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

***Abstract*—****.**

***Index Terms*—****convolution neural networks, self-attention, deep adversarial learning, 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。

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。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。

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。在测试效率上和测试目标无关性上都达到预期的结果。

分点展开本文的贡献。

介绍本文的各部分框架的结构。Section 1-5。

Benefiting from its recurrent structure, (LSTM) [[50](#_bookmark8)], as an alternative type of neural networks, shows great power in precise timing of sequence data [51]. Generative Adversarial Networks (GANs) [[7](#_bookmark6)] is particularly famous for generating highly simulated images [[8](#_bookmark7)]. Inspired by this, we attempted to apply it to generate fuzz testing data and make the trained model, replacing engineers, to design test cases. Thus, we can break the limitations above. In this paper, we propose and design a fuzz testing methodology based on DCGAN (Deep Convolution Generative Adversarial Networks) [[9](#_bookmark8)]. The contributions are summarized as follows:

1. We propose a methodology based on Bi-directional LSTM and DCGAN to deal with fuzzing data generation, in which it can intelligently learn to generate testing data by itself.
2. On top of the approach, we build a universal fuzzing framework, the BLSTM-DCNNFuzz, which can deal with most ICPs’ fuzz testing. Also, in data processing, character- level data conversion is implemented.
3. To evaluate its effectiveness, we apply it to fuzzing several ICPs. The results reveal that our method has a good performance.

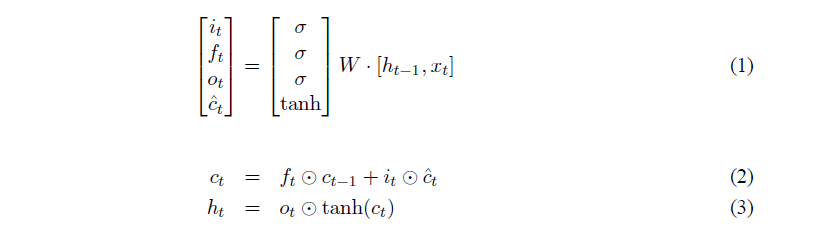
The remainder of this paper is organized as follows. Section II presents preliminary knowledge. Section III details optimized DCGAN algorithm and the entire methodology design. Section IV presents the evaluation results. Section V discusses the related work. Section VI concludes the paper and discusses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

1. PRELIMINARY

In this section, we introduce some preliminary knowledge. First, the basis of LSTM, GAN and DCGAN is presented. We then introduce the preliminary knowledge of ICPs and their common features. Lastly, we give an overview of fuzz testing and its application in finding ICP’s vulnerabilities.

1. *Long Short-Term memory*

Hochreiter and Schmidhuber (1997) first proposed LSTM to overcome the gradient vanishing problem of RNN (Recurrent Neural Networks). It is a special RNN that introduces an adaptive gating mechanism which can decides the degree to keep the previous state and avoid the problem of long-tern dependencies. Given a sequence *S* = {*x1, x2, …, xl*}, where *l* is the length of input text, LSTM processes it word by word. At time-step *t*, the memory *ct* and the hidden state *ht* are updated with the following equations:



where *xt* is the input at the current time-step, *i*, *f* and *o* is the input gate activation, forget gate activation and output gate activation respectively, *ˆc* is the current cell state, *σ* denotes the logistic sigmoid function and ⊙ denotes element-wise multiplication.

1. *Generative Adversarial Networks*

Generative Adversarial Networks, proposed by Goodfellow and others, is a promising framework to guide the training of generative models. Specifically, in GAN a discriminative network(discriminator) learns to distinguish whether a given data instance is real or not, and a generative network(generator) learns to confuse discriminator by generating fake but plausible data.

In this adversarial network framework, the goal of the training generation model is to obtain a probability distribution of Q, which approximates the real data X. In order to achieve this goal, a known distribution (such as Gaussian distribution, uniform distribution) is used to sample and get the "noise data" in the first place. Then, these "noise data" are input into the generation model, the training of the generation model is carried out. After the training of generator is finished, simulated data is output.

In this study, we utilize this feature to generate simulated sequence message. Notably, when applying deep adversarial learning to fuzz testing ICPs, we expect the generated data to have a correct message format but with various message content so that the code coverage and testing depth can be guaranteed.

