2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)

SeqFuzzer: An Industrial Protocol Fuzzing Framework from a Deep Learning Perspective

Hui Zhao†, Zhihui Li†, Hansheng Wei†, Jianqi Shi†*, Yanhong Huang†‡,
† National Trusted Embedded Software Engineering Technology Research Center
East China Normal University, Shanghai, China
* Hardware/software Co-Design Technology and Application Engineering
Research Center, Shanghai, China
‡ Shanghai Key Laboratory of Trustworthy Computing, Shanghai, China
Email: {hui.zhao, zhihui.li, hansheng.wei}@ntesec.ecnu.edu.cn
{jqshi, yhhuang}@sei.ecnu.edu.cn

***Abstract*—Industrial networks are the cornerstone of modern industrial control systems. Performing security checks of industrial communication processes helps detect unknown risks and vulnerabilities. Fuzz testing is a widely used method for performing security checks that takes advantage of automation. However, there is a big challenge to carry out security checks on industrial network due to the increasing variety and complexity of industrial communication protocols. In this case, existing approaches usually take a long time to model the protocol for generating test cases, which is labor-intensive and time-consuming. This becomes even worse when the target protocol is stateful. To help in addressing this problem, we employed a deep learning model to learn the structures of protocol frames and deal with the temporal features of stateful protocols. We propose a fuzzing framework named *SeqFuzzer* which automatically learns the protocol frame structures from communication traffic and generates fake but plausible messages as test cases. For proving the usability of our approach, we applied SeqFuzzer to widely-used Ethernet for Control Automation Technology (EtherCAT) devices and successfully detected several security vulnerabilities.**

***Index Terms*—industrial safety, deep learning, vulnerability mining, self-learning, fuzzing, EtherCAT**

I. Introduction

With the development of networks and technologies, the industrial control system (ICS) has developed rapidly, especially in terms of industrial network communication. Technologies like "Industrial Ethernet" have contributed greatly to the development of industrial communication systems. However, these advantages also pose new threats to the ICS [1]. Originally, the ICS was isolated from the Internet, which ensured its communication security to a large extent. Nowadays, however, the ICS network is interconnected with the outside world, thus exposing itself to various network attacks.

Performing security testing on industrial network protocols can help decrease the security risks of the ICS. Fuzzing [2] is a widely used technology for handling network security. It is a kind of testing technology which inputs the target system with random and malformed test data in order to trigger system abnormalities so that potential vulnerabilities can be identified. Fuzzing is often combined with other technologies (e.g., context-free grammar) in order to achieve better fuzzing results. When it comes to the application of fuzzing in the industrial circle, researchers have developed various fuzzing tools for revealing security loopholes in network protocols or industrial devices. Wang et al. proposed a fuzzing technology for Open Platform Communications protocol [3]. Voyiatzis et al. designed a fuzzing technology for Modbus protocol [4]. A vulnerability scanner SimaticScan was developed for Siemens SIMATIC Programmable Logic Controllers (PLC) [5]. In addition, Zhang et al. proposed a fuzzing approach for Profinet Discovery and Configuration Protocol [6]. These efforts have contributed greatly to the development of fuzzing technology and have improved industrial security. However, they also have some limitations. On the one hand, these fuzzing tools are developed for specific protocols. On the other hand, these tools generally require some understanding of the formats or skeleton of the target protocol, which is difficult for private protocols. The situation is even worse when the protocol is stateful.

Stateful network protocols are required in order to keep track of a server's internal state and can be modeled as finite state machines, where the entire communication process consists of a series of state transitions. The current state of a network communication depends on its previous state and the protocol message in each state must conform to a predetermined format [7]. In this case, the protocol that has an ordered data transfer feature is more complex to understand, which makes fuzzing harder to perform.

Therefore, we are committed to developing a general industrial network protocol fuzzing approach that figures out protocol formats automatically and deals with the temporal features of stateful protocols. Sequence-to-sequence (seq2seq) [8] is an encoder-decoder model structure that can handle input and output sequences of different lengths. In addition, the Long Short-Term Memory (LSTM) [9] model is a widely-used deep learning model that has shown great power in learning the structure and precise timing of sequence data [10]. Here, we combined deep learning with fuzzing technology. The key contributions of this paper are as follows:

1. A seq2seq-based fuzzing approach is proposed, which

*Corresponding Author

978-1-7281-1736-2/19/$31.00 ©2019 IEEE
DOI 10.1109/ICST.2019.00016

can be used to learn the formats and state transition relations of protocols and perform security testing. We apply LSTM as the encoder and the decoder of seq2seq. The encoder LSTMs model learns the syntax of real protocol sequences, while the decoder LSTMs model generates real-looking but fake protocol messages which are used as fuzzing data.

2. A general industrial network protocol fuzzing framework, *SeqFuzzer*, is proposed, which is protocol-independent. We perform testing experiments on the Ethernet for Control Automation Technology (EtherCAT) protocol using SeqFuzzer.

3. Several security vulnerabilities of the EtherCAT protocol were detected by SeqFuzzer. To the best of our knowledge, this paper presents the first attempt applying fuzzing to EtherCAT. EtherCAT is a real-time Ethernet-based stateful industrial protocol. It is well known for its high speed and efficiency. Many researchers have used EtherCAT to develop or improve the communication performance of industrial systems [11], [12], [13]. The security of the EtherCAT protocol is very important for the entire ICS. We found a number of potential vulnerabilities in EtherCAT, such as packet injection attack, Man-in-the-Middle attack, Media Access Control address spoofing, and other unknown attacks. The experimental results demonstrate the effectiveness of SeqFuzzer in mining stateful protocol vulnerabilities with no prior knowledge.

The rest of this paper is organized as follows. Section 2 introduces the LSTM and seq2seq models. Section 3 details the deep-learning-based fuzzing method and the SeqFuzzer framework. Section 4 presents an experiment on EtherCAT. Related work is then discussed in Section 5. Finally, we come to a conclusion and discuss some ideas about future work in Section 6.

II. Preliminary

In this paper, we utilize the deep learning model seq2seq. A deep learning LSTM network is used as the encoder and decoder of seq2seq to learn the stateful protocol syntax. In this section, we will introduce the LSTM and seq2seq models.

A. Long Short-Term Memory Network

The Recurrent Neural Network (RNN) is a type of neural network for processing sequential data. It performs the same operation for each element in the sequence and has the same weight parameters at each time step. Therefore, when learning long-term sequences, it will cause gradient disappearance or explosion problems [14].

An LSTM [9] is a special RNN that can avoid the problems of long-term dependencies [10]. Its structure is shown in Fig. 1. In addition to the original hidden unit h_t, the LSTM adds several architectures of units: a cell state c_t and three control gates which are the forget gate f_t, the input gate i_t, and the output gate o_t. c_t is used to save the long-term state and the three control gates determine whether the data is added to c_t [15] according to a pass rate between 0 and 1. Therefore, each LSTM cell has different weight parameters and the effects of hidden cells on the next cells are controllable.

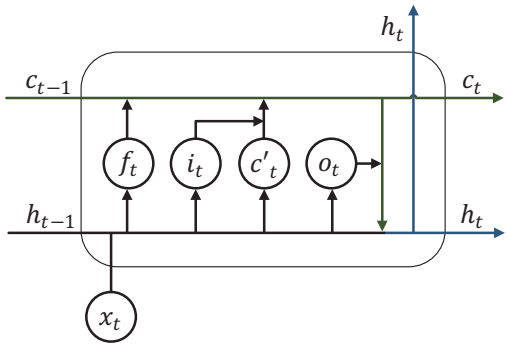

Fig. 1. Long Short-Term Memory network structure

For example, a long sequence $x : \langle x_0, x_1, ..., x_t, ..., x_n \rangle$ is entered into the LSTMs. At time t, f_t determines the degree of forgetting of the previous c_{t-1} based on the x_t and output of the last hidden cell, h_{t-1}. i_t determines the update of the current cell state by h_{t-1} and x_t. c'_t updates its status based on the h_{t-1} and the i_t. Finally, o_t determines the output of the current cell h_t by filtering c_t. Compared with other neural network models, the mechanisms of cell state and gate control allow the LSTM to better learn long sequence relationships. Therefore, the LSTM has achieved great success in many fields, such as large vocabulary speech recognition [16], semantic representation, [17] and information retrieval [18]. In this paper, we utilize the great potential of an LSTM to learn and predict the precise timing of complex network protocols.

B. Sequence-to-Sequence Network Model

Seq2seq [8] is an encoder-decoder network model structure. The input and output of seq2seq are sequences which can be of different lengths. The encoder model learns the information of the input sentence $x : \langle x_0, x_1, ..., x_n \rangle$, and outputs a latent vector C containing the input sentence information. The C is learned by the decoder model for predicting the output sentence $y : \langle y_0, y_1, ..., y_m \rangle$.

For the encoder and decoder models, RNN, LSTM, and other neural network models can be used. The input and output data can be in the form of images, voices, videos, etc. In this paper, we utilized the LSTM as the encoder and decoder of seq2seq to learn the stateful protocol grammar.

III. SeqFuzzer Framework

As stated above, the purpose of this work is to search for effective solutions for the automatical fuzz testing of private protocols whose formats are unknown and to properly handle the temporal features of stateful protocols. To achieve this, we utilize a deep learning method. By training the seq2seq model on the network traffic of a real industrial environment, we can derive the structure characteristics (formats) included in the model parameters. The temporal features are also learned for stateful protocols.

In this paper, we propose the SeqFuzzer framework, as shown in Fig. 2. Under the workflow of the SeqFuzzer, the

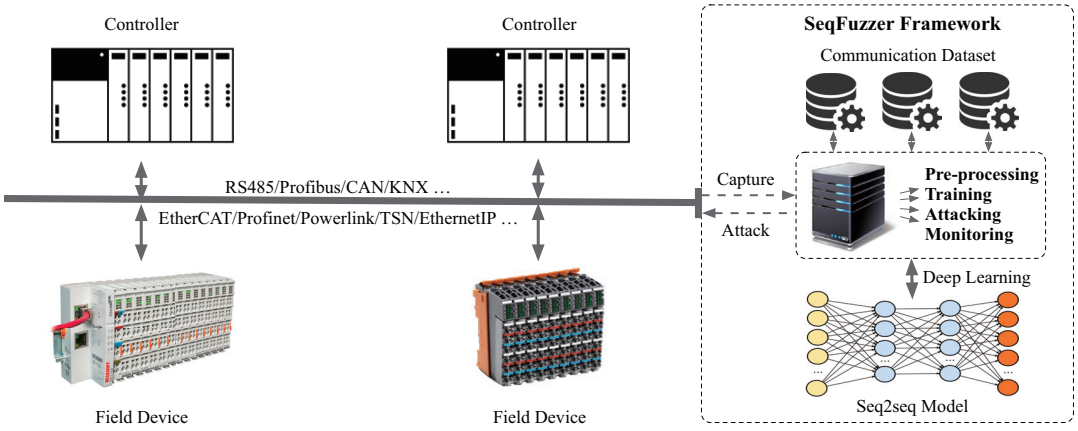

Fig. 2. SeqFuzzer framework

introduction can be divided into the following steps. The first step is to capture network traffic in the real industrial scenario and to perform some pre-processing steps on the captured data. The second step is to construct and train the seq2seq model. The third step is to generate test cases using the trained seq2seq model. The final step is to perform testing in order to find vulnerabilities where monitoring is needed to watch for irregular ICS behaviors. The details of these steps are provided in the following subsections.

A. Training Data Capture and Pre-processing

Since we leverage deep learning techniques, it is important to collect training data. Equipment is needed for capturing original messages to learn in the real industrial control environment. In the present work, we captured the real protocol messages as the training data.

After we have captured enough real protocol messages, we perform two simple data pre-processing steps on them which are converted into a decimal sequence file and normalized to standard decimal data containing the temporal features of the protocol. The details of the two pre-processing steps are given below.

- Feature data conversion. Using capture tools, we can obtain a digital protocol message file where each sequence represents one protocol message. We transfer the digital messages into decimal sequences. The decimal file containing the real protocol messages is obtained.
- Adding special symbols. In data communication, the lengths of the messages are different. Before starting the seq2seq model with decimal sequences, we need to add some special characters in order to standardize the training. In this paper, we use PAD (pad) to fill short decimal sequences, STA (start) as the sequences start flag, and END (end) as the sequences end flag.

At this point, the processed decimal feature data is stored in the communication dataset as input data for the deep learning seq2seq model.

This subsection embodies the encapsulation of SeqFuzzer. It only requires simple processing and does not require an understanding of the true meaning of decimal feature data.

B. Seq2seq Model Construction and Training

In the literature on network protocol fuzzing, the quality of test cases has a direct impact on the final fuzzing result. High-quality test cases may have more chances to find target vulnerabilities. Researchers generally go to great lengths to acquire high-quality test cases. In this paper, obtaining high-quality test cases depends on two parts, one is the initial sequence captured, which was introduced above, and the other is to build a network model to generate test cases.

This subsection embodies the versatility and automation of SeqFuzzer. It utilizes the deep learning models of seq2seq and LSTM to obtain favorable test cases.

1) Seq2seq Model Building: We assume s_i is the input sequence that indicates the decimal feature, and our aim is to obtain s_{i+1} that conforms to the state transition relations of the stateful protocol. At the same time, we use X and Y to describe the input feature sequences and the corresponding next feature sequences, respectively.

We apply a three-layer deep LSTMs in the seq2seq model whereby an encoder LSTMs model learns the syntax of the protocol data, a decoder LSTMs model predicts the corresponding receiving protocol sequence, and the semi-effective protocol data (which is fake but plausible) along with the sequential syntax structure is obtained for fuzzing. A three-layer deep LSTMs model can express features more abstractly at a higher level, improve recognition accuracy, and reduce training time [19]. The seq2seq network structure is shown in Fig. 3. It includes an input layer, an embedding layer, a multilayer LSTMs, and an output layer. Both the input and output are sequences and are not required to be equal in length.

After data pre-processing, each message is converted into a decimal feature sequence, such as '$[1, 15, 232, 0, ...]$', and converted into a vector by the embedding layer before being input into the LSTMs. We use the feature sequence

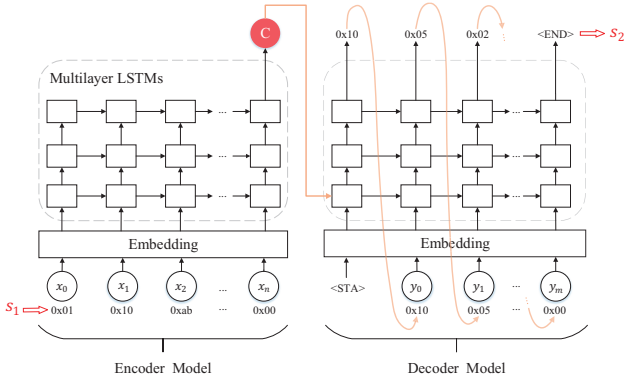

Fig. 3. Seq2seq structure

$X(x_i|\langle x_0, x_1, ..., x_n\rangle)$ which is processed by the embedding layer as the input data for the first layer of the LSTMs. Each LSTM cell h_i has two inputs, o_i and h_{i-1}, that contain the previous protocol statement information. h_i can be expressed as the following formula (1):

$$h_i = \begin{cases} \xi(o_i), & i = 0 \\ \xi(o_i, h_{i-1}), & i \neq 0 \end{cases} \tag{1}$$

where ξ is a non-linear activation function, such as tanh, sigmoid, etc., and o_i is the output of the previous LSTM cell. When LSTM layer=1, $o_i = x_i$. The output of the first layer is used as input data for the second layer to convey the state transition relations of the stateful protocol. In the third layer of the encoder LSTMs model, the output of each LSTM cell is only used for the input of the next cell. Only the output of the last cell is saved, which is the latent vector $C = \kappa(x_0, x_1, ..., x_n)$. It represents all the state and structure information of the input protocol sequence X.

The decoder LSTMs model decodes C and predicts the next message s_{i+1} that matches the features of the stateful protocol. It is different from the encoder LSTMs model in which each cell output $Y(y_i|\langle y_0, y_1, ..., y_m\rangle)$ generated by the third layer of the decoder LSTMs model is saved and input to the next stage, $y_i : \gamma(STA, y_0, y_1..., y_{i-1})$.

As shown in Fig. 3, the stateful protocol feature message $s_1 = \langle 0x01, 0x10, 0xab, ..., 0x00\rangle$ is the input of the encoder LSTMs model, and the latent vector C representing s_1 is obtained. In the decoder LSTMs model, $y_0 = 0x10$ is obtained based on decoding the inclusion of stateful protocol information C. y_0 is also used as the input for the next stage. $y_1 = 0x05$ is then obtained. Finally, $s_2 = \langle 0x10, 0x05, 0x02, ...0x00\rangle$ is obtained. s_2 is the next sequence which matches the state transition relationship of s_1, as predicted by the seq2seq model.

2) Seq2seq Model Training: The input feature data is divided into three parts, a train set, a validation set, and a test set. We utilize the Adam gradient optimization algorithm [20] which is an effective random optimization algorithm. It has a small memory requirement and is very effective for large datasets. In the training and validation phase, we utilize the N-gram [21] as a criterion for judging the similarity between the generated data Y and the true data. The more N-grams that the generated data share with the real data s_{i+1}, the better the generated data.

The parameters of the training, validation and test phases are shared, and the training and validation phases are designed to learn ideal parameters. In order to train the model well, we use the decoder LSTMs model differently for training and validation.

In the training phase, the real sequence s_i is used as the input of the encoder LSTMs model as it can learn the sequence syntax better. Instead of generated data Y, the next real data s_{i+1} is used as the input of the decoder LSTMs model. Instead of generating, the decoder learns how to predict during the training process.

The validation phase can prevent overfitting and make the model more robust. Parameters trained by the training phase are used for prediction. In this phase, we value the results predicted by the decoder, and the prediction result y_i of each stage is used as the output of the next stage.

Before formally starting, we need to set the parameters for the seq2seq model training randomly. Dai et al. [22] found that setting parameters by pre-training is better than random initialization for deep learning network models, which can significantly stabilize the training. In SeqFuzzer, the pre-training includes an input layer, an embedding layer, a multilayer LSTMs, and an output layer, which is the same structure as the formal training. Batch size data is used for pre-training with low dimensional training. The weights obtained from the pre-training are used as initialization parameters Ws for the encoder LSTMs model and decoder LSTMs model.

C. Test Cases Generation

In the predicting step, the input of the decoder LSTMs model requires C and Y to participate. After operating the training and validation, favorable parameters were obtained. The data sequence $X(x_i|\langle x_0, x_1, ..., x_n\rangle)$ was entered into the encoder LSTMs model. The output y_i of each decoder LSTM model stage is the result of the prediction. There is a normalization recovery step at the output layer of the decoder LSTMs model. $Y(y_i|\langle y_0, y_1, ..., y_m\rangle)$ is normalized to become the semi-effective fuzzing data we need.

Our goal is to automatically learn the temporal features of stateful protocols and generate real-looking fake messages for fuzzing. In this subsection, semi-effective fuzzing data is acquired and stored in the communication dataset for fuzzing.

D. Testing

This subsection is the ultimate embodiment of SeqFuzzer's high efficiency. Semi-effective fuzzing data is sent to the field device for attacking and monitoring. Before performing the fuzzing, we need to set up a fake controller to attack the field device. Capture tools are utilized to monitor the response of the field device to the fuzzing. Finally, the outbound fake packets and the inbound real packets are analyzed by the detection program for vulnerabilities.

IV. Experiment Evaluation

In this section, the experiment on the EtherCAT protocol is discussed. This includes the EtherCAT message capture, message data pre-processing, seq2seq model of the self-learning syntax, generation of test data, fuzzing, and result analysis. Before introducing the experiment, we will provide a brief introduction to EtherCAT.

A. EtherCAT Real-Time Industrial Ethernet Protocol

EtherCAT is a real-time industrial Ethernet technology. EtherCAT uses master/slave mode media access control and sends data in a specific ring topology, depending on the implementation of the master. It uses standard Ethernet packets with a data frame type of 0x88A4. EtherCAT is stateful and has four states, including Init, Pre-Operational, Safe-Operational, and Operational. Its transition from initialization to running state must follow the order of an Init → Pre-Operational → Safe-Operational → Operational sequence and cannot be done by skipping the steps. The conversion relationship between states is shown in Fig. 4.

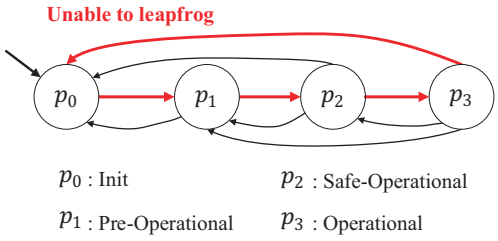

Fig. 4. EtherCAT state machine

B. Real EtherCAT Data Capture

In this experiment, the controller was a Beckhoff [23] PLC, and the field devices were a Beckhoff bus coupler EK1100, digital input EL1004, digital output EL2004, and system terminal EL9011. The capture tools were the Beckhoff Listener ET2000 [24] and the terminal capture software Wireshark [25]. ET2000 is an industrial Ethernet multi-channel listener and has a software interface that can display the captured packets via the Wireshark software on the terminal equipment. ET2000 can support all real-time Ethernet standards, such as EtherCAT, Profinet, etc. Moreover, these tools have little impact on the actual industrial communication environment, which make them good candidates for capturing network protocol messages.

The capture environment is shown in Fig. 5. The ET2000 is connected between the master and the salves. The Wireshark software on the PC is connected to the ET2000 interface and displays the data captured by the monitor ET2000. Using Wireshark, we obtain a hexadecimal array file, where each array represents a message. We capture millions of protocol messages with the ET2000 and display them on Wireshark.

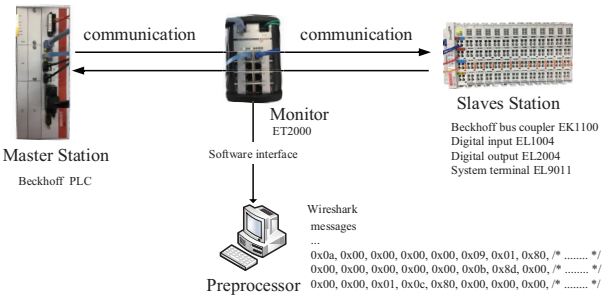

Fig. 5. Capture environment

C. Data Pre-processing

Protocol messages are exported as a hex C array file on Wireshark and processed into hexadecimal feature data. The data are then converted into a decimal feature data file. For example, 0x00 :→ 0, 0xff :→ 255. The data conversion is shown in Fig. 6. Since the lengths of the EtherCAT protocol messages are not uniform, we insert the special character PAD mentioned above to fill the shorter decimal feature sequences, which makes the length of all messages consistent with the longest length max. The input feature sequences are then obtained, as shown in Fig. 7.

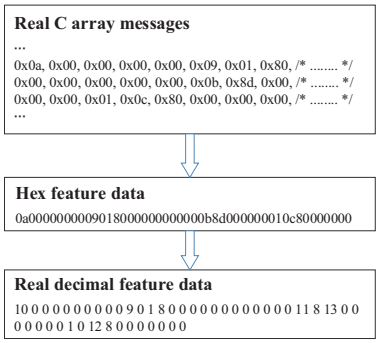

Fig. 6. Data conversion

D. Training of Seq2seq and Generation

The batch input feature sequences are used as input to the seq2seq model for pre-training and obtain the initialization parameters for the formal training. The input feature data is divided into three parts, training data, validation data, and test data. After the training and validation steps are completed, relevant parameters are obtained for generating semi-effective sequences. The special character STA indicates the beginning of the prediction, and the special character END indicates the end of the prediction. We utilize the Adam algorithm to train feature data for multiple epochs (epoch = 1,3,8,13,18,23,28). The data is saved for different epoch training. All generated sequences are of length max, and the special character PAD in each sequence is removed and restored to the original length of the sequence.

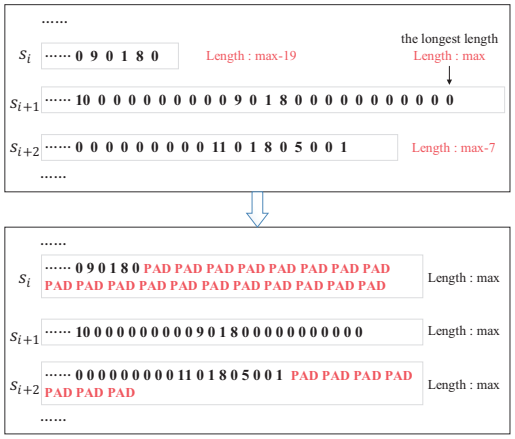

Fig. 7. PAD fill shorter decimal feature sequences

E. Fuzzing and Results

SOEM (Simple Open EtherCAT Master) [26] is a free, open source EtherCAT software library that sends and receives EtherCAT frames through bare sockets. Based on the SOEM source code, we developed the EtherCAT test master station on the Ubuntu17 Linux operating system. We utilized a USB network port hub as the master PC interface to implement the hardware capabilities of the Beckhoff master. In order to have the same monitoring conditions as the capture environment, the ET2000 is used to connect the master station and the slaves station EK1100.

The semi-effective test data generated by SeqFuzzer is sent to the slaves station via the SOEM master station, and the messages are displayed on the Wireshark as the ET2000 listens. During the experiment, we found red dataframe error flags in Wireshark. In addition, in order to determine whether the transmitted message is received, we have two criteria: the Diagnostic LEDs for the EtherCAT fieldbus and the WKC (Working Counter) processing flag in the EtherCAT sub-messages.

EtherCAT protocol messages are input into SeqFuzzer, which learns their syntax and constructs fake fuzzing data.

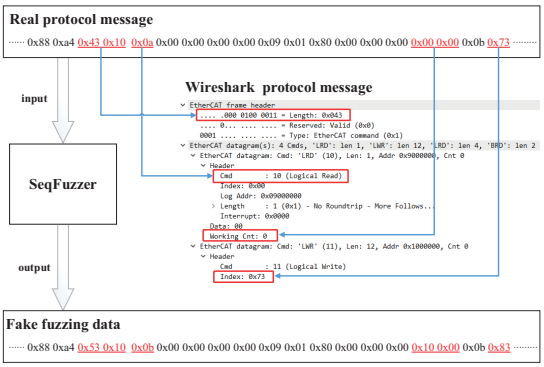

Fig. 8. Generate data and real data

Compared to protocol messages, fuzzing data changed some bytes by SeqFuzzer, as shown in Fig. 8. We judged the results of the experiment with the following criteria:

- Recognition rate. Whether the sent message is recognized by the Wireshark as the EtherCAT protocol.
- Acceptance rate. The proportion of messages processed by the slaves.
- Detection ability. The ability of the EtherCAT protocol in detecting vulnerabilities.

1) Recognition Rate: The protocol flag is displayed in the syntax column *protocol* of Wireshark. The frame type of the transmitted message data is 0x88A4, but some failed data is misidentified. The recognition rate for different epochs are shown in Table I. When the epoch is equal to 1, the result does not show a high recognition rate. A large amount of packet data is not recognized as an EtherCAT message. This was probably because of insufficient training on the LSTMs. When the epoch is e3, e8, or e13, the ideal recognition rate is obtained. This shows that SeqFuzzer is highly automated and accessible given that the data generation process does not require human involvement, not prior knowledge of the protocol is required, and it does not depend on a specific protocol.

TABLE I
RECOGNITION RATE

	e1	e3	e8	e13	e18	e23	e28
Recognition Rate	52%	100%	98.7%	99.93%	95.4%	96.3%	92.3%

* Where ex indicates epoch = x.

2) Acceptance Rate: According to the *index* field in the sub-message, the transmit and receive message pairs are obtained. The acceptance rate is obtained by calculating the number of *WKC* field changes in the message pairs, as shown in Table II. It shows that the generated data satisfies the state transition relationship of EtherCAT. This is strong proof that SeqFuzzer can generate test data for a stateful protocol. The acceptance rate is favorable when the epoch is between 3 and 13. As the epoch increases, the acceptance rate decreases.

TABLE II
ACCEPTANCE RATE

	e1	e3	e8	e13	e18	e23	e28
Acceptance Rate	92.24%	99.99%	99.28%	98%	97.2%	93.11%	90.86%

3) Detection Ability: There are many known vulnerabilities, such as MITM, MAC address spoofing, slave address attack, packet injection and so on. The MAC address is important for protocol detection. With regards to EtherCAT, we found that Wireshark and LEDs would behave differently when sending different MAC addresses. For better analysis, we discuss detection ability in two situations:

- Real MAC address plus constructed content. The real MAC address instead of the MAC address spoof combine with the protocol data content constructed by SeqFuzzer to become fuzzing data for testing for vulnerabilities.
- Constructed MAC address plus constructed content. Protocol data constructed by SeqFuzzer was utilized to test the MAC address spoofing vulnerability.

Real MAC Address plus Constructed Content The MAC address of the generated protocol data is replaced with the real MAC address in order to obtain the message pairs. After analysis, we detected the following vulnerabilities:

1 Packet injection attack. SeqFuzzer modified the *data* field of the sub-message, but the *length* field in the *EtherCAT frame header* was not equal to the number of bytes in the sub-message. However, the message was received, which is a packet injection attack.
2 MITM attack. In the message pairs, we checked a situation where the message data generated by SeqFuzzer only changed the *data* field, but the slaves processed it. If the master is manipulated to use the same MAC address as the PLC and modifies the data in the message, once the slaves accept it, it will lead to MITM attack.
3 Unknown attacks. There are other potential crises that may be dangerous, such as the *data* field changed, but the *WKC* did not.

Constructed MAC Address plus Constructed Content We send the protocol data generated by SeqFuzzer from the SOEM master to the slaves. Watching the LEDs flashing and looking at the changes in *WKC* to determine whether the slaves have received a message from an unknown MAC address. If the message is received, there is a MAC address spoofing vulnerability.

DetectEcat is a program that was developed for detecting vulnerability statistics. The program algorithm is shown in Alg. 1. After sending 132,200 generated fake messages, the statistics of the attacks are shown in Table III. Among them, the number of packet injection attacks and MITM attacks is relatively high. When the epoch ranges from 5 to 13, the result shows favorable detection ability. However, in order to obtain better experimental results, the epoch should be set between 8-13.

Algorithm 1 DetectEcat

for each $< s_i, r_i > \in Message\ Pairs$ **do**
  if WKC in $s_i \neq WKC$ in r_i **then**
    if MAC is $True$ **then**
      if $Length$ in s_i = $Length$ in r_i **then**
        $Packet\ injection$
      end if
      if $Data$ in $s_i \neq Data$ in r_i **then**
        $MITM\ attack$
      end if
    else
      $MAC\ address\ spoofing$
    end if
  else
    if $Data$ in $s_i \neq Data$ in r_i **then**
      $Unknown\ attacks$
    end if
  end if
end for

TABLE III
SEQFUZZER DETECTS ETHERCAT VULNERABILITIES STATISTICS

	e1	e3	e8	e13	e18	e23	e28
Packet injection	19	1724	282	83	93	91	64
MITM attack	13	1420	1001	285	72	43	44
MAC address spoofing	0	57	102	69	52	11	3
Unknown attacks	3	6	0	2	0	0	0

In this experiment, SeqFuzzer automatically learned the EtherCAT payload frame structure, generated fuzzing data with a high receiving rate and detection ability even when the EtherCAT format is unknown, and successfully detected several security vulnerabilities.

V. RELATED WORK

Fuzz testing for network protocol. Fuzzing is not a new technology. In fact, it has been prevalent for more than 20 years. Miller [2] was the pioneer of fuzzing through his work testing UNIX programs. Fuzzing has since been applied widely and developed greatly. More and more technologies were introduced to enhance the power of fuzzing. Because of its effectiveness, fuzzing has many applications in the field of network protocol testing. Aitel et al. [27] introduced a block-based approach that divides a network protocol packet into several blocks and tests the target by removing all known factors in the network protocol. Some researchers conduct security testing by synthesizing abstract behavioral models from target protocols [28] or by constructing a finite state automaton [29]. The reverse engineering method is also used to perform dynamic monitoring of the target protocol and analyze the processing [30], [31]. Comparetti et al. [32] applied the state machine to reverse engineering, which enhanced the results.

Deep learning for grammar-based fuzzing. With the rapid development of deep learning technology, fuzzing was also combined with it. Learn&fuzz [33] uses the seq2seq model to learn the syntax of PDF (a complex input format) objects and uses the learned syntax to produce test data for testing PDF parsers. Rajpal et al. used a neural network model to predict good and bad locations in the input file to perform fuzzing mutations [34]. Deep learning is also applied in network protocol fuzzing. Fan et al. [35] used deep neural networks to learn a generative input model of proprietary network protocols, and used the learned model to generate new messages for testing. Chockalingam et al. proposed a deep-learning-based method to detect abnormalities for the Controller Area Network bus [36]. These efforts have made a

number of contributions to network security. In the present work, we utilized the seq2seq model to learn the protocol format. SeqFuzzer can properly handle the temporal features of stateful protocol automatically, especially in the context of an industrial environment. This work is different from others.

Vulnerability detection for real-time Ethernet protocol and EtherCAT. Granat et al. [37] tested packet injection and MITM attacks in the EtherCAT environment and defined a preprocessor for the open source intrusion detection system Snort. The attack is constructed and detected by the developer under the understanding of the protocol. Ovaz Akpinar et al. [38] also proposed the EtherCAT preprocessor for Snort. The EtherCAT vulnerability is known in advance, attack vectors such as MAC spoofing and data injection are generated, then the attack is executed. However, SeqFuzzer is an automatic learning framework that uses a common fuzzing approach to effectively solve the security problems of network protocols. It is an intelligent vulnerability detection tool that does not require knowledge of the protocol rules.

VI. Conclusion and Future Work

Industrial network protocols play very import roles in the industrial environment. To improve industrial security, we have to carry out appropriate testing on industrial network protocols.

However, the testing work is not easy as many industrial network protocols are private, meaning that the protocol format must be interpreted manually, which requires a large amount of labor.

Moreover, there are many stateful protocols. Understanding the temporal features is even more difficult. Current work is almost protocol-specific and cannot handle the feature well. To improve this situation, we leveraged a deep learning model to learn the protocol format and deal with the state transition relations of the stateful protocols .

We have proposed a fuzzing framework SeqFuzzer, which can automatically learn the protocol format from network traffic and generate fake but plausible messages which serve as test cases. We applied SeqFuzzer to test the EtherCAT protocol. SeqFuzzer automatically generated fuzzing data with high receiving rates and detection capabilities even when the format of EtherCAT messages was unknown, and successfully detected several security vulnerabilities of EtherCAT.

