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

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

***Abstract*—The Industrial Control Protocol (ICP) is the cornerstone of the communication of the Industrial Control System (ICS), and its importance to the ICS is self-evident. If the security of ICPs can be guaranteed, the security of ICSs can be guaranteed to a large extent. However, due to the strong diversity of the industrial control environment applicable to ICPs which causes the meaning of the same parameter in different applications varies greatly, it is difficult for testers to formulate a series of universal security rules. Therefore, fuzz testing (fuzzing) has already become the main methods of detecting vulnerabilities in ICPs. However, the process of fuzzing relies heavily on the specificatnalyze and understand the speion of ICPs. And it will take a lot of time and manuzal engineering to acification. In this paper, we proposed a new simple and smart sequence generation neural network framework based on Improved Self-Attention Generative Adversarial Networks (SAGAN), called SAGANFuzzer, to solve the problems. Moreover, we put forward a series of performance metrics in the field of fuzzing to evaluate different models. Compared with traditional methods, our framework can generate massive fake but plausible test protocol message automatically in a short time without protocol specification; compared with other deep learning works for fuzzing, our framework can not only test multi-dimensional input effectively but also increase the probability of vulnerabilities being triggered while being more parallelizable and requiring significantly less time to train. We evaluate its performance by testing several typical ICPs, including MQTT and Modbus. Extensive experiments demonstrate significant improvements over test effectiveness and efficiency. ######## Here can specify specific values.**

***Index Terms*—deep adversarial learning, self-attention, convolution neural networks, fuzz testing, industrial control protocol**

I. Introduction

With the arrival of the Industry 4.0 and Internet+, some new trends in the development of manufacturing. Strategic plans have been put forward to build the next generation of manufacturing. These strategies intend to empower the industry through information technology, such as optimizing the processes, reducing the costs and increasing the efficiency, to unlock greater productivity. The ICS, as the basic component of the automated production of the national economy and the people's livelihood, is a key part for national safe strategy. The interaction between the ICS and its network environment becomes more frequent. The system is facing increasing external security threats. When applying the ICP into actual production, it is necessary to discover potential protocol vulnerabilities in time and prevent them in advance.

Currently, applying traditional fuzz testing techniques to detect loopholes in ICPs is an effective method. However, there are some limitations on applying the techniques: (i) High demand for the testers. The tester is required to design appropriate test cases according to the communication protocol specification running in the system. (ii) Long test cycle. The entire testing process will last a long time. It is impossible to complete the test task efficiently when it is in urgent needs. (iii) No versatility. 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, reducing the workload and breaking the border of different protocols to achieve the generality. However, poor machine learning algorithms not only consume a lot of computing resources during the training of the model, but also tend to generate a large number of error-formatted protocol message sequences after the training, resulting 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 there is a a challenge in distinguishing what of them are vulnerabilities and how to find real vulnerabilities from these crashes. In order to explore the the balance between normal and abnormal exceptions, 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 for sequences of discrete, and introduce a penalty term to ensure the diversity of generated test case.

(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 allow significantly more parallelization which requires significantly less time to train, makes it superior in quality of generating fake but plausible testing data in the process of model design.

(3) On top of the approach, we build a universal fuzzing framework based on improved SAGAN, called SAGANFuzzer, 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. Judge this part exist or not: Moreover, in the process of sampling, this paper introduces a series of anti-random strategies to improve the probability of anomalies throwing

(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

To evaluate its performance, we apply it to test several ICPs. The experimental results show that our method has better performance than and can obtain excellent test results in different ICPs. In terms of test effectiveness and efficiency, the expected results are achieved.

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

II. Related Works

The prototype of fuzzing is Random Testing. Duran and Ntafos [2] are pioneers in this field. In the early research, arbitrary input was used to test computer programs and good results were achieved. The concept of fuzzing was formally proposed by Miller et al. [3] in 1990 and was initially applied to the testing of Unix programs. Fuzzing is to conduct 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, analyze the system according to abnormal behavior, and find the exploitable vulnerabilities in the system.