1. *Deep Convolution Generative Adversarial Networks*

Prior to introducing DCGAN, it is necessary to briefly introduce CNNs (Convolution Neural Networks) which have recently enjoyed great success in image and video recognition. Its success is mainly due to the large public image repositories, such as ImageNet, and high-performance computing systems, such as GPUs (Graphics Processing Units) or large-scale distributed clusters. Because the pooling layer has no weights, no parameters and only a few hyper parameters, one convention comprises of one convolutional layer part and one pooling layer part in general. A typical CNN consists of many layers, including the input layer, conventions, the fully-connected layer, the output layer, etc.

Recently, NLP communities pay more and more attention to CNN and have achieved favorable results in various of text classification tasks [38, 39]. Different from RNNs accomplished in time-related problems, CNN is good at learning spatial structure features. Actually, most ICPs’ message have the following features: concise format, limited length and compact structure. This makes CNN a better way to solve this kind of problem [52]. After proper preprocessing via Bi-LSTM which adds the location feature in the input, each filter in CNN can be regarded as a detector that detects whether a functional unit in the data frame is correct [40], which is conducive to grasping the format features of the sequence data in ICS. Hence in this study, CNN serves as the generator and the discriminator in DCGAN accordingly.

DCGAN is proposed by Alec Radford et al. [[9](#_bookmark8)] to bridge the gap between the success of CNNs for supervised learning and unsupervised learning. They extend GAN to the CNN domain and invent a structure called deep convolution generative adversarial networks (DCGAN) that have certain architectural constraints. This innovation combines the advantages of CNN in processing multidimensional feature and the idea of deep adversarial learning. Due to the constraints, DCGAN largely overcomes the problem of unstable training of GANs, such as non-convergence, vanishing gradient and mode collapse. These constraints are listed in Table I. We designed our model architecture based on the constraints.

TABLE I

ARCHITECTURE GUIDELINES FOR STABLE DEEP CONVOLUTIONAL

GENERATIVE ADVERSARIAL NETWORKS

|  |  |
| --- | --- |
| **#** | **Architecture constraints** |
| 1 | Replace any pooling layers with strided convolutions (dis-criminator) and fractional-strided convolutions (generator). |
| 2 | Use batch normal in both the generator and the discriminator. |
| 3 | Remove fully connected hidden layers for deeper architec-  tures. |
| 4 | Use ReLU activation in the generator for all layers except for  the output, which uses Tanh. |
| 5 | Use LeakyReLU activation in the discriminator for all layers. |

1. *Industrial Control Protocols*

ICPs refers to the communication protocol deployed in ICSs. As a class of systems, ICS has its characteristics, such as requiring high real-time performance and just providing several specific functions. Correspondingly, ICP’s message format tends to be concise. ICS consist of master stations and slave stations. The data transmission and operation control between them are realized through the ICP in it. Currently, various ICPs operate in a wide variety of ICSs around the world. Therefore, it is important to maintain their security. Except for the popular ICPs, Some ICPs are modified from the existing protocols or solely designed. These ICPs may have no clear specifications. Thus, the manual-based fuzz testing method will suffer from this.



Fig. 1. General workflow of fuzzing test

1. *Fuzz Testing*

Fuzz testing is a quick and cost-effective method for finding severe security defects in software. Traditionally, fuzz testing tools apply random mutations to well-formed inputs of a program and test the resulting values. Besides, fuzzing is a brute force vulnerability method, in which it uses a large amount of malicious input to have stress tests on the target. As industrial control networks become more and more interconnected, flaws in the implementations of ICP could allow a malicious party to attack vulnerable systems remotely over the internet. To avoid this, we use fuzzing to discover the flaws, in advance. The workflow is shown in Fig. 1.



Fig. 2. BLSTM-DCNNFuzz Framework

1. BLSTM-DCNNFUZZ FRAMEWORK

In this section, we first present an overview of the BLSTM-DCNNFuzz and then describe the main aspects in details. The overall fuzzing framework can be seen from Fig. 2.

1. *Framework Overview*

The purpose of this framework is to conquer the aforementioned limitations of traditional fuzz testing methods. We make it meet these requirements: (i) Highly intelligent. It can design and generate test cases by itself. (ii) Protocol independence. It can deal with most ICPs without knowing their protocol specifications. (iii) Very efficient. The entire testing can be completed in just a few days. As for the general process, first, the framework fetches network traffic data from the target ICS. Then, it learns from the traffic data and generates various testing data. Finally, the data is injected into the target system for testing. The flowing describes the main steps.