In terms of future work, we first plan to combine SeqFuzzer with other deep learning models, such as Convolutional Neural Networks and Generative Adversarial Networks, to achieve better learning of protocol features. Second, we plan to test more industrial protocols, such as Powerlink and Profinet, in order to future evaluate SeqFuzzer.

Acknowledgment

This work is partially supported by Shanghai Science and Technology Committee Rising-Star Program (No.18QB1402000), National Natural Science Foundation of China (No. 61602178 and No. 61602177), China HGJ Project under Grant (No. 2017ZX01038102-002), and National Defense Basic Scientific Research Program of China (No. JCKY2016204B503).

References

[1] M. Cheminod, L. Durante, and A. Valenzano, "Review of security issues in industrial networks," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 277–293, 2013.

[2] 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.

[3] 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.

[4] 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.

[5] R. Antrobus, S. Frey, B. Green, and A. Rashid, "Simaticscan: Towards a specialised vulnerability scanner for industrial control systems." BCS, 2016.

[6] D. Zhang, J. Wang, and H. Zhang, "Peach improvement on profinet-dcp for industrial control system vulnerability detection," in *2015 2nd International Conference on Electrical, Computer Engineering and Electronics*. Citeseer, 2015.

[7] R. Ma, D. Wang, C. Hu, W. Ji, and J. Xue, "Test data generation for stateful network protocol fuzzing using a rule-based state machine," *Tsinghua Science and Technology*, vol. 21, no. 3, pp. 352–360, 2016.

[8] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[11] S. Yan, L. Zhang, Z. Chen, J. Wang, and B. Wang, "A micro-grid control system based on ethercat," in *International Conference on Information Technology, Computer Engineering and Management Sciences*, 2011, pp. 385–388.

[12] D. Orfanus, R. Indergaard, G. Prytz, and T. Wien, "Ethercat-based platform for distributed control in high-performance industrial applications," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–8.

[13] L. Wang, J. Qi, H. Jia, and B. Fang, "The construction of soft servo networked motion control system based on ethercat," in *International Conference on Environmental Science and Information Application Technology*, 2010,

pp. 356–358.
[14] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.
[15] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.
[16] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth annual conference of the international speech communication association*, 2014.
[17] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.
[18] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, "Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 24, no. 4, pp. 694–707, 2016.
[19] S. Zhang, X. Liu, and J. Xiao, "On geometric features for skeleton-based action recognition using multilayer lstm networks," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2017, pp. 148–157.
[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[21] G. Doddington, "Automatic evaluation of machine translation quality using n-gram co-occurrence statistics," in *Proceedings of the second international conference on Human Language Technology Research*. Morgan Kaufmann Publishers Inc., 2002, pp. 138–145.
[22] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in neural information processing systems*, 2015, pp. 3079–3087.
[23] BECKHOFF https://www.beckhoff.com/.
[24] BECKHOFF ET2000 https://www.beckhoff.com/english.asp?ethercat/et2000.htm/.
[25] Wireshark https://www.wireshark.org/.
[26] SOEM https://openethercatsociety.github.io/.
[27] D. Aitel, "The advantages of block-based protocol analysis for security testing," *Immunity Inc., February*, vol. 105, p. 106, 2002.
[28] Y. Hsu, G. Shu, and D. Lee, "A model-based approach to security flaw detection of network protocol implementations," in *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*. IEEE, 2008, pp. 114–123.
[29] S. Gorbunov and A. Rosenbloom, "Autofuzz: Automated network protocol fuzzing framework," *IJCSNS*, vol. 10, no. 8, p. 239, 2010.
[30] G. Wondracek, P. M. Comparetti, C. Kruegel, E. Kirda, and S. S. S. Anna, "Automatic network protocol analysis." in *NDSS*, vol. 8, 2008, pp. 1–14.
[31] W.-M. Li, A.-F. Zhang, J.-C. Liu, and Z.-T. Li, "An automatic network protocol fuzz testing and vulnerability discovering method," *Jisuanji Xuebao(Chinese Journal of Computers)*, vol. 34, no. 2, pp. 242–255, 2011.
[32] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol specification extraction," in *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE, 2009, pp. 110–125.
[33] P. Godefroid, H. Peleg, and R. Singh, "Learn&fuzz: Machine learning for input fuzzing," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2017, pp. 50–59.
[34] M. Rajpal, W. Blum, and R. Singh, "Not all bytes are equal: Neural byte sieve for fuzzing," *arXiv preprint arXiv:1711.04596*, 2017.
[35] R. Fan and Y. Chang, "Machine learning for black-box fuzzing of network protocols," in *International Conference on Information and Communications Security*. Springer, 2017, pp. 621–632.
[36] V. Chockalingam, I. Larson, D. Lin, and S. Nofzinger, "Detecting attacks on the can protocol with machine learning."
[37] A. Granat, H. HÖFKEN, and M. Schuba, "Intrusion detection of the ics protocol ethercat," *DEStech Transactions on Computer Science and Engineering*, no. cnsce, 2017.
[38] K. Ovaz Akpinar and I. Ozcelik, "Development of the ecat preprocessor with the trust communication approach," *Security and Communication Networks*, vol. 2018, 2018.