Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Trust Aware Ensemble Secure Protocol in IoT using ZigBee Optimized Mode

Pushpa Latha Thumma[1], Dr. Prasadu Peddi[2]

[1]Research scholar, Shr iJJT University, Jaipur
[2]Professor, Shri JJT University Jaipur

Zigbee is extensivelyused for its efficiencyin using wireless resources the Internet of Things. Recently, Manufacturer-manufactured IoT devices were recently affected because of serious flaws in Zigbee standards. Testing security of Zigbee protocols is getting more important. However, applying the existing methods of vulnerability detection like fuzzing to the Zigbee protocol isn't an easytask.Handling of low-level hardwareevents is a major issue. In the case of those usingthe Zigbee protocol which is communicated via a radio channel most of the existing tools for fuzzing protocols don't have the right execution environment. ZigBee ensures the security ofsensitive data on networks and devices by implementing the Advanced Encryption Standard (AES) which is the best encrypted symmetric key block While AES is currently considered highly secure, there is ongoing research and anticipation that it may become vulnerable to security lapses in the future as computing power increases and new vulnerabilities are discovered. Additionally, symmetric cryptosystems, like AES, face challenges with authentication and key management, particularly in large-scale systems. To address security concerns in wireless sensor networks, especially those related to Zigbee communications, we have developed Z-Fuzzer. Z-Fuzzer is a program designed to detect security weaknesses in Zigbee protocols and can be deployed on any compatible device. In this research, we utilized Z-Fuzzer, which employs an embedded commercial system simulator. This simulator comes equipped with pre-defined hardware and accessories that interact with the fuzzy engine to accurately mimic the operations of the Zigbee protocol. Z-Fuzzer utilizes code coverage techniques to enhance the quality of tests it provides. In our research, we compared Z-Fuzzer's simulation platform with other recent protocol fuzzing tools, such as BooFuzz and Peach Fuzzer. The results demonstrate that Z-Fuzzer achieves higher code coverage within Z-Stack, a widely adopted Zigbee protocol implementation. Furthermore, Z-Fuzzer identified more flaws compared to BooFuzz or Peach Fuzzer, despite conducting fewer tests. They found three significant flaws, with severity scores ranging from 7.5 to 8.2. The analysis evaluates the effectiveness ofthesystemacrossvariousaspects,includingsecurityfeatures,communications,and computational costs.

Keywords: ZigBee protocol, advanced encryption standard (AES), internet of things (IoT), cryptography, Zigbee protocol, fuzzing.

1. Introduction

Today, the use of the Internet has become an essential part of everyday life. The idea of a comprehensive system administration phase based upon the relation to thearticle's brilliant content is officially a huge leap. The supposed Internet of Things (IoT) technologyis evolving into a foundation for a modern world where humans and their objectscanbecoordinated[1].Theadvancementswillallowthecreationofnew apps as well as administrations that are likely to make use of the abundance of both physical and digital substances. The IoT model relies on available communication technology, such as Wi-Fi Bluetooth, Zig-Bee, among others to name some [2]. But, establishing a suitable and desired IoT network, which is based upon these different technologies is a difficult and laborious problem.

IoT devices have the ability to communicate to multiple communication protocols simultaneously within the IoT network. Some devices require a higher speed connectioninordertocarryouttheirfunctionseffectively.However,therearedevices with fewer resources that need connectivity that is low in usage of resources. It is the Zigbee protocol [5] an omnipresent wireless protocol that helps reduce costs forpower and memory and has seen widespread use for home manufacturing, automated systems, and in manufacturing. Zigbee is very dependent on hardware setup since it uses the radio channel to transmit data instead of the internet. The Zigbee protocol, which offers a range of security measures and options inside the Zigbee network, is defined by the Zigbee Alliance. The shortcomings and imperfections present in different Zigbee stacks have been exposed by recent study [37, 46, 53]. The flaws might be used to launch a denial-of-service attack or send remote executable malware to a Zigbee-enabled Philips lighting system. Although the latest iterations of the Zigbee protocol have addressed some security concerns, the scientific community is not giving the protocol much attention. Thus, it is crucial and very useful to discover security vulnerabilities in Zigbee protocols.