* 1. *Preprocessing of ICP Communication Data:* Data preprocessing has a significant impact on model training. We take two steps to process the raw data. First, we cluster the raw data according to specific criteria. Then, we augment some special test cases with a small percentage to increase their influence on the model. Lastly, we encode the raw data into a fully digital matrix form.
  2. *Model Architecture and Training Strategies:* We use DCGAN as the underlying architecture. Our purpose is to generate aggressive fuzz testing data to attack the target ICP for triggering more bugs. To achieve this purpose, we carefully design the network structure of the generator and the discriminator. As to the model training, we take measures to handle the unstable training problem such as mode collapse and non-convergence. Lastly, we use the trained model to generate enough fuzzing data.
  3. *Fuzz Testing The Target ICP:*In this step, we use the generated fuzzing data to attack a specific ICP. In the testing, we deliberately collect test cases that caused exceptions. They can enrich the original training data and help improve the model performance. The following gives more detailed information about these steps.

1. *Preprocessing of ICP Communication Data*

The current mainstream ICPs include Modbus, EtherCAT, Powerlink, Porfinet, Ethernet/IP, TSN (Time Sensitive Networks) [16] and so on. There are various ways to capture data packet from different ICPs. The most direct way is to use the appropriate terminal capture tool from the real industrial control network environment to capture the data packets generated by the ICSs as training data. After obtaining the raw data, Data Preprocessing Module(DPM) are divided into three steps to preprocess data. The details are as follows:

1）Data Frame Clustering

The effect of fuzz testing depends predominantly on testing depth and high code coverage. Protocol messages captured from the ICP vary in length and message type. The better our model comprehends the differences between protocol messages, the better testing depth we can achieve. We leverage a variety of clustering strategies to improve the classification scores, such as frame length clustering, K-means clustering and so on. Frame length clustering is to cluster the sequences based on their length. Since sequences which have the same length always tend to share the same message type. K-means, using Euclidean distance as similarity benchmark, is also applied in the clustering of data frames based on function code of data frames. Under these strategies, we can do data augmentation to a class of messages with a small percentage. This makes the generated data more diverse, which can help improve code coverage.

2) Adding Special Symbols

The other step is to add special symbols to guide DPM to obtain higher quality training data.

The process of capturing data by DPM can be divided into two categories, as shown in Fig. 3. First, for a known protocol stack, such as the TCP/IP protocol stack which Modbus-TCP running on, the IP header can be used as a demarcation point for capturing. We truncate the IP header (Including IP source address, destination address and additional information for some other delivery requests) and retain the file that holds the IP header for further packet injection attack. Then we insert *STA* (start) as the sequences start flag at the beginning of the packet body, *END* (end) as the sequences end flag. The operation eliminates the influence of irrelevant information on the model and improves the quality of the captured data. Second, if the protocol is unknown, the learning with address is performed, and *STA* and *END* are added directly at the beginning and end of the entire captured data. Finally, the processed data is stored in the training data set.

Moreover, pad the short sequence with uniform character *PAD*(pad) to the maximum frame length. It helps unify the sequence length to standardize the training.



Fig. 3. Process of capturing data

3) Data Feature Conversion

The captured sequence data is in the native features which cannot be processed directly. In order for the model to perceive these features, raw digital protocol message need to be converted to an appropriate pattern. Due to few ways to learn word embeddings [45] about data frames, we transfer the network traffic in the ICS into a hexadecimal sequence based on an alphabet of n characters.

In addition to the sequence length alignment, we use a character level preprocessing, inspired by [[10](#_bookmark9)], to deal with the converted data feature in this work. In the alphabet of data frame, there are 10 digit characters and 6 English letter characters as shown in the flowing:

***0 1 2 3 4 5 6 7 8 9 a b c d e f***

According to the alphabet, each character in the sequence is encoded as a one-hot vector of the n dimension *x∈R1×n*. As depicted in Fig. 4, the position where the character locates in the alphabet is one, and the rest are zeros. Thus, a sequence data with length *l0* is encoded into a matrix *X∈Rl*0*×n*, as a stack of character embedding.



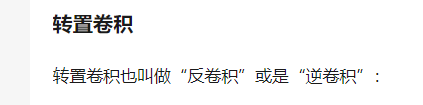
Fig. 4. A sample of raw ICS data and its feature quantization

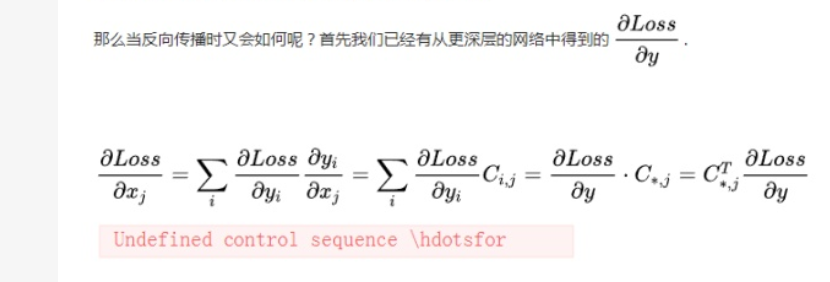
1. *Model Architecture and Training Strategies*

In this section, we give the model architecture details of Model Training and MSG(message) Generating Module(**MTMGM**). Later, we present the training strategies about the model in this study.