Since then, a variety of different techniques were proposed to improve the efficient of fuzzing. These techniques include static analysis [4] [5], dynamic analysis [6] [7] [8]. Because of its effectiveness, fuzzing has been studied in the network protocol testing field to enhance the reliability of the computer network [9] [10] [11] [12] [13] [14]. And some of these works are for ICPs and have made a certain contribution to the improvement of the safety and security of ICPs. Greg Banks et al. proposed a fuzzy test tool called SNOOZE [15] for stateful network protocols. Devarajan et al. [16] released a fuzzy test module based on Sully tool for Modbus, DNP3 and other industrial control protocols. The Peach Fuzzing [17] designed a Modbus-TCP fuzzy test tool called MTF which builds the test model by Modbus official instructions. Boofuzz [18], proposed by Pereyda et al., aims not only for numerous bug fixes but also for extensibility in the field of network protocol fuzzing.

However, fuzzing test still faces many challenges, such as how to mutate seed inputs, how to increase code coverage, and how to effectively bypassing verification [19]. With the advancement of machine learning in the field of cybersecurity, it has also been adopted by many studies for vulnerability detection[20] [21] [22], including the applications in fuzzing [23] [24] [25] [26] [27]. ICPs have much in common features such as a short-data frame and no encryption. They are designed to meet the real-time requirements of ICSs. As was expected, some studies have incorporated fuzzing algorithms of ICPs based on deep learning into the fuzzing process of ICPs [28] [29]. Machine learning technology is introduced into fuzzing to provide a new idea for solving the bottleneck problems of the traditional fuzzing technology and also makes the fuzzing technology intelligent. Using machine learning for fuzzing testing will become one of the critical points in the development of vulnerability detection technology with the explosive growth of machine learning research. These efforts all contribute to deep learning based fuzzing.

However, 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.

Self-attention [30], a attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence, allows attention-driven, long-range dependency modeling for generation tasks. GAN, with a discriminative network D (discriminator) learns to distinguish the reality of a given data instance, and a generative network G (generator) learns to confuse discriminator, can generate fake but plausible data via an adversarial learning process. Inspired by these, 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 SAGANFuzzer. And with using Wasserstein distance and a gradient penalty (WGAN-GP) [31], our framework can not only solve the problem of poor performance for discrete data but also ensure the diversity of generated test case.

III. SAGANFuzzer Framework

In this section, we first acquire a large number of communication data from the ICS to be tested and then preprocess the data as the training data of the deep adversarial learning model established. Secondly, a generation model and a discriminant model are designed to generate the adversarial network, and a specific model is obtained by training and retraining the model with the acquired data. Finally, a series of performance metrics are put forward to evaluate our model.

A. Preprocessing of ICP Message Frames

In order to get diverse test cases, reach high code coverage and desirable fuzzing results, the data preprocessing is a necessary part. There are two steps in preprocession: **Message Frame Clustering** and **Data Conversion**.

*1) **Message Frame Clustering**:* After message frames collection, there are many different length message frames of MQTT. We adopted the Length Clustering method to preprocess the data frames. It is noted that most message frames with the same length share the same frame type. First, we gather

the data frames which have the same length. In order not to get too many groups of data frames, groups whose quantity is less than a threshold (e.g. 5000) will be move the longer adjacent group, while groups contains more data than threshold are left unchanged. In such a way, there may be message frames with different length in one group. After clustering, we select the max length of messsage frames for every group and add special token 'u' at the end of message frames whose length is less than max length.

*2) **Data Conversion**:* Raw data frames of most protocols are hexadecimal, which can not be fed into the model directly. So after clustring, we will map the hexadecimal data frames into the digital vectors. The vocabulary we use is as followed:

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

The size of vocabulary is 17, there are ten digits and seven characters according to the hexadecimal message frames and the supplement token 'u'. Based on the vocabulary, we convert our message frames $x \in R^{seq\_size \times 1}$ 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 will be the input of the discriminator which will be disscused in 4.2.

B. Model Design

In this part, we describe our test case generation model in detail. As the figure shows, 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 evaluates 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 Wassertein distance between real message frame distribution and our fake message frame distribution will be shorten and our model grasps the knowledge of the protocol grammar.

*1) **Generator**:* The structural of generator is as followed. 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 a hyper parameter. 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 (NCR) $\tilde{z} \in \mathbb{R}^{ss \times dm}$ where ss is the max sequence length of current group and dm is number of feature dimension.

