

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

***Abstract*—** **超过80%的涉及国计民生的关键基础设施依靠工控系统来实现自动化作业[1].而工控网络协议是工控网络通信的基石，其对工控系统的重要性不言而喻。如果工控网络协议的实现安全性能够得到保证，工控系统的安全性在很大程度上也能够得到保证。但ICP所适用的环境多元性较强，导致同一参数在不同应用中的含义千差万别，因此，安全开发者难以制定一系列通用的安全规则，Fuzz testing (fuzzing) 成为探查ICP 漏洞的主要手段。 Fuzz testing (fuzzing) has already become the main means of probing vulnerabilities in ICPs. However, the process of fuzzing relies heavily on the specification of ICPs. And it will take a lot of time and manual engineering to analyze and understand the specification. In this paper, we proposed a new simple and smart sequence generation neural network architecture based on** **Improved Self-Attention Generative Adversarial Networks (SAGAN). Based on this model, we present an** **automatically and intelligent fuzzing framework, called SAGANFuzzer, to solve the problems. Compared with traditional methods****,** **our framework can generate** **massive fake but plausible** **test protocol message automatically** [**in a short time**](http://dict.cnki.net/dict_result.aspx?searchword=%e5%9c%a8%e7%9f%ad%e6%97%b6%e9%97%b4%e5%86%85&tjType=sentence&style=&t=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 effectiveness by testing several typical ICPs, including MQTT and Modbus. Extensive experiments demonstrate significant improvements over test effectiveness and efficiency.**

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

1. INTRODUCTION

金融危机后，在制造业的发展出现了一些新动向，各国政府纷纷提出战略计划兴建下一代制造业，如工业互联网，中国制造2025 等，这种战略意在通过信息技术赋能工业，使其流程优化，降低成本，提高效率，从而释放更大的生产力。由于工业互联网往往存在安全攸关行业，确保工业互联网的安全具有特别重要的意义。为了方便工业系统中子系统之间的协作，工业互联网中的不同子系统之间互联程度越来越高。系统将面临更多的外来安全威胁。

在工控系统投入实际运行前，运用有效的测试技术及时发现整个系统可能存在的漏洞，提前修补预防，避免实际运行中的风险意义重大。当前，将传统的模糊测试技术运用于工控系统漏洞的发现是一种有效的方法；但存在一些限制之处：（1）对测试人员要求较高，需要测试人员根据系统中运行的通讯协议规范设计恰当的测试用例，来实施测试。（2）测试周期较长，从测试用例的设计到测试结束，需要花费较长的时间，在面临比较亟待投入运行的系统时，无法高效的完成测试任务。（3）不具有普适性；传统方法每次都需要根据特定的测试目标设计相应的测试用例，不能达到一次设计多处使用的效果。

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 . 然而，较差的机器学习算法不仅在模型的训练时极大的消耗计算资源 而且在训练完成后容易生成大量格式错误的协议消息序列，造成正常抛出(**crashes and error messages**)，从而被服务器迅速拒绝，无法进行测试。

分点展开本文的贡献。

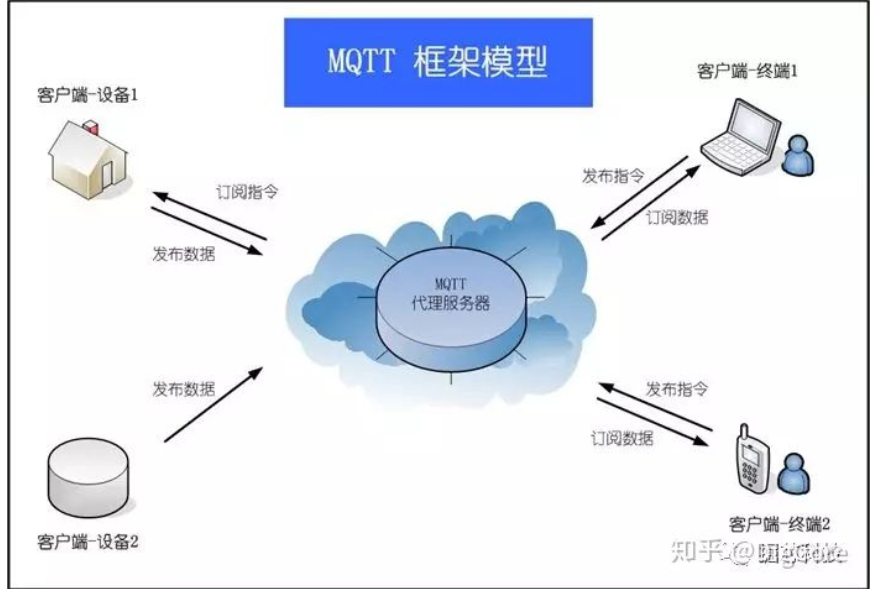
为了探索正常、异常抛出(In fuzzing, there are a lot of crashes and error messages, but a few of them are vulnerabilities. How to find real vulnerabilities from these crashes is a challenge)的平衡点，本文基于深度对抗学习提出一种打破上述限制的模糊测试用例生成方法。The contributions are summarized as follows:

