IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 29, 2019, accepted July 12, 2019, date of publication July 25, 2019, date of current version August 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2931061

Finding Sands in the Eyes: Vulnerabilities Discovery in IoT With EUFuzzer on Human Machine Interface

JIAPING MEN[1], GUANGQUAN XU[2,3], (Member, IEEE), ZHEN HAN[1], ZHONGHAO SUN[4], XIAOJUN ZHOU[5], WENJUAN LIAN[6], AND XIAOCHUN CHENG[7], (Senior Member, IEEE)

[1]Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China
[2]Big Data School, Qingdao Huanghai University, Qingdao, China
[3]Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, Tianjin 300350, China
[4]National Computer Network Emergency Response Technical Team/Coordination Center of China, China
[5]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[6]College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China
[7]Department of Computer Sciences and Faculty of Science and Technology, Middlesex University, London NW4 4BE, U.K.

Corresponding author: Zhonghao Sun (sunzhonghao@cert.org.cn)

This work was supported in part by the Natural Science Foundation of China under Grant U1736114 and Grant U1736115, and in part by the CNCERT/CC under Grant K19GY500020.

ABSTRACT In supervisory control and data acquisition (SCADA) systems or the Internet of Things (IoT), human machine interface (HMI) performs the function of data acquisition and control, providing the operators with a view of the whole plant and access to monitoring and interacting with the system. The compromise of HMI will result in lost of view (LoV), which means the state of the whole system is invisible to operators. The worst case is that adversaries can manipulate control commands through HMI to damage the physical plant. HMI often relies on poorly understood proprietary protocols, which are time-sensitive, and usually keeps a persistent connection for hours even days. All these factors together make the vulnerability mining of HMI a tough job. In this paper, we present EUFuzzer, a novel fuzzing tool to assist testers in HMI vulnerability discovery. EUFuzzer first identifies packet fields of the specific protocol and classifies all fields into four types, then using a relatively high efficiency fuzzing method to test HMI. The experimental results show that EUFuzzer is capable of identifying packet fields and revealing bugs. EUFuzzer also successfully triggers flaws of actual proprietary SCADA protocol implementation on HMI, which the SCADA software vendor has confirmed that four were zero-day vulnerabilities and has taken measures to patch up.

INDEX TERMS Protocol format parsing, vulnerability mining, fuzzing test, HMI security, IoT.

I. INTRODUCTION

In SCADA system, PLCs (Programmable Logic Controllers) are connected to a central control terminal (i.e., the HMI) through which operators can supervise and control the system. It is necessary to maintain the proper functioning of SCADA system, especially the HMI, which is essential to ensure the safe and reliable operation of the critical infrastructure. Experimental results showed that SCADA equipment can be crashed easily even by randomly assembled inputs, not to mention some well-designed data packets. To make things worse, HMI has its own inherent flaws, including design and implementation of vulnerabilities, such as no communication authentication and input verification, which makes HMI vulnerable to attacks. Some possible attack scenarios are: 1)Disclosure of sensitive data; 2)Lost of View; 3)Remote commands execution.

Security vulnerabilities typically arise from incomplete design and/or poor implementation, which is true for the protocols used in SCADA systems. Securing HMI requires testing for such vulnerabilities. Fuzz-testing is a widely-used security assessment method to discover bugs of input validation and the application logic by passing crafted hostile inputs to the communication target. However, using fuzz-testing methodologies in HMI vulnerability discovery is difficult. HMI has the following characteristics:

The associate editor coordinating the review of this manuscript and approving it for publication was Jun Wu.

1) HMI usually supports several industrial protocols, such as Modbus-TCP, IEC61850 and even some proprietary protocols.
2) HMI performs real-time communication, and the session effective time is very short. What's more, HMI usually keeps a persistent TCP/IP connection for hours even days to periodically acquire real-time data of the field equipment, differing from normal communication between client and server.
3) To communicate with HMI continuously, the values of some fields within the packets need to be changed accordingly. For example, the sequence number should be in an ascending order by adding one at a time and the length fields need to be recalculated in every reassembled packet.
4) The packets passing to HMI should pass through simple validation check and reach the internal processing logic before they are rejected.

However, most of the current fuzzing methods work better when detailed protocol specifications are available. When facing proprietary protocols, they cannot even establish connections with the target. What's more, most of these fuzzing tools act as clients only testing servers (i.e., PLCs). Some fuzzing tools work inline between HMI and PLC, performing two-way testing, but experimental results have shown that the efficiency and precision are not satisfactory [1].

In this paper, we have made great improvement to the fuzzing method in vulnerability discovery on HMI. We present EUFuzzer (called Easily-Using-Fuzzer), an appliance designed to effectively perform fuzzing test on HMI. EUFuzzer differs from other fuzzers in the following aspects and can be applied in fuzzing HMI of high efficiency, which is also the innovation and contribution of EUFuzzer.

1) Input. The input of EUFuzzer is not protocol specification, but the original packets. Thus EUFuzzer is available in fuzzing protocols both public and proprietary. EUFuzzer identifies packet fields of a specific protocol and divides these fields into four types: constant fields, session-related fields, length fields and mutable fields.
2) Assembling Packets. EUFuzzer only assigns the value of mutable fields with elaborately constructed data. Thus EUFuzzer has a rapidly assembling process and within the validity time window of the SCADA protocol. For the other three kinds of fields, EUFuzzer fills them accordingly. For instance, the session-related fields will be changed according to different sessions; the length fields will be recalculated; while constant fields remain the same within all packets. As a result, test suite is relatively small and the majority of these crafted packets can pass through the validation check and reach the internal processing logic of HMI.
3) Communication. EUFuzzer communicates with HMI directly rather than inline, and keeps a persistent connection until a crash occurs.

EUFuzzer provides 26 different mutators to perform the mutation according to the data types, whether it is number, string or array. Simultaneously, EUFuzzer closely monitors the state of HMI. Once HMI crashes or something abnormal happens, EUFuzzer will keep a log of the session and throw an exception to the tester. In general, when a fuzzer displays an error that contains a vulnerability, it will declare success. However, for critical infrastructure, the definition of success will be broader: discovering software errors that could cause any disruption or disturbance. We care about all interruptions and disturbances, because any interruption can affect the stability of the whole system [2].

The remainder of this paper is organized as follows. Section II gives an overview of previous work. After presenting the architecture of EUFuzzer in detail in Section III, we conduct experiments to verify the validity of EUFuzzer in Section IV. Concluding remarks follow in Section V.

II. RELATED WORK

Fuzz-testing is a popular and effective choice for vulnerabilities mining. Since Miller [3], [4] firstly introduced the fuzzing technique in 1990, Fuzzing has been part of the overall quality assurance employed by many big companies, such as Adobe [5], Microsoft [6], and Google [7], as well as by security companies and consultants.

Proell [8] discovered that SCADA equipment can be crashed easily even by randomly assembled inputs, not to mention some well-designed data packets. There are some popular and widely used open source fuzzing tools, such as PeachFuzzer [9], Sulley [10], SPIKE [11], ProFuzz [12] etc. These tools have made a great contribution to vulnerability mining, but they all have limitations in the SCADA systems. The main drawback of these tools is that the tester must have some prior knowledge of the protocol under test, which is not always true. For fuzzing non-proprietary SCADA protocols, there also exist some commercial tools, such as Wurldtech Achilles Platform and Codenomicon Defensics, both of which are prestigious. The former is a testing, validation and certification platform for industrial devices, while the latter is a world famous fuzzing tool. These two tools represent the most advanced fuzzing technology in use and are widely used in industrial systems, but they do not have a good support of proprietary protocols in SCADA systems. Ganesh Devarajan has improved the open source Devarajan [13] fuzzing tool and released new fuzzing modules for DNP3, ICCP and Modbus. SecuriTeam extends its commercial suite, beSTORM fuzzer [14], to support DNP3 fuzz testing. Bond et al. developed the ICCP test commercial tool suite called ICCPSic [15]. In addition, Mu Dynamics has developed the Mu Test Suite [16] for fuzzing of IEC61850, Modbus and DNP3, which are standard industrial protocols. In academic circles, many researchers have made great efforts to improve the fuzzing efficiency. Nick Stephens et al. present Driller [17] to augment fuzzing through selective symbolic execution. Haller et al. invent Dowser [18], using static analysis to identify regions of code that are likely to lead to a vulnerability involving a buffer overflow. Maverick Woo et al. find a more efficient fuzzing

scheduling algorithm [19]. Rebert et al. has made a great contribution to optimize the seed selection for fuzzing [20]. However, these tools above are not specially for SCADA testing. Henrique Dantas et al. introduce eFuzz [21], to perform fuzz testing for DLMS/COSEM electricity meters. There also exist work on static or dynamic analysis for the detection of malicious applications [**?**], [22]–[28]. Intrusion detection or anomaly detection of user behaviors, program behaviors or network traffic behaviors in related work [29]–[34] also provide useful inspiration for the design and the development of the experiments in this paper.

Usually, most existing fuzzing tools can only test against the servers, while in the industrial control system, clients such as HMI are in a security blind spot. The reason is that when designing the fuzzing tools, manipulated packets are sent to the server-specific IP address and port. But they cannot send targeted response packets according to the received packets from the client. What's more, the time-sensitive, session-oriented nature of many SCADA equipment requires fuzzer to perform in high-speed within the time limits. As a result, some inline fuzzer have been developed. QueMod [35] transmits random data or makes random mutations, whose efficiency and accuracy is extremely low. Rebecca Shapiro et al. has created a tool—lZFuzz [1], [36], an inline tool using arp spoofing to test the proprietary protocol, which is claimed to be the first inline fuzzer that goes beyond random strings and mutations. However, LZFuzz is simple and error prone.

III. DESIGN OF EUFUZZER

The better a protocol is modeled, the more chances are to reveal vulnerabilities. On the other hand, the complexity of a data model significantly influences the run-time of a fuzzing process. This is the key factor for the success of a good fuzzer. Before the application logic of the target software is reached, the packets sent by the fuzzing tool usually need to be processed multiple times [2]. In order to reach the application processing logic, the input must meet the check condition as much as possible so it can pass through the simple verification in the early stage; in order to trigger the possible error, the input needs to be manipulated to a degree. In order to have a clear fields identification, we take advantage of the algorithms found in the bioinformatics field(See in III-A). And we describe the architecture of EUFuzzer in detail in III-B.

A. THEORETICAL BASIS

Very short or very similar sequences can be aligned by hand easily. However, most packets of a protocol are of great length and highly variable that cannot be aligned solely by human efforts. Therefore we turn to sequence alignment [37] for help. Sequence alignment is aligning two strings from beginning to end. If the two sequences are already considered to be similar, then a global alignment algorithm is used. Because local alignment algorithm inserts too many gaps, the aligned segments will lose their original meaning. So we make some assumptions that the request packets have almost the same structure of a specific protocol, and so do the response packets. We don't handle some very peculiar protocols whose packet structure is dynamically changing. Fortunately, for the sake of convenience and easy of use, the protocols in SCADA systems are typically clear and clean, including proprietary protocols. Marshall et al. use bioinformatics algorithms in Protocol Informatics project [38], which attempts to identify protocol fields in poorly network protocols. And we are inspired by the endeavour of Weidong Cui et al.,who uses bioinformatics algorithms to identify the fields of packets and replay attacks and multi-stage infection process [39]. The application of sequence alignment is detailed in Section III-B.

B. ARCHITECTURE OF EUFUZZER

The basic idea of the EUFuzzer is straightforward: given some samples of an application session, EUFuzzer identifies fields in the ADUs and adjusts the mutable fields in a controlled way before sending the response packets. It plays the role of server in the C/S architecture in SCADA systems, e.g. PLCs, communicating with HMI directly. The work of EUFuzzer can be divided into two stages: identification stage and fuzzing stage. The identification stage is off-line and the fuzzing stage is on-line. The architecture of EUFuzzer is illustrated in Fig.1.

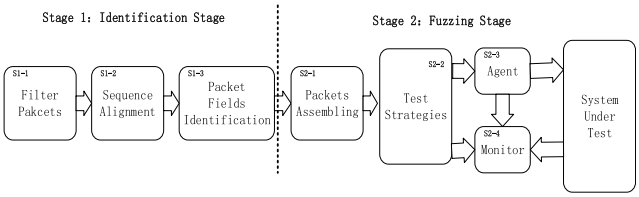


FIGURE 1. The architecture of EUFuzzer.

1) STAGE 1: IDENTIFICATION STAGE

In the identification stage, EUFuzzer needs to parse the network pcap fields, filters the dialogs of the protocol and removes the irrelevant packets. Then EUFuzzer identifies the four types of fields within the packet.

S1-1: Filter Packets.

First, we filter the pcap files from the wireshark or tcpdump, selecting the packets of the protocol that will be handled later. In our experiments, we use regular expression of "tcp.port == PORT && ip.host == HOST", indicating the port and IP address. After that, we will get a clean sample of a particular protocol. Then, we gather the requests and the responses respectively. After that, we get the original packets. We use two sessions as a sample in the following section.

S1-2: Sequence Alignment.

The cornerstone of our processing method is to compare a series of dialogs(i.e. byte streams) to determine the fields. Because the ultimate purpose of EUFuzzer is to identify fields in packets of proprietary protocols, the semantics of the protocol for sequence alignment cannot be used to guide the

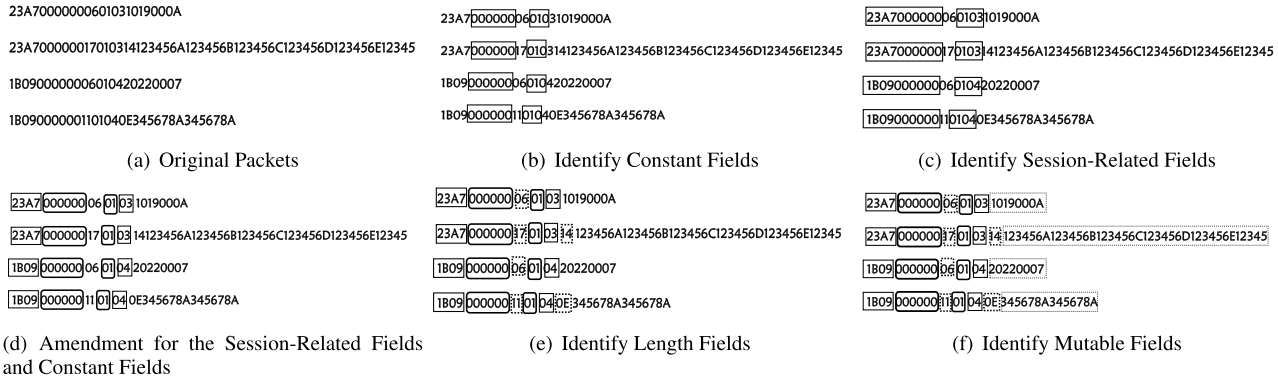


FIGURE 2. The detailed process of fields identification and classification.

matching process. Two different forms of sequence alignment are employed: global alignment and local alignment. Global alignment refers to the overall alignment of two sequences considered as similar, where standard Needleman-Wunsch algorithm is being used. Local alignment means matching two sequences with large differences to find the sequence fragment with the highest similarity. The packets follow the same protocol but may differ greatly in packet length, ADU and so on. Smith-Waterman algorithm is selected to signify the shared information that of the two sequences (byte streams).

These two alignment algorithms set different weights for each paired character, depending on whether the two characters are the same, different, or one of them is a gap. If they are the same, the weight is set to m; if they are different, it is n; if one of them is a gap, it is g. The total score for a possible alignment is the sum of the corresponding weights. Note that there may be multiple alignments for the same score. Details of the algorithms can be found in [38]. In general, Smith-Waterman algorithm sets the score for the same pair to be 2, for the different pair is -1, and for one gap is -2. While for Needleman-Wunsch algorithm, the score for the one gap pair is set to 0 to obtain an accurate result when comparing two sequences that are already considered to be similar. First, the Smith-Waterman algorithm is used to select similar sequences belonging to one protocol from a packet database. Then Needleman-Wunsch algorithm is utilitized for further analysis. To gain the highest accuracy, we set $m = 2, n = 0, g = 0$ for Needleman-Wunsch algorithm and $m = 2, n = -1, g = -2$ for Smith-Waterman algorithm. Then similarity matrices are derived based on observations of evolutionary data on many sequences using Markov chains among other techniques. The most common matrices in bioinformatics is Percent Accepted Mutation (PAM) [37] and BLOcks SUbstitution Matrix (BLOSUM) [40]. In this paper, after a lot of experiments, PAM is selected.

S1-3:Packet Fields Identification.

Fig.2 demonstrates the process of fields identification in detail. We process the requests and the responses respectively, figuring out the constant fields in the first place. By using "tcp.port == PORT", we can get the responses classification. And we can get the constant fields by using string alignment algorithms, as is shown in Fig.2(b). Constant fields remain the same during the test, such as the protocol identifier.

Secondly, we process packets in a dialog, then we can find out the session-related fields(See in Fig.2(c)). Session-related fields are highly correlated with different sessions, such as session identifier and the sequence number of the packets. The session-related fields remain the same within a session, which is the case for session identifier, or in an ascending order by adding one at a time, which is the case for sequence number.

After the first and second steps, we utilize the results accumulated to amend the constant fields and session-related fields. Fig.2(d) depicts the process.

Then, We use packets of the requests to find the length fields, and so do the responses, which is demonstrated in Fig.2(e). Length fields indicate the whole or partial packet, and need to be recalculated in every reassembled packet. There are maybe more than one length fields in one packet and the size of length fields themselves vary from one byte to more bytes.

The left segments are classified as mutable fields. Mutable fields are the key elements in our test, which mean the value of these fields are always manipulated in every reassembled packet, aiming to trigger bugs in the target under test(See in Fig.2(f)).

The instance above is an alignment within two simple sessions(or dialogs). Nonetheless, to better understand the structure of packets, it is of great significance to align them against multiple sequences. A phylogenetic tree is created to guide the multiple sequence alignment [41], which is an evolutionary tree and demonstrates mutations as time goes on [42]. Network packets change the values of different fields, which is analogous to the evolutionary process. Unweighted Pairwise Mean by Arithmetic Average (UPGMA) [43] algorithm is chosen to generate the phylogenetic trees, which is commonly used in this field. The algorithm is defined as following

E.q.1:

$$d_{ij} = \frac{1}{C_i C_j} \sum_{p \in C_i, q \in C_j} D_{pq} \quad (1)$$

where d_{ij} is the distance between clusters C_i and C_j and $D_{pq}$is the Smith-Waterman score.

The tree building process is relatively explicit and can be refered to see more details.

2) STAGE 2: FUZZING STAGE

During the fuzzing stage, EUFuzzer first parses the incoming packet, setting the values of different fields and assembles the packet. Then if sends crafted packets to HMI in a controlled way and monitors any exceptions or abnormal behaviors of HMI. The logic process is illustrated in Fig 3.

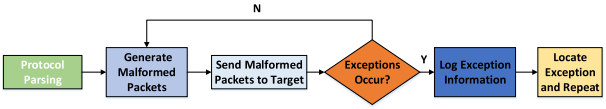


FIGURE 3. The logic Process of EUFuzzer.

S2-1: Packets Assembling.

First, EUFuzzer figures out the value of session-related fields, and copy the value to the same fields of the response packets. Then EUFuzzer assigns the value of mutable fields with elaborately constructed data. The values of constant fields are pre-given. The assembling process of response packets is shown in Fig 4.

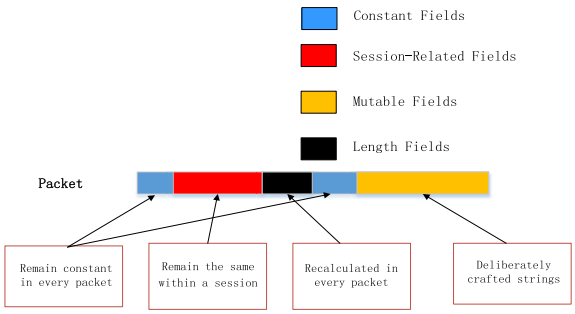


FIGURE 4. Packet assembling process.

S2-2: Test Strategies. In order to carry out the fuzzing process effectively, we have tried some modes and strategies. Mode 1 is that when HMI initializes a request, EUFuzzer responses with one packet, which is the normal communication mode and can run smoothly. Mode 2 is that when HMI initializes a request, EUFuzzer responses with two or more packets. Mode 3 is that when HMI initializes two or more requests, EUFuzzer responses with only one packet. Experiments show that mode 2 and mode 3 do not work well but it can reveal bugs occasionally.

As for the packets sending strategies, 2 different methods are employed. One is random strategy, which means responding packets randomly with the mutable fields crafted. The other is bit sequential strategy, which means responding packets sequentially with the mutable fields crafted one bit at a time. Other options include starting sending packets from a specific test case or only test a subset of crafted packets. All the test process can be repeated easily if using the same seed number again.

S2-3: Agent. Agent guarantees that EUFuzzer plays the role of server and listens to a specific port. It keeps the long connection between EUFuzzer and HMI. When there is a crash, agent is responsible for reconnection and carries on the test continuously.

S2-4: Exception Monitoring and Recording. EUFuzzer takes the Pcap monitor and Ping monitor.Pcap monitor performs network data capture during the fuzzing test iteration, and the captured data is updated when each iteration starts. If an error occurs, the captured packet is logged, otherwise the captured packet is discarded. Pcap files are compatible with Wireshark and tcpdump so that the contents of the packet can be analyzed using wireshark or tcpdump to determine the causes. The Ping monitor will not block until Timeout is hit. It is useful for validating a target is still up. The last step of EUFuzzer is to locate the packet(s) that causes the crash of HMI. EUFuzzer employs binsearch to accelerate this process. For instance, when the HMI stops work at the 200th packet, we need to find out which packet is the inducing factor. EUFuzzer will retest the HMI with 100th to 200th, then 150th to 200th, then 175th to 200th...until it locates the packet that causes the crash, for example, the 190th packet. For the sake of accuracy, EUFuzzer continues this binsearch from 190th to 200th, to determine whether the last 10 packets are benign or crash-inducing.

IV. EXPERIMENTAL EVALUATION

Three different experiments have been conducted to evaluate the performance of EUFuzzer. First, we validate the packet identification and classification capability of EUFuzzer. Then, the efficiency of vulnerability discovery of EUFuzzer is compared with three open source fuzzers: PeachFuzzer(in souceforge) [9], [44], Fuzzer (maintained by rmadair) [45] and Radamsa(by Aoh) [46]. The last experiment conducted is to validate EUFuzzer in real proprietary SCADA protocol implementation in HMI.

A. THE 1ST EXPERIMENT: PACKET FILEDS IDENTIFICATION AND CLASSIFICATION

EUFuzzer is similar with PI [47] to some extent in filed identification. PI intends to determine fields in protocol packets as detailed as possible, while EUFuzzer just wants to figure out mutable fields, which is more preferable in fuzzing. For illustrative purpose, Modbus [48] and ICMP [49], [50] are selected as an example of "blackbox" protocol as they are comprehensive and easy to follow. We compare the results generated by EUFuzzer and PI dissector on captures of these two public protocols. Both the packet captures of Modbus and ICMP are consisted of 2000 packets. We randomly separate these packets into 10 partitions with equal number of packets

TABLE 1. Running times for ICMP and modbus captures.

Method	PI			EUFuzzer		
	real	user	sys	real	user	sys
ICMP	0m4.797s	0m4.512s	0m0.232s	0m4.587s	0m4.292s	0m0.224s
Modbus	0m37.506s	0m36.856s	0m0.356s	0m13.636s	0m13.462s	0m0.095s

in each partition respectively. Then we conduct the experiment and calculate the mean value.

As for the case of ICMP, the results are as follows ($1B = 1byte = 8bits$):

- Original ICMP format: [1 B][1 B][2 B][2 B][2 B]
- ICMP format identified by PI: [1 B][1 B][2 B][2 B][1 B] [1 B]
- ICMP format identified by EUFuzzer: [2 B][2 B][2 B] [2 B]

EUFuzzer divides four types of fields within a packet: constant fields, session-related fields, length fields, mutable fields. As the case of ICMP, the fields are tokened in order as: constant field, mutable field, session-related field, mutable field.

Due to the fact that the response and request packet format is not always the same. We identify the fields of packet of requests and responses respectively of modbus:

- Original Request structure: [2 B][2 B][1 B][1 B][1 B] [1 B][x B]
- Request fields identified by PI: [2 B][2 B][1 B][1 B] [1 B][1 B][x B]
- Request fields identified by EUFuzzer: [2 B][3B][1 B] [1 B][1 B][x B]
- Original Response structure: [2 B][2 B][1 B][1 B] [1 B][1 B][y B]
- Response fields identified by PI: [2 B][2 B][1 B] [1 B][2 B][y B]
- Response fields identified by EUFuzzer: [2 B][3 B][1 B] [1 B][1 B][1 B][y B]

The variables x and y indicate that the field length is not fixed.

We record the time of PI and EUFuzzer for processing 200 packets and calculate the mean value. Table 1 shows that EUFuzzer is much faster than PI when dealing with Modbus protocol. However, both of them have an approximative running time with ICMP. Why? The reason is that ICMP protocol does not have length fields, where the advantage of EUFuzzer cannot be utilized. Experiments show that the results are consistent with four types of fields when using EUFuzzer.

B. THE 2ND EXPERIMENT: VULNERABILITY DISCOVERY CAPABILITY

Open source library libmodbus [48] is used to build a modbus client with some simple input validation and some embedded bugs to verify RUFuzzer's capability of vulnerability discovery. The modbus client

TABLE 2. Bug distribution in simulation environment.

Bug Type \ Func Code	0x01	0x02	0x0F	0x06	0x03	0x10	0x17
buffer overflow	Y	Y	Y	Y	Y	Y	Y
incomplete packets	Y	Y	Y	Y	Y	Y	Y
wrong number	N	Y	Y	N	Y	Y	Y
format error	N	N	N	Y	Y	Y	Y
wrong function code	Y	Y	Y	Y	Y	Y	Y
wrong starting address	Y	Y	Y	Y	Y	Y	Y
wrong ending address	Y	Y	Y	Y	Y	Y	Y

TABLE 3. Statistics of packets generated by different fuzzers.

Tools	total packets(T)	number of answered packets(A)	number of bug-inducing packets(B)	running time	number of bugs
Randomfuzzer	too many	11	2	72,000s	1
PeachFuzzer	31,786	10,955	1679	39,733s	34
Fuzzer	3,723	531	149	4,852s	9
Radamsa	10,000	2,154	874	13,221s	29
EUFuzzer	13,561	10,252	5398	16,273s	44

contains all major functions of libmodbus: write_coil(0x01), read_bits(0x02), write_coils(0x0F), write_register(0x06), read_register(0x03), write_registers(0x10), read_registers(0x17). Bug types and their numbers are presented in Table 2:

Three open source fuzzers: PeachFuzzer [9], [44], Fuzzer [45] and Radamsa [46] are chosen as comparison. All these three fuzzers can be used as server to perform vulnerability mining on clients. We use the Modbus protocol specification as an input of PeachFuzzervalid packets as inputs of Fuzzer and Radamsa, for both Fuzzer and Radamsa are mutation-based fuzzing, while PeachFuzzer is generation-based fuzzing. Meanwhile, EUFuzzer is feeded with Modbus captures to classify fields itself. RandomFuzzer, which is totally randomly mutating the bit, is severed as a basis. EUFuzzer uses bit sequential strategy. The RandomFuzzer mutates all fields of the packet and generates so many packets that we just run RandomFuzzer for 20 hours as a baseline.

We define P_1, P_2, P_3 to describe the efficiency of fuzzers.

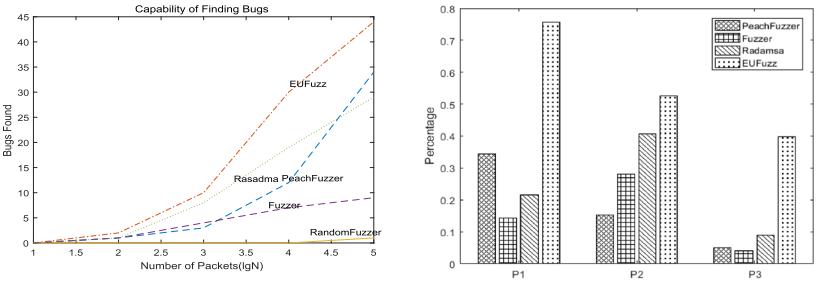
$$P_1 = A/T \tag{2}$$

$$P_2 = B/A \tag{3}$$

$$P_3 = B/T \tag{4}$$

P_1 indicates the percentage of answered packets in total, P_2 represents the percentage of bug-inducing packets in answered packets, and P_3 reveals the percentage of bug-inducing packets in total.

RandomFuzzer plays the role of baseline, so we just analyze the left four tools. As illustrated in Tabel 3, we can find PeachFuzzer generates the most packets. However, the total number of packets generated by EUFuzzer is not the least. We analyze the scheme of the other three fuzzers and draw



(a) Capability and Speed of Finding Bugs

(b) Performance and Efficiency

FIGURE 5. Four tools in comparison.

the conclusion that: PeachFuzzer has the sophisticated mutating mechanism, while the mutation methods of Fuzzer and Radamsa are relatively simple. Nonetheless, EUFuzzer finds all bugs while other three tools only find partial. As is shown in Fig.5(a), EUFuzzer proves itself in vulnerability mining with high speed. In the light of the statistics presented in Fig.5(b), EUFuzzer has the highest percentage (up to 75.6%) of packets to pass the input check and answered by HMI. By the sharp contrast, Fuzzer only has 14.3%. Of all the answered packets, EUFuzzer retains 52.6% packets to be effective in triggering bugs, while PeachFuzzer just has 15.3%. What we concern most is the total efficiency, i.e. the bug-inducing packets of all packets. EUFuzzer can achieve 39.8% while the other three are very low to be below 10%.

And we also verify EUFuzzer on a proprietary protocol–Step7, which is used in Siemens PLC. Because it is proprietary, we use Snap7 [51] instead. Snap7 is an open source, 32/64 bit, multi-platform Ethernet communication suite for interfacing natively with Siemens S7 PLCs. The bug types we set are (1)wrong data length, (2)wrong data number, (3)wrong starting address,(4)wrong parameter number, (5)wrong parameter length. There are only one bug of each bug type. EUFuzzer uses 79 test cases to trigger all bugs.

C. THE 3RD EXPERIMENT: PRACTICAL APPLICATION OF PROPRIETARY PROTOCOL IMPLEMENTATION

We conducted this experiment in our lab testbed. This type of HMI is widely used in Oil and Petrochemical in China, using a proprietary protocol and we don't have the specification. EUFuzzer communicates with HMI and responds with crafted packets. Experiment configuration is as follows:

- CPU:Intel XEON3050(2.13)
- Memory:8G
- Disk:500G
- Operation System: WIN 7(x64)
- HMI version:4.0

EUFuzzer and HMI software are implemented on a laptop of the above configure respectively.

EUFuzzer totally generates 15,361 crafted responses, of which 11,437 packets are answered. Totally 12 severe vulnerabilities are found by EUFuzzer. These flaws have been submitted to the SCADA software vendor and they have conformed that 4 flaws were zero-day vulnerabilities and have taken measures to patch up. We repeat our experiment on different configuration of platforms and find that the performance of the EUFuzzer depends on the configuration of the platform. Nonetheless, whatever the platform is, the total packets and bugs are always the same.

V. CONCLUDING REMARKS

EUFuzzer has been tested on some well known protocols and a few proprietary protocols. The test performance meets our expectation. Experimental results show that it has good performance both in packet field identification and vulnerability discovery. However, EUFuzzer also has its own limitations:

1) EUFuzzer has a relatively coarse granularity.
2) EUFuzzer cannot find vulnerabilities caused by several packets together.

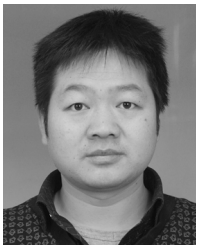
These two problems are interrelated. In our future work, we will refine the granularity of EUFuzzer to improve the proportion of semi-valid packets in a more detailed way and to extend the testing to protocols used in the Internet of Things [52]–[55] and also investigate the possibility of increasing the test accuracy [56].

REFERENCES

[1] S. Bratus, A. Hansen, and A. Shubina, "LZfuzz: A fast compression-based fuzzer for poorly documented protocols," Tech. Rep., 2008.
[2] R. Shapiro, S. Bratus, E. Rogers, and S. Smith, "Do-it-yourself SCADA vulnerability testing with LZFuzz," Tech. Rep., 2017.
[3] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of UNIX utilities," *Commun. ACM*, vol. 33, no. 12, pp. 32–44, Dec. 1990.
[4] B. P. Miller, D. Koski, C. P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl, "Fuzz revisited: A re-examination of the reliability of UNIX utilities and services," Dept. Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep., 1995.
[5] P. Uhley. *A Basic Distributed Fuzzing Framework for FOE*. [Online]. Available: https://blogs.adobe.com/security/2012/05
[6] P. Godefroid, M. Y. Levin, and D. Molnar, "SAGE: Whitebox fuzzing for security testing," *Commun. ACM*, vol. 55, no. 3, pp. 40–44, 2012.
[7] D. J. Conger, K. Srinivasamurthy, and R. S. Cooper, "Distributed file fuzzing," U.S. Patent 7 743 281, Jun. 22, 2010.
[8] T. Proell, "Fuzzing proprietary protocols: A practical approach," in *Proc. Secur. Educ. Conf. Toronto*, 2010.
[9] M. Eddington, "Peach fuzzing platform," *Peach Fuzzer*, vol. 34, 2011.
[10] G. Devarajan, "Unraveling SCADA protocols: Using sulley fuzzer," in *Proc. Defon 15 Hacking Conf*, 2007, pp. 1–40.

[11] D. Aitel, "An introduction to spike, the fuzzer creation kit," Tech. Rep., Aug. 2002.
[12] R. Koch. (2016). *Profuzz*. [Online]. Available: https://github.com/HSASec/ProFuzz
[13] G. Devarajan. *Sulley Fuzzing Framework*. [Online]. Available: http://code.google.com/p/sulley
[14] SecuriTeam. *Securiteam Software Testing Framework*. [Online]. Available: http://www.beyondsecutity.com/black-box-testing.html
[15] Digital Bond. *iccpsic Assessment Tool*. [Online]. Available: http://www.digitalbond.com/2007/08/28/iccpsic-assessment-tool-set-relaeased/
[16] Mu Dynamics. *Mu Test Sutie*. [Online]. Available: http://mudynamics.com/products/Mu-Test-Suite.html
[17] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Driller: Augmenting fuzzing through selective symbolic execution," in *Proc. NDSS*, vol. 16, 2016, pp. 1–16.
[18] I. Haller, A. Slowinska, M. Neugschwandtner, and H. Bos, "Dowsing for overflows: A guided fuzzer to find buffer boundary violations," in *Proc. 22nd USENIX Secur. Symp. (USENIX Secur.)*, 2013, pp. 49–64.
[19] M. Woo, S. K. Cha, S. Gottlieb, and D. Brumley, "Scheduling black-box mutational fuzzing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 511–522.
[20] A. Rebert, S. K. Cha, T. Avgerinos, J. Foote, D. Warren, G. Grieco, and D. Brumley, "Optimizing seed selection for fuzzing," in *Proc. 23rd USENIX Secur. Symp. (USENIX Secur.)*, 2014, pp. 861–875.
[21] H. Dantas, Z. Erkin, C. Doerr, R. Hallie, and G. van der Bij, "eFuzz: A fuzzer for DLMS/COSEM electricity meters," in *Proc. 2nd Workshop Smart Energy Grid Secur.*, 2014, pp. 31–38.
[22] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," *Future Gener. Comput. Syst.*, vol. 78, pp. 987–994, Jan. 2018.
[23] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy risk analysis and mitigation of analytics libraries in the Android ecosystem," *IEEE Trans. Mobile Comput.*, to be published.
[24] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in Android applications for malicious application detection," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 11, pp. 1869–1882, Nov. 2014.
[25] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "Droidensemble: Detecting Android malicious applications with ensemble of string and structural static features," *IEEE Access*, vol. 6, pp. 31798–31807, 2018.
[26] X. Wang, W. Wang, Y. He, J. Liu, Z. Han, and X. Zhang, "Characterizing Android apps' behavior for effective detection of malapps at large scale," *Future Gener. Comput. Syst.*, vol. 75, pp. 30–45, Oct. 2017.
[27] X. Liu, J. Liu, W. Wang, Y. He, and X. Zhang, "Discovering and understanding Android sensor usage behaviors with data flow analysis," *World Wide Web*, vol. 21, no. 1, pp. 105–126, 2018.
[28] W. Wang, M. Zhao, and J. Wang, "Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J. Ambient Intell. Hum. Comput.*, to be published.
[29] W. Wang, J. Liu, G. Pitsilis, and X. Zhang, "Abstracting massive data for lightweight intrusion detection in computer networks," *Inf. Sci.*, vols. 433–434, pp. 417–430, Apr. 2018.
[30] W. Wang, Y. He, J. Liu, and S. Gombault, "Constructing important features from massive network traffic for lightweight intrusion detection," *IET Inf. Secur.*, vol. 9, no. 6, pp. 374–379, 2015.
[31] W. Wang and R. Battiti, "Identifying intrusions in computer networks with principal component analysis," in *Proc. 1st Int. Conf. Availability, Rel. Secur. (ARES)*, Apr. 2006, pp. 270–279.
[32] W. Wang, T. Guyet, R. Quiniou, M.-O. Cordier, F. Masseglia, and X. Zhang, "Autonomic intrusion detection: Adaptively detecting anomalies over unlabeled audit data streams in computer networks," *Knowl.-Based Syst.*, vol. 70, pp. 103–117, Nov. 2014.
[33] W. Wang, X. Guan, and X. Zhang, "Processing of massive audit data streams for real-time anomaly intrusion detection," *Comput. Commun.*, vol. 31, no. 1, pp. 58–72, 2008.
[34] W. Wang, X. Guan, X. Zhang, and L. Yang, "Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data," *Comput. Secur.*, vol. 25, no. 7, pp. 539–550, 2006.
[35] C. Rohlf. *Quemod*. [Online]. Available: https://github.com/struct/QueMod
[36] R. Shapiro, S. Bratus, E. Rogers, and S. Smith, "Identifying vulnerabilities in SCADA systems via fuzz-testing," in *Proc. Int. Conf. Crit. Infrastruct. Protection*. Springer, 2011, pp. 57–72.
[37] M. Dayhoff, R. Schwartz, and B. Orcutt, "22 a model of evolutionary change in proteins," in *Atlas of Protein Sequence and Structure*, vol. 5. National Biomedical Research Foundation Silver Spring, 1978, pp. 345–352.
[38] M. A. Beddoe, "Network protocol analysis using bioinformatics algorithms," in *Proc. ToorCon*, 2004.
[39] W. Cui, V. Paxson, N. Weaver, and R. H. Katz, "Protocol-independent adaptive replay of application dialog," in *Proc. NDSS*, 2006, pp. 1–15.
[40] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proc. Nat. Acad. Sci. USA*, vol. 89, no. 22, pp. 10915–10919, 1992.
[41] Wikipedia. *Multiple Sequence Alignment*. [Online]. Available: https://en.wikipedia.org/wiki/Multiple-sequence-alignment
[42] M. Gouy, S. Guindon, and O. Gascuel, "SeaView version 4: A multiplatform graphical user interface for sequence alignment and phylogenetic tree building," *Mol. Biol. Evol.*, vol. 27, no. 2, pp. 221–224, 2009.
[43] M. Steinbach, M. S. G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *Proc. KDD Workshop Text Mining*, Boston, MA, USA, vol. 400, 2000, pp. 525–526.
[44] Peach Community. *Peach Fuzzer Community Edition*. [Online]. Available: https://sourceforge.net/projects/peachfuzz/files/Peach/
[45] Rmadair. *Fuzzer Maintained by Rmadair*. [Online]. Available: https://github.com/rmadair/fuzzer
[46] Radamsa. *A General-Purpose Fuzzer Called Radamsa*. [Online]. Available: https://github.com/aoh/radamsa
[47] M. Beddoe, "The protocol informatics project," Tech. Rep., 2004.
[48] Modbus Organzation. *A General-Purpose Fuzzer Called Radamsa*. [Online]. Available: http://www.modbus.org/
[49] Wikipedia. *Internet Control Message Protocol*. [Online]. Available: https://en.wikipedia.org/wiki/Internet-Control-Message-Protocol
[50] A. B. Forouzan, *Data Communications and Networking*. New York, NY, USA: McGraw-Hill, 2006.
[51] Davide Nardella. *Step7 Open Source Ethernet Communication Suite*. [Online]. Available: http://snap7.sourceforge.net/
[52] R. H. Weber and R. Weber, *Internet of Things*, vol. 12. Springer, 2010.
[53] G. Q. Xu, Y. Zhang, A. K. Sangaiah, X. H. Li, A. Castiglione, and X. Zheng, "CSP-E2: An abuse-free contract signing protocol with low-storage TTP for energy-efficient electronic transaction ecosystems," *Inf. Sci.*, vol. 476, pp. 505–515, Feb. 2019.
[54] G. Q. Xu, J. Liu, Y. R. Lu, X. J. Zeng, Y. Zhang, and X. M. Li, "A novel efficient MAKA protocol with desynchronization for anonymous roaming service in global mobility Networks," *J. Netw. Comput. Appl.*, vol. 107, pp. 83–92, Apr. 2018.
[55] X. Zeng, G. Xu, X. Zheng, Y. Xiang, and W. Zhou, "E-AUA: An efficient anonymous user authentication protocol for mobile IoT," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1506–1519, Jun. 2018.
[56] C. Zhang, C. Liu, X. Zhang, and G. Almpanidis, "An up-to-date comparison of state-of-the-art classification algorithms," *Expert Syst. Appl.*, vol. 82, pp. 128–150, Oct. 2017.

JIAPING MEN received the B.S. degree from the North China University of Science and Technology, China, in 1999, and the M.S. degree from Sichuan University, China, in 2010. He is currently pursuing the Ph.D. degree with the School of Computer and Information Technology, Beijing Jiaotong University, China. His main research interests include in security and privacy in cloud computing.



GUANGQUAN XU received the Ph.D. degree from Tianjin University, in March 2008. He is currently pursuing the Ph.D. degree with the Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, China, where he is also a Full Professor. He is also the Director of Network Security Joint Lab and the Network Attack and Defense Joint Lab. His research interests include cyber security and trust management. He has published 70+ papers in reputable international journals and conferences, including the IEEE IoT-J, FGCS, IEEE Access, PUC, JPDC, and the IEEE Multimedia. He is a member of the CCF. He served as a TPC Member for the IEEE UIC 2018, SPNCE2019, IEEE UIC2015, IEEE ICECCS 2014, and reviewers for journals, such as IEEE Access, ACM TIST, JPDC, IEEE TITS, Soft Computing, FGCS, and Computational Intelligence.

ZHEN HAN received the Ph.D. degree from the China Academy of Engineering Physics, in 1991. He is currently a Professor with the School of Computer and Information Technology of Beijing Jiaotong University. He has authored or co-authored over 100 papers in various journals and international conferences. His main research interests include information security architecture and trusted computing.

ZHONGHAO SUN received the Ph.D. degree from the Northwestern Polytechnical University, in 2017. He is currently an Engineer with the National Computer Network Emergency Response Technical Team/Coordination Center of China (known as CNCERT or CNCERT/CC). He has authored or co-authored over 20 papers in various journals and international conferences. His main research interests include cyber security and industrial control system (ICS) security.

XIAOJUN ZHOU received the B.S. degree in electric power engineering from the Shanghai University of Electric Power, in 2008, and the M.S. and Ph.D. degrees in cyberspace security from the University of Chinese Academy of Sciences, in 2015 and 2018, respectively.

He is currently an Assistant Researcher with the Institute of Information Engineering of the Chinese Academy of Sciences and an Application Engineer of DCS. His research interests include the industrial control protocol security analysis, industrial internet security, the Internet of Things security. He is also responsible for or participated in several industrial security projects, including a series of major scientific research projects, such as the National Natural Science Foundation Projects, the National Key Research and Development Plan, and the Key Research and Development Projects of the Chinese Academy of Sciences. He led the research team to conduct in-depth research on protocol security and achieved many results, and published several papers in domestic and foreign journals. In the 2015 EICS+ industrial control system information security attack and defense competition, he received the third prize. And he achieved the Excellence Award in the 2016 ''BEWG Cup'' second national industrial control system information security attack and defence competition.

WENJUAN LIAN received the master's degree and doctor's degree from the Shandong University of Science and Technology, in 2002 and 2011, respectively. She is currently pursuing the Ph.D. degree with the College of Computer Science and Engineering, Shandong University of Science and Technology, where she is also an Associate Professor. She has published 20+ papers in core journals and international conferences, finished 10+ national and provincial projects, published 3 books. Her research interests include deep learning and cyber security.

XIAOCHUN CHENG (SM'04) received the B.Eng. degree in computer software engineering and the Ph.D. degree in computer science from Jilin University, in 1992 and 1996, respectively. He has been with the Computer Science EU Project Coordinator, Middlesex University, since 2012. He is currently a member of IEEE SMC Technical Committee on Enterprise Information Systems, IEEE SMC Technical Committee on Computational Intelligence, IEEE SMC Technical Committee on Cognitive Computing, IEEE SMC Technical Committee on Intelligent Internet Systems, IEEE Communications Society Communications and Information Security Technical Committee, BCS Information Security Specialist Group, BCS Cybercrime Forensics Specialist Group, and BCS Artificial Intelligence Specialist Group.