*1 ) Model* *Architecture:* There are two components, namely the generator model and the discriminator model. One of our design philosophies is to design lightweight models based on attaining its effect. In virtue of the distinguishing feature of reducing the consumption of computing resources, it is convenient to deploy to embedded devices and lays the groundwork for continuous online learning in the future. Referring to the aforementioned DCGAN architecture constraints and our requirements, we reasonably designed the architecture of DCGAN which are given in Fig. 5 (b).

1. ***Generator.*** The generator uses a deconvolution [[42](#_bookmark8)] neural network architecture. We adopt batch normalization (BN) [11] right after each convolution and before activation, following [11]. In addition, The generator consists of multiple fractional convolutional layers. Specifically, it replaces the pooling layer with fractional convolution layer, which is different from the traditional convolutional network.





Deconvolutions, also called transposed convolutions or fractional-stride convolution [44], work by swapping the forward and backward passes of a convolution. Based on



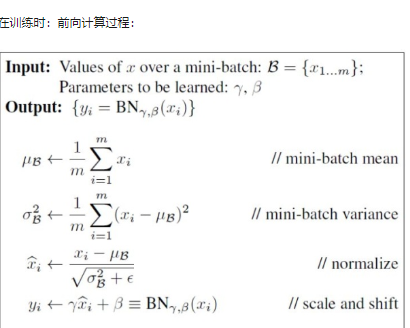
Fig. 5. The architecture of DCGAN: (a) generator network, (b) architecture of DCGAN, (c) and discriminator network. The first part of (c) is a BLSTM2DCNN for the 260 byte input sequence. One hots have size 16, and BLSTM has 260 hidden units.

Zero padding and non-unit strides in G, the following formula formalizes the output size of a deconvolution:

***a = (i + 2p − k)* % s**  ( 1)

*o` = s(i` - 1 ) + k – 2p + a*  ( 2 )

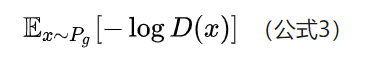




where o’ represents the square output size, i’ represents square input size, k represents square kernel size, s represents same strides along both axes, p represents same padding along both axes, i represent the input size of the next convolution layer, and *a* represents the number of zeros added to the bottom and right edges of the input.

Fig. 5 (a) depicts the network structure of the generator. Except the last layer, we select BN and ReLU (Rectified Liner Units) as the activation in rest layers [[9](#_bookmark8)]. The last layer applies Tanh as the activation. The generator takes a uniform noise distribution Z as input, and output a matrix which will be an input to the discriminator model. Notably, no pooling layers or fully connected layers are used.

G损失函数：（李 -Access）

the - log D alternative  


1. ***Discriminator***

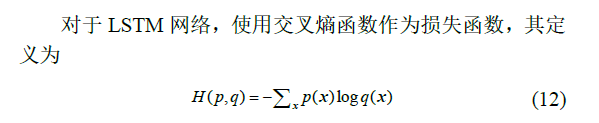
In adversarial training, the discriminator model is mainly designed to guide the training of the generator. One hot [representation](javascript:showjdsw('showlj_0','lj_0')) converts distributed representations of bytes of each protocol message into vectors, which form a matrix. The matrix includes two dimensions: the time-step dimension and the feature vector dimension. Most existing models just take notice of the time-step dimension of texts to obtain a fixed-length vector, which ignore the spatial structure features. However, the time-step dimension and the feature vector dimension are not mutually independent of each other. To integrate the features on both dimensions of the matrix, we leverage BLSTM and CNN so that the discriminator can hold not only the time-step dimension but also the feature vector dimension information.