In order to help IoT developers assess security issues associated with the Zigbee protocol used for their applications We have developed Z-Fuzzer an open protocolthat is device independent. simulation system that allows the test of fuzzing in Zigbee protocols. Z-Fuzzer comprises two primary parts: a fuzzing engine as well as a test harness performing protocols stack. Z-Fuzzer initially aims at creating superior test scenarios that adhere to protocol packet format requirements. Thus, we use coverage- based feedback to enhance grammar-based fuzzing[25] and generate high-qualitytest cases, which we subsequently rank in order of increasing difficulty. The coverage measure is employed within AFL [58] in our estimation of code coverage.

We created Z-Fuzzer and tested its efficacyin identifying securityweaknesses. We've picked two of the most effective protocols fuzzing software tools, BooFuzz [41] and Peach [38], that will be our test tools to test this technique of fuzzing. Experts in the field Sulley [21] as well as Peach [38] use protocol fuzzers. BooFuzz replaces them. Peach is an extremely popular commercial protocol fuzer. In addition to Peach is Zigbee protocols simulation system we use BooFuzz and Peach. We use the common Zigbee protocol, Z-Stack (29), for fuzzing in order to test them against Z-Fuzzer.

Three of the vulnerabilities mentioned were given As of the publication date of this article, CVE IDs with extremely high CVSS scores [20] were still being examined. Our research provides insight into the vulnerabilities in the Zigbee protocol whenused in a software simulation environment without actual hardware access. What we provide is as follows:

Proposed Work:

Z-Fuzzeroffersadevice-independentfuzzingplatformthatsupportsZigbeeprotocols. It achievesthisbyrunningtheprotocolstack on afullsoftwaresimulatorandutilizing an intermediate server to facilitate communication between the fuzzing engine and simulator.Byleveragingcodecoveragestatisticstoprovidefeedbackandimprovethe test-generation process of the grammar-based fuzzing approach, Z-Fuzzer enhances the quality of tests generated.

Compared to tools like BooFuzz or Peach, Z-Fuzzer can potentially achieve higher code coverage with fewer tests. During testing, six previously unknownvulnerabilities were discovered within Z-Stack, three of which have been classified as high-severity CVE IDs.

Problemstatement:

It's not an easy undertaking to apply fuzzing methods to Zigbee protocols. In the beginning,fuzzersmaybreak compilerinspection whenaddinginstruments tocodein the Zigbee protocols source code in the event that it is they are available. Zigbee protocol providers generally design protocols for specific embedded devices using a certain toolchain for development [2425. Regarding the availability of protocols, it is common for vendors to implement checks against compilers within their protocols to block compilers that aren't on the list of accepted, specifically those that are general- purpose compilers (e.g., GCC and LLVM) employed by numerous coverage-guided fuzzyers.

The remaining portion of the paper is set up as follows. The duties of fuzzing testing and Zigbee security analysis are discussed in Section 2. The history of the Zigbee protocol and the foundations for current protocol fuzzing are given in Section 3. Z- Fuzzer's idea and implementation are described in Section 4. In Section 5, Z-Fuzzer's performance is analysed in comparison to two cutting-edge protocols for fuzzers. The drawbacks of the existing model and the possibility of future improvements are examinedinSection6.Section7endswitha fewremarksandideasformore research.

2. Related Work

Fuzz testing is a widely employed method for identify security weaknesses with tools like AFL and its extensions such as AFL++ gaining popularity in recent years for automatic security analysis. These tools use algorithms for code coverage to guide their mutation-based approach. However, they do not directly support validation ofthe Zigbee protocol's source code. Due to the diverse nature of Zigbee protocol implementations bydifferent providers usingvarious toolchains, compiler inspections tailored to embedded systems are often required. This presents a challenge for integratingAFL-likefuzzersthatrelyonuniversalcompilerslikeGCCandLLVMfor instrumentation. Z-Fuzzer addresses this challenge by incorporating measurement directly into the source code using the IAR Workbench's embedded compiler, afeature compatible with most Zigbee protocol vendors. Similar to AFL and other coverage-guided fuzzers, Z-Fuzzer utilizes coverage measurement techniques to determine edge coverage.

The security flaws in the the Zigbee protocol may be studied in a number of ways.