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

The output block is a fully connected layer and a softmax layer. After the conversion block and the transformer block, the generator will not output the message frame data directly, instead, it outputs a probability vectors $\tilde{x} \in \mathbb{R}^{ss \times vs}$. Where vs is the size of vocabulary, which is 17 in our situation. If we want to check the quality of message frames generated by generator, we can apply $argmax$ function on $\tilde{x}$ out of the model and then translate to hexadecimal data.

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 sequence. For example, the input words of the Transformer [Attention is all you need] is converted into word embedding. In our model, we feed NCR to our transformer block. Why don't we use word embedding? There is a problem that the outout of noise block is random noise vector, it is hard to get the corresponding word embedding. In the NLP tasks, discrete data is usually regarded as the results 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 function 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 transfromer block. NCR looks like word embedding, but it is fake. NCR maps the one-dimension noise vector to high dimensional space information. Comparing with one-hot vector, NCR can cover more information of high dimension space because most digits of ont-hot vector are zero. 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. What's more, the self-attention layer in the transformer block supports parallel training, which can accelerate the training process. The loss function of generator is

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

where $\tilde{x} \in \mathbb{R}^{ss \times vs}$ 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.

*2) **Critic**:* It looks like discriminator in normal GAN model, but it is not a clssfier, we call it critic here, which is the same as WGAN[cite WGAN]. The structure of critic is as Fig.2 shows. The critic includes five conv1d layer, five self-attention layer and one fully connected layer. For the critic, there are two types of inputs. One is one-hot vector from the real message frames, and another is the ouputs of the generator which represents the fake message frames. The second dimension of the output of generator means the probability of each word

in vocabulary, and one-hot vector can also be understanded as a special case of it, that is, the probability of one word is 1 and the probability of other words is 0. From the table, we can see that the first conv1d layer changes the second dimension of inputs from propability of word to the feature representation, which is the same as dimension of NCR. It is noted that every conv1d layer followed by a self-attention layer. There is a little difference that the filter size of conv1d is not the same. The filter size of conv1d fs are (dm, w) where ds is the number of features and $w \in 1, 2, 3, 4, 5$. It is well known that protocols have message header, which specify the content of the message body in the message frame. More simply, the front part of message frames affects the back part of the message frame.

If we don't know the grammer of the protocol, it is hard to distinguish the boundary of two parts. If the model have learned which part is the message header, it can pay more attention to the message header part. Here we use conv1d layer to figure out. Due to different protocols have different length of the message header, we change the filter size of conv1d to capture this information better. As for the content in the message header, there are several parts affect several parts content in the message body. As Fig.4 illustrated, message header are divided into three parts and each part will affect the corresponding part in the message body. In order to capture how the content in the message header affects specific part of the message body, we use self-attention layers. Self-attention considers the attention weight of every position of the input and is good at learning long-range dependencies. With the boundary infromation leaned by conv1d layers, self-attention layer will pay more attention to the correlation between the content in the message header and message body. After repeated self-attention and conv1d layers five times, there is a reshape layer and a fully connected layer. Finally, the critic will output a scalar score. The critc loss is:

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

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 1-Lipschitz constraint, we adopted the solution of WGAN-GP. We use gradient penalty instead of the weight clipping to enforce the 1-Lipschitz constraint. It makes training more stable and therefore easier to train.

3) ***Training strategy****:* Once the model design is completed, we begin to train our model. Under normal circumstances, the training of the GAN is adversarial, which means the generator and discriminator (critic) should be on the same level. If the imbalamce is too serious, the another one will learn nothing. And this is the reason why the GAN model is difficult to train and the training is unstable. Due to the properties of the Wassterin distance, we can train the critic better first, and then narrow the wassterin 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. We use Adam Optimizer with learning rate 0.0004, beta1 0.5 and beta2 0.9 in the generator and the critic. In order to judge the convergence of the model, we use the following equation, which is the approximate value of the wassertein distance.

$$wd = \mathop{\mathbb{E}}_{x\sim\mathbb{P}_r}[D(x)] - \mathop{\mathbb{E}}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] \tag{4}$$

Even we want to generate the fake message frames that share a high similarity with the real message frames, the ulmate goal of our model is to achieve effective fuzzing results and identify as many bugs as possible. In order to obtain the desired results, we need some differences between the real data and the fake data. So we not only save the final version of the model, which is converged, but also the intermediate generator model. We save the generator model every 10 training epoch deliberately. In such a training strategy, we can achieve the goal of the high code coverage and deeper testing depth.

