Lin *et al. Cybersecurity* (2021) 4:28
https://doi.org/10.1186/s42400-021-00087-5

Cybersecurity

RESEARCH Open Access

ICPFuzzer: proprietary communication protocol fuzzing by using machine learning and feedback strategies

Pei-Yi Lin, Chia-Wei Tien*, Ting-Chun Huang and Chin-Wei Tien

Abstract

The fuzzing test is able to discover various vulnerabilities and has more chances to hit the zero-day targets. And ICS(Industrial control system) is currently facing huge security threats and requires security standards, like ISO 62443, to ensure the quality of the device. However, some industrial proprietary communication protocols can be customized and have complicated structures, the fuzzing system cannot quickly generate test data that adapt to various protocols. It also struggles to define the mutation field without having prior knowledge of the protocols. Therefore, we propose a fuzzing system named ICPFuzzer that uses LSTM(Long short-term memory) to learn the features of a protocol and generates mutated test data automatically. We also use the responses of testing and adjust the weight strategies to further test the device under testing (DUT) to find more data that cause unusual connection status. We verified the effectiveness of the approach by comparing with the open-source and commercial fuzzers. Furthermore, in a real case, we experimented with the DLMS/COSEM for a smart meter and found that the test data can cause a unusual response. In summary, ICPFuzzer is a black-box fuzzing system that can automatically execute the testing process and reveal vulnerabilities that interrupt and crash industrial control communication. Not only improves the quality of ICS but also improves safety.

Keywords: Industrial communication protocol, Network protocol fuzzing, Long short-term memory (LSTM), Industrial control system (ICS)

Introduction

With the promotion of Industry 4.0, the demand for a device that can operate automatically has become increasingly large. Not only do factory devices need to be automated to increase efficiency and reduce manual error rates and labor costs but also the electrical and hydraulic systems need to be automated. Industrial communication systems (ICSs) (Poletykin 2018) occupy a very important position in modern automation systems. Supervisory control and data acquisition (SCADA) and the distributed control system (DCS) are widely used to control and monitor the production process, and local operations can receive instructions from a remote system; thus, the protocol that is used for dialogue becomes important (Su et al. 2017).

If ICSs can be attacked through the communication protocol (McLaughlin et al. 2016), it may lead to unsustainable communication between the devices and cause serious problems. One of the most famous attacks is Trisis (Wikipedia contributors 2020a), a well-known malware that disrupted production processes and was able to manipulate security systems. In addition, some devices that were originally isolated from the network can now also be connected externally to facilitate remote control or query data. Many connection functions are not protected by security measures (Nan et al. 2013), such as operating the function without continuing to support or update it, or the logical processing of the protocol is not completed,

*Correspondence: emmily@iii.org.tw
Cybersecurity Technology Institute, Institute for Information Industry Taiwan, China, No. 133, Sec. 4, Minsheng E. Rd., Songshan Dist. 105, Taipei City Taiwan, China

with even several pieces of content being transmitted in plain text. In a factory, if a motor or robotic arm operates excessively, it may cause the motor to burn out and damage the entire machine or even the factory.

A secure device requires many tests to ensure its security and defensiveness, but the source code of software is difficult to obtain, and the details of the hardware architecture are limited to professional knowledge. Therefore, how to protect the security of communication with a device in industry is a key matter. The fuzzing test (Liang et al. 2018) is a kind of preliminary and uncomplicated testing that can be performed on a system. A fuzzer sends data with an abnormal format to the device (Shapiro et al. 2011) to find potential or known threats and assist users in checking the tolerance of the DUT to the abnormal data. The zero-day vulnerability (You et al. 2019) can also be determined through the fuzzing test, serving as the goalkeeper of the system. By sending content of different formats to the device, a user can not only check whether the logical processing inside the system is correct but also perform the robustness test (Grubbs 2018) on the system and test its ability to reply. This is a test that does not cause much harm to the logical core of the system.

Many international testing standards mention the importance of fuzzing tests, such as the Embedded Device Security Assurance (EDSA) (GISA Security Compliance Institute 2020). The security development lifecycle (SDLC) defined by Microsoft (GMicrosoft 2020) also indicates that performing the fuzzing test during the verification phase is important; thus, the fuzzing test is necessary. At present, there are many types of industrial proprietary communication protocols, such as Modbus/TCP or the DLMS/COSEM, which can be used to regularly transmit information about a device or system. Modbus/TCP includes some function codes, as shown in Table 1, that can be customized to meet a specific demand. The DLMS/COSEM is a complicated protocol that few fuzzing systems can test. In the future, more varieties of protocols will be developed to adapt to devices with different functions. However, many fuzzing systems currently require prior knowledge of the format of the protocol for mutation. Users need to define the rule of the protocol and decide whether the field can be mutated or not. In addition, some fuzzers can generate test data automatically but can only test a single protocol, such as MTF (Voyiatzis et al. 2015). Multiple different industrial proprietary communication protocols may be developed on the same device (Lin and Liu 2019). If users need to perform fuzzing tests on those protocols, much time will be required to understand the structure of each protocol, and the mutation rule will have to be defined by the users themselves. A tester who has professional knowledge is even required to assist in testing, and the testing process will require substantial labor costs.

Table 1 Modbus function code categories

Function Code (DEX)	Categories
1∼64	Public function codes
	(Define reading, writing…etc.)
65∼72	User-Defined
73∼99	Public function codes
100∼110	User-Defined
111∼127	Public function codes
128∼255	Reserved

Therefore, we propose a black box and an automated fuzzing system to test the industrial proprietary communication protocol. Our contributions are as follows:

- We automatically created test data by using LSTM to predict the similar data of proprietary protocol and defined the field that can be mutated. The amount of data predicted by LSTM that can be identified as the real protocol content is up to 90% or more. With this feature, testers do not need to spend time understanding the structure of different protocols.
- We used the feedback strategies to adjust the weight of the mutation type to create test data with more possibilities to cause the unusual connection status. Compared with the first round of test results, it can not only help users find 69% more data able to cause an abnormal connection with the device but also highlight the scope of the data that can trigger these connection statuses.
- Finally, we designed five experiments to verify the effectiveness of our system and compared our system with three other well-known fuzzing systems. The tested devices and fuzzing systems include open-source and commercial versions. After comparison, it was found that ICPFuzzer can trigger the DUT to crash with fewer test cases. The number of sending times of ICPFuzzer is less than the commercial version of the fuzzing system. Moreover, we tested the device which uses DLMS/COSEM with no knowledge about this protocol and obtains a good result.

The remainder of this paper is organized as follows. "Learning-based fuzzer architecture design" section describes the architecture and operating process of ICPFuzzer, including the prediction methods related to LSTM and the feedback strategy mutation used to create test data. In "Experiment and evaluation" section, we evaluate our method from different aspects. Five experiments are conducted to verify the system efficacy and compare it with that of other well-known fuzzing

systems. "Discussion" section discusses the limitations of ICPFuzzer. "Related work" section reviews related studies. Finally, "Conclusions" section concludes the paper.

Learning-based fuzzer architecture design

To solve the problem of users having to define the mutation fields and methods and to increase the effectiveness of the test data, we use machine learning and a feedback strategy to create the test data. Therefore, in this paper, we propose a black-box fuzzing system named ICPFuzzer, which is an automated fuzzing test system for industrial proprietary communication protocols. Algorithm 1 shows the procedure of ICPFuzzer. Users do not need knowledge of the protocol; they need to input only the information I (IP or port, etc.) of the device and the PCAP file, which is related to the contents of the protocol that the DUT uses to carry out communication. After starting the process, ICPFuzzer uses machine learning to predict the communication behavior of the DUT and creates test data that are similar to the original communication contents to avoid excessive useless data for testing. Then, we classify and distinguish the unknown format of the protocol. The contents of the data are divided into many blocks. One of those blocks is selected for mutation and can be used to determine the data that have more possibilities of resulting in the DUT connection being reset or the system crashing. In addition, we analyze the results of the first round of the test to focus on the special status of the DUT connection, such as connection reset or timeout, and then the test data can be effectively mutated by using feedback strategies for further testing. ICPFuzzer includes two major phases, as shown in Fig. 1. The details of each phase are introduced in the following.