Snout[36],createdbyMikulskisandothers,andZ3Sec[37]byMorgneretal.are systems for penetration testing, which uses spoofing and packet replaying to identify Zigbeeflaws and vulnerabilities. Two tools havebeen createdto assess thesecurityof the Zigbee protocol, specifically in relation to embedded devices: IoTcube [31] and beSTORM (48). Zigator, a security analysis tool that examines encrypted Zigbee packets to detect specific jamming and spoofing attempts, has also been proposed by Akestoridis et al. [3]. Security evaluation devices are black-box techniques that examine and alter data flowing via the Zigbee network in order to identify security holes in the protocol.

Devices that use different protocols for communication like WiFi, Bluetooth Low Energy (BLE) as well as Zigbee can coexist in the same space in the Internet of Things system. To address this the researchers recently released novel attackstrategies that use using the Zigbee protocol. It is known that the BLE protocol is employed in Cayre et al. (12) to initiate the first pivoting attack WazaBee that targets Zigbee enabled devices. Bluetooth is a protocol. Since WiFi as well as Zigbee each use their own 2.4 frequency range, Chi and co. [14] identified a variety of new risks that could be exploited through hidden jamming. The researchers showed that attackers might send WiFi packets to interfere with or even completely stop the connectivity between Zigbee devices.

Unlike the other vulnerability-exploiting methods, Z-Fuzzer looks for unknown flaws in the source code of the Zigbee protocol rather than in the active Zigbee network and the several protocols that are part of the same system. The application does not need physical hardware or a specific understanding of the fundamental hardware architecture. Our results from our experiments show vulnerability in the protocol stack's top layer may also result in devastating failures and risks to the IoT applications' capabilities.

Zhao and Co. [10] first presented an IoT mutual authentication solution intended to enhance security inside IoT devices. In addition to implementing mutual authentication for devices and gateways, dynamic password generation, this scheme aimstoimprovethe securityofauthentication. It wasevaluatedusingcalculationsand simulations, showing superior results in terms of efficiency and security when compared to existing systems. However, it's important to note that the scheme comes with a few drawbacks since it is unable to take into account essential properties like the ability to hide, not link ability, and non-traceability.

Discussion

Numerous research studies have examined the possibility of using lightweight cryptographictools,includingbitswitchfunctionsorhashfunctionsXOR andbitwise XOR, to meet various requirements. Yet, these research studies do not take into account the need for solid mutual authentication that is low in computation and communication costs vital to ZigBee devices. As a result, creating a mutual authentication system that works well is a significant task for the Internet of Things community. It's also important to be awarethat most of these solutionsconcentrate on the strength of two-way authentication, ignoring important factors like anonymity, traceability, and link ability, or improving encryption. Our proposed method does not just focus on an option for D2TC as well as D2D authentication within the ZigBee protocolbutisalsoamethodtoimproveencryption.Inaddition,ourstrategy addressescriticalsecurityfeaturessuch astransactionanonymity aswellaslinkability in the absence of it as well and the lack of traceability

3. Background

Zigbee is a wireless standard technology designed to provide the use of low-cost and low-power wireless machine-to-machine (M2M) as well as the Internet of Things (IoT) networks.

Zigbee is an open standard that is optimised for low data rate applications andrequires little power. It is possible, in theory, to combine implementations from several vendors. Interoperability problems plague Zigbee devices in the real world, where they have been altered and adapted by several vendors. As opposed to Wi-Fi networks which connect the endpoints of devices to networks with high speeds, Zigbee supports a much lower rate of data transfer and utilizes an interconnected network protocol that allows users to eliminate hub devices and build an autonomous architecture.

With only four layers, ZigBee is an inexpensive wireless sensor network withminimal power consumption. The control of media access (MAC) layer and the physical (PHY) layer are the first two levels. The MAC layer performs fundamental radio operations and allows communication between two devices through a one hop link. Their compliance with IEEE 802.15.4 standard is guaranteed. Additionally, the networks (NWK) layer was created to handle duties including address management and packet routing. The application (APL) layer, which is the topmost layer, is where a node in the network performs its main function. It also ensures the installation and management of safe connections between nodes [21,23]

In the ZigBee protocol, there are three types of key keys that serve to protect the network: (1) A master secret that is shared by all of the equipment in a ZigBee network is called the network key, or NWK key for short. It offers encryption capabilities and network-wide encryption.