C. Performance Metrics

Some quantitative criteria [32] [33] [34] have emerged only recently assessing 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: [33] [34] in the field of fuzzing based on deep learning [35] 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.

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* in the same data set. 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 characters for correctly generated test case packages versus non-essential 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:

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

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

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

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

$$Recall = \frac{TP}{TP + FN} \tag{9}$$

$$F - measure = 2 \times precision \times \frac{Recall}{Precision + Recall} \tag{10}$$

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

*2) **Effectiveness of Vulnerability Detection (EVD)**:* EVD refers to the ability to trigger anomalies of the test target based on the test target accepting the test data. The effectiveness evaluation of the fuzzing methods can be divided into two aspects: testing depth and code coverage. The effectiveness of fuzz testing depends predominantly on the testing depth and high code coverage. Therefore, we propose the effectiveness evaluation metric EVD of the model from the above two aspects.

When the generated data can be accepted by the test target, it indicates that the generated data is similar with real data in terms of format, embodying that the model has the ability to learn the format of data from the real world. And this phenomenon also reflects a certain depth of testing. Our ultimate goal is to trigger as many anomalies as possible to the ICS to find vulnerabilities, so we need to take two sub-metrics into account when we finally calculate the testing depth for EVD: *Acceptance Rate of Test Cases (ARTC)* and *Ability of Vulnerability Detected (AVD)*:

a. ARTC. ARTC refers to the acceptance rate at which the generated test data is received by the test target when it is sent to the test target. The specific formula is as follows:

$$ARTC = \frac{N_{accepted}}{N_{all}} \times 100\% \tag{11}$$

where $N_{accepted}$ is the total number of test cases recognized and N_{all} is the total number of test cases sent.

b. VDE. VDE refers to the ability to trigger anomalies of the test target when the generated test data is sent to the test target. We define tVDE as follows:

$$VDE = \frac{N_{anomalies}}{N_{all}} \times 100\% \tag{12}$$

where $N_{anomalies}$ is the total number of test cases which trigger anomalies and N_{all} is the total number of test cases sent.

Diversity of Generated Cases (DGC) is a sub-metric proposed as the extent of code coverage of fuzz testing in EVD. It refers to the ability to maintain the diversity of the training data. Furthermore, The diversity-based approach is a useful test case selection criterion for code coverage [36] [37] . More diverse generated test data frames are likely to cause more exceptions. And this indicator focuses on the number of message types in the generated data, see the following formula:

$$DGC = \frac{N_{gen\_categories}}{N_{all\_categories}} \times 100\% \tag{13}$$

where $N_{gen\_categories}$ is the total number of message categories in the generated data frames, and $N_{all\_categories}$ is the total number of message categories in the training set.

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 is converted into a value that can be directly added or subtracted. We normalize the submetrics and add them up to get the comprehensive scores for EVD:

$$EVD = (\frac{ARTC_i - ARTC_{\min}}{ARTC_{\max} - ARTC_{\min}} + \frac{VDE_i - VDE_{\min}}{VDE_{\max} - VDE_{\min}} + \frac{DGC_i - DGC_{\min}}{DGC_{\max} - DGC_{\min}}) \times \frac{100}{3} \tag{14}$$

setting the values of different effectiveness sub-metrics of a model as $\{submetric_1, submetric_2, ..., submetric_n\}$, then values of ARTC, VDE and DGC of the ith model are: $ARTC_i$, VDE_i and DGC_i. $submetric_{max}$ and $submetric_{min}$ are respectively the maximum and minimum values of the corresponding sub-metric.

*3) **Efficiency of Vulnerability Detection (EFVD)**:* EFVD is an important indicator of the model Efficiency. On the one hand, EFVD refers to the *Training Time (TT)* it takes to train a model.

a. TT. Short TT correspondingly improves the efficiency of testing.

On the other hand, it reflects the *Time of Anomalies Triggered (TAT)* after sending a fixed number of test cases.

b. TAT. TAT refers to the time interval from the first request (t_1) to the third check out of the anomalies (such as error code 3, no response from the server, etc.) for each fuzzy test as a formula for TAT in this paper:

$$TAT = t_{3_{\text{th}}\_\text{anomaly}} - t_1 \tag{15}$$

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, in this study, we only focus on the efficiency of the method. The specific formula is as follows:

$$EFVD = (\frac{TT_i - TT_{\min}}{TT_{\max} - TT_{\min}} + \frac{TAT_i - TAT_{\min}}{TAT_{\max} - TAT_{\min}}) \times \frac{100}{2} \tag{16}$$

similar to aforementioned effectiveness sub-metrics, the values of TT and TAT of the ith model are: TT_i and TAT_i. TT_{max} and TAT_{min} are respectively the maximum and minimum values of the corresponding sub-metric.

IV. Experiment and Result Analysis

In this section, MQTT and Modbus are choosed as our test targets from a variety of ICPs in the experiment. We generate a lot of test cases using the generative model. Then the generated data is used to stress test the target and trigger anomalies of the ICP. Finally, according to the anormalies triggered, vulnerabilities in the anomalies are submitted for the improvement of the ICP after analysis to find the reason for the target anomalies. We evaluate the effectiveness of the proposed method by experimentation.

A. Environment construction of MQTT and Modbus

To show effectiveness and efficiency of our framework, we apply it to test MQTT, one of the mainstream ICPs. To indicate the versatility of our method, another ICP, Modbus, is also used to test.

1) MQTT: MQTT (Message Queuing telemetry Transport) protocol is an application layer ICP based on publication and subscription under the ISO (Intemetional Organization for Standardization) standard and works on the TCP/IP protocol cluster. It is simple to implement, provides the transmission data with the quality of service, has the characteristics of lightweight, high bandwidth utilization efficiency, data independence and so on. The message format of MQTT is illustrated in Fig. 1.

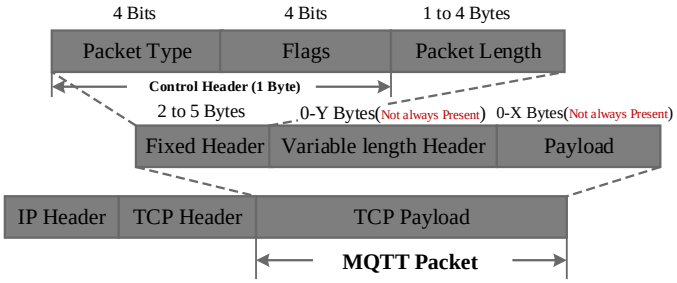

Fig. 1. Message format of MQTT

In the experiment, we utilize Mosquitto v1.5.8 [38] as the open source message broker of message push ICP MQTT v3.1 and Paho-MQTT v1.0.2 [38] as the MQTT client.

2) Modbust-TCP: There are many variants of Modbus, including Modbus-TCP, Modbus-UDP and so on. We choose Modbus-TCP as another fuzzing ICP in this study. It uses master-slave communication mode, in which the master communicates with the slave by sending and receiving data frames [39].

B. Evaluation Setup

1) Experimental Environment: Tensorflow, one of the popular deep learning framework in the industrial community, is adopted to implement the model architecture. To improve the training efficiency, we train the model 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. When launching an attack, the simulators run on another machine with 4 processors (Intel (R) Core (TM) i5-5300U CPU@2.30GHz) 16.00GB memory (RAM) and 64-bit Microsoft Windows 10 Professional Operating System, x64-based processor.

2) Model Training Setting: As for the parameter setting, we initialize all weights from zero-centered Normal distribution with a standard deviation of 0.2, which can increase diversity to a certain extent. The mini-batch size is set to 32 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 G and D is set to 0.0001 and 0.0004 in the Adam optimizer. As to the check point in generator model, the interval of model saveing frequency is set to 10. We train the models for 100 epochs and save the generator model for every 10 epochs to get plentiful test cases.

C. Fuzz Testing The Target ICP

With the trained model, we generate test cases, input the system under test, and 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, we record and restart the test process at the current point. At the end of the entire test, we analyzes the causes of antecedent anomalies and the logs of server systems to find the running status of potential vulnerabilities.

1) Training Data: Training data in deep learning significantly influence the model training. In the experiment, training data about the two industrial control protocol is collected separately.

a. MQTT. In order to capture the MQTT communication data, a MQTT network environment based ICS is prepared as illustrated in Fig. 2. Wireshark [40] is applied to grab packets during communication. These grabbed frames serve as the training data for the SAGANFuzzer framework. After fetching the connection information between the legal clients and the MQTT broker, we construct a disguised MQTT client to send the same connection information to connect to the broker in the IOT network with no authentication security measures. The disguised MQTT client launches the attack by sending fake MQTT messages which are generated by fuzzy test models.

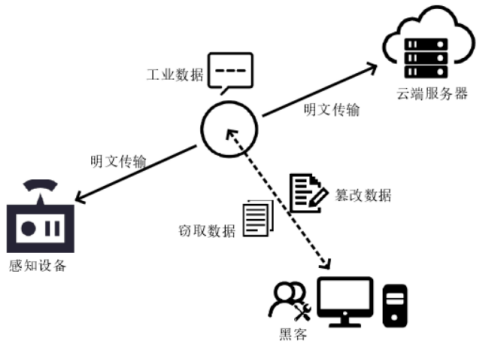

图 2-7 信息窃取和篡改

Fig. 2. MQTT enviroment

b. Modbus-TCP. Pymodbus [41], a python package that implements the Modbus protocol, is used to generate the training data frames. Enough different types of data frames can be generated quickly via Pymodbus, which is practical and convenient. In the experiment, a Modbus-TCP implementation, Modbus RSSim v8.20, is applied as the fuzzing targets. The generated test cases are sent to the Modbus-TCP implemen-

TABLE I
PERFORMANCE METRICS COMPARISON

Dataset (number of training cases)	Methods	Year	F-measure	SE	SP	AC
10,000	GPF	2007	–	0.7252	0.9798	0.9474
	CNN-1D model	2014	0.6701	0.8437	0.9743	0.9626
	LSTM-based model	2018	0.6300	0.9563	0.9007	0.9254
	WGAN-based model	2019	0.7960	0.9696	0.9115	0.9165
	SAGANFuzzer	2020	0.8208	0.8274	0.9775	0.9608
500,000	GPF	2007	–	0.7720	0.9730	0.9510
	CNN-1D model	2014	0.8373	0.8270	0.9842	0.9690
	LSTM-based model	2018	0.8388	0.8203	0.9856	0.9700
	WGAN-based model	2019	0.8396	0.8108	0.9871	0.9706
	SAGANFuzzer	2020	0.8502	0.8538	0.9878	0.9771

tation by a python script over sockets to test the effects in simulated applications.

2) Logging and Evaluation: Logging is the basis for further analysis of model performance and fuzzing effectiveness. By analyzing logs of communication process is an effective method to find ICPs' anomalies. Some vulnerabilities may be manifested according to the obvious abnormal behavior of the system, and some behaviors need to be further analyzed. Based on the statistical analysis of the log file, we evaluate experimental results. Furthermore, we artificially analyze specific anomalies to get more details. Test data that causes target anomalies will be recollected and put into the training data set again. Data augmentation and value mutation operation will be applied to these data before putting it into the retraining data set. The retrained model with these mutated data can improve the capability of the model to detect vulnerabilities.

D. Result Analysis

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

1) Statistical Analysis And Results: The widely used General Purpose Fuzzer(GPF) [42], 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 500,000 generated test cases to MQTT implementations through the TCP/1883 port.

According to the three aforementioned evaluation metrics, the effectiveness and efficiency of our fuzzing framework SAGANFuzzer and fuzzers in the control group are evaluated and represented graphically. Details are as follows.

a. F-measure. Table I 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 seen from the table that the algorithm in this paper has Higher scores 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 CNN-1D model and LSTM-based seq2seq model is higher than that of the method in this paper, 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 0.37% higher than the WGAN-based model, and the Sensitivity of this method is 4.82% higher than that of the WGAN-based model. On the dataset of 500,000 training data, our method is 0.27% higher in F-measure and 2.4% 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.

c. EFVD.

REFERENCES

[1] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[2] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," *IEEE transactions on Software Engineering*, no. 4, pp. 438–444, 1984.

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

[4] S. Sparks, S. Embleton, R. Cunningham, and C. Zou, "Automated vulnerability analysis: Leveraging control flow for evolutionary input crafting," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE, 2007, pp. 477–486.