As shown in Fig. 5(c), the BLSTM Layer contains two sub-networks for the forward and backward sequence context respectively. The output of the ith byte is shown in the following equation:

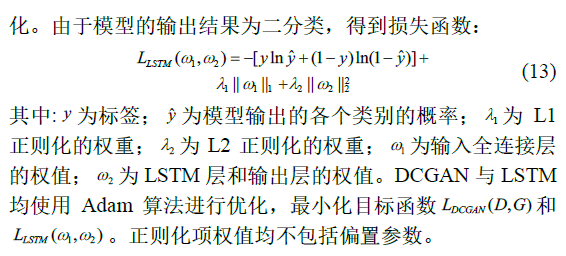


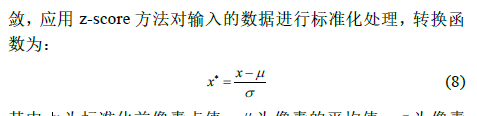
A matrix *H = {h1, h2, …, hl\_max}*, *H ∈ Rl\_max × d*, is obtained ×from BLSTM Layer, where *d* is the size of hidden units of BLSTM and *l\_max* is the length of maximum frame length of the ICP. In this study, we make *d* equal *l\_max* to get a square input size which contains sequential information [53].

(paper : 结合DCGAN 和LSTM 的)

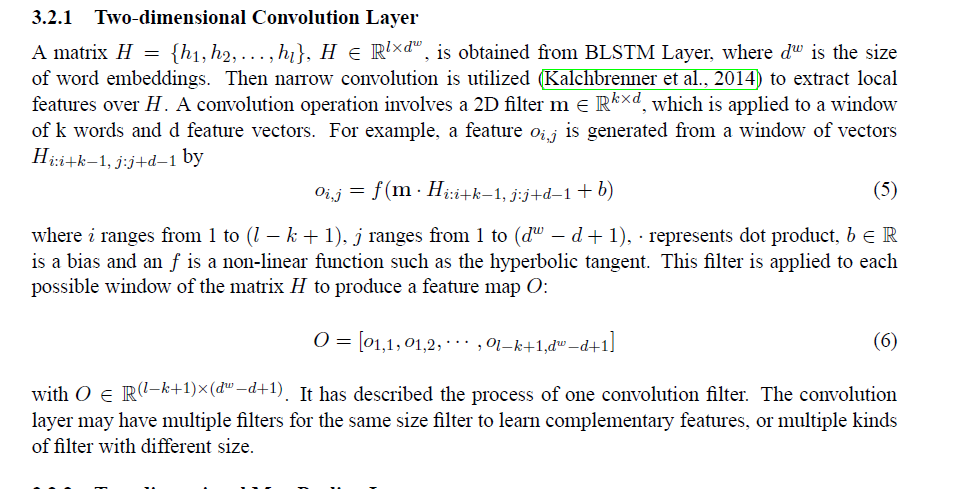






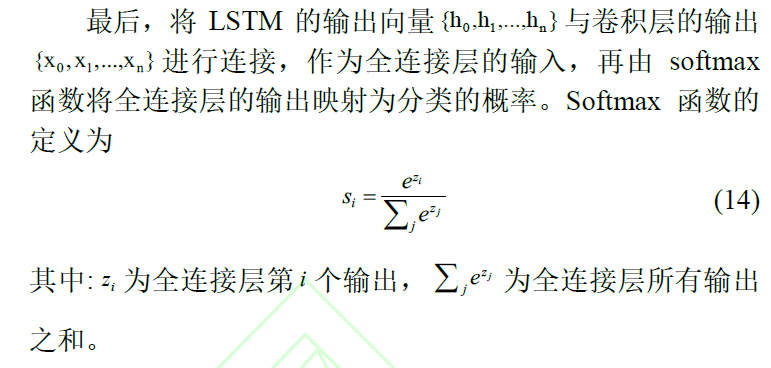


The discriminator applies strided convolution layers to replace any pooling layers instead of deconvolution layers.



Different from the generator, discriminator does not apply BN in the input layer to avoid sample oscillation and model instability [9]. It applies BN and Leaky ReLU [[12](#_bookmark11)] as an activation instead of ReLU. At the end of the network, we use one fully-connected layers to make the output convert to a *1x1* discrimination probability.

(paper : 结合DCGAN 和LSTM 的)



判别器最后的全连接层只有一个输出节点，使用sigmoid

函数将网络的输出值映射为概率。

D损失函数：（李 -Access）



The detail layout of CNN can be seen from the right part of Fig. 5 (c). The discriminator takes real-world processed matrix from BLSTM or the output matrix from the generator as its input.

整体的损失函数：（李 -Access）

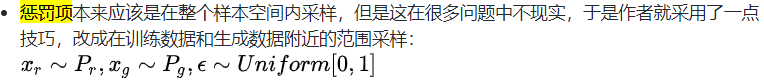


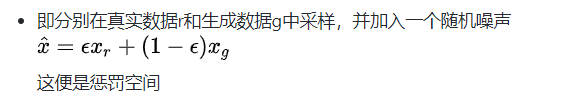
(paper : 结合DCGAN 和LSTM 的)

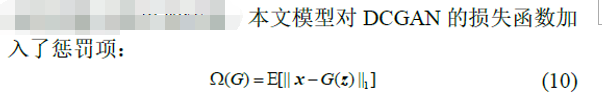
（来源：

<https://github.com/sssste/DeepLeraningNotes/wiki/GAN%E7%9B%AE%E5%89%8D%E5%B8%B8%E7%94%A8%E7%9A%84loss%E5%87%BD%E6%95%B0>）

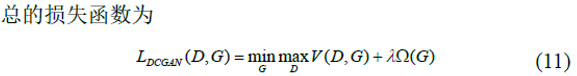
为了让样本更有多样性，c











最小化目标函数( , ) *DCGAN L D G* 和 1 2 ( , ) *LSTM L*  (来自Paper 结合DCGAN 和 LSTM)

*2）Model Training Strategies:* To get a well-trained model, appropriate training strategies should be taken. DCGAN Model training is difficult because the two models need to be trained synchronously. We have adopted a reasonable strategy to avoid the emergence of aforementioned problems such as model collapse and non-convergence. Dai et al. [43] found that setting parameters by pre-training is better than random initialization for deep learning network models, which can significantly stabilize the training. Therefore, we pre-train the discriminator for several epochs, getting parameters of D which helps form a gradient to guide the generator firstly. Second, we choose Adam optimizer [[13](#_bookmark12)] with tuned hyperparameters, which takes advantage of adaptive learning rates and momentum. All models were trained with mini-batch stochastic gradient descent (SGD) [[14](#_bookmark13)] with a fairish mini-batch size. All weights initialized from a normal distribution. These tactics help reduce the training oscillation and instability.

Our ultimate goal is to leverage the generated fake data to test the target and trigger more bugs. One factor that affects the effectiveness of fuzz testing is the test data diversity. Rich test data tends to find more bugs. In addition to data augmentation [15], we save the generator model for several training epochs. Data generated in different epochs can enrich the fuzzing data diversity. There exists a tradeoff between the correct data format and the data diversity. We intend to not only make the generated data formats correct but also make the generated data more diverse in data content.

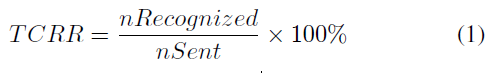
1. *Fuzz Testing The Target ICP*

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

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.



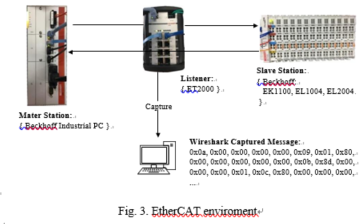
Fig. 5. Message format of Modbus-TCP

*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.**

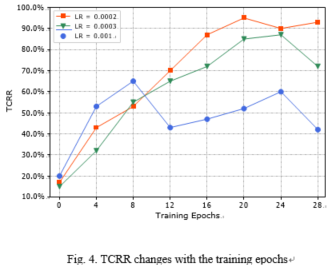
从4 个实验整体来看，合法性指标上都有GPF ≈

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。

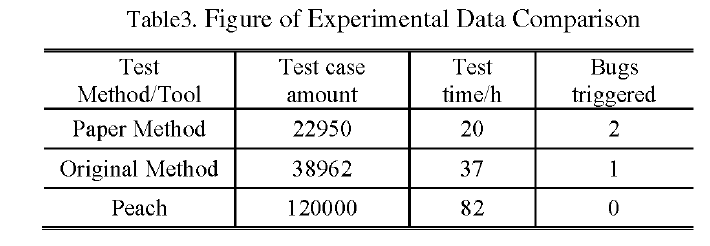
GPF 算法的合法率平均数为58． 5%，而生成对抗学习算法最终到达的合法率在75% ～ 90%。LSTM-based model算法的最终合法率明显低于其他用于深度学习的2 组，其原因可能在于无法有效地学习到数据的空间特征。协议消息的功能码与参数范围有限，增加了随机生成中正常抛出的可能性。

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.

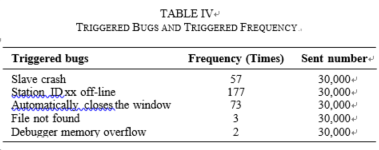
利用四个模型对四个系统进行Fuzzing 测试， **Fuzzing** 测试结果如表Table 3。



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.

