HexGANFuzzer: 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] Zhihui Li[4]

***Abstract*—The Industrial Control Protocol (ICP) is the cornerstone of the communication of the Industrial Control System (ICS). If the security of ICPs can be guaranteed, the security of ICSs can be guaranteed to a large extent. However, the industrial control environment applicable to ICPs has a strong diversity, 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 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. Compared with traditional methods, our framework can generate massive fake but plausible test protocol messages automatically in a short time without protocol specifications. Compared with other deep learning works for fuzzing, our framework can not only increase the probability of triggering vulnerabilities but also be more parallelizable and require significantly less time to train. 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.**

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

I. Introduction

With the arrival of Industry 4.0 [1] 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. These strategies intend to empower the industry through information technology to unlock greater productivity. 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 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[2] 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 [2], 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

To evaluate the performance of our model, we test it on 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 the 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 are pioneers in this field [3]. 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. [4] in 1990 and was initially applied to the testing of Unix programs. 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, analyze the system according to abnormal behavior, and find the exploitable vulnerabilities in the system.

Since then, a variety of different techniques have been proposed to improve the efficiency of fuzzing. These techniques include static analysis [5] [6], dynamic analysis [7] [8] [9]. Because of its effectiveness, fuzzing has been studied in the network protocol testing field to enhance the reliability of the computer network [10] [11] [12]. citetsankov2012secfuzz [13] 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 [14] for stateful network protocols. Devarajan et al. [15] released a fuzzy test module based on the Sully tool for Modbus, DNP3 and other industrial control protocols. Voyiatzis et al. [16] designed a Modbus-TCP fuzzy test tool called MTF which builds the test model by Modbus official instructions. Boofuzz [17], 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 still faces many challenges, such as how to mutate seed inputs, how to increase code coverage, and how to effectively bypass verification [18]. With the advancement of machine learning in the field of cybersecurity, it has also been adopted by many studies for vulnerability detection[19] [20] [21], including the applications in fuzzing [22] [23] [24] [25] [26]. 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 [27] [28]. 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. With the explosive growth of machine learning research, using machine learning for fuzzing will become one of the critical points in the development of vulnerability detection technology. 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.

Self-attention [29], an attention mechanism relating to different positions of a single sequence, allows attention-driven, long-range dependency modeling for generation tasks. GAN, with a discriminative network D (discriminator) which learns to distinguish the reality of a given data instance, and a generative network G (generator) which 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 HexGANFuzzer. And with using Wasserstein distance and a gradient penalty (WGAN-GP) [30], our framework can not only solve the problem of poor performance for discrete data but also ensure the diversity of generated test cases.

III. HexGANFuzzer 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 generative 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, data preprocessing is a necessary part. There are two steps in preprocessing: **Message Frame Clustering** and **Data Conversion**.

*1) **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. 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 moved to the longer adjacent group, while group contains more data than the threshold are left unchanged. In such a way, there may be message frames

with different lengths in one group. 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.

*2) **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:

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

The size of the vocabulary is 17, there are ten digits and seven hexadecimal characters according to the hexadecimal message frames and the supplement token 'u'. 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.

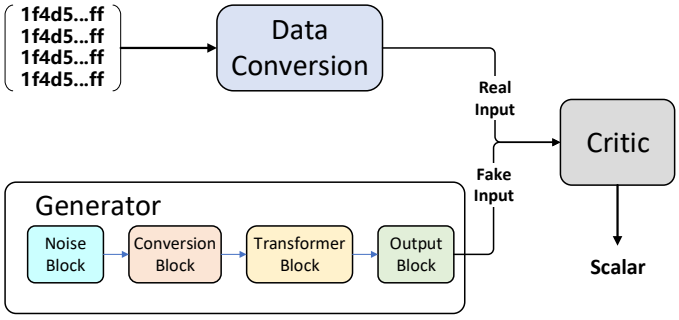

Fig. 1. The architecture of HexGANFuzzer

B. Model Design

In this part, we describe our test case generation model in detail. As Fig. 1 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 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.

*1) **Generator**:* The structure of the generator is as followed in Fig. 2. 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 (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.

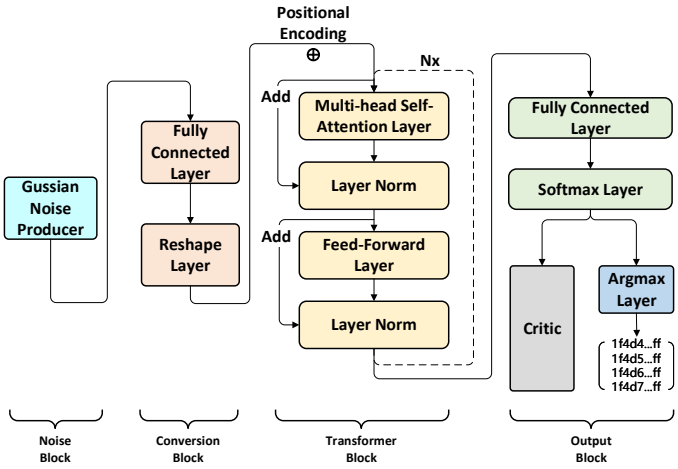

Fig. 2. Generator Network

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 [29]. 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 conversion block and the transformer 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. For example, the input words of [29] are converted into word embedding. 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. NCR maps the one-dimension noise vector to high dimensional space information. Compared with the one-hot vector, NCR can cover more information of high dimension space.

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.

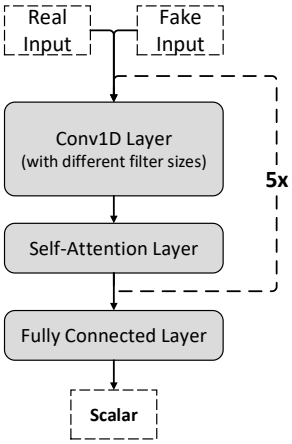

Fig. 3. Generator Network

2) ***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 [2]. The structure of the critic is as Fig. 3 shows. 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.

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:

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

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 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.

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 imbalance is too serious, 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 Wasserstein distance, we can 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. Adam Optimizer was used both in the generator and the critic. 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.

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

Though the model wants to generate the fake message frames that share high similarity to the real message frames, the ulmate goal is to achieve effective fuzzing results and identify as many bugs as possible. In order to obtain the desired results,

there should be some differences between the real data and the fake data. So we not only save the final version of the model, which is convergent, but also the intermediate generator model. The model was 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.

C. Performance Metrics

Some quantitative criteria [31] [32] [33] 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 [32] [33] in the field of fuzzing based on deep learning [34] 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*. 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:

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

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

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

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

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

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

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

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, embodying that the model has the ability to learn the format of data frames from the real world. This phenomenon also 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)*:

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

b. AVD. AVD refers to the ability of triggering anomalies for the test target when the generated test data is sent to the test target. We define AVD as follows:

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

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 for fuzzing in EVD. It refers to the ability to maintain the diversity of the generated data. The diversity-based approach is a useful test case selection criterion for code coverage [35] [36]. Furthermore, 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 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 normal-

ize aforementioned sub-metrics 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 $\{sm_1, sm_2, ..., sm_n\}$, then values of ARTC, AVD and DGC of the ith model are: $ARTC_i$, AVD_i and DGC_i. sm_{max} and sm_{min} are respectively the maximum and minimum values of the corresponding sub-metric.

*3) **Efficiency of Vulnerability Detection (EFVD):*** EFVD is an important metric of model Efficiency. On one hand, it refers to the *Training Time (TT)* required to train a model. On the other hand, it indicates the *Time of Anomalies Triggered (TAT)* after sending a fixed number of test cases.

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

b. TAT. 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 ($t_{3_{\text{th\_anomaly}}}$) . The following formula is for TAT in this paper:

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

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, 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. The specific formula is as follows:

$$EFVD = [(1 - \frac{TT_i - TT_{\min}}{TT_{\max} - TT_{\min}}) + (1 - \frac{TAT_i - TAT_{\min}}{TAT_{\max} - TAT_{\min}})] \times \frac{100\%}{2} \tag{16}$$

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 chosen as our target protocols from a variety of ICPs in the experiment. We generate large amounts of test cases by our generative model. Then the generated data is used to stress test the target and trigger anomalies of the ICP. Finally, according to the anomalies triggered, vulnerabilities in the anomalies will be submitted for the improvement of the ICP after analysis. We evaluate various aspects of performance of our method by experimentation.

A. Environment construction of MQTT and Modbus

To show the 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 (International 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. 4.

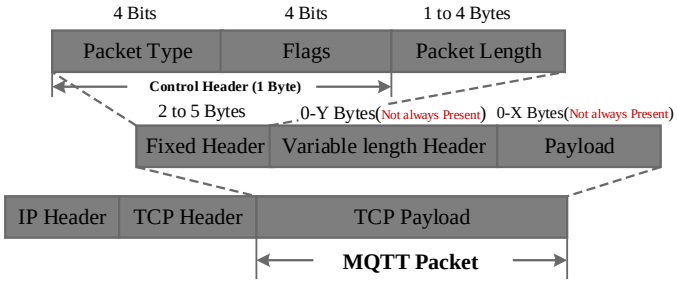

Fig. 4. Message format of MQTT

In the experiment, we utilize Mosquitto v1.5.8 [37] as the open-source message broker of message push ICP MQTT v3.1 and Paho-MQTT v1.0.2 [37] 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 slaves by actively sending data frames [38].

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 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

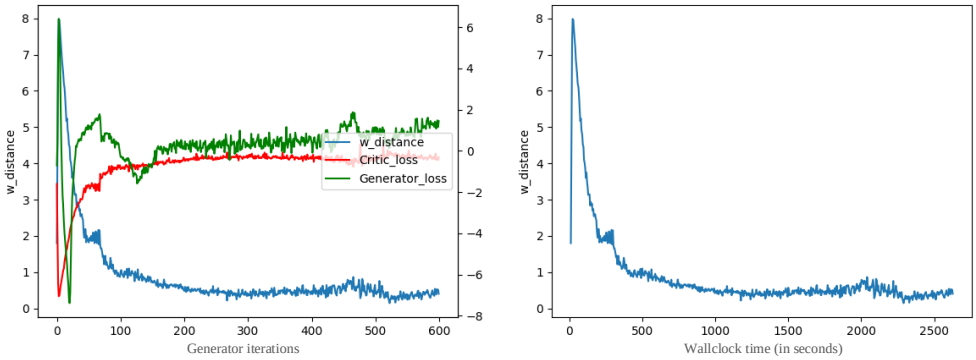

Fig. 5. Dataset(500,000 test cases) W-distance over generator iterations (left) or wall-clock time (right) for HexGANFuzzer

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.

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 analyze 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 model training. In the experiment, training data about the two industrial control protocols is collected separately.

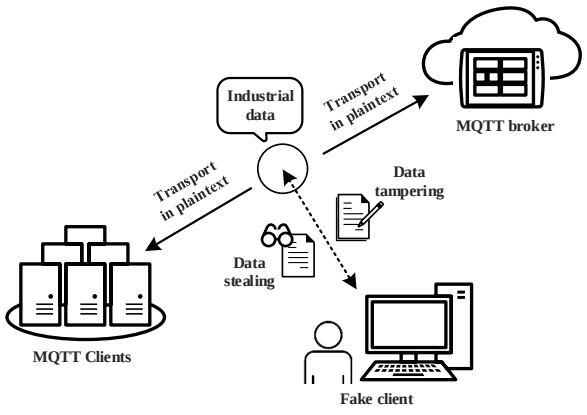

Fig. 6. MQTT enviroment

a. MQTT. In order to capture the MQTT communication data, an MQTT network environment based ICS is prepared as illustrated in Fig. 6. Wireshark [39] is applied to grab packets during communication. These grabbed frames serve as the training data for the HexGANFuzzer 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.

b. Modbus-TCP. Pymodbus [40], 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 implementation by a python script over sockets to test the effects in simulated applications.

2) Model Training: 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. We design our model architecture based on these architectural constraints and improve training speed and sample quality. 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 of training in Figure 5.

3) Logging and Evaluation: Logging is the basis for further analysis of model performance and fuzzing effectiveness. By analyzing logs of the 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

TABLE I
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**

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 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.

1) Statistical Analysis And Results: The widely used General Purpose Fuzzer(GPF) [41], 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 effectiveness and efficiency of our fuzzing framework HexGANFuzzer 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 observed 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 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. 7. 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 $<$ GAN-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. Due to the unsupervised characteristic of generative adversarial algorithms, it may result in generating a large number of malformed protocol message sequences, leading to normal throwing which may

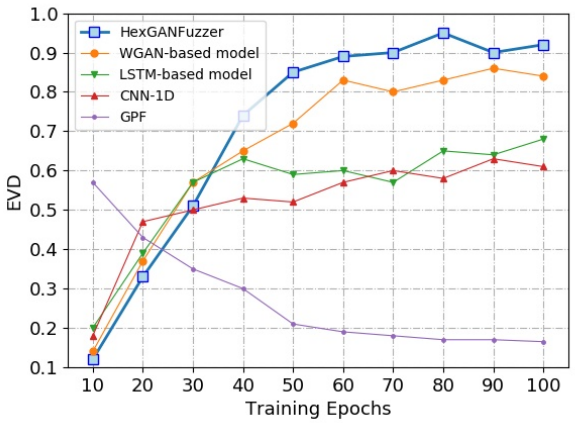

Fig. 7. EVD changes with the training epochs

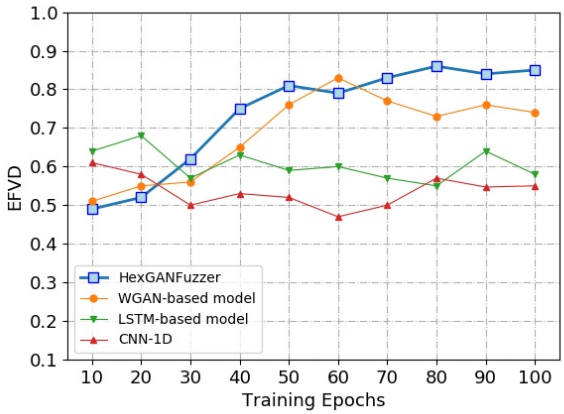

Fig. 8. EFVD changes with the training epochs

cause EVD can not be further promoted. It's worth going into detail that our trained model can effectively detect kinds of anomalies as presented in Table II, which shows we are likely to achieve the higher code coverage and the stronger ability of our model has to detect anomalies.

TABLE II
TRIGGERED ANOMALIES AND TRIGGERED FREQUENCY OF HEXGANFUZZER

Triggered Anomalies	Frequency (Times)	ATITA (Mins)
Using abnormal function code	104	5.78
Data length unmatched	121	4.52
Integer overflow	21	13.46
Broker memory overflow	2	285.05
Unknown attack	53	10.64

where Frequency represents the number at which the specified type of anomalies is triggered by the model during test time and average time interval of triggered anomalies (ATITA) is the quotient of the number of triggered anomalies and test time.

c. EFVD. There is a huge difference in the training time of different GPF and the time of anomalies triggered by GPF is constant under different iterations, so it is not discussed in this part. The normalization is applied to EFVD, which is the same as the EVD. The testing depth has increased as illustrated in Fig. 7 at the expense of reducing the code coverage of fuzz testing. But what makes a difference, rather, is that high test efficiency is maintained on the premise of attaining high EVD rates. The variation trend of EFVD of different deep learning based models is presented in Fig. 8. It can be seen from the figure 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.

2) Potential Vulnerabilities Found: In this part, we mainly talk about one exception that we analyzed, which is ***Buffer Exception***.

When the mosquitto broker receives message frames from the client, it will store data into the buffer. And it will parse the message frames based on the MQTT protocol grammar without checking the whole message length. That is, 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.

As the Fig. 9 shows, in $step1$, we send the generated message frames formed by green and red box part to the broker, where the green box part is a complete MQTT message frame and the red box part is additional. The mixed message frame is stored in the buffer first. Then, the broker only parses the green part of the message frames correctly and the red part is left in the buffer. In this situation, the broker has treated **0x32 0xff 0xff 0xff 0x7f** as the part of the message header of the next message frame. In $step2$, when real next message frames arrived at the buffer, the broker has considered 268 million as the remaining length of this message frame. It means the following message frames in green boxes and the previous red part should form a message frame with a length of about 268 million. 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.

3) Applying The Method to Modbus-TCP: As shown in Table III, we detect these potential vulnerabilities, including

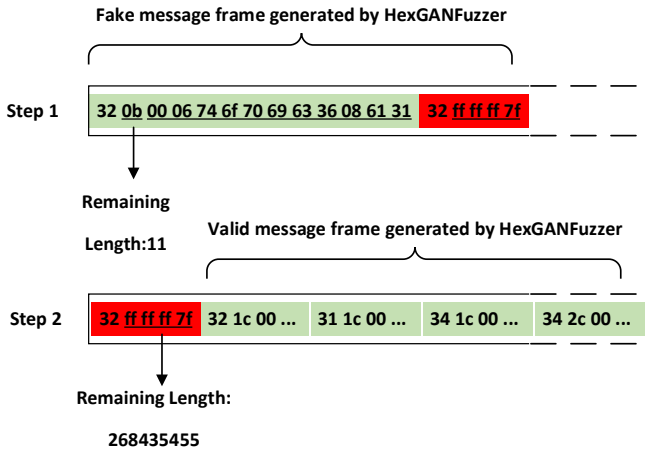

Fig. 9. Potential vulnerabilities analysis

slave crash, station off-line, working counter attack, using abnormal function code and so on, in Modbus-TCP via the new trained HexGANFuzzer. In the experiment, we send the generated data messages S_i to the slave stations and record the corresponding received message R_i. Massive message pairs $< S_i, R_i >$ are obtained. According to the abnormal protocol characterization above, we analyze and compare the specified field values and obtained the following statistical results. Experiments on the Modbus-TCP protocol prove that our method has great versatility.

TABLE III
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
Working counter attack	14 times	28.13
Using abnormal function code	207 times	0.52
Automatically closes window	13 times	28.47
Data length unmatched	190 times	0.63
Debugger memory overflow	4 times	41.85
Unknown attack	197 times	0.59

where number of Triggered anomalies(NTA) represents the total number of anomalies triggered by the models during test time.

V. CONCLUSIONS AND FUTURE WORKS

Fuzzy test based on deep adversarial learning is of great practical significance to the security verification of ICP. 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.

Considering the current situation, we intend to perform the study in the following aspects in future studies. First, GAN Compression [42] is a good way to keep the model more lightweight. And it is a promising application scenario that the lightweight model with online learning capabilities is deployed in embedded devices, which can learn protocol specifications or message formats of different protocols automatically. Second, a series of anti-random strategies after model training to improve the probability of anomalies triggering. Finally, we plan to apply the improved WGAN for more complex unsupervised learning NLP tasks including stateful protocols and non-stateful protocols.

REFERENCES

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

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

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

[13] 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.

[14] 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.

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

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

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

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

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

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

[42] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "Gan compression: Efficient architectures for interactive conditional gans," *arXiv preprint arXiv:2003.08936*, 2020.

[43] 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.

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