Algorithm 1: Fuzzing with ICPFuzzer

```
   Input: PCAP, I
1  while not end of PCAP do
2      Filtering(PCAP, I)
3  LSTM_Predicting_And_Separating_Block()
4  while the DUT is not crashed do
5      while not end of the output of
       LSTM_Predicting_And_Separating_Block() do
6          Execution_Testing()
7      Feedback_Strategy()
8      while not end of the output of
       Feedback_Strategy() do
9          Execution_Testing()
10 Generating_Report()
```

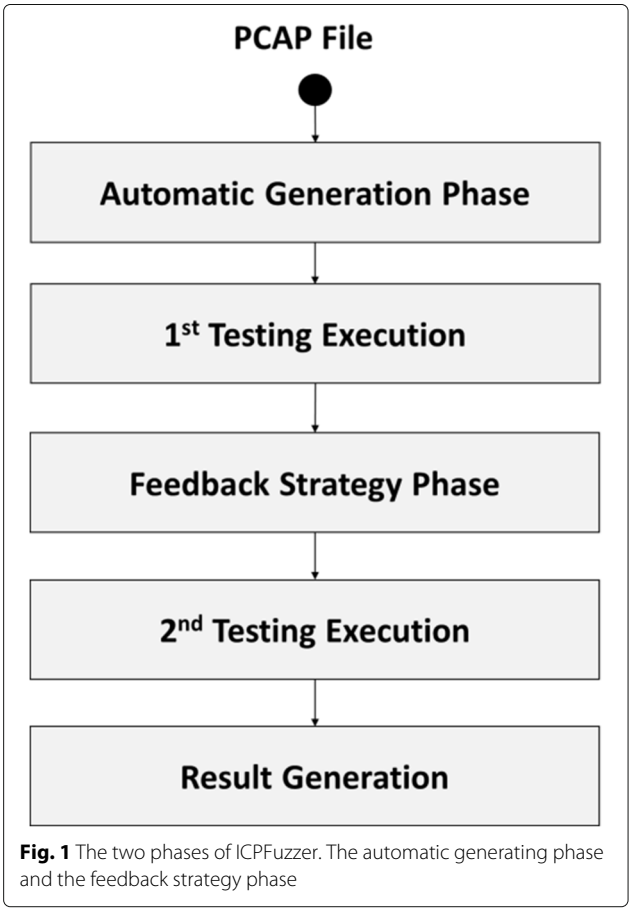

Fig. 1 The two phases of ICPFuzzer. The automatic generating phase and the feedback strategy phase

Automatic generating phase

The aim of the automatic generating phase is to generate test data based on machine learning and define the format of the data to mutate it.

Content filtering

As the beginning of the whole testing process, we first filter the contents of the protocol. The operation process is shown in Fig. 2. Initially, ICPFuzzer receives a PCAP file and the setting information of the DUT, including the IP and port. Then, we convert the PCAP file to a JSON file by using tshark and extract the contents of the application layer with which it communicates based on the information of the DUT. Like we select the contents by using the destination IP and port. If the value of the destination information is same with we expect, than we select the application payload as the contents. For example, the payload's destination IP and port are same with the DUT's. We also choose the data depending on the character of the DUT, such as the server or the client. According to the role of ICPFuzzer (in Modbus/TCP, ICPFuzzer is the client), we add " ** " to the beginning of each data point, which is used as a separation point between data points,

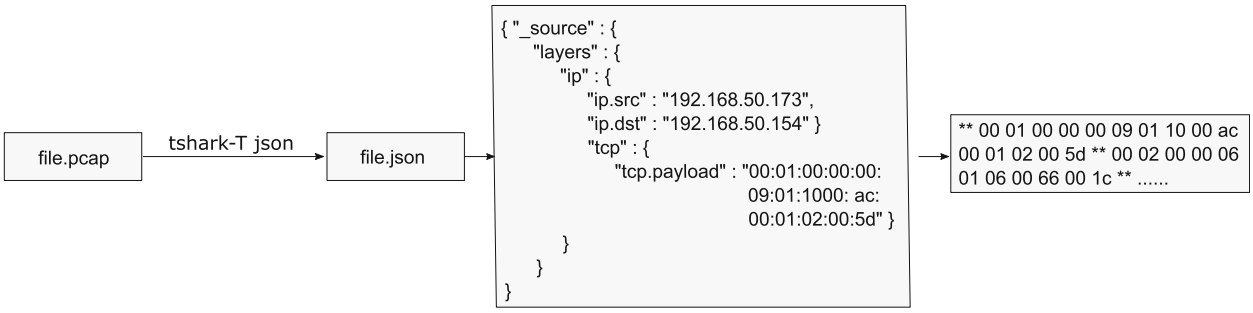

Fig. 2 The filtering process. Conversion of the PCAP file into a JSON file, and then extraction of the information of tcp.payload according to the role of the system

and then connect all the data points in series. In addition, we replace " : " with a space.

To create more suitable test data with content similar to the actual communication behavior, we use LSTM to learn the data and create more data similar to the original communication behavior of the protocol. Before this step, the data must be transformed into data that are suited for the LSTM model. The process is shown in Fig. 3. The data, except for the separation points, are converted from HEX to DEC, similar to the conversion from 0xac to 172. The separation points help the LSTM module become aware of breakpoints in messages. After completing these data, they are sent to the LSTM module.

Fuzzing data generation

To solve the data quantity problem and create high-quality test data, we choose machine learning to learn the communication content between the server and client and simulate more details to create the basic data of the test data. Among the many machine learning methods, the feature of LSTM (Hochreiter and Schmidhuber 1997) is that it can remember the semantics of the sentences and that it is able to produce continuous data for strings, which can help us generate test data that are similar to the content of the packet data.

LSTM is an advanced version of the RNN (Lipton et al. 2015), which can solve the problem of memory loss. In the standard RNN, the core has only one layer containing tanh. Although it is widely used to process sequential data, it is easy for gradient disappearance to occur (Bengio et al. 1994), and LSTM adds a sigmoid to each layer for control to improve the RNN problem. The sigmoid unit outputs a value between 0 and 1, describing how much data can be passed: 0 means that data are not allowed to pass, while 1 means that data are allowed to pass. LSTM mainly uses the cell state and three-layer structure to control which feature needs to be remembered, including the forget gate, the input gate and the output gate. The forget gate is the first layer of LSTM, and the content contains a sigmoid. This layer can determine what proportion of the new and old information needs to be forgotten and should not enter into the next layer. The second layer is the input gate, which determines how much information from the first layer will be recorded and updated in it. This layer contains a sigmoid and tanh. The output gate contains a sigmoid and tanh as well. It decides what information needs to be output for prediction. The cell is used to store the calculated value of each gate and can affect the next cell state.

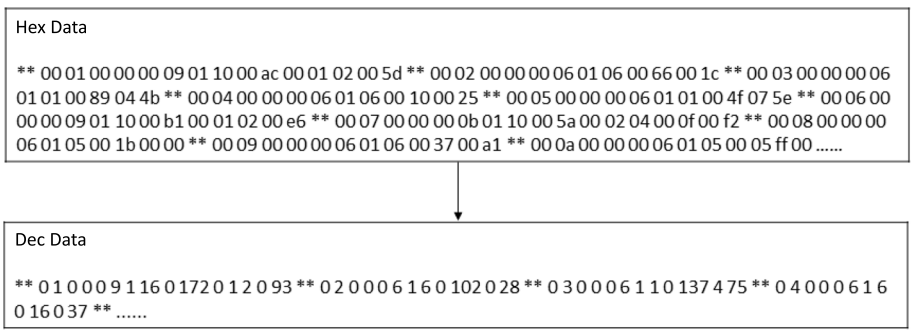

Fig. 3 The conversion process. Conversion of data from HEX to DEC for LSTM learning