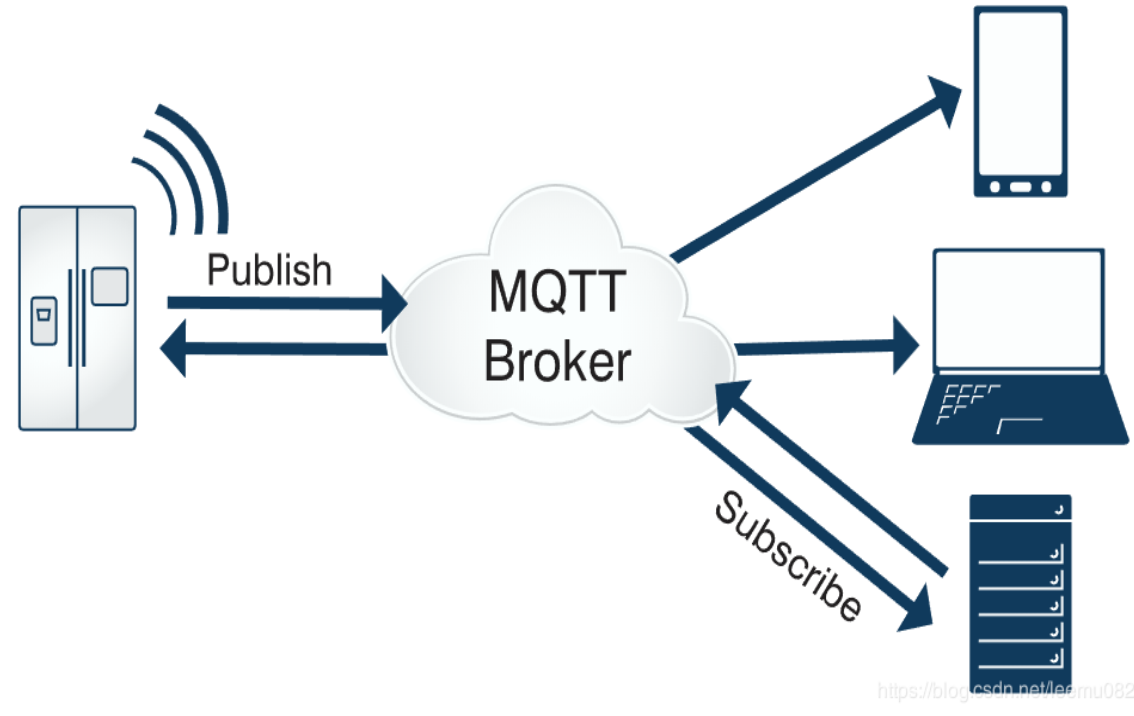
1. We propose a methodology based on GAN to deal with fuzzing data generation, in which it can intelligently learn to generate testing data by itself. We 运用Wasserstein distance来解决GAN的limitations for generating sequences of discrete，同时引入惩罚项来保证生成数据的多样性
2. 在模型设计过程中，在保证模型轻量节省计算资源的前提下，引用self-attention mechanism which dispenses with recurrence and convolutions entirely and (allows for) 允许significantly more parallelization which requires significantly less time to train, makes it superior in quality of generating fake but plausible testing data；
3. On top of the approach, we build a universal fuzzing framework, **called SAGANFuzzer**, which can deal with most ICPs’ fuzz testing. Also, 在采样过程中，本文引入一系列反随机策略，提升异常抛出的概率。