[5] J. Kinder, F. Zuleger, and H. Veith, "An abstract interpretation-based framework for control flow reconstruction from binaries," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2009, pp. 214–228.

[6] M. Höschele and A. Zeller, "Mining input grammars from dynamic taints," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 720–725.

[7] O. Bastani, R. Sharma, A. Aiken, and P. Liang, "Synthesizing program input grammars," *ACM SIGPLAN Notices*, vol. 52, no. 6, pp. 95–110, 2017.

[8] F. M. Kifetew, R. Tiella, and P. Tonella, "Generating valid grammar-based test inputs by means of genetic programming and annotated grammars," *Empirical Software Engineering*, vol. 22, no. 2, pp. 928–961, 2017.

[9] D. Aitel, "An introduction to spike, the fuzzer creation kit," *presentation slides), Aug*, vol. 1, 2002.

[10] P. Amini and A. Portnoy, "Sulley fuzzing framework," 2010.

[11] S. Gorbunov and A. Rosenbloom, "Autofuzz: Automated network protocol fuzzing framework," *IJCSNS*, vol. 10, no. 8, p. 239, 2010.

[12] M. Eddington, "Peach fuzzing platform," *Peach Fuzzer*, vol. 34, 2011.

[13] P. Tsankov, M. T. Dashti, and D. Basin, "Secfuzz: Fuzz-testing security protocols," in *2012 7th International Workshop on Automation of Software Test (AST)*. IEEE, 2012, pp. 1–7.

[14] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing." in *NDSS*, 2018.

[15] G. Banks, M. Cova, V. Felmetsger, K. Almeroth, R. Kemmerer, and G. Vigna, "Snooze: toward a stateful network protocol fuzzer," in *International Conference on Information Security*. Springer, 2006, pp. 343–358.

[16] G. Devarajan, "Unraveling scada protocols: Using sulley fuzzer," in *Defon 15 Hacking Conference*, 2007.

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

[18] J. Pereyda, "boofuzz: Network protocol fuzzing for humans," *Accessed: Feb*, vol. 17, 2017.

[19] J. Li, B. Zhao, and C. Zhang, "Fuzzing: a survey," *Cybersecurity*, vol. 1, no. 1, p. 6, 2018.

[20] G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, and L. Mounier, "Toward large-scale vulnerability discovery using machine learning," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, 2016, pp. 85–96.

[21] F. Wu, J. Wang, J. Liu, and W. Wang, "Vulnerability detection with deep learning," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 2017, pp. 1298–1302.

[22] B. Chernis and R. Verma, "Machine learning methods for software vulnerability detection," in *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*, 2018, pp. 31–39.

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

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

[25] J. Wang, B. Chen, L. Wei, and Y. Liu, "Skyfire: Data-driven seed generation for fuzzing," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 579–594.

[26] D. She, K. Pei, D. Epstein, J. Yang, B. Ray, and S. Jana, "Neuzz: Efficient fuzzing with neural program smoothing," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 803–817.

[27] X. Liu, R. Prajapati, X. Li, and D. Wu, "Reinforcement compiler fuzzing," 2019.

[28] Z. Hu, J. Shi, Y. Huang, J. Xiong, and X. Bu, "Ganfuzz: a gan-based industrial network protocol fuzzing framework," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*, 2018, pp. 138–145.

[29] Z. Li, H. Zhao, J. Shi, Y. Huang, and J. Xiong, "An intelligent fuzzing data generation method based on deep adversarial learning," *IEEE Access*, vol. 7, pp. 49 327–49 340, 2019.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[31] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, 2017, pp. 5767–5777.

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

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

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

[35] K. Shmelkov, C. Schmid, and K. Alahari, "How good is

my gan?" in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 213–229.

[36] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 1, pp. 1–42, 2013.

[37] D. Mondal, H. Hemmati, and S. Durocher, "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)*. IEEE, 2015, pp. 1–10.

[38] R. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.

[39] A. Swales *et al.*, "Open modbus/tcp specification," *Schneider Electric*, vol. 29, 1999.

[40] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[41] G. Collins, "Pymodbus documentation," 2013.

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

[43] E. Paho-MQTT, "Mqtt-sn software," 2018.