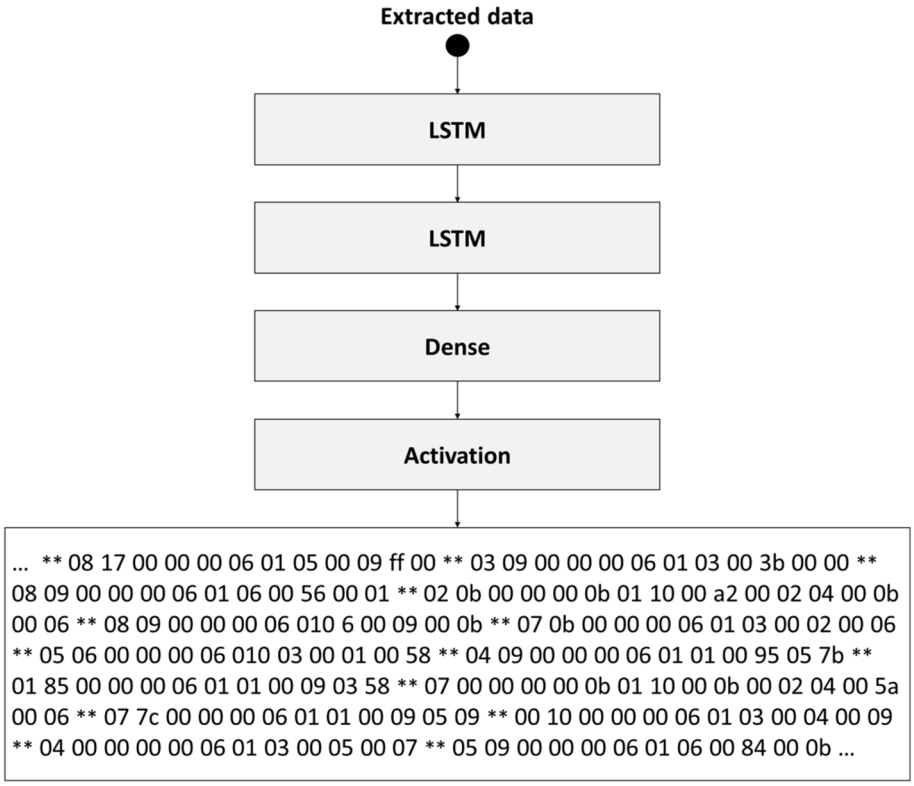

Fig. 4 The Learning process about layers. There are two LSTM layers we use to learn and predict the data

As shown in Fig. 4, we use two LSTM layers to learn and predict the data. Our activation function is softmax, and we use RMSprop as the optimizer. At least 2000 payloads which start with ** are sent into LSTM. After inputting the data from the content filter into LSTM, many ** symbols are added at the beginning of the predicted data points first, and then the subsequent data points are predicted in order on the basis of **. The predicted data are then converted to HEX as the final output. The operation process is shown in Fig. 5, and the predicted result is shown in Fig. 6.

Machine learning is a good way to simulate data and has the possibility to mutate the data. This method can make data not only abnormal relative to the DUT but also reduce the amount of useless data. To increase the test intensity and obtain better test data, ICPFuzzer mutates the data predicted by the LSTM module. However, before the data are mutated, it is necessary to distinguish the blocks of the data that need to be mutated. For unknown protocols, it is difficult to appropriately distinguish the blocks. Therefore, we use the Needleman and Wunsch (1970) algorithm to solve the problem of classifying the blocks. The aim of this algorithm is to align protein or nucleotide sequences. We exploit this algorithm to separate the predicted data into blocks and then calculate the difference rate of each block. If the difference rate is lower, it means that the fault tolerance of the device for that block may be lower. Therefore, the probability of choosing this block for mutation is increased. Because some values may be fixed, for example, the Unit ID in Modbus/TCP, we add a threshold. This threshold reduces the chance of mutation of these blocks to avoid generating too much invalid test data.

The method of counting the block difference rate is shown in Eq. 2, where D is the dataset of the block difference rate. P is the number of different values in a column, as shown in Eq. 1, where q is the total number of blocks and M is the total amount of data. The aim is to calculate the proportion of the difference value in each block. Equation 3 shows the resulting block difference rates. According to Fig. 7, with the previous five sets of values and a column with a block index of 1 as an example, there are four different values in the first column. Thus, p_1 is 4, M is 5, and the block difference rate is 0.8.

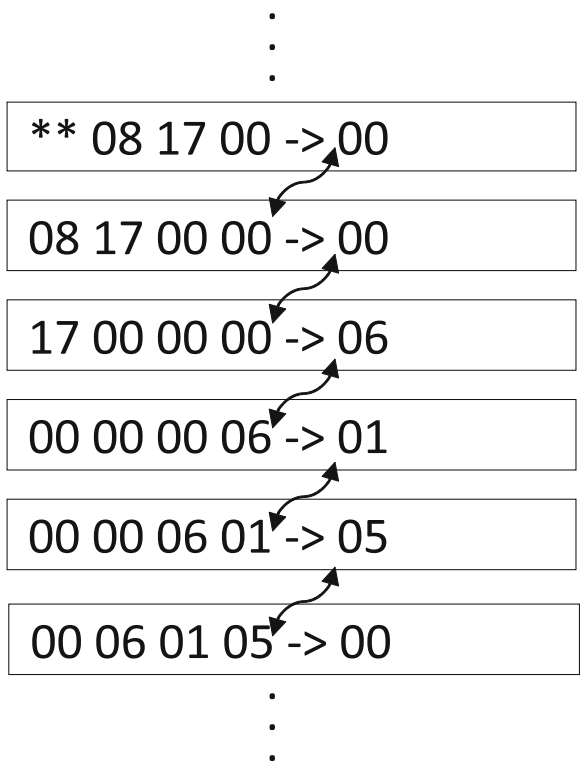

Fig. 5 The prediction process. Prediction of the next value by LSTM based on the previous content

$$P = \{p_1, p_2, \ldots, p_q\} \tag{1}$$

$$F_D = \frac{P}{M} \tag{2}$$

$$D_{result} = \left\{\frac{p_1}{M}, \frac{p_2}{M}, \ldots, \frac{p_q}{M}\right\} \tag{3}$$

If the total difference value is $4+n_1$ and the total number of columns is $5 + m$, then the similarity of the block is $\frac{4+n_1}{5+m}$, as shown in Fig. 8. n_1 is an integer that represents the amount of the rest of the different types of data in the first column. m is an integer that represents the amount of test data except for the first five data points.

After calculating the block difference rate, it can be seen that block index 2, block index 4, and block index 6 are significantly lower. The reason is that for Modbus/TCP, block index 2 is the protocol number, which is basically fixed to two 0x00, and block index 4 is the Unit ID of the DUT. If it changes, the DUT has a great chance of not accepting it and not responding. Block index 6 is part of the contents of the data. The reason that most of this block is 0x00 is because the value at this position in the original data is almost 0x00. After the calculation of the block difference rate, a block will be selected according to the block difference rate to combine the random mutation methods into a single mutation type. ICPFuzzer has seven mutation methods, including bit flips, byte exchange, byte copies, and byte removal (Jääskelä 2016). This combination will be brought into the next test and is related to the feedback strategy phase. Taking Fig. 9 as an example, the block index of these data is 3, and the mutation method is an increment. This combination becomes one of the mutation types. Through our system, if we analyze different types of protocols, ICPFuzzer will generate different numbers of mutation types, as different protocols have different total numbers of blocks.

Feedback strategy phase

The feedback strategy phase aims to create more efficient data depending on the connection status from the DUT and to help users find more suspicious data. We gather the result of the first testing round and define different weight value of the connection result. The more serious connection result, such as connection timeout, which has bigger weight value. Compared with the first round of mutations type, the probability of occurrence is the same. The probability of the second round of mutations will be given different weights based on the test results of the first round, so that the certain mutations will occur more often.

The method is to arrange the weight values of the mutation types and then use the binary search method (Wikipedia contributors 2020b) to select the mutation type that can cause a certain situation by the weighted probability. Since it is a probabilistic choice, other mutation types also have the opportunity to be selected. The feedback strategy phase can not only increase the number of triggers of those special statuses of the DUT but

…** 08 17 00 00 00 06 01 05 00 09 ff 00 ** 03 09 00 00 00 06 01 03 00 3b 00 00 **
08 09 00 00 00 06 01 06 00 56 00 01 ** 02 0b 00 00 00 0b 01 10 00 a2 00 02 04 00 0b
00 06 ** 08 09 00 00 00 06 01 06 00 09 00 0b ** 07 0b 00 00 00 06 01 03 00 02 00
06 ** 05 06 00 00 00 06 01 03 00 01 00 58 ** 04 09 00 00 00 06 01 01 00 95 05 7b **
01 85 00 00 00 06 01 01 00 09 03 58 ** 07 00 00 00 00 0b 01 10 00 0b 00 02 04 00 5a
00 06 ** 07 7c 00 00 00 06 01 01 00 09 05 09 ** 00 10 00 00 00 06 01 03 00 04 00
09 ** 04 00 00 00 00 06 01 03 00 05 00 07 ** 05 09 00 00 00 06 01 06 00 84 00 0b ...