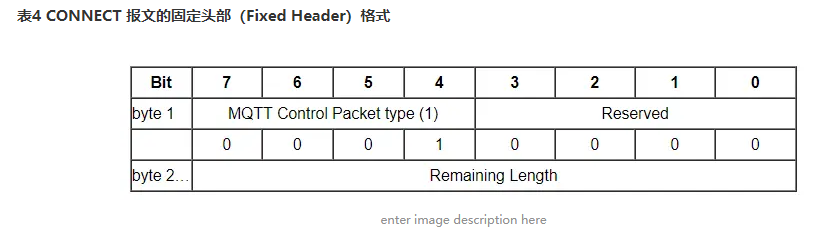
To evaluate its effectiveness, we apply it to fuzzing several ICPs. 实际环境中的实验结果证明该方法表现出了较好的性能，在不同的工控系统测试中均能获得较高的测试效果，达到测试目的，能有效引发系统异常行为。在测试effectiveness上和测试效率上都达到预期的结果。

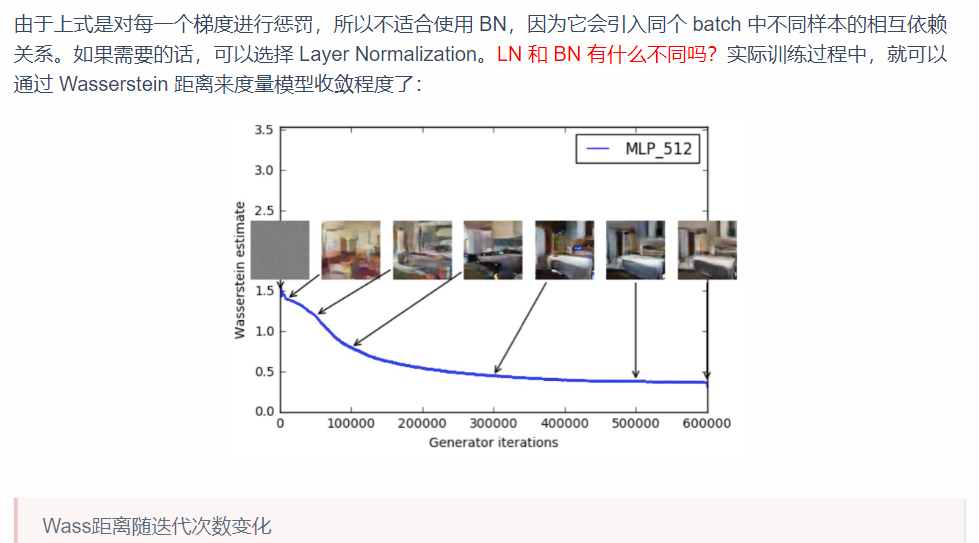
介绍本文的各部分框架的结构。Section 1-5。

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.

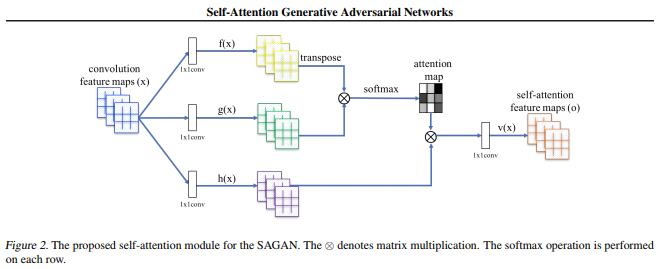








Reference : <http://www.iterate.site/post/02-%E8%AE%A4%E8%AF%86%E4%B8%96%E7%95%8C/01-%E6%95%B0%E5%AD%A6/99-%E7%AE%97%E6%B3%95/05-%E7%AE%97%E6%B3%95%E5%BC%80%E5%8F%91/11-%E7%AE%97%E6%B3%95%E6%A8%A1%E5%9E%8B/03-%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0-dl/33-%E5%AF%B9%E6%8A%97%E5%AD%A6%E4%B9%A0-gan/5.13-wgan/>



Reference：

<https://arxiv.org/pdf/1805.08318.pdf>

1. RELATED WORKS

// \* 里面cite 没表明来源的均来自 a systematic ...... \* //

模糊测试的原型是任意性测试（Random Testing）。Duran 和Ntafos[3] 是这方面研究的先驱者，在他们的早期研究中，他们采用任意的输入测试计算机程序，取得了不错的效果。模糊测试这一概念是由Miller 等人[4] 在1990 年正式提出的，最开始应用于Unix 程序的测试。它通过对测试目标输入大量非预期的畸形输入[11] (e.g., files, network packets, program codes) 来进行压力测试，以期引发目标系统异常行为。并依据异常行为对系统进行分析，找到系统存在的可利用漏洞（exploitable vulnerabilities）。（李大论文）

Since then, a variety of different techniques were proposed to improve the efficient of fuzzing. These techniques include static analysis (Sparks et al. 2007; Kinder et al. 2008), dynamic analysis (Höschele and Zeller 2016; Bastani et al. 2017; Kifetew et al. 2017). Because of its effectiveness, fuzzing has

been studied in the network protocol testing field to enhance the reliability of the computer network.(加一些最新的普通的network的test Aitel et al. Aitel (2002) ) And some of these works are for ICPs and have made a certain contribution to the improvement of the safety and security of ICPs.

/ \* 下面一段cite来自自己小论文

Greg Banks et al. proposed a fuzzy test tool called SNOOZE Banks et al (2006) for stateful network protocols such as SIP, TCP/IP, etc.

Devarajan Devarajan (2007) released a fuzzy test module based on Sully tool for Modbus, DNP3 and other industrial control protocols.

Voyiatzis et al. Voyiatzis et al (2015) designed a Modbus-TCP fuzzy test tool called MTF which builds the test model by Modbus official instructions.

M. Eddington, “Peach fuzzer,” Accessed: Sep. 1, 2017. [Online]. Available: https://www.peach.tech

J. Pereyda, “boofuzz: Network protocol fuzzing for humans,” Accessed: Feb. 17, 2017. [Online]. Available: https://boofuzz.readthedocs.io/en/latest/

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 (Li et al. 2018).

With the advancement of machine learning in the field of cybersecurity, it has also been adopted by many studies for vulnerability detection. (Grieco et al. 2016; Wu et al. 2017; Chernis and Verma 2018), including the applications in fuzzing (Godefroid et al. 2017; Rajpal et al. 2017; Wang et al. 2017; She et al. 2018; Liu Xiao, Prajapati, Rupesh, Li Xiaoting 2019). As was expected, some studies have incorporated The Fuzzing algorithm of ICPs based on deep learning into the fuzzing process of ICPs.

/ \* 下面一段cite来自自己小论文

Chockalingam et al.Chockalingam et al (2016)

utilized an LSTM model to do intrusion detection about CAN

bus protocol. Rajpal et

al. Rajpal et al (2017) applied a sequence-to-sequence neural

network model to enhance the AFL (American Fuzzy Lop)

Zalewski (2017) fuzzer. It uses RNN as an assistive technology

to improve the AFL’s performance toward stand-alone

programs. 把学长学姐的小论文说一下

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. However, there are still some limitations, such as unbalanced training samples and difficult to extract the characteristics related to vulnerabilities.

These efforts all contribute a lot to deep learning based fuzzing. In general, most of them use RNN model and prior knowledge to deal with their fuzzing problem.

However, in this study, we use the CNN model as a core technique and attempt to deal with ICP fuzzing problem without knowing the prior knowledge.



Fig. 2. BLSTM-DCNNFuzz Framework

1. BLSTM-DCNNFUZZ FRAMEWORK

In this section, we首先从待测试的系统中抓取大量的通讯数据，对数据进行预处理作为方法中所建立的深度对抗学习模型的训练数据。其次，设计建立生成对抗网络中的生成模型和判别模型，用所获得的数据对模型进行训练得到特定的模型。用生成模型生成大量的测试用例数据。再次，用生成的数据对系统进行压力测试，引发系统异常。最后，根据系统的异常，找到系统异常的原因，进行修补改进。

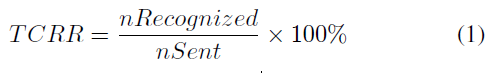
1. *Fuzz Testing The Target ICP*

With the trained model, we can generate as much test data as we want. When conducting a fuzz testing, MSG Sending and Receiving Module (\textbf{MSRM}) is in charge of monitoring interactive states, sending the test data to the target and receiving the feedback. Besides recording the relevent logs during the fuzzing process, the Logging Module(\textbf{LM}) is applied for abnormal MSGs and logs analysis based on the following performance metrics.