）

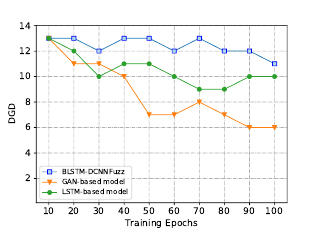


1. ***DGD.***

*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*

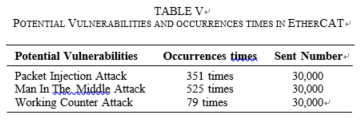
To demonstrate our methodology is not just akin to a particular ICP, we apply BLSTM-DCNNFuzz to test EtherCAT, another widely used ICP. We retrain the model with captured EtherCAT data frames. With the newly trained model, massive test cases are generated expediently to detect the potential vulnerabilities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



V. RELATED WORKS

Fuzzing has developed for decades, and practice has proven its effectiveness. In 1988 Professor Miller et al. [[4](#_bookmark3)] developed a fuzzing tool to test Unix programs’ robustness. The goal of the tool is not to evaluate the safety of the system [49], but to evaluate the code quality and reliability of the system. At that time, fuzzing was simply feeding a program with random inputs. Subsequently, some researchers proposed various methods to improve fuzz testing. (i) Model-based fuzzing [[24](#_bookmark23)–[26](#_bookmark24)] models the input data based on a specific model. (ii) Grammar-based fuzzing [[27](#_bookmark25)–[29](#_bookmark26)] utilize the input data grammar to guide the test data generation. Because of the effectiveness, fuzzing has been studied in network protocol testing field. Aitel et al. developed a block-based approach by divide the network packet into several blocks. Y. Hsu et al. [[31](#_bookmark28)] conducted the testing by abstracting a behavioral model from target protocols. These constant efforts make fuzz testing more and more mature.

Nowadays, with strong learning ability, deep learning is being applied to various fields. Without exception, some studies have incorporated deep learning into fuzzing. P. Godefroid et al. [[32](#_bookmark29)] proposed a sequence-to-sequence model to learn the input grammar of PDF object to help produce fuzzing data for PDF parser. William Blum et al. [[33](#_bookmark30)] also applied sequence-to-sequence neural network model to enhance the AFL (American Fuzz Lop) [[34](#_bookmark31)] fuzzer in which the model at- tempts to learn the optimal mutation locations in the input files. It uses RNN as an assistive technology to improve the AFL’s performance toward stand-alone programs. Chockalingam [[35](#_bookmark32)] uses LSTM model to do intrusion detection about CAN bus protocol. 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.

VI. CONCLUSIONS AND FUTURE WORKS

In this study, we propose an effective fuzzing methodology based on DCGAN 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.
  2. J. P. McDermott, “Attack net penetration testing,” in *Proceedings of the 2000 workshop on New security paradigms*. ACM, 2001, pp. 15–21.
  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. B. P. Miller, D. Koski, C. P. Lee, V. Maganty, R. Murthy,
     1. Natarajan, and J. Steidl, “Fuzz revisited: A re- examination of the reliability of unix utilities and ser- vices,” Technical Report CS-TR-1995-1268, University of Wisconsin, Tech. Rep., 1995.
  5. T. Wang, Q. Xiong, H. Gao, Y. Peng, Z. Dai, and

S. Yi, “Design and implementation of fuzzing technology for opc protocol,” in *Intelligent Information Hiding and Multimedia Signal Processing, 2013 Ninth International Conference on*. IEEE, 2013, pp. 424–428.

* 1. A. G. Voyiatzis, K. Katsigiannis, and S. Koubias, “A modbus/tcp fuzzer for testing internetworked industrial systems,” in *Emerging Technologies & Factory Automa- tion (ETFA), 2015 IEEE 20th Conference on*. IEEE, 2015, pp. 1–6.
  2. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu,

D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

* 1. T. Karras, S. Laine, and T. Aila, “A style-based generator architecture for generative adversarial networks,” *arXiv preprint arXiv:1812.04948*, 2018.
  2. A. Radford, L. Metz, and S. Chintala, “Unsupervised reresentation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.
  3. X. Zhang, J. Zhao, and Y. LeCun, “Character-level con- volutional networks for text classification,” in *Advances in neural information processing systems*, 2015, pp. 649– 657.
  4. S. Ioffe and C. Szegedy, “Batch normalization: Accelerat- ing deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
  5. B. Xu, N. Wang, T. Chen, and M. Li, “Empirical eval- uation of rectified activations in convolutional network,” *arXiv preprint arXiv:1505.00853*, 2015.
  6. D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
  7. I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *International conference on machine learn- ing*, 2013, pp. 1139–1147.
  8. J. Salamon and J. P. Bello, “Deep convolutional neural networks and data augmentation for environmental sound classification,” *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017.
  9. M. Wollschlaeger, T. Sauter, and J. Jasperneite, “The fu- ture of industrial communication: Automation networks in the era of the internet of things and industry 4.0,” *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17– 27, 2017.
  10. G. Collins, “Pymodbus pakage,” [https://pypi.org/project/ pymodbus/](https://pypi.org/project/pymodbus/).
  11. Beckhoff, “Beckhoff,” <https://www.beckhoff.com/>.
  12. Beckhoff-EK1100, “Ek1100,” [http://www.beckhoff.com.](http://www.beckhoff.com/) cn/cn/default.htm?Ethercat/ek1100.htm/.
  13. Beckhoff-EL1004, “El1004,” [http://www.beckhoff.com.](http://www.beckhoff.com/) cn/cn/default.htm?Ethercat/el1004.htm/.
  14. Beckhoff-EL2004, “El2004,” [http://www.beckhoff.com.](http://www.beckhoff.com/) cn/cn/default.htm?Ethercat/el2004.htm/.
  15. Beckhoff-ET2000, “Et2000,” [https://www](http://www.beckhoff.com/).beckhof[f.com/](http://www.beckhoff.com/) english.asp?ethercat/et2000.htm/.
  16. Wireshark, “Wireshark,” <https://www.wireshark.org/>.
  17. M. Peroli, F. De Meo, L. Vigano`, and D. Guardini, “Mobster: A model-based security testing framework for web applications,” *Software Testing, Verification and Reliability*, vol. 28, no. 8, p. e1685, 2018.
  18. M. Utting, A. Pretschner, and B. Legeard, “A taxonomy of model-based testing approaches,” *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.
  19. A. Lunkeit and I. Schieferdecker, “Model-based security testing-deriving test models from artefacts of security engineering,” in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2018, pp. 244–251.
  20. R. Hodova´n, A´ . Kiss, and T. Gyimo´thy, “Grammarinator:

a grammar-based open source fuzzer,” in *Proceedings of the 9th ACM SIGSOFT International Workshop on Au- tomating TEST Case Design, Selection, and Evaluation*. ACM, 2018, pp. 45–48.

* 1. S. Jero, M. L. Pacheco, D. Goldwasser, and C. Nita- Rotaru, “Leveraging textual specifications for grammar- based fuzzing of network protocols,” *arXiv preprint arXiv:1810.04755*, 2018.
  2. T. Guo, P. Zhang, X. Wang, and Q. Wei, “Gramfuzz: Fuzzing testing of web browsers based on grammar analysis and structural mutation,” in *Informatics and Ap- plications (ICIA), 2013 Second International Conference on*. IEEE, 2013, pp. 212–215.
  3. D. Aitel, “The advantages of block-based protocol anal- ysis for security testing,” *Immunity Inc., February*, vol. 105, p. 106, 2002.
  4. Y. Hsu, G. Shu, and D. Lee, “A model-based approach to security flaw detection of network protocol implemen- tations,” in *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*. IEEE, 2008, pp. 114–123.
  5. P. Godefroid, A. Kiezun, and M. Y. Levin, “Grammar- based whitebox fuzzing,” in *ACM Sigplan Notices*, vol. 43, no. 6. ACM, 2008, pp. 206–215.
  6. M. Rajpal, W. Blum, and R. Singh, “Not all bytes are equal: Neural byte sieve for fuzzing,” *arXiv preprint arXiv:1711.04596*, 2017.
  7. MichaZalewski, “American fuzzy lop,” [http://lcamtuf. coredump.cx/afl/](http://lcamtuf.coredump.cx/afl/).
  8. V. Chockalingam, I. Larson, D. Lin, and S. Nofzinger, “Detecting attacks on the can protocol with machine learning.”
  9. Bell, Ron. "Introduction to IEC 61508." *Proceedings of the 10th Australian workshop on Safety critical systems and software-Volume 55*. Australian Computer Society, Inc., 2006.
  10. Piggin, R. S. H. "Development of industrial cyber security standards: IEC 62443 for SCADA and Industrial Control System security." *IET Conference on Control and Automation 2013: Uniting Problems and Solutions*. IET, 2013.
  11. Yoon Kim. 2014. Convolutional neural networks for sentence classification. arXiv preprint arXiv: 1408.5882 (2014).
  12. Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In Advances in neural information processing systems. 649–657.