Fig. 6 The result after LSTM prediction. The prediction includes "**", and it separates the test data automatically

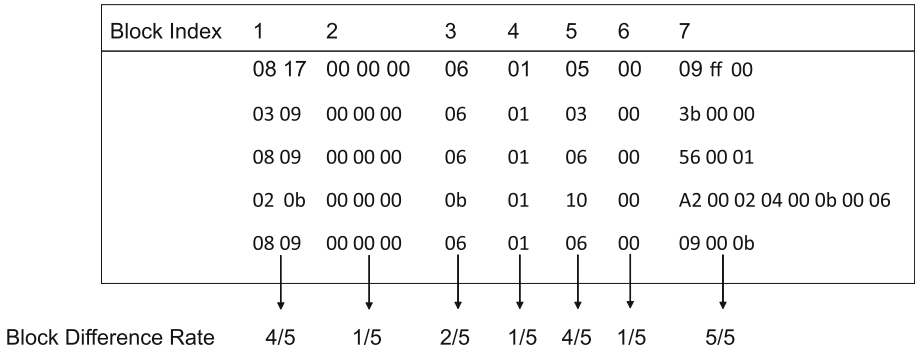

Fig. 7 The example of the block difference rate. After the data are divided, the block difference rate can be calculated from the different types of values in the same column. Each column represents a block. Taking the first column as an example, the block difference rate is $\frac{4}{5}$

also test other mutation types. We record the test results of the test data created by these seven mutation methods and blocks after the first round of testing. We use Eq. 4 to determine the adjusted weight of each mutation type; T is the number of times that the connection status occurs, S is the weight of each connection status, K is the dataset of the new weight of the mutation type, and h is the total number of occurrences of the connection status. Then, we use Eq. 5 to count the possibilities after adjusting the weight value, where l is the total number of mutation types and W is the dataset of the new possibilities.

$$\alpha_K = \sum_{i=1}^{h} \left(S_i * T_{S_i}\right) \tag{4}$$

$$F_W = \frac{\alpha_K}{\sum_{i=1}^{l} \alpha_i} \tag{5}$$

Algorithm 2 shows how we analyze the results of the previous round and generate test data for the next test round to further test some specific statuses of the DUT. Take Table 2 as an example. Before adjusting the weights, all mutation types have the same probability to be used. The total number of mutation types in the first round is 49, including seven mutation methods corresponding to seven blocks. We first take the results of 5 mutation types for explanation purposes. In the first round of mutations, the probability of occurrence of various mutation types is $\frac{1}{49}$, which means that the probability of all mutation types is equal at first.

After the first round of testing, taking mutation type 1 as an example, it caused a total of three statuses for the DUT: status 1, status 2 and status 3. It was indicated that status 2 was triggered the most. The status description after considering Table 3 shows that the test data created by mutation type 1 can result in a success response, connection timeout and no response relative to the DUT. The connection timeout status is the most serious state; thus, we raise the weight of this status to 7. The no response status is abnormal but not too serious; thus, we raise the weight of this status to 3. The success response weight is 1. Before starting the second round of testing, it is necessary to go through the feedback strategy phase. We add

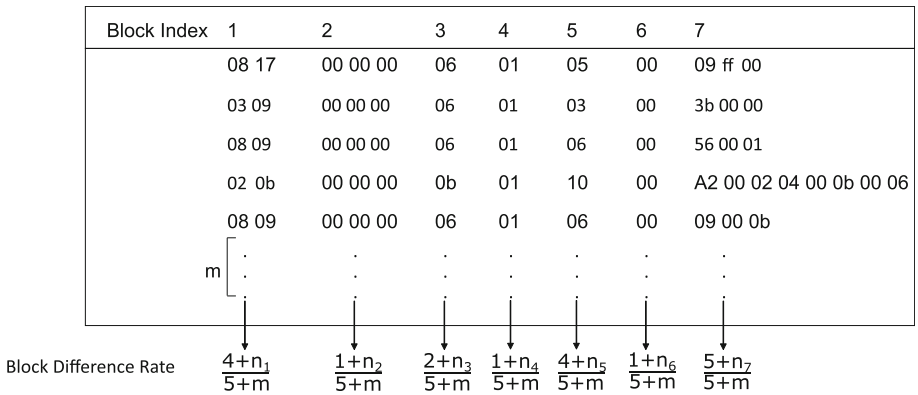

Fig. 8 The example of the block difference rate for total data. If the total amount of test data is 5+m, the block difference rate in the first column will be $\frac{4+n_1}{5+m}$. There are $4 + n_1$ different values in the first column

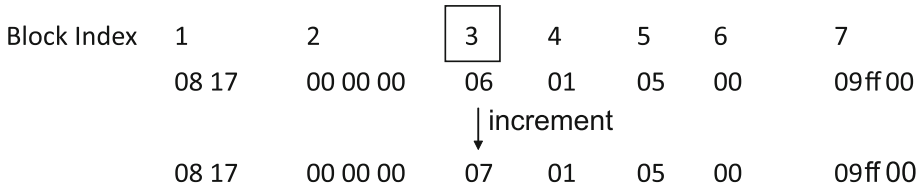

Fig. 9 The combination of the block number an the mutation method. When the block number is 3 and the mutation method is an increment, the value of the third block is chosen to be increased

Algorithm 2: Feedback counting

Input : S, T, R
Ouput: W

Send the test data and aggregate the result R
while *not end of R* **do**
  $temp = 0$
  for $i \leftarrow 1$ *to* h **do**
    $temp = S_i * T_{S_i} + temp$
  $\alpha_K = \alpha + temp$
for $i \leftarrow 1$ *to* K **do**
  $W_i = \frac{\alpha_i}{\alpha_K}$
return W

the total value of the weight according to the status corresponding to mutation type 1. The number of occurrences of status 1 is 13, the number of occurrences of status 2 is 28 and the number of occurrences of status 3 is 1; thus, the total weight of mutation type 1 in the second round is $13 \times 1 + 28 \times 7 + 1 \times 3 = 218$, and so on for the other mutation types. When the total weights of all mutation types have been calculated, the calculation of the occurrence probability of each mutation type is based on all the total weights; then, the weighted occurrence probability of mutation type 1 is $\frac{214}{514+j}$, where j is the total weight of the rest of the mutation types. The same value for mutation types 2 to 5 are $\frac{81}{514+j}$, $\frac{57}{514+j}$, $\frac{143}{514+j}$, and $\frac{15}{514+j}$, respectively.

After a comparison, we find that the probabilities are different after using the mutation strategy. The probability of occurrence of mutation types containing status 2 after adjusting the weight is indeed higher than that of other mutation types.

After mutating the data, ICPFuzzer sends test data to the DUT and monitors the received responses. The order of the test data is tested one by one according to the sequence of data created by mutating. Exceptions will be captured from the DUT, which is under abnormal conditions. There are currently three main connection statuses: success response, abnormal connection and connection refused. The process used by our system to judge the status of a DUT is shown in Fig. 10. When the test is started, the system first connects to the DUT and then sends packets. It also monitors the response status of the DUT by capturing the connection. There are three statuses that are abnormal: connection timeout, connection reset and no response. These three statuses will be taken as the basis for the number of times to restart the connection. If the maximum number of times set by the user is finally reached, the system will restart the connection. If the connection is refused after restarting the connection, ICPFuzzer will

Table 2 The probability of occurrence of five mutation types in the first round and the second round depending on the status of the feedback from the DUT

Mutation type number	Original possibility	First round test result (occurrence)	Adjusted Weight	Possibility after adjusted weight
1	$\frac{1}{49}$	Status 1 : 13 times Status 2 : 28 times Status 3 : 1 time	218	$\frac{218}{514+j}$
2	$\frac{1}{49}$	Status 1 : 66 times Status 3 : 5 times	81	$\frac{81}{514+j}$
3	$\frac{1}{49}$	Status 3 : 19 times	57	$\frac{57}{514+j}$
4	$\frac{1}{49}$	Status 1 : 1 time Status 2 : 16 times Status 3 : 10 times	143	$\frac{143}{514+j}$
5	$\frac{1}{49}$	Status 3 : 5 times	15	$\frac{15}{514+j}$
.....	.....	.....	.....	.....

Table 3 Different weight values defined according to the severity of the connection status relative to the DUT

Status	Description	Weight
1	Success Response	1
2	Connection Timeout	7
3	No Response	3

directly end the connection and terminate the test. If the test is in the first round, the results will be aggregated and analyzed to generate the test data for the second round. If the test is in the second round, the testing report will be generated.

After completing the fuzzing test, ICPFuzzer sets the information of the DUT, including the IP, port, name, and test information such as the file location where the evidence will be saved, the training time and the testing time for each round. The report also includes the outline and details of the test data and results. JSON files and PDF files are generated so that users can refer to them and further test the DUT.

Experiment and evaluation

In this section, we conducted five experiments to verify the effectiveness of ICPFuzzer. The first four experiments were based on Modbus/TCP, while the fifth experiment verified the DLMS/COSEM. In order to collect the data for training, we sent messages that conform to the protocol format but have random values to the tested device, and collect messages that can cause effective responses as the training data. First, we compared the recognition rate of the test data produced by LSTM and that obtained using different tested device samples with different numbers of epochs. The main results can be used to prove that the use of machine learning to create test data can fit in the format of the protocol. Then, we considered the difference between using the feedback strategy phase or not. At the same time, we compared the time taken by different weight random selection algorithms and compared the effects that those algorithms achieve. In the third experiment, we compared the effect of three well-known fuzzers with open-source and commercial versions. In the fourth experiment, we tested the physical equipment with Modbus/TCP in different fields and found the existing vulnerabilities. In the final experiment, we performed learning prediction and fuzzing tests for the DLMS/COSEM in the smart grid field.

ICPFuzzer evaluation

We completed two experiments to prove that LSTM can actually create similar protocol data and that our feedback strategy can trigger more vulnerabilities.

Recognition rate evaluation

Our test environment was set in VMWare which OS was Ubuntu 16.04 and memory was 8096 mb. The tested devices use Modbus/TCP simulation programs, i.e., ModbusPal v1.6 (Darkweb and nnovic 2011) and Modbus_tk v1.0.0 (Luc Jean 2019). These DUTs have better receiving capabilities and will not crash during the test. We sent the data predicted by the LSTM for different numbers of epochs to the DUT and used tcpdump to record the testing process at the same time. Then, we saved the data as a PCAP file and used tshark to convert the data into the JSON format for analysis. If the transmitted message can be recognized as Modbus/TCP data, then the valid data can be included in the recognition counts. The results are as shown in Figs. 11 and 12.

According to the first experiment, we find that LSTM can well simulate the communication format of the protocol. It can lessen the time needed by testers to understand the protocol. The average recognition rate is more than 90%. After testing with two different devices, we find that the change in a number of epochs will not largely affect the recognition rate. Even when mutating the test data, the recognition rates are still greater than 60.

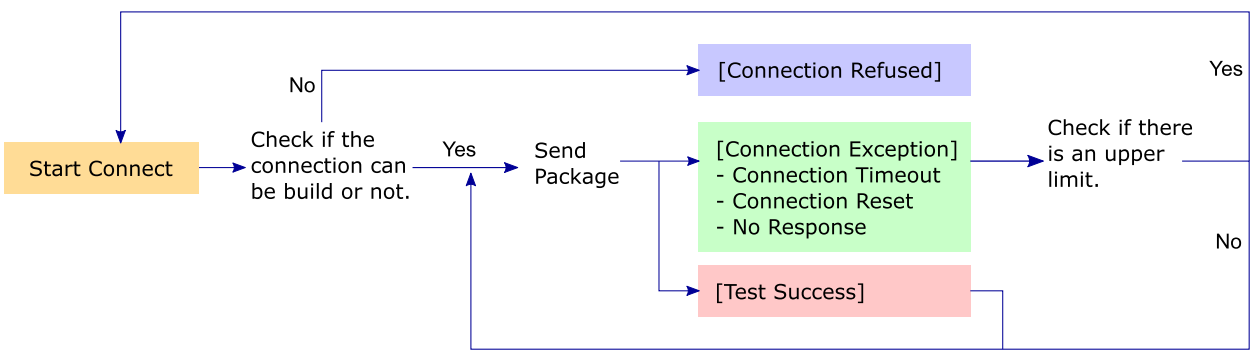

Fig. 10 The connection statuses. After sending the test data, the system will perform different processes according to the different response statuses of the DUT

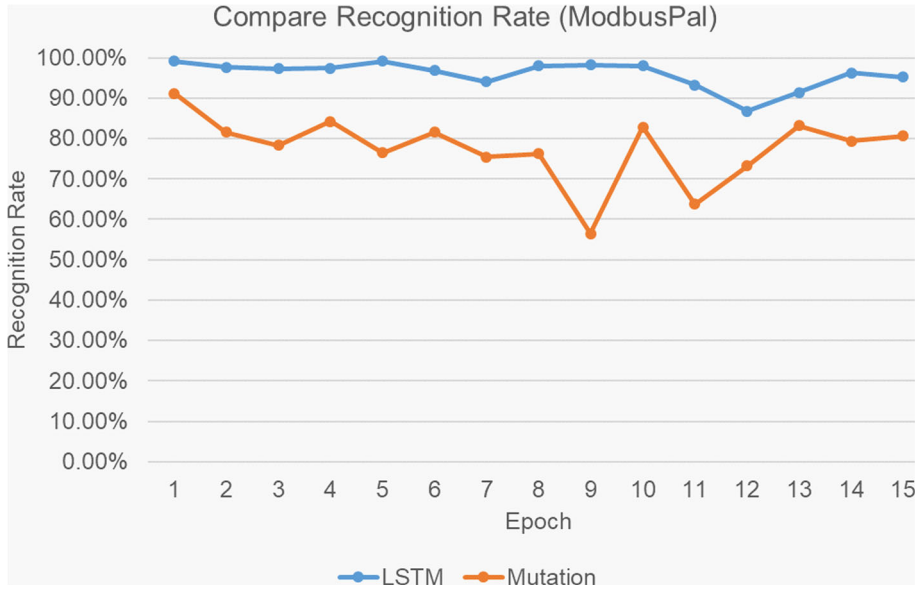

Fig. 11 The recognition rate of ModbusPal. The recognition rate of the data predicted by LSTM and using ModbusPal as the DUT

In addition, we take the predicted data, ModbusPal as the DUT, and epoch 1 to train the data, as shown in Fig. 13. According to the predicted data, one can find that the probability that the predicted data is fixed in the protocol format is high. Taking ** 00 06 00 00 00 06 01 05 00 06 ff 00 as an example, the position of 0x05 is the function code, and the value represents the Modbus/TCP function for writing a single coil. With the function 0x05, the content can be filled only with "0xff 0x00" or "0x00 0x00"; thus, the prediction is correct. In addition, in terms of length, the position of 0x06 is the length of the message, and the length of the subsequent values is exactly 6. Taking ** 00 ec 00 00 00 0b 01 10 00 25 00 02 04 00 08 00 09 as an example, 0x0B is converted to DEX (i.e., 11), which is also representative of the length, and the value of the next 11 values is predicted. Therefore, it is indeed suitable to use LSTM to predict the content of unknown protocols.

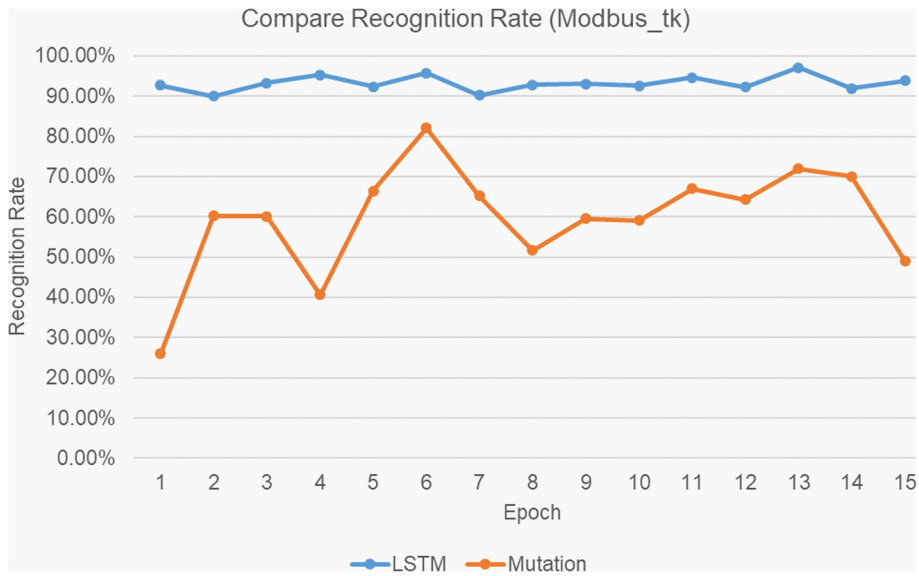

Fig. 12 The recognition rate of Modbus_tk. The recognition rate of the data predicted by LSTM and using Modbus_tk as the DUT

```
.....** 00 6d 00 00 00 06 01 03 00 06 00 05 ** 03 06 00 00 00 06 01 01 00 47 03 09
** 02 40 00 00 00 06 01 03 00 00 00 06 ** 00 06 00 00 00 06 01 05 00 06 ff 00 ** 04
05 00 00 00 06 01 03 00 06 00 08 ** 09 01 00 00 00 06 01 06 00 03 00 01 ** 07 01 00
00 00 06 01 05 00 05 ff 00 ** 08 00 00 00 00 06 01 03 00 09 00 00 ** 09 03 00 00 00
06 01 06 00 00 00 06 **.....** 00 ec 00 00 00 0b 01 10 00 25 00 02 04 00 08 00 09
** 00 1f 00 00 00 06 01 05 00 2f 00 00 ** 04 5f 00 00 00 06 01 03 00 a5 00 9e ......
```

Fig. 13 The predicted results of ModbusPal. The data predicted by LSTM when the epoch is 1. The DUT is ModbusPal

Validating the feedback strategy phase

Our test environment was set in VMWare which OS was Ubuntu 16.04 and memory was 8096 mb, and we used ModbusPal v1.6 as the DUT. There are two outcomes: 1) connection timeout and 2) the situation in which the DUT does not reply after receiving the message. In the second round of testing, three different weight random selection algorithms (Wikipedia contributors 2020c) were used: the random selection method (R) (Müller 2016), Alias method (A) (Walker 1977) and binary search method (B). At present, these three algorithms are often used. The time needed to create the test data was compared separately, as shown in Fig. 14. Clearly, the binary search method requires the shortest amount of time.

After implementing the three different weight random selection algorithms in the system, the differences in the results and calculation times between the first round and the second round were compared. The results are shown in Tables 4 and 5. According to the experiment, we find that after completing the feedback strategy phase, the possibility of triggering the special status is higher, and more different data contents can be found and cause the connection timeout status of the DUT to occur or the connection to be reset. Comparing the three different random weight selection algorithms, it is found that the test data created by these three methods can yield good results. Compared with the time period, the binary search method is the fastest, but in terms of effect, the random selection method is the best and can achieve a higher effect in the second round. Excessive no response statuses will result in connection reset. The reset status occurs because the DUT has closed the connection, and the sending side is still reading and writing. Even if the sending side is well controlled, the connection must be reconnected to continue the communication. In the end, the binary search method is chosen because the three methods have similar effects in terms of results, but the calculation time is almost half that of the others.

Comparison to other fuzzers

In addition to ModbusPal and Modbus_tk, we additionally tested Modbus-slave (Witte Software 2020), pymodbus (RiptideIO 2020), ModbusTool (Graham ross and matt Sargent 2020) and diagslave (ProconX Pty Ltd 2020). The first three simulators were installed on Windows 10, and the last one was installed on Ubuntu 16.08. These environments were all set in VMWare which memory was 8096 mb. To compare the performances of the fuzzers,

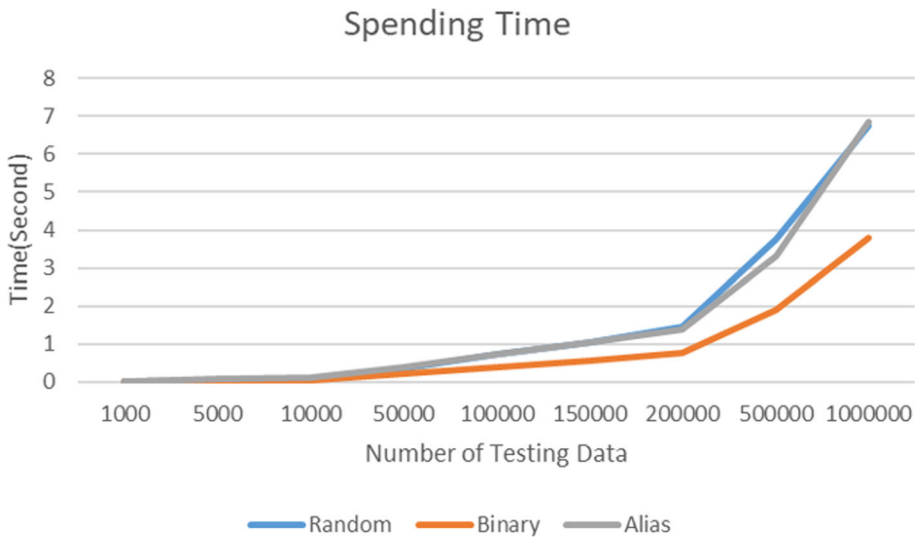

Fig. 14 The comparison of spending times. Comparison of the spending times of the random selection method, Alias method, and binary search method

Table 4 Comparison of the probabilities of causing connection timeout and the execution time of the whole test

Round	1^{st}	2^{nd} (R)	2^{nd} (A)	2^{nd} (B)
Connection timeout	23.1%	46.5%	39.6%	42.8%
Period	–	0.0131 s	0.0192 s	0.0046 s

we compared ICPFuzzer with Peach Tech (2020), booFuzz (Joshua 2020) and beStorm (Beyond security 2020). Peach and booFuzz were installed on Ubuntu 16.08, and beStorm (30-minute trial version) was installed on Windows 10. Our purpose is to compare how many packets the fuzzer needs to send to make the DUT inoperable such that it must be restarted for operation.

In this experiment, ICPFuzzer is able to find the error data and cause the DUT to crash with the least number of packages. If the DUT crashes, it should be restarted, and some other devices may even need the configuration to be reset, as shown in Table 6. According to the results of this experiment, the test cases of pymodbus, ModbusTool and Diagslave have excellent test results. ICPFuzzer can use fewer packets to cause the DUT to crash. Modbus-slave is related to the expiration of the applicable time, resulting in the DUT being automatically disconnected. ModbusPal and Modbus_tk have a better ability to receive data with the incorrect format; thus, all fuzzers cannot crash their system.

Fuzzing performance evaluation

The above experiments all tested the communication of the Modbus/TCP simulator. Moreover, we also tested three different devices: an air monitor that can be set at home and two programmable logic controllers (PLC) that are used in factories, one to control a hydraulic system with a motor (IDEC 2020) and one to control a robot arm (ICPDAS 2020). After testing the air monitor, we found that it could not process the unsupported function code, and sending similar data would cause the air monitor to crash. After testing the second DUT, we found that the motor can be controlled with an additional function code, but this function is not shown on the instruction manual. When there is no water, sending a command to start the motor will cause it to burn, the consequences of which are very serious. The third DUT has a problem with an instruction that can cause the robotic arm to operate excessively and the entire system to stop operation; even if it is restarted after unplugging, it cannot resume operation.

Table 5 Comparison of the probabilities of the DUT responding with "No Response" or "Connection Reset" and the execution time of the whole test

Round	1^{st}	2^{nd} (R)	2^{nd} (A)	2^{nd} (B)
No Response	38%	44.3%	40.4%	43.5%
Connection Reset	13.7%	17.2%	18%	16.2%
Period	–	0.0117 s	0.0222 s	0.0031 s

In addition, we used ICPFuzzer to test a device that communicates with an unencrypted DLMS/COSEM. The test environment was set in VMWare which OS was Ubuntu 16.04 and memory was 8096 mb, and the test simulator was openMUC (2020). We ran at least 10,000 test cases and did not test the HLS phase or the authorization phase. We simply tested the communication phase after authorization. Therefore, we sent the HDLC layer data first, and then the authentication data of the DLMS/COSEM layer were transmitted. The contents of the data were not transmitted until the authentication was confirmed to be successful. ICPFuzzer could send data in a format similar to that of the DLMS/COSEM. Even though this proprietary protocol is complicated, the system was able to complete the test. We also found that the feedback strategy phase worked with the protocol. For the no response status, the number of data points in the second round was significantly greater than that in the first round, as shown in Table 7.

Discussion

Although ICPFuzzer has the ability to detect messages that can cause the DUT to crash or the connection to be reset, compared with other fuzzers, it has better capabilities; however, there are still many functions that need to be improved. This section discusses the capabilities and weaknesses of our method observed through the above experiments and how we can improve in the future. According to the results of the first experiment, although the data predicted by LSTM are highly recognizable and can indeed help testers save time in understanding the protocol, it is shown that the epoch has no major influence on the result. It is found that the sample data generated by different devices yield different training results. In addition, the recognition rate after a mutation still needs to be improved. A classifier will be added later to automatically select the mutation method. This is one of the future research topics. From the second experiment, we know that the feedback strategy phase can effectively increase the amount of data needed to find error messages. However, although more rounds of testing may indeed yield more messages, they may also cause the test time to be too long. In addition, the fastest and most effective weight random selection algorithm was also found. The fuzzing test cannot be used to test the crashed status of a DUT that cannot be started by using the command line, as this is something that the feedback strategy phase cannot do.

The third experiment clearly shows that the test data created by ICPFuzzer are more likely to cause device crashes than the test data created by the rules set by

Table 6 Comparison of the number of frames of different fuzzers before the DUT crashed

	ICPFuzzer	Peach	booFuzz	beStorm
Pymodbus	<50 frames	>100 frames	>50 frames	>500 frames
Modbus-slave	DUT timeout	DUT timeout	DUT timeout	DUT timeout
ModbusTool	<30 frames	50∼100 frames	>100 frames	>5000 frames
ModbusPal	Not crash	Not crash	Not crash	Not crash
Diagslave	>800 frames	–	–	>800 frames

the users. The limitation is that we need to learn from the existing communication data. How to obtain more effective data is a limitation of our system. Taking Modbus/TCP as an example, the function code may include more data or the test data cannot be set for different devices; for example, different devices have different Unit IDs.

In addition to using the simulator as the DUT, we also tested a real ICP device. After the fourth experiment, it could be proved that in addition to the good detection of the Modbus/TCP simulator by ICPFuzzer, the test results of the physical devices can also yield good outcomes. However, compared to the fast response speed of the simulator, the response speed of the physical device is very slow, and the test time must be considered. In addition, if the device crashes, it is more difficult to directly restart a physical device using the command line, and to get the device back on-line, it may even be necessary to restart the power source. The fifth experiment shows that ICPFuzzer can test the DLMS/COSEM, which is a complex protocol and can be used in a smart grid. Although it can simulate intermediate communication, the previous handshake phase needs to be achieved by the playback function. Therefore, how to use machine learning to predict the handshake phase is still a problem. At present, there is a fuzzer use state machine that can record the phase of a protocol to complete the fuzzing test for the stateful protocol (Ma et al. 2016). The handshake phase and communication phase can also be tested separately. Therefore, how to complete the fuzzing test for the stateful protocol will also be the focus of one of our future studies.

Related work

At present, many fuzzing methods for industrial proprietary communication protocols have been proposed. How to mutate data and maintain the high performance of testing is the main research direction.

Table 7 The result of testing the DLMS/COSEM by using ICPFuzzer

Round	No COSEM response	Correct
1^{st}	131	241
2^{nd}	249	135

Grammar-based fuzzer

Modbus/TCP is a very widely used protocol in industry. It can be used in air monitors for homes or robotic arms for communication in many factories. To test Modbus/TCP more effectively, many studies have researched and developed many good methods. To pursue efficient results, most mutation methods use grammar-based mutation to change the data content. MTF (Voyiatzis et al. 2015) is a fuzzing system that can test eight implementations of Modbus/TCP. To avoid too many invalid test data points, MTF first investigates what function codes the DUT has and then tests for specific types of mutation. There is another Modbus/TCP fuzzing system (Xiong et al. 2015), which is a black-box testing technology. To reduce the number of tests, it first confirms which function codes the DUT has and then does not send the related test data. However, the disadvantage of both systems is that the users need to traverse all function codes before they can start performing mutation and testing. MTF-Storm (Katsigiannis and Serpanos 2018) was based on MTF. Compared to randomly generating data, MTF-Storm uses some special values to trigger the vulnerabilities. Another white-box fuzzing system (Yoo and Shon 2016) uses dynamic information extracted from program execution and finds the input field to generate the test data. Additionally, there are famous commercial fuzzing systems for testing communication protocols, i.e., beStorm and Defensics (Synopsys 2020). The disadvantage of these fuzzing systems is that a protocol testing module needs to be specified, the test data are fixed, and the vendor needs to be updated first. Moreover, Peach and boofuzz are useful open-source fuzzing systems that users can utilize to define the rules of the testing protocol and monitor the DUT.

Computing-based fuzzer

ProFuzz (Niedermaier et al. 2017) uses the Ratcliff/Obershelp pattern recognition algorithm to analyze the structure of proprietary industrial protocols and create the package objects. It can determine the proprietary handshake between devices, including the TCP handshake.

Mutation-based fuzzer

Some fuzzing systems use existing information to perform mutations, such as the communication contents of the DUT or the test data of other fuzzing systems. This method is called mutation-based fuzzing. LZFUZZ (Shapiro et al. 2011) creates a token using the Lempel-Ziv compression algorithm and creates the test data based on the packet contents to test the unknown protocol. T-Fuzz (Peng et al. 2018) uses the test data generated by a known coverage-guided fuzzing system such as AFL as its basis and analyzes the state of the protocol. Vuzzer (Rawat et al. 2017) monitors the data flow features of the SUT to generate the input and uses the feedback loop to create a new input. Another fuzzing system (Han et al. 2012) proposes an approach to fuzzing the protocol in layer 2 by using the PRDF module to describe the communication structure and using the RATM module to analyze the relationship between the fields.

Learning-based fuzzer

Machine learning is helpful for finding vulnerabilities. Because the effectiveness of a mutation can be a problem for fuzzing tests, most learning-based fuzzers are used to generate test data or define the block of data to mutate. An increasing number of studies have found significant effects in triggering a bug by using machine learning to generate test data. SeqFuzzer (Zhao et al. 2019) uses LSTM to learn the communication contents of the DUT and creates the test data. Some fuzzing systems have also been developed in other fields. One method (Rajpal et al. 2017) first runs a general fuzzer and then uses LSTM to learn the packet contents to achieve the purpose of creating test data. Learn&Fuzz (Godefroid et al. 2017) uses the seq2seq algorithm to learn the format of the PDF and generates test data to test the system that displays the PDF. In addition, some fuzzing methods test a system by analyzing the response from the DUT and generate the new test data depending on the feedback. A method that tunes the attributes of machine learning based on the feedback generated by different inputs has also been developed (Böttinger et al. 2018).

Conclusions

The advent of Industry 4.0 has caused more devices to be connected externally for more effective control, but this increase may also usher in more attacks. Since many different devices have the same connection protocol for communication, attacking the communication content has become a threat that needs to be prevented. The fuzzing test can be used to test whether a communication system is defective or a device is easy to crash, and it is considered by many standards. Moreover, there are many fuzzers that need the tester to have knowledge of the protocol, but many devices used in factories may use many different protocols at one time. It takes much time to fully understand their architectures. How to effectively mutate the data becomes the most important consideration. With the popularity of machine learning, an increasing number of researchers have begun to study what role machine learning can play in fuzzing tests. Research also shows that machine learning can indeed help a fuzzing system establish test data and obtain good results.

In this paper, we discussed the importance of the communication protocols that need to be tested and proposed an automated fuzzing system for industrial proprietary communication protocols. We used LSTM to learn the communication content of a device. This approach solves the problem of testers needing to spend time to learn about a protocol. We also used the Needleman algorithm to determine the block and then mutated it to create the test data. After testing a round, we used the results to adjust the weights and used the binary search algorithm to select the mutation type for some specific connection statuses. It is found that more test data can cause the device to crash or the connection to be reset. The most important thing is that all processes are operated automatically. Users need to input only the corresponding PCAP file and the setting information of the DUT, which can reduce the problem of users having to define the rules of the protocol. Our evaluation data can be downloaded at the cloud (Institute for Information Industry 2020).

Acknowledgements

Not applicable.

Authors' contributions

P.Y.L conceived of the presented idea, developed the theory, performed the computations and verified the analytical methods. C.W.T (Chia Wei Tien) conceived of the presented idea, verified the analytical methods. T.C.H developed the theory and performed the computations. C.W.T (Chin Wei Tien) conceived of the presented idea, investigate and supervised the findings of this work. The author(s) read and approved the final manuscript.

Funding

Not applicable.

Availability of data and materials

Institute for Information ndustry (2020) ICPFuzzer evaluation data. https://reurl.cc/VXedlR. Accessed 13 Dec 2020.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

To the best of our knowledge, the named authors have no conflict of interest, financial or otherwise.

Received: 12 January 2021 Accepted: 7 April 2021
Published online: 03 August 2021

References

Bengio Y, Simard P, Frasconi P (1994) Learning long-term dependencies with gradient descent is difficult. IEEE Trans Neural Netw 5(2):157–166

Beyond security (2020) BeSTORM: Black box testing. https://beyondsecurity.com/solutions/bestorm.html?cn-reloaded=1. Accessed 13 Dec 2020

Böttinger K, Godefroid P, Singh R (2018) Deep reinforcement fuzzing. In: 2018 IEEE Security and privacy workshops (SPW). pp 116–122. https://doi.org/10.1109/SPW.2018.00026

Darkweb and nnovic (2011) ModbusPal - Java MODBUS simulator. http://modbuspal.sourceforge.net/. Accessed 13 Dec 2020

GISA Security Compliance Institute (2020) IEC-62443-CSA-Certification. https://www.isasecure.org/en-US/Certification/IEC-62443-CSA-Certification#tab2. Accessed 13 Dec 2020

GMicrosoft (2020) Security engineering. https://www.microsoft.com/en-us/securityengineering/sdl/. Accessed 13 Dec 2020

Godefroid P, Peleg H, Singh R (2017) Learn fuzz: Machine learning for input fuzzing. In: 2017 32nd IEEE/ACM International conference on automated software engineering (ASE). pp 50–59. https://doi.org/10.1109/ASE.2017.8115618

Graham ross and matt Sargent (2020) ModbusTool - A modbus TCP and RTU master and slave tool with import and export functionality. https://github.com/graham22/ModbusTool. Accessed 13 Dec 2020

Grubbs HL (2018) Field programmable gate array high capacity technology for radar and counter-radar drfm signal processing. Calhoun. https://calhoun.nps.edu/handle/10945/59670. Accessed 13 Dec 2020

Han X, Wen Q, Zhang Z (2012) A mutation-based fuzz testing approach for network protocol vulnerability detection. In: Proceedings of 2012 2nd International conference on computer science and network technology. pp 1018–1022. https://doi.org/10.1109/ICCSNT.2012.6526099

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780

ICPDAS (2020) Programmable Automation Controller (iP-8441-MTCP). https://www.icpdas.com/root/product/solutions/pac/ipac/ip-8x41-mtcp.html. Accessed 13 Dec 2020

IDEC (2020) IDEC programmable logic controller. http://tw.idec.com/zht/p/c60/. Accessed 13 Dec 2020

Institute for Information Industry (2020) ICPFuzzer evaluation data. https://reurl.cc/VXedlR. Accessed 13 Dec 2020

Jääskelä E (2016) Genetic algorithm in code coverage guided fuzz testing

Joshua P (2020) Boofuzz - network protocol fuzzing for humans. https://github.com/jtpereyda/boofuzz. Accessed 13 Dec 2020

Katsigiannis K, Serpanos D (2018) Mtf -storm: a high performance fuzzer for modbus/tcp. In: 2018 IEEE 23rd International conference on emerging technologies and factory automation (ETFA) Vol. 1. pp 926–931. https://doi.org/10.1109/ETFA.2018.8502600

Liang H, Pei X, Jia X, Shen W, Zhang J (2018) Fuzzing: State of the art. IEEE Trans Reliab 67(3):1199–1218

Lin J, Liu L (2019) Research on security detection and data analysis for industrial internet. In: 2019 IEEE 19th international conference on software quality, reliability and security companion (QRS-C). pp 466–470. https://doi.org/10.1109/QRS-C.2019.00089

Lipton ZC, Berkowitz J, Elkan C (2015) A critical review of recurrent neural networks for sequence learning. 1506.00019

Luc Jean (2019) Modbus-tk: Create Modbus app easily with Python. https://github.com/ljean/modbus-tk. Accessed 13 Dec 2020

Ma R, Wang D, Hu C, Ji W, Xue J (2016) Test data generation for stateful network protocol fuzzing using a rule-based state machine. Tsinghua Sci Technol 21(3):352–360

McLaughlin S, Konstantinou C, Wang X, Davi L, Sadeghi A, Maniatakos M, Karri R (2016) The cybersecurity landscape in industrial control systems. Proc IEEE 104(5):1039–1057

Müller K (2016) Accelerating weighted random sampling without replacement

Nan C, Eusgeld I, Kröger W (2013) Hidden vulnerabilities due to interdependencies between two systems. In: Hämmerli BM, Kalstad Svendsen N, Lopez J (eds). Critical information infrastructures security. Springer Berlin Heidelberg, Berlin Vol. 7722. pp 252–263

Needleman SB, Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 48(3):443–453

Niedermaier M, Fischer F, von Bodisco A (2017) Propfuzz – an it-security fuzzing framework for proprietary ics protocols. In: 2017 International conference on applied electronics (AE). pp 1–4. https://doi.org/10.23919/AE.2017.8053600

openMUC (2020) JRxTx - Java serial communication library. https://github.com/openmuc/jrxtx. Accessed 13 Dec 2020

Peach Tech (2020) Peach fuzzer. https://www.peach.tech/. Accessed 13 Dec 2020

Peng H, Shoshitaishvili Y, Payer M (2018) T-fuzz: Fuzzing by program transformation. In: 2018 IEEE Symposium on security and privacy (SP). pp 697–710. https://doi.org/10.1109/SP.2018.00056

Poletykin A (2018) Cyber security risk assessment method for scada of industrial control systems. In: 2018 International russian automation conference (RusAutoCon). pp 1–5. https://doi.org/10.1109/RUSAUTOCON.2018.8501811

ProconX Pty Ltd (2020) Diagslave modbus slave simulator. https://www.modbusdriver.com/diagslave.html. Accessed 13 Dec 2020

Rajpal M, Blum W, Singh R (2017) Not all bytes are equal: Neural byte sieve for fuzzing

Rawat S, Jain V, Kumar A, Cojocar L, Giuffrida C, Bos H (2017) Vuzzer: Application-aware evolutionary fuzzing. In: Network and Distributed System Security Symposium. Sourced from Microsoft Academic - https://academic.microsoft.com/paper/2613534458

RiptideIO (2020) PyModbus - A python modbus stack. https://github.com/riptideio/pymodbus. Accessed 13 Dec 2020

Shapiro R, Bratus S, Rogers E, Smith S (2011) Identifying vulnerabilities in scada systems via fuzz-testing. In: Butts J, Shenoi S (eds). Critical Infrastructure Protection V. Springer Berlin Heidelberg, Berlin. pp 57–72

Su W, Antoniou A, Eagle C (2017) Cyber security of industrial communication protocols. In: 2017 22nd IEEE International conference on emerging technologies and factory automation (ETFA). pp 1–4. https://doi.org/10.1109/ETFA.2017.8247769

Synopsys (2020) Defensics fuzz testing. https://www.synopsys.com/software-integrity/security-testing/fuzz-testing.html. Accessed 13 Dec 2020

Voyiatzis AG, Katsigiannis K, Koubias S (2015) A modbus/tcp fuzzer for testing internetworked industrial systems. In: 2015 IEEE 20th Conference on emerging technologies factory automation (ETFA). pp 1–6. https://doi.org/10.1109/ETFA.2015.7301400

Walker AJ (1977) An efficient method for generating discrete random variables with general distributions. ACM Trans Math Softw 3(3):253–256

Wikipedia contributors (2020) Triton (malware) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Triton_(malware). Accessed 13 Dec 2020

Wikipedia contributors (2020) Binary search algorithm. https://en.wikipedia.org/wiki/Binary_search_algorithm. Accessed 13 Dec 2020

Wikipedia contributors (2020) Pseudo-random number sampling — Wikipedia, The free encyclopedia. https://en.wikipedia.org/wiki/Pseudo-random_number_sampling. Accessed 13 Dec 2020

Witte Software (2020) Modbus slave simulator. https://www.modbustools.com/download.html. Accessed 13 Dec 2020

Xiong Q, Liu H, Xu Y, Rao H, Yi S, Zhang B, Jia W, Deng H (2015) A vulnerability detecting method for modbus-tcp based on smart fuzzing mechanism. In: 2015 IEEE International conference on electro/information technology (EIT). pp 404–409. https://doi.org/10.1109/EIT.2015.7293376

Yoo H, Shon T (2016) Grammar-based adaptive fuzzing: Evaluation on scada modbus protocol. In: 2016 IEEE International conference on smart grid communications (SmartGridComm). pp 557–563. https://doi.org/10.1109/SmartGridComm.2016.7778820

You W, Wang X, Ma S, Huang J, Zhang X, Wang X, Liang B (2019) Profuzzer: On-the-fly input type probing for better zero-day vulnerability discovery. In: 2019 IEEE symposium on security and privacy (SP). pp 769–786. https://doi.org/10.1109/SP.2019.00057

Zhao H, Li Z, Wei H, Shi J, Huang Y (2019) Seqfuzzer: An industrial protocol fuzzing framework from a deep learning perspective. In: 2019 12th IEEE Conference on software testing, validation and verification (ICST). pp 59–67. https://doi.org/10.1109/ICST.2019.00016

Publisher's Note