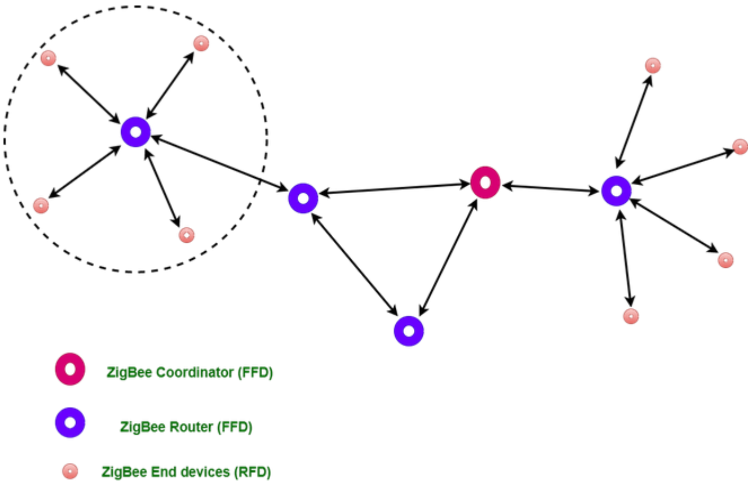

Fig1:Tyoesof ZIgBee Devices

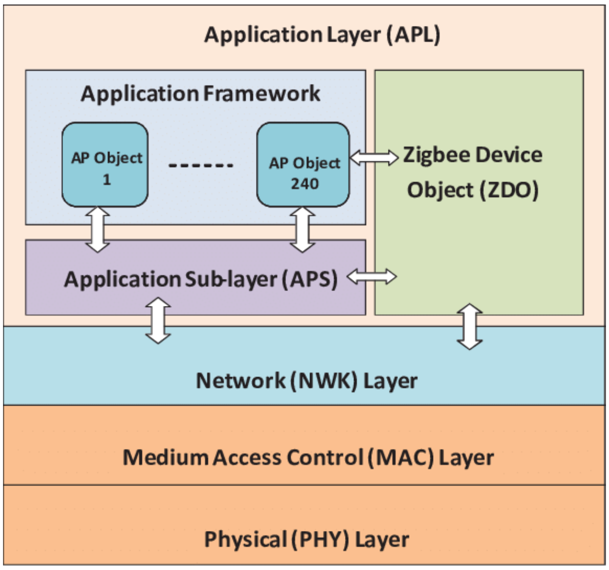

Fig2: ArchitectureofZigBee

NetworkModel

The three main types of topologies seen in ZigBee networks are thought to be star, tree, and mesh topologies. ZigBee networks often employ mesh or star topologies as their primary architecture. Trust Centre (TC) is in charge of managing devices linked to the network and traffic management for mesh-based networks. Endpoints may use the Trust Centre to establish a network connection. Since the TC is aware of every device in the network, it is responsible for both setting up and maintaining thenetwork as well as handling the data of each device. Every ZigBee device must authenticate with the TC during the startup procedure. The purpose of the TC is to produce a key for encryption for the session and to provide authentication for all communications between devices. Thus, every ZigBee network has to be equipped with the TCT. ZigBee devices generally connect to the physical world and other devices within the network.

Mutationin protocolfuzzing tools.

Various black-box protocol fuzzing methods have been proposed and utilized to generate structured and logical packets compliant with network protocol standards. These methods often rely on grammar-based fuzzing techniques. Test inputs are constructed from scratch, adhering to input requirements that specify information formatandintegrityrestrictions.Protocolfuzzersutilizeabstractrepresentationblocks to construct protocol frames, a method sometimes referred to as block-based representation. These blocks consist of basic or nested blocks compatible with the protocol's format. By employing a script for format definition, the fuzzer canrepresent the protocol's message using basic data organized according to their positions.

The fuzzer typically modifies only one message field during each iteration of the fuzzingprocess.Atransformationinstanceofmanystate-of-the-artprotocolfuzzing tools is depicted in Figure 3. Fuzzing a command which triggers various brand- specificcodes resulted inTestCase1.Thefuzzerreverts thisfieldtoitsoriginalvalue of zero after allowing sufficient time for the modification to take effect. It then proceeds to modify the subsequent fields. In the second test scenario, the Command Identifier (CmdID) field is fuzzed, executing a new line in the command using the value 0x05.

Assumptionsand Model Threat

The goal of our paