Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 762891, 12 pages
http://dx.doi.org/10.1155/2014/762891

Research Article

A Novel Fuzzing Method for Zigbee Based on Finite State Machine

Baojiang Cui,[1] Shurui Liang,[1] Shilei Chen,[1] Bing Zhao,[2] and Xiaobing Liang[2]

[1] *Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2] *China Electric Power Research Institute, Beijing 100876, China*

Correspondence should be addressed to Baojiang Cui; cuibj@bupt.edu.cn

Received 6 October 2013; Accepted 18 October 2013; Published 20 January 2014

Academic Editor: Fatos Xhafa

With the extensive application of Zigbee, some bodies of literature were devoted into finding the vulnerabilities of Zigbee by fuzzing. According to earlier test records, the majority of defects were exposed due to a series of testing cases. However, the context of malformed inputs is not taken account into the previous algorithms. In this paper, we propose a refined structure-based fuzzing algorithm for Zigbee based on FSM, FSM-fuzzing. Any malformed input in FSM-Fuzzing is injected to the tested sensor against a specific initial state. If the sensor transferred to the next state of FMS or crashed, there would be a defect of Zigbee in dealing with the input under the state. The final state of the sensor is verified by an UIO sequence. After a round of tests, the sensor is regressed to the specific state to prepars for receiving the next mutation. All of the states would be traversed in FSM-fuzzing. A fuzzing tool, ZFSM-fuzzer, is designed for evaluating the performance of FSM-fuzzing. Experiment results show that there is a vulnerability of Zigbee in dealing with the frames without destination addresses. Further, the quality of cases of FSM-fuzzing is higher than the previous algorithms. Therefore, FSM-fuzzing is powerful in finding the vulnerabilities of Zigbee.

1. Introduction

Zigbee attracts much attention in recent years due to the growth of the demand for low power consumption [1]. Therefore, Zigbee sensors are lightweight with slow operation speed and low storage ability. Moreover, the sensors are usually deployed in the outside [2]. As a result, Zigbee networks are faced with multiple attacks, such as data monitor, data tampering, and Denial of Service (DoS) attack [3].

Some bodies of literature focused on the security analysis and the security framework of Zigbee. They proposed key management schemes for encrypting data and identifying impersonated nodes with low cost [4]. They also proposed privacy protection schemes [5] and security routing protocols [6] to enhance the security of Zigbee networks. The previous research result shows that there are several vulnerabilities with Zigbee protocol, which would lead to the jamming or even crashing of the networks. Thus, mining the potential defects of Zigbee and refining them are important for the security of networks.

Fuzzing test is widely used in finding defections of network protocols and file applications [7]. Fuzzing test is implemented by injecting malformed testing cases to the target system and detecting the operation status of the target [8, 9]. If the target system crashed, there may be vulnerability in it. Previously, the test cases of fuzzing test were generated in the random algorithm. Time cost and resource cost of random fuzzing test are both huge. Furthermore, little defects are exposed in this way. Some researchers proposed testing cases generation algorithms in different ways for improving the efficiency of fuzzing test. The general refined fuzzing algorithms are structure-based algorithm [10], key-field-based algorithm [11], and boundary-based algorithm [12]. However, the relationship between testing cases and the transformation of operation state of the target system are not taken into account in the refined algorithms. And several vulnerabilities are exposed owing to the interaction between special states and particular testing cases. For these reasons, a refined fuzzing algorithm based on state transferring is proposed.

In this paper, we present a refined algorithm for generating fuzzing test cases for Zigbee, FSM-fuzzing. It combines finite state machine (FSM) with structure-based fuzzing algorithm according to MAC protocol of Zigbee. The sensors

transfer from one state to another in line with Zigbee on working. Before checking the defects of Zigbee in one process, FSM-fuzzing makes the tested sensor work in the previous state. Then a test case against the state transferring process which is semicompliance with the frame construction rules of Zigbee is sent to the target sensor. If the semicompliance case makes the sensor transfer to the next state successfully or makes the sensor or network crash, there may be vulnerability in Zigbee or stack. The unique input and output (UIO) sequence is used for checking the state of the target sensor after dealing with the test case. Finally, the test sequences are regressed to the initial state by regression process so that FSM-fuzzing can implement the next round of test. Regression process improves the automation level of FSM-fuzzing.

In order to verify the advantage of FSM-fuzzing in finding potential defects, a fuzzing framework named ZFSM-fuzzer is designed. ZFSM-fuzzer is implemented on Z-Stack which is a widely used Zigbee stack. In this paper, we take the process of connection and disconnection as an example to verify the usability of FSM-Fuzzing. Experiment results show that there is an abnormality of Z-Stack in dealing with a kind of malformed packets that don not contain the destination address. A series of performance tests are also implemented among random-based algorithm, structure-based algorithm, and FSM-fuzzing. Compared with random-based algorithm, the scale of testing cases and time cost of FSM-fuzzing is greatly reduced. Compared with structure-based algorithm, the quality of test cases of FSM-fuzzing is higher. In addition, it is easier for FSM-fuzzing to locate the reasons of vulnerabilities than structure-bases algorithm.

The rest of this paper is organized as follows. Section 2 introduces some related works in the security of Zigbee. Section 3 describes the theory of FSM and the main idea of FSM-fuzzing. Section 4 presents the framework of ZFSM-fuzzer. After designing the topology of tested network, experiments are conducted and the results are shown in Section 5. Finally, we conclude this paper and point out the future work in Section 6.

2. Related Work

The attributes of poor storage ability and low computing capacity lead to vulnerability of Zigbee networks. The threats against Zigbee networks can be divided into six classifications: attacks against data privacy, DoS attacks, node compromise, side-channel attacks, impersonation attacks, and protocol-specific [13]. These threats may lead to disclosure of sensitive data, increasing of network load, or even communications blocking.

In order to resist primary attacks, four kinds of security services are ordered by the MAC layer of Zigbee.

(i) Data encryption: the payload of MAC frame can be encrypted in AES, if the security bit is set to 1 in the frame control field of the MAC frame. There are three encryption modes offering different levels, which are AES-CTR, AES-CBC-MAC, and AES-CCM [14, 15].

(ii) Data freshness: every frame owns an increasing sequence number in the head of MAC frame. The receiver checks the freshness of the sequence number. This measure is against reply attacks and data forgery.

(iii) Frame check sequence (FCS): there is FCS occupying two bytes at the end of MAC frame. The sequence is unique for each frame. If the FCS does not match with the frame, the frame is probably tampered with by attackers during transmission.

(iv) Access control list (ACL): it is an optional security service on MAC layer. The upper layer of MAC maintains a list of authentic nodes. If the access control service is started, the sensor will reject all data from unauthentic sensors [16].

Some researchers paid attention to other schemes to enhance the security of wireless sensor network (WSN) including Zigbee. Mogre et al. [17] researched the security requirement of WSNs and proposed a security framework. They declared that the security of WSN depended on the comprehensive detection of misbehavior, security routing protocol, and reputation management. McCusker and O'Connor [18] presented a symmetric key distribution scheme for WSN. The scheme met the strict energy constraint through porting the pairing component to hardware. Karlof and Wagner [19] summarized the threats against routing protocols, such as selective forwarding, Sybil attacks and wormholes, and suggested countermeasures.

Some researchers also focused on finding vulnerabilities of protocol of WSN. Shin Jung et al. [20] found a defect of Zigbee in beacon networks. The coordinator would not verify the real identity of a node with an authentic ID in a GTS request. The requests from the legitimated nodes would be ignored if the attackers cloned large scale of legal nodes and sent GTS requests. DoS attack was achieved.

In order to increase the automation level of vulnerability mining, fuzzing was introduced by Mendonça and Neves [12] to test Wi-Fi, which was early used in finding defects of wire network protocols. Lahmadi et al. [21] also applied fuzzing on discovering vulnerabilities in 6LoWPAN networks and designed a testing framework. Peng et al. [22] firstly designed architecture for fuzzing test on MAC layer of Zigbee, called ZBCBT.

However, testing cases are generated in random-based algorithm in black-box Fuzzing test. The time cost and resource cost of random fuzzing test are huge. Therefore, improving the efficiency of Fuzzing test for WSN protocols becomes a hot area. Peng et al. proposed a refined algorithm, ZBCA, employed in ZBCBT to generate fuzzing test cases. The subfields of the cases in ZBCA were set with the boundary value so that the probability of exposing vulnerabilities was increased. A vulnerability of Zigbee that would lead to the dissociation between end devices and coordinators was also presented. Cui et al. [23] also proposed a refined fuzzing algorithm based on node clone.

Research results show that the vulnerabilities ordinarily expose in the special state. However, the state transferring of the tested nodes is not taken into account in the above refined algorithms. Several abnormalities in deep paths are

not exposed. Therefore, a refined fuzzing algorithm based on FSM is proposed to deal with the problem in this paper.

3. FSM-Fuzzing

Fuzzing test is a frequently used technique for finding vulnerabilities of protocols and programs. It makes exceptions exposed by inserting specific inputs like malformed data or faults into the target system which is monitored by testers simultaneously [24]. Fuzzing test was first proposed by Miller et al. [25] for finding vulnerabilities of UNIX programs in 1990. This test tool generated a battery of inputs randomly and inserted them into the applications, which then caused more than 24% of applications to crash [7]. However random-based algorithm which generates a huge number of test cases is inefficient. In order to reduce the number of invalid test cases and improve the path coverage of target system, smart fuzzing testing frameworks were proposed. At present, for various applications and protocols, corresponding fuzzing testing tools can be used. For example, COMRaider and AxMan [26] are designed for ActiveX controls, as well as SPIKE [27] for network protocols and Peach [28] for files and protocols.

In this section, we propose a refined fuzzing algorithm for Zigbee, FSM-fuzzing. The algorithm is based on the theory of FSM. The testing object in this method is the state transition of the tested sensor. Before sending malformed inputs, the tested sensor is changed to an initial state. Through verifying the final state by the UIO sequence, defects of the tested stack can be found. If the sensor transfers to the next state successfully or even the sensor is out of work, there would be a defect in dealing with the kind of mutations.

3.1. Finite State Machine (FSM). The essence of protocol is that the sensors transfer from one state to another complying with the rules. FSM is widely used in describing state migration of sensors by a digraph [29]. A FSM M is a 5-tuple $M = (S, I, O, \delta, \gamma)$, where

(i) $S = \{s_0, s_1, \ldots, s_{n-1}\}$ is a finite set of states,

(ii) $I = \{i_0, i_1, \ldots, i_{n-1}\}$ is a set of inputs,

(iii) $O = \{o_0, o_1, \ldots, o_{n-1}\}$ is a set of outputs,

(iv) $\delta : S \times I \rightarrow S$ is a state migration function,

(v) $\gamma : S \times I \rightarrow O$ is an output function.

At any moment, FSM can only work on one state in S. The digraph $G = (V, E)$ can be determined by FSM of the protocol. All of the states are taken as the vertices in G and the transferring of states is a directed arc.

(i) $V = \{v_0, v_1, \ldots, v_{n-1}\}$ is the set of vertices. The set represents S of M.

(ii) $E = \{(v_i, v_j; i/o) : v_i, v_j \in V\}$ is the set of arcs. It represents a series of state migrations, in which an element $(v_i, v_j; i/o)$ means the transferring from s_i to s_j. The input of the event is i and the output is j.

An example of G is shown in Figure 1.

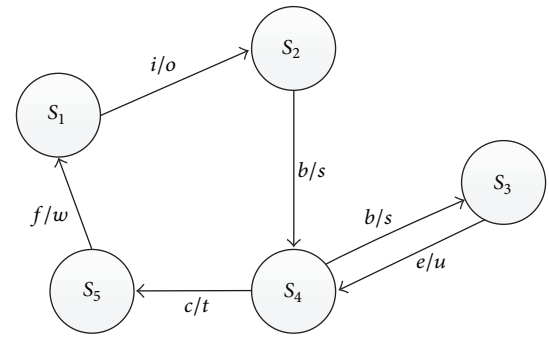

Figure 1: Example of digraph G.

The purpose of building FSM is getting the testing sequences according to the state migration digraph G. There are four general methods of generation testing sequences based on FSM, that is, transition tour method (T method), distinguishing sequences method (D method), charactering sequences method (W method), and unique input/output (UIO) sequences method (U method) [30].

X is assumed as the UIO sequence of s_i. X exists if and only if the output of s_i under X is different from the output of any state under X,

$$\lambda(s_i, X) \neq \lambda(s_j, X), \quad i \neq j. \tag{1}$$

The UIO sequence of s_i is represented by UIO(s_i). UIO sequence is widely used for verify the current state of FSM. The state of FSM can be sure only if there is an UIO sequence under the state.

3.2. FSM-Fuzzing for Zigbee. FSM-fuzzing is a refined fuzzing algorithm for Zigbee based on FSM. The main idea of FSM-fuzzing is making the tested nodes work on a special state before injecting a malformed frame to it. Therefore, the process of FSM-fuzzing can be conducted as follows.

(i) Create FSM for each process of Zigbee [31] and draw the state transition diagram [32]. Each arc of the diagram is treated as a test object. If the sensor successfully achieves the transition or crashes under a malformed input, vulnerability is exposed.

(ii) Obtain the UIO sequence of each state. The malformed input and the UIO sequence of the next state form the testing sequence against one arc. The UIO sequence is used for verifying the current state of the sensor.

(iii) Implement regression process. Reset the sensor into the previous state so that the next malformed input can be directly injected into it. The regression sequence is also a part of the testing sequence against the migration.

(iv) Generate a set of malformed inputs for the migration in structure-based algorithm. The set is the negation of the normal inputs.

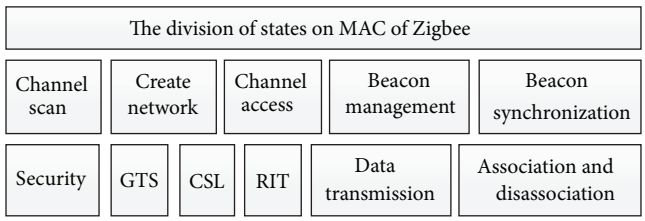

Figure 2: The division of states on MAC.

Table 1: Sates of connection and disconnection.

State	Comment
S_1	Disassociation state
S_2	Coordinator waiting state
S_3	Association state

Table 2: Input of state transition.

Input	Comment
Tr_1	Sending association request and receiving ACK
Tr_2	Sending data request and receiving ACK and the response of successful association
Tr_3	Sending association request and receiving ACK and the response of failure association or waiting timeout
Tr_4	Sending disassociation request and receiving ACK

Table 3: Set of arcs.

Title	Arc
e_1	$e(s_1, s_2, \mathrm{Tr}_1)$
e_2	$e(s_2, s_3, \mathrm{Tr}_2)$
e_3	$e(s_2, s_1, \mathrm{Tr}_1)$
e_4	$e(s_2, s_1, \mathrm{Tr}_4)$

According to IEEE 802.15.4, the state transition on the MAC layer of Zigbee can be divided into twelve processes as shown in Figure 2.

The process of association and disassociation is the most simple among the processes. In this part, it is taken as an example to explain FSM-fuzzing. The process of connection and disconnection is shown in Figure 3.

3.2.1. Creating FSM. The states of the tested protocol in the process of connection and disconnection are listed in Table 1.

The inputs of state transitions are shown in Table 2. They are described in the angle of the testing system.

The state transition digraph is shown in Figure 4.

The set of the arcs E is shown in Table 3.

3.2.2. Obtaining UIO Sequence. The digraph of association and disassociation is simple. Therefore, the UIO sequence of each state can be obtained manually. The shortest UIO sequences are listed in Table 4. A sequence is represented by a 3-tuple:

$$\mathrm{UIO}\left(s_i\right) = f\left(s_i, s_j; \mathrm{Tr}_n\right), \tag{2}$$

Table 4: UIO sequence of each state.

State	UIO sequence
s_1	$f(s_1, s_2, \mathrm{Tr}_1)$
s_2	$f(s_2, s_3, \mathrm{Tr}_2)$ or $f(s_2, s_1, \mathrm{Tr}_3)$
s_3	$f(s_2, s_1, \mathrm{Tr}_4)$

Table 5: Input subsequences for testing.

Transition function	Input sequence
$e(s_1, s_2, s_1; \sim\mathrm{Tr}_1)$	$\sim\mathrm{Tr}_1$, Tr_3
$e(s_1, s_2, s_3; \sim\mathrm{Tr}_1)$	$\sim\mathrm{Tr}_1$, Tr_2
$e(s_2, s_3, s_1; \sim\mathrm{Tr}_2)$	$\sim\mathrm{Tr}_2$, Tr_4
$e(s_2, s_1, s_2; \sim\mathrm{Tr}_3)$	$\sim\mathrm{Tr}_3$, Tr_1
$e(s_3, s_1, s_2; \sim\mathrm{Tr}_4)$	$\sim\mathrm{Tr}_4$, Tr_1

Table 6: Input sequences for each transition function.

Transition function	Regression sequence
$(s_1, s_2; \sim\mathrm{Tr}_1)$	$\sim\mathrm{Tr}_1$, Tr_3
$(s_1, s_2; \sim\mathrm{Tr}_1)$	$\sim\mathrm{Tr}_1$, Tr_2, Tr_4
$(s_2, s_3; \sim\mathrm{Tr}_2)$	Tr_1, $\sim\mathrm{Tr}_2$, Tr_4
$(s_2, s_1; \sim\mathrm{Tr}_3)$	Tr_1, $\sim\mathrm{Tr}_3$, Tr_1, Tr_3
$(s_3, s_1; \sim\mathrm{Tr}_4)$	Tr_1, Tr_2, $\sim\mathrm{Tr}_4$, Tr_1, Tr_3

where s_i presents the verified state. S_j presents the final state due to the unique input. Tr_n presents the unique input sequences.

The input subsequence for testing each transition in FSM-fuzzing is shown in Table 5. The transition function is represented by $e(s_i, s_j, s_k; \mathrm{Tr}_n)$. s_i presents the initial state of the tested transition. s_j presents the next state of the tested transition. s_k presents the terminal state under the UIO sequence. Tr_n presents the type of malformed inputs for fuzzing test.

3.2.3. Regression Process. A transition is tested repeatedly by the malformed inputs in FSM-fuzzing. Therefore, it is necessary to implement regression process. Before implementing the new round of fuzzing test, the tested sensor is set into the initial state of the tested transition from s_1. After implementing the round, the tested sensor should be set into the initial state of the stack s_1. The whole input regression for each transition is shown in Table 6.

3.3. Mutation of Data. There are a series of filter rules to verify the legality of frames on the MAC layer of Zigbee. Most of the testing cases generated in structure-based algorithm are in compliance with the rules. Therefore, FSM-fuzzing is deployed with structure-based algorithm to generate the malformed inputs.

The set of the malformed inputs is the negation of the normal inputs. Except the field of command type, the other fields are fuzzing in structure-based algorithm. The pseudo code of generation malformed Tr_1 is listed in Pseudocode 1.

The pseudocode of generation malformed Tr_2 is listed in Pseudocode 2.

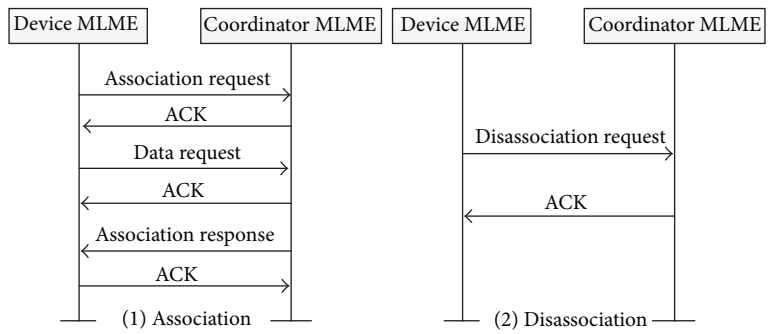

Figure 3: Process of association and disassociation.

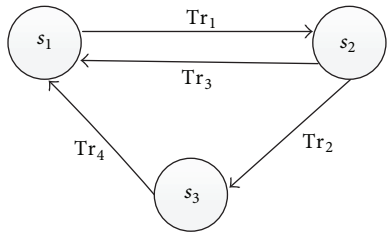

Figure 4: State transition digraph of association and disassociation.

The process of generation malformed Tr_4 is listed in Pseudocode 3.

4. ZFSM-Fuzzer

ZFSM-fuzzer is a fuzzing tool for Zigbee that is deployed with FSM-fuzzing. It consists of two parts as shown in Figure 5. The two parts are fuzzing-controller and fuzzing-executer.

Fuzzing-controller is working on a personal computer (PC). It orders graphical interface for user to control the address of the tested node and the beginning of fuzzing test. It also presents testing results on the interface timely. Fuzzing-executer is working on a Zigbee sensor. It implements a test according to the command from fuzzing-controller. Another duty of this section is listening for communications and transporting the packets to fuzzing-controller for further analyzing.

4.1. Fuzzing-Controller. Fuzzing-controller is designed for comfortably controlling the beginning and the end of fuzzing test. FSM-fuzzing is deployed on it. The testing sequences for fuzzing including the malformed inputs are also generated by this module. Therefore, the burden of fuzzing-Executer is reduced and the rate of injecting testing cases is improved.

Fuzzing-controller is divided into seven sections. UI controller is the top module of fuzzing-controller. It offers the interface for users to configure the parameters of fuzzing, such as the modes of the source address and the destination address, the address of ZFSM-fuzzer and the address of the tested node, and so on. The system also presents the state of the tested node in I/O last field. The interface of ZFMS-fuzzer is shown in Figure 6.

Packet definer is a text file that describes the theory of Zigbee. The structure of the frame and the flows of communications are both defined in it. Testers create FSM of Zigbee according to the definer.

Packet generator is the core module of ZFSM-fuzzing. The testing sequences and the value of the subfields of each case are calculated in this part. The type of testing and the key parameters of the tested network are obtained from the UI controller.

After producing the testing cases, the packet controller informs the serial port controller to send the cases to fuzzing-executer.

The serial port controller deals with the events of the serial port. The controller is in charge of transporting the testing cases to the Zigbee node. It also receives the packets from a sniffer through another serial port and delivers them upper.

The responsibility of packet analyst is analyzing the binary data coming from the serial port according to IEEE 802.15.4.

The analytical results are submitted further to packet decoder. The part picks up the abnormal behavior of the tested network based on packet definer and generates a testing report.

4.2. Fuzzing-Executer. Fuzzing-executer is deployed on the Zigbee nodes. The function of the part is lightweight owing to fuzzing-controller. The primary responsibility of it is injecting testing cases to the tested sensor.

There is a module working on application layer (APL) receiving data from fuzzing-controller. These packets are directly sent to the tested node after setting the FCS.

There is another Zigbee node deployed with fuzzing-Executer. The node plays the role of listener in the network. It is set in the mode of promiscuous mode and captures the packets in the network. The packets are delivered to fuzzing-controller through the serial port either.

5. Experiment

5.1. Topology. Figure 7 shows the basic network topology for implementing fuzzing test with ZFSM-fuzzer.

It needs three Zigbee sensor nodes for implementing the test. One of them is the tested node. And the other two nodes are deployed with ZFSM-fuzzing. One plays as the listener and another is the executer. They both communicate with

```
(1) //Tr1:the generation of association request in Fuzzing
(2) seqPkt[seqLen].flag = PKT_SEND_FUZZ;
(3) seqPkt[seqLen].type = PKT_TYPE_CMD;
(4) seqPkt[seqLen].sec = PKT_FUZZ;
(5) seqPkt[seqLen].Pnd = PKT_FUZZ;
(6) seqPkt[seqLen].Ackr = PKT_FUZZ;
(7) seqPkt[seqLen].Panc = PKT_FUZZ;
(8) seqPkt[seqLen].DA_type = PKT_FUZZ;
(9) seqPkt[seqLen].SA_type = PKT_FUZZ;
(10) seqPkt[seqLen].sqNumber = sqNumber++;
(11)
(12) seqPkt[seqLen].DPan = DPan;
(13) seqPkt[seqLen].SPan = "FFFF";
(14) seqPkt[seqLen].DAddr = "0000";
(15) seqPkt[seqLen].SAddr = "0000";
(16) //The address of Fuzzing tool
(17) seqPkt[seqLen].SAddr_l = "00124B0001301DA2";
(18) //The assress of coordinator
(19) seqPkt[seqLen].DAddr_l = "00124B0001301DFF";
(20) //the command type of association request
(21) seqPkt[seqLen].Remarks = "1";
(22)
(23) seqPkt[seqLen].length = 0;
(24) seqPkt[seqLen].randLength = PKT_NON_FUZZ;
(25)
(26) seqLen++;
(27)
(28) //Tr1:request for ACK
(29) seqPkt[seqLen].flag = PKT_RECV;
(30) seqPkt[seqLen].type = PKT_TYPE_ACK;
(31) seqLen++;
```

Pseudocode 1: Pseudocode of Tr_1.

```
(1) //Tr2:the generation of data request in Fuzzing
(2) seqPkt[seqLen].flag = PKT_SEND_FUZZ;
(3) seqPkt[seqLen].type = PKT_TYPE_CMD;
(4) seqPkt[seqLen].sec = RKT_FUZZ;
(5) seqPkt[seqLen].Pnd = PKT_FUZZ;
(6) seqPkt[seqLen].Ackr = PKT_FUZZ;
(7) seqPkt[seqLen].Panc = PKT_FUZZ;
(8) seqPkt[seqLen].DA_type = PKT_FUZZ;
(9) seqPkt[seqLen].SA_type = PKT_FUZZ;
(10) seqPkt[seqLen].sqNumber = sqNumber++;
(11)
(12) seqPkt[seqLen].DPan = DPan;
(13) seqPkt[seqLen].Span = DPan;
(14) seqPkt[seqLen].DAddr = "0000";
(15) seqPkt[seqLen].SAddr = "FFFF";
(16)
(17) seqPkt[seqLen].SAddr_1 = "00124B0001301DA2";
(18) seqPkt[seqLen].DAddr_1 = "00124B0001301DFF";
(19) seqPkt[seqLen].Remarks = "4";
(20)
(21) seqPkt[seqLen].length = 0;
(22) seqPkt[seqLen].randLength = PKT_NON_FUZZ;
(23)
(24) seqLen ++;
```

Pseudocode 2: Pseudocode of Tr_2.

```
(1) //Tr4:the generation of disaccociation request in Fuzzing
(2) seqPkt[seqLen].flag = PKT_SEND_FUZZ;
(3) seqPkt[seqLen].type = PKT_TYPE_CMD;
(4) seqPkt[seqLen].sec = PKT_FUZZ;
(5) seqPkt[seqLen].Pnd = PKT_FUZZ;
(6) seqPkt[seqLen].Ackr = PKT_FUZZ;
(7) seqPkt[seqLen].Panc = PKT_FUZZ;
(8) seqPkt[seqLen].DA_type = PKT_FUZZ;
(9) seqPkt[seqLen].SA_type = PKT_FUZZ;
(10) seqPkt[seqLen].sqNumber = sqNumber++;
(11)
(12) seqPkt[seqLen].DPan = DPan;
(13) seqPkt[seqLen].SPan = DPan;
(14) seqPkt[seqLen].DAddr = "0000";
(15) seqPkt[seqLen].SAddr = "FFFF";
(16)
(17) seqPkt[seqLen].DAddr_l = "00124B0001301DFF";
(18) seqPkt[seqLen].SAddr_l = "00124B0001301DA2";
(19) seqPkt[seqLen].Remarks = "3";
(20)
(21) seqPkt[seqLen].length = 0;
(22) seqPkt[seqLen].randLength = PKT_NON_FUZZ;
```

Pseudocode 3: Pseudocode of Tr_4.

Table 7: Addresses of sensors.

Sensor	IEEE address
Tested node	00124B00013001DFF
ZFSM-executer	00124B00013001D2A
ZFSM-listen	00124B00013001D48

a computer through a serial port. The IEEE address of the three nodes is listed in Table 7. The type of the nodes is CC2530 produced by Texas Instruments (TI) and the stack employed on the chips is Z-Stack.

5.2. Fuzzing Test. The process of association and disassociation is taken for example as the tested object. The rules of mutating inputs under each state are described in Table 8. The state transition from s_2 to s_1 is hard to implement in experiment. Therefore, the rules of Tr_3 are not included in the table.

The testing report of fuzzing-controller showed that there were four times abnormal state transition in the fuzzing test. All of the malformed inputs in the four rounds lead to the state transition from s_2 to s_3 defined in Figure 4. The detail of the four mutations is presented in Table 9. The same point of the four inputs is that the destination addresses are not included in them. Therefore, there may be vulnerability of Z-Stack in dealing with the packets without destination addresses under the state of s_2.

5.3. Performance Test. There are four parameters being used for describing performance of Fuzzing test, which are the total number of testing cases, the rate of VS-Cases, redundant rate of testing cases, and the valid rate of the testing cases. The valid rate of testing cases is calculated based on the rate of VS-Cases and redundant rate of testing cases. The quality of the cases can be described by it in quantitative. The higher the valid rate of the algorithm is, the higher quality of testing cases generated by the algorithm is. In this part, random-based algorithm, structure-based algorithm, and FSM-fuzzing are compared in the four ways.

5.3.1. The Amount of Testing Cases. For a long time, reducing the number of cases in random-based algorithm has become the primary goal of proposing a refined algorithm.

Structure-based algorithm is an example. The key fields such as frame type, addresses, and payload of MAC are specified according the filter rules. The mutated subfield occupies 1 byte in a frame. Therefore, the amount of testing cases covering all possibility is little.

FSM-fuzzing is refined based on structure-based algorithm. Therefore, the number of testing sequences is equal to the amount in structure-based algorithm.

In comparison, the number of Fuzzing test cases in the three algorithms is separately recorded and shown in Figure 8. Abscissa presents the length of the cases L. Ordinate presents the amount of testing cases N. The three curves in the figure belong to the three algorithms separately.

It is obvious that the curve of random-based algorithm exponentially rises. The scale of testing cases in it is the largest among the three algorithms. The amount of structure-bases algorithm is fixed with the increasing of length. The amount of FSM-fuzzing is almost 4 times more than the amount in structure-based algorithm. The multiple is related to process of UIO verifying and regression. Statistical results show that nearly 3 additional inputs are needed in order to inject one malformed case.

Table 8: Rules of mutating inputs.

State	Type	ACK (0/1)	Dest.addr type (Byte)	Src.addr type (Byte)	payload
$\sim Tr_1$	CMD	FUZZ	FUZZ	FUZZ	$\sim 0 \times 0180$
$\sim Tr_2$	CMD	FUZZ	FUZZ	FUZZ	$\sim 0 \times 04$
$\sim Tr_4$	CMD	FUZZ	FUZZ	FUZZ	$\sim 0 \times 0302$

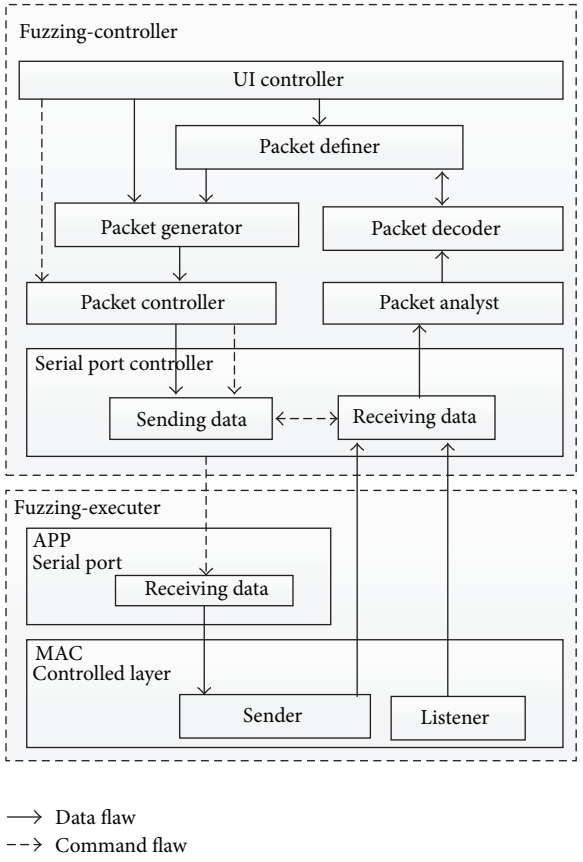

Figure 5: The architecture of ZFSM-Fuzzer.

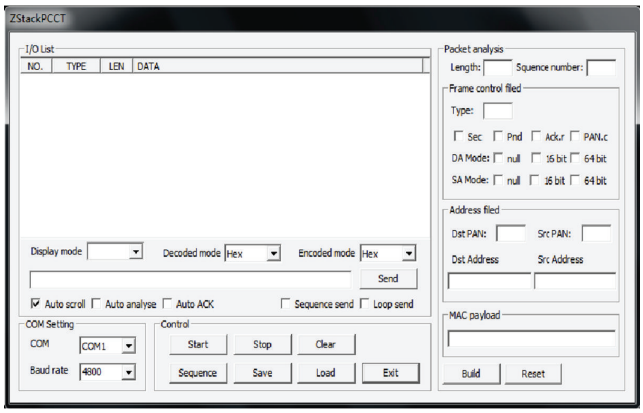

Figure 6: The interface of ZFMS-fuzzer.

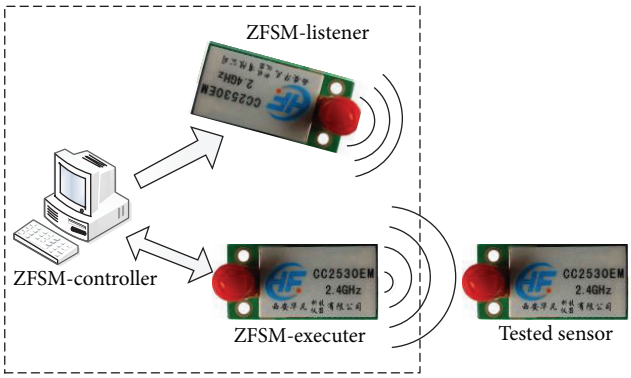

Figure 7: Topology of fuzzing test.

5.3.2. The Rate of VS-Cases. In previous fuzzing tests, reasons for vulnerabilities were hard to locate because the state of sensors was not within the scope of monitoring. Furthermore, the vulnerabilities are always exposed owing to a specific previous state. Therefore, implementing the fuzzing test against a specific state is a solution for locating the reasons for vulnerabilities. However, the abnormal inputs may lead the sensor transfering to an unknown state. Although a defect is exposed during a later test, the previous sensor is also unknown. It is knocked into the previous problem. The rate of VS-Cases is proposed for describing the valid rate of cases in exposing vulnerability.

VS-Case presents the cases generated under a valid state that is specified by testers. For example, s_2 is the specific valid state. The purpose of the fuzzing test is finding the vulnerabilities of the sensor in dealing with abnormal inputs under s_2. Only the cases generated under s_2 are VS-Cases. Some of the fuzzing cases may make the sensor transfer from s_2 to the other states. Until the sensor transfers to s_2 again, the cases generated by fuzzer are invalid.

The comparison of random-based algorithm, structure-based algorithm, and FSM-fuzzing is shown in Figure 9. Abscissa presents the amount of testing cases N. Ordinate presents the amount of VS-Cases N_{VS}.

According to the testing result, almost 100% of the cases in random-based algorithm are VS-Cases. The vulnerabilities exposed in this method are analyzed.

After a long period of tests, the probability of state transition owing to a testing case in structure-based algorithm is 5%. It nearly needs 15 cases to make the sensor transform to other states in the first time. For this reason, the curve of the structure-based algorithm is almost a constant equal to 15.

The rate of VS-Cases of FSM-fuzzing is nearly equal to 15%. The malformed cases generated by FSM-fuzzing are

Table 9: Detail of inputs causing abnormal transitions.

Input	Type	Sec	Pending	ACK	PAN ID compression	Dest addr Mode	Src addr Mode	Dest PAN ID	Dest addr	Src PAN ID	Src.addr	MAC payload
$\sim Tr_2$	CMD	0	1	1	1	16	64	007F	0000	—	00124B0001301DA2	04
$\sim Tr_2$	CMD	0	0	1	0	64	64	007F	00124B0001301DFF	007F	00124B0001301DA2	04
$\sim Tr_2$	CMD	1	1	1	1	—	64	—	—	007F	00124B0001301DA2	04
$\sim Tr_2$	CMD	0	0	1	1	—	64	—	—	007F	00124B0001301DA2	04

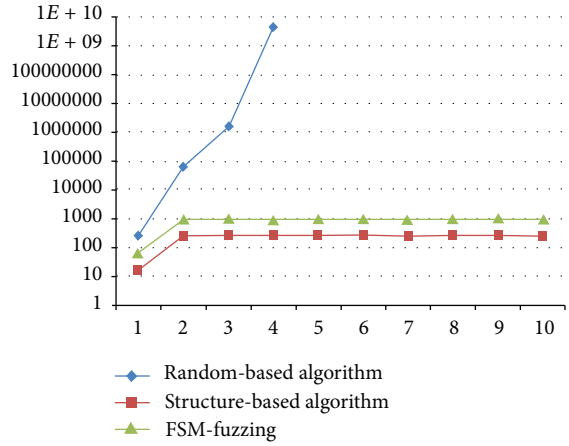

Figure 8: The amount of testing cases.

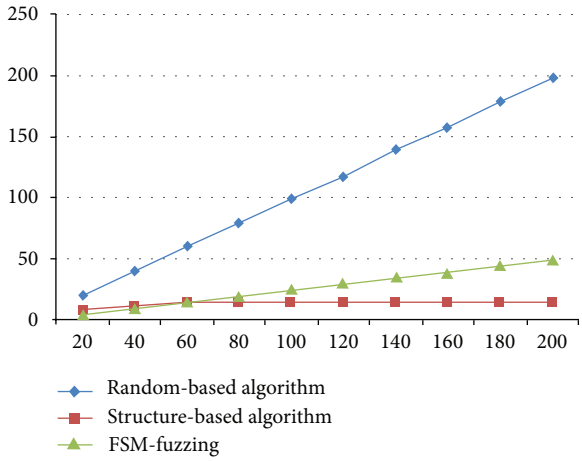

Figure 9: The amount of VS-Cases.

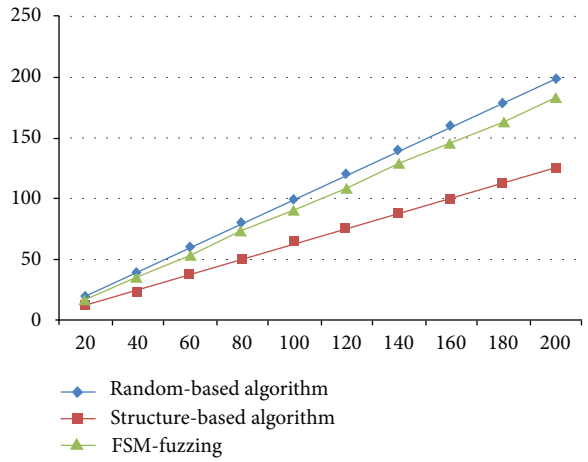

Figure 10: The amount of redundant cases.

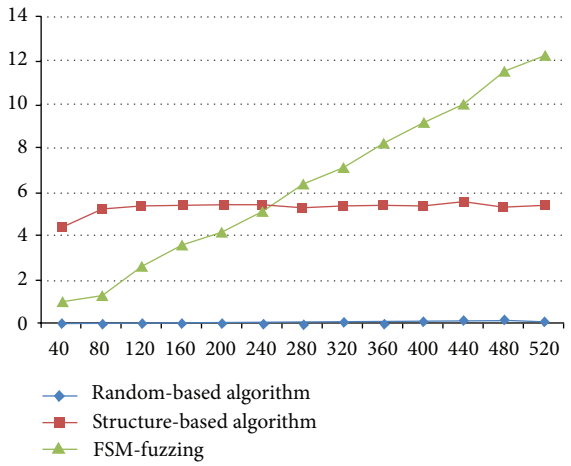

Figure 11: The amount of valid cases.

against the tested state. The non-VS-Cases include UIO cases and regression cases. The result is a match with the previous experiment result.

5.3.3. Redundant Rate of Testing Cases. Redundant rate is the percent of redundant cases. Redundant case refers to the case that is abundant by the tested sensor. According to IEEE 802.15.4, there are a series of rules to filter illegal packets. The abundant cases are not submitted to higher layers of the tested stack. They are redundant for the deep paths of the tested stack. The existence of redundant cases involves the efficiency of fuzzing test greatly. In this experiment, the number of cases that are abundant is counted. The results are shown in Figure 10. In this coordinate, abscissa presents the amount of cases N and ordinate presents the amount of redundant cases N_r.

Almost 100% of the cases generated by random-based algorithm are redundant. The amount of structure-based algorithm is the least. Because all of the cases generated in this method are strictly complied with the filter rules. The redundant rate of FSM-fuzzing is 10% lower than random-based algorithm for the reason that most of the malformed inputs are failure in making the tested sensor transform from one state to another and the contribution of UIO sequences and regression sequences is little. The cases of UIO sequences and regression sequences devote a lot in increasing the rate of redundant cases.

5.4. Valid Rate of Testing Cases. Valid rate of testing cases is a comprehensive parameter to evaluate the quality of the cases generated by a fuzzing algorithm. It is commonly determined by the rate of VS-Cases and redundant rate of testing cases. The formula of valid rate is

$$p = \left(1 - \frac{N_r}{N}\right) \times N_{\text{VS}}. \tag{3}$$

Therefore, the number of valid packets is

$$N_V = p \times N. \tag{4}$$

The number of valid packets of the three algorithms is shown in Figure 11. Abscissa presents the amount of cases N and ordinate presents the amount of valid cases N_V.

Although the rate of VS-Cases of random-based algorithm reaches 100%, the comprehensive valid rate of it is nearly 0. The efficiency of random-based is extremely poor.

The number of valid cases of structure-based algorithm is a constant equaled to 5. No matter how many testing cases the fuzzer tool injects under a specific state, the number of valid cases is not increased. Furthermore, the valid rate is decreased with the increasing of N. Therefore, the state of the tested sensor should be changed manually after receiving 5 inputs. The automation of the fuzzing test is reduced. The efficiency is also negatively affected.

The valid rate of FSM-fuzzing keeps rising with the increasing of the amount of testing cases. And the slope of the curve is constant according to the figure. When the number of cases is beyond 240, the amount of valid cases is larger than the other algorithms. Therefore, the quality of the cases of FSM-fuzzing is the highest among the three algorithms. Overall, the performance of FSM-fuzzing is the best.

6. Conclusion

In this paper, we propose a scheme FSM-fuzzing to generate Fuzzing cases based on FSM. The context of the cases and the state transition of the tested sensor are both taken account into FSM-fuzzing. It is easy for testers to locate and analyze the vulnerabilities of Zigbee. Meanwhile, the process of regression raises the automation level of fuzzing test based on FSM. In order to comply with the filters rules of Zigbee on MAC, the malformed inputs are generated in structure-based algorithm. The methods of generating testing sequences and the malformed packets are explained through taking the process of association and disassociation as an example.

The architecture of fuzzing tool deployed with FSM-fuzzing is designed. The testing sequences are generated by computers so that the burden of the testing node is reduced. According to experiment results, we concluded that there was vulnerability in Z-Stack in dealing with the frames without destination addresses. Thus, FSM-fuzzing is useful in find defects of Zigbee.

We also proposed four parameters for evaluating the performance of FSM-fuzzing, the amount of fuzzing test, the rate of VS-Cases, the redundant rate of cases, and the valid rate of cases. Although the amount of Fuzzing test of FSM-fuzzing is 4 times more than the amount of structure-based algorithm, the amount of VS-Cases of FSM-fuzzing is more than structure-based algorithm. The redundant rate of FSM-fuzzing is almost 1.5 times more than the rate of structure-based algorithm. Yet the comprehensive results show that the valid rate of FSM-fuzzing is more than structure-based algorithm when the number of cases is more over than 240. Overall, efficiency of fuzzing test taken in the context of cases account is more efficient than structure-based algorithm.

In the future work, we will devote to build the FSMs of the other processes and interconnect each part. The fuzzing test is implemented based on the whole FSM of Zigbee. The other deep defects are expected to be found by ZFSM-fuzzer.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by National Natural Science Foundation of China (nos. 61170268, 61100047, and 61272493), International S&T Cooperation Special Projects of China (nos. 2013DFG72850), and The National Basic Research Program of China (973 Program) (Nos. 2012CB724400).

References

[1] Y. Peng, Y. Li, Z. Lu, and J. Yu, "Method for saving energy in Zigbee network," in *Proceedings of the 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '09)*, pp. 1–3, IEEE, September 2009.

[2] S. Farahani, *Zigbee Wireless Networks and Transceivers*, Elsevier, Burlington, Mass, USA, 2007.

[3] Y. Yu, K. Li, W. Zhou, and P. Li, "Trust mechanisms in wireless sensor networks: attack analysis and countermeasures," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 867–880, 2012.

[4] Y. Zhang, X. Li, and J. Liu, "A secure hierarchinal key management scheme in wireless sensor network," *International Journal of Distributed Sensor Networks*, vol. 2012, Article ID 547471, 8 pages, 2012.

[5] D. He, J. Bu, S. Zhu et al., "Distributed privacy-preserving access control in a single-owner multi-user sensor network," in *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM '11)*, pp. 331–335, Shanghai, China, April 2011.

[6] I. K. Rijin, N. K. Sakthivel, and S. Subasree, "Development of an enhanced efficient secured multi-hop routing technique for wireless sensor networks," *Development*, vol. 1, no. 3, pp. 2320–9801, 2013.

[7] Q. Liu and Y. Zhang, "TFTP vulnerability finding technique based on fuzzing," *Computer Communications*, vol. 31, no. 14, pp. 3420–3426, 2008.

[8] H. C. Kim, Y. H. Choi, and D. H. Lee, "Efficient file fuzz testing using automated analysis of binary file format," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 259–268, 2011.

[9] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proceeding of the 29th IEEE International Conference on Computer Communications (INFOCOM '10)*, IEEE, March 2010.

[10] S. Yong Kim, S. Cha, and D.-H. Bae, "Automatic and lightweight grammar generation for fuzz testing," *Computers & Security*, vol. 36, pp. 1–11, 2013.

[11] H. Cheng and Y. Zhang, "Blutooth OBEX vulnerability discovery technique based on fuzzing," *Computing Engineering*, vol. 34, no. 19, pp. 151–156, 2008.

[12] M. Mendonça and N. F. Neves, "Fuzzing Wi-Fi drivers to locate security vulnerabilities," in *Proceedings of the 10th IEEE International Symposium on High Assurance Systems Engineering (HASE '07)*, pp. 379–380, November 2007.

[13] J. Lopez, R. Roman, and C. Alcaraz, "Analysis of security threats, requirements, technologies and standards in wireless sensor

networks," in *Foundations of Security Analysis and Design V*, vol. 5705, pp. 289–338, 2009.

[14] IEEE 802.15.4, "Wireless medium access control (mac) and physical Layer (PHY) specifications for low-rate wireless personal area networks (WPANs)," 2006.

[15] J. Li, Q. Huang, X. Chen, S. S. M. Chow, D. S. Wong, and D. Xie, "Multi-authority ciphertext-policy attribute-based encryption with accountability," in *Proceedings of the 6th International Symposium on Information, Computer and Communications Security (ASIACCS '11)*, pp. 386–390, March 2011.

[16] Y. Xiao, H.-H. Chen, B. Sun, R. Wang, and S. Sethi, "MAC security and security overhead analysis in the IEEE 802.15.4 wireless sensor networks," *Eurasip Journal on Wireless Communications and Networking*, vol. 2006, Article ID 93830, 2006.

[17] P. S. Mogre, K. Graffi, M. Hollick, and R. Steinmetz, "A security framework for wireless mesh networks," *Wireless Communications and Mobile Computing*, vol. 11, no. 3, pp. 371–391, 2011.

[18] K. McCusker and N. E. O'Connor, "Low-energy symmetric key distribution in wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 3, pp. 363–376, 2011.

[19] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," *Ad Hoc Networks*, vol. 1, no. 2-3, pp. 293–315, 2003.

[20] S. Shin Jung, M. Valero, and A. Bourgeois, "Attacking beacon-enabled IEEE 802.15.4 networks," in *Social-Informatics and Telecommunications Engineering*, Lecture Notes of the Institute for Computer Sciences, pp. 253–271, Springer, Singapore, 2010.

[21] A. Lahmadi, C. Brandin, and O. Festor, "A testing framework for discovering vulnerabilities in 6LoWPAN networks," in *Proceedings of the 8th IEEE International Conference on Distributed Computing in Sensor Systems*, pp. 335–340, 2012.

[22] S. Peng, B. Cui, and R. Jia, "A novel vulnerability dectection method for zigbee MAC layer," in *The 6th International Conference on Network and System Security*, 2012.

[23] B. Cui, S. Liang, and S. Peng, "An dynamic detection technology for IEEE 802.15.4 protocol based on node replication," *Journal of Tsinghua University (Science and Technology)*, vol. 52, no. 10, pp. 194–200, 2012.

[24] C. Holler, K. Herzig, and A. Zeller, "Fuzzing with code fragments," in *Proceedings of the USENIX Security*, pp. 445–458, 2012.

[25] B. P. Miller, L. Fredriksen, and B. So, "Study of the reliability of UNIX utilities," *Communications of the ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[26] Z. Wu, H. Wang, and L. Sun, "Survey of fuzzing," *Application Research of Computers*, vol. 27, no. 3, pp. 829–832, 2010.

[27] http://www.immunitysec.com/resources-freesoftware.shtml.

[28] http://peachfuzzer.com/.

[29] A. Simao and A. Petrenko, "Checking completeness of tests for finite state machines," *IEEE Transactions on Computers*, vol. 59, no. 8, pp. 1023–1032, 2010.

[30] A. Takeshi Endo and A. Simao, "Evaluating test suite characteristics, cost, and effectiveness of FSM-based testing methods," *Information and Software Technology*, vol. 55, no. 6, pp. 1045–1062, 2013.

[31] G. Shu, Y. Hsu, and D. Lee, "Detecting communication protocol security flaws by formal fuzz testing and machine learning," *IFIP International Federation for in Formation Processing*, vol. 2008, pp. 299–304, 2008.

[32] B. Huang and Q. Wen, "An automatic fuzz testing method designed for detecting vulnerabilities on all protocol," in *Proceedings of the International Conference on Computer Science and Network Technology (ICCSNT '11)*, pp. 639–642, December 2011.