1. *Performance Metrics*

Some quantitative criteria [57-59] have emerged only recently assessing GAN on image generation. There is no unified validation metrics and benchmarks in this field. Therefore, in accordance with our research purpose and specific situation, we proposed the flowing metrics. Among them, *TCRR* and *BTE* serve as the training performance metrics, and *DGD* serve as the fuzzing effectiveness metrics.

*a.* *Test Case Recognition Rate (TCRR):* *TCRR* refers to the percentage of test cases recognized by the test target. It indicates the proportion of valid test cases. In the fuzz testing of ICP, if the target can recognize the test case, we consider the test case is correct in format and necessary information. The higher *TCRR* indicates more generated test cases are similar to the real-world traffic sequence in format. Conversely, the lower *TCRR* means more test cases are dropped directly by the target, which needs to adjust or retrain the model. The formula is shown below:



where *nRecognized* is the total number of test cases recognized, and *nSent* is the total number of test cases sent.

*b.* *Bugs Triggered Efficiency (BTE):* On the one hand, *BTE* refers the specific bugs found. On the other hand, it reflects the number and tme of bugs triggered after sending a fixed number of test cases. It is an important indicator of the model effectiveness. Since the ultimate goal is to find as many vulnerabilities as possible, we consider not only what bugs were found in the testing but also the testing efficiency. It should be noted that the number of errors found is also related to the test target. Weak target will highlight the method effectiveness. However, in this study, we only focus on the effectiveness of the method. The specific formula is as follows.

BTE =nBugs / nAllCases +

where nBugs indicates the number of bugs found, and the denominator nAllCases is the number of all the test cases sent, t*abnormal* records the interval from the last normal request initiation to the next check-out of bug (five maximum values and five minimum values are discarded), M is the total number of time intervals, ti is the interval of discoving the ith bug, t*abnormal= {t1, t2, …., tm}* and *a == 1/(e^8)* （empirical value）. The larger value indicates the stronger bug trigger ability.

*c.* *Diversity of Generated Data (DGD):* *DGD* refers to the ability to maintain the diversity of the training data. More diverse generated test data frames are likely to cause more exceptions. This indicator focuses on the number of message types in the generated data. It is also an important indicator of the method effectiveness.

DGD = (*nGCategory/* *nACategory)* x 100%

where *nGCategory* is the total number of message categories in the generated data frame, and *nACategory* is the total number of message categories in the training set.

1. *Logging and Evaluation*

We construct the Logging Module(LM) to record the feedback from the ICP. The module, as shown in Fig. 4.7, consists of two parts: one of which is the system logging of the tool itself; the other part records the feedback of the send**/**receive data to the log file. In the communication process, normal communication data and occurred anomalies will be logged into a log file by the module.

The log file saved by LM 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 training data set. We assume that retraining the model with these data can improve the capability of the model to discover vulnerabilities.



Fig. 4.7. Construction of Logging Module

1. EXPERIMENT AND RESULT ANALYSIS

In this section, we evaluate the effectiveness of the proposed method by experimentation. To show its effectiveness, we apply it to test Modbus, one of the widely used ICPs. To indicate the versatility of our method, another ICP, EtherCAT, will also be used to test.

1. *Modbus and EtherCAT*

We choose Modbus and EtherCAT as our test targets from a variety of ICPs in the experiment. ICPs have much in common features such as a short-data frame, no encryption. They are designed to meet the real-time requirements of the control system.

*1) Modbus-TCP:* Modbus protocols have many variants, including Modbus-TCP and Modbus-UDP. Here, we use Modbus-TCP as one of the fuzzing targets，as illustrated in Fig. 5. It uses master-slave communication mode, in which the master communicates with the slave by sending and receiving data frames. In the experiment, different Modbus-TCP implementations, including Modbus RSSim v8.20, Modbus Slave v6.0.2 and xMasterSlave v.156 are applied as the fuzzing targets. Finally, in order to better demonstrate the effectiveness of our approach, we use the serial communication mode between MCU [46] and PC, and adopt RS485 bus [47] for signal transmission to build the real Modbus network environment. The generated test cases are sent in real environment to test the effects in real applications.



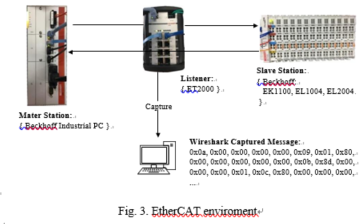
Fig. 5. Message format of Modbus-TCP

*2) EtherCAT:* EtherCAT offers high real-time performance and provides a master-slave communication mode among the industry devices. A typical EtherCAT network consists of one master and several slaves. The master generates datagrams and sends them to the loop network. These datagrams are reflected at the end of each network segment and sent back to the master. We test EtherCAT to prove the versatility of our method.

1. *Training Data*

Training data in deep learning significantly influence the model training. Thus, we accurately collect and preprocess the training data. In the experiment, training data about the two industrial control protocol is collected separately.

1. *Modbus-TCP:* We use the Pymodbus [[17](#_bookmark16)], a python package that implements the Modbus protocol, to generate the training data frames. Through it, we can quickly generate enough different types of data frames, which is practical and convenient. Specifically, 300,000 pieces of data including various type are used as the training data.
2. *EtherCAT:* In order to capture the EtherCAT communication data, we prepare an EtherCAT based ICS as illustrated in Fig. 3. The master station is a Beckhoff [[18](#_bookmark17)] industrial PC, and the slave station includes EK1100 [[19](#_bookmark18)], EL1004 [[20](#_bookmark19)] and EL2004 [[21](#_bookmark20)]. ET2000 [[22](#_bookmark21)] is used as the online Listener between the master and the slaves. The Wireshark [[23](#_bookmark22)], running on a computer, can fetch and display the massive communication data messages from the listener. After processing, these messages will serve as the training data for the EtherCAT protocol.



1. *Evaluation Setup*
   1. *Experimental Environment:* We adopted Tensorflow, one of the popular deep learning framework, to implement the model architecture. To improve the training efficiency, we train the model on a Windows machine with 8 processors (Intel(R) Core (TM) i7-6700K CPU@4.00GHz) 16.0GB memory (RAM) Nvidia GeForce GTX 1080 Ti (11GB). When launching an attack, the simulators run on another machine with 4 processors (Intel (R) Core (TM) i5-5300U CPU@2.30GHz) 8.00GB memory (RAM).
   2. *Model Training Setting:* As for the parameter setting, we initialize all weights from zero-centered Normal distribution with a standard deviation of 0.02. The mini-batch size is set to 256 in all models. The learning rate is set to 0.0002 in the Adam optimizer. As to the Leaky ReLU function in discriminator model, the slope of the leak is set to 0.2. We train the models for 1000 epochs and save the generator model for every 100 epochs to get plentiful test cases.
2. *Experiment Results*

In this section, we show the experimental results in three aspects. We first present the bugs occurred in fuzzing the Modbus implementations. We then reveal statistical results and its analysis. Lastly, to show the methodology’s protocol independence in ICP’s fuzz testing, we apply it to test EtherCAT protocol.

1. *Exception Founded:*

We send the generated data frames to the aforementioned Modbus implementations which serve as Modbus slave stations. A total of 30,000 test cases generated was sent to each Modbus implementations. The effect is exciting that we successfully triggered bugs. The following describes these bugs in detail.

Much abnormal information is displayed at the console of the simulation software when the Modbus Rssim is attacked by the generated data frames. For a while, it goes crash. In detail, the software pop ups windows prompt box after we sent about 3500 data frames, indicating that the program has crashed. We send data frames range from 3450th to 3500th to the other two simulation softwares, Modbus Slave and xMasterSlave, no abnormality occurs. This comparison shows that Modbus Rssim has some errors in the emulating Modbus-TCP protocol.

Another exception worth discussing is “Station ID xx off-line, no response sent” in Modbus Slave. The log indicates that “Station ID 32 off-line, no response sent” after sending about 6540 data frames. But we observe that the station 32 is still online. This phenomenon makes us believe that there is an implementation flaw with the slave state judgment of Modbus Slave.

In fuzz testing the xMasterSlave, we find that the software automatically closes the window itself at times. Through the analysis of the system log, we conclude that memory overflow is the cause of the software crash, which suggests that the programmer may not consider the exception of populating with data boundary values when implementing the simulator.

In further attacks of fuzz testing the three [simulation software](http://dict.cnki.net/dict_result.aspx?searchword=%e6%a8%a1%e6%8b%9f%e8%bd%af%e4%bb%b6&tjType=sentence&style=&t=simulation+software)s and the real environment, anomalies such as “Using Abnormal Function Code”, “Data length Unmatched”, “Integer Overflow”, and “Abnormal Address” occur on a regular basis. We record the test cases that cause these abnormalities. All the abnormal feedbacks from the three softwares and slaves in real environment are counted for further analysis. Other anomalies such as “File not Found” and “Debugger Memory Overflow” are found only once or twice and thus are not discussed in detail.

1. *Statistical Analysis And Results*

In the study, we choose the widely used GPF (General Purpose Fuzzer) [48], GAN-based model and LSTM-based seq2seq mode as fuzzers in the control group. The systems to be tested are 3 modbus simulation softwares, namely Modbus Slave, xMasterSlave, pymodbus and MOD\_RSSIM, and the real modbus network environment we put up. In order to better evaluate the overall effect of the model on the protocol, we combined the experimental results of the four systems. The weights of the data obtained in these four experiments are 20%, 20%, 20%, 40% in the holistic data.

After fuzzers in the experimental group and control group are fully trained, fuzzing test is conducted by sending generated test cases through the TCP/502 port.

According to the three evaluation indicators mentioned above, we evaluate the effectiveness and efficiency of our fuzzing framework BLSTM-DCNNFuzz. Details are as follows.

1. ***TCRR.*** We choose Modbus Slave as the target and send the generated test cases to it. In the experiment, we tried three different learning rate when training the model. Experiment shows that model training is stable when the learning rate is set to 0.0002. From Fig. 4, we can see that TCRR rises with increasing training epochs. This indicates that an increasing volume of generated data has the correct message format. Initially, TCCR increases significantly; with further training, it tends to increase slowly and eventually flattens. The highest point of TCCR is about 95%. It means that most of the generated data can be identified by the target.

**GPF is compared with three fuzzing model based on depth learning sampling, and the experimental results are shown in figure 6.** **The horizontal line represents the performance of GPF on the systems. Due to not involving the learning process, there is no changing trend of GPF.**

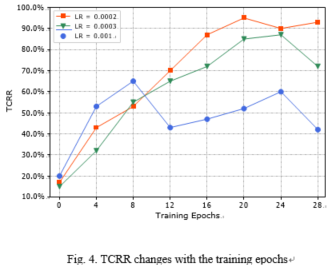
从4 个实验整体来看，合法性指标上都有GPF ≈

LSTM-based model< GAN-based model < BLSTM-DCNNFuzz.

生成对抗学习算法在训练超过30 代之后，合法率将明显超越GPF 算法，其合法率上升趋势在60 代以后显著放缓。

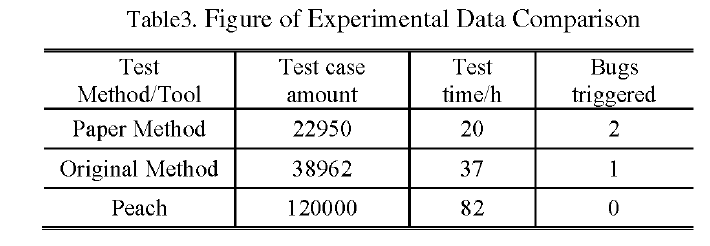
GPF 算法的合法率平均数为58． 5%，而生成对抗学习算法最终到达的合法率在75% ～ 90%。LSTM-based model算法的最终合法率明显低于其他用于深度学习的2 组，其原因可能在于无法有效地学习到数据的空间特征。协议消息的功能码与参数范围有限，增加了随机生成中正常抛出的可能性。

对于50代～ 100 代的生成对抗学习算法算法，BLSTM-DCNNFuzz的平均合法率比GAN高8%。从侧面说明了BLSTM-DCNNFuzz适用于针对该协议的预测.



1. ***BTE.***When testing the modus implementations, we recorded triggered bugs and triggered frequency.

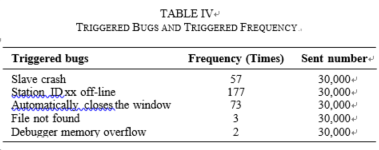
利用四个模型对四个系统进行Fuzzing 测试， **Fuzzing** 测试结果如表Table 3。



BLSTM\_DCNNFuzz 具体表现如表4所示、（ 可删 ： 实验结果表明the proposed methodology can trigger

vulnerabilities in high frequency.

）

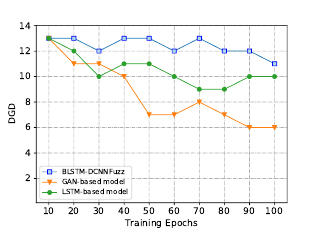


1. ***DGD.***

*由于GPF的趋势是不变的， 并且不同GPF的覆盖度不同， 因此在此不讨论。* A total of 13 types of Modbus data frames are prepared in the original training data. When the training epochs is 10, the diversity of 3 models matins the best. After training, some message categories are generally lost, which 意味着fuzzing test 的 code coverage 降低

The effect of fuzz testing depends predominantly on testing depth and high code coverage.

Fig. 5 shows that the generated test cases maintain the diversity of the original training data. Therefore, the model has good performance on maintaining the test case diversity. Usually, the richer the data type, the stronger the ability to detect anomalies. Thus, the trained model can effectively detect bugs as presented in Table IV.



1. *Applying The Method to Another ICP-EtherCAT*

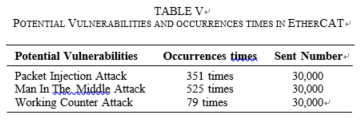
To demonstrate our methodology is not just akin to a particular ICP, we apply BLSTM-DCNNFuzz to test EtherCAT, another widely used ICP. We retrain the model with captured EtherCAT data frames. With the newly trained model, massive test cases are generated expediently to detect the potential vulnerabilities of EtherCAT protocol. The following presents the details.

1. ***Communication details of*** ***EtherCAT***.

EtherCAT uses a dedicated Ethernet data frame type definition to transport EtherCAT packets of Ethernet data frames. The master station initiates the communication process and controls the working status of the slave stations via process data communication. The EtherCAT slave stations extract the control information and commands from the protocol messages sent by the master station, and insert the relevant information and the collected data of the local industrial field device associated with itself. Then this Ethernet data message is transferred to the next EtherCAT slave stations. The operation repeats until all slave stations are traversed.

1. ***Detected potential vulnerabilities***.

We detected these potential vulnerabilities including man-in-the-middle (MITM), MAC address spoofing, slave address attack, packet injection, working counter (WKC) attack and so on. In the experiment, we send the generated data messages *Si* to the slave stations and record the corresponding received message *R*i. We get massive message pairs < *Si, Ri* >. According to the protocol abnormal characterization above, we analyzed and compared the specified field values and obtained the following statistical results. Experiments on the EtherCAT protocol prove that our method has great versatility.



VI. CONCLUSIONS AND FUTURE WORKS

In this study, we propose an effective fuzzing methodology based on SAGAN to generate fake but plausible fuzzing data about ICPs. This methodology utilizes CNN to learn the spatial structure and distribution of real-world message and generate similar data frames without knowing the detailed protocol specification. Allowing the convolution neural networks to learn the message format can save human effort and reduce time. In this manner, when testing other network protocols, we do not need to understand their specifications, which is convenient. We ultimately evaluate this method by fuzzing two safety-critical ICPs, including Modbus-TCP and EtherCAT. The results indicate that the proposed method has application potential to test a series of ICPs.

In future studies, we expect to create a more intelligent and more automated network protocol fuzzing system deployed to embedded devices. The system can apply the manner of online learning to learn protocol specifications or message format of different protocols automatically. Considering the current situation, we intend to perform the study in the following aspects. First, we want to further explore other architectures to enhance our approach. Second, we will use our method to test other stateful ICPs, such as Profibus, Powerlink and future TSN. These protocols constitute an important part of most current ICPs. Finally, we intend to integrate each excellent architecture and processing module to form a hybrid model and a complete software system, which can deal with most network protocols, including stateful protocols and non-stateful protocols.

REFERENCES

* 1. T. Braibant, “Coquet: a coq library for verifying hard- ware,” in *International Conference on Certified Programs and Proofs*. Springer, 2011, pp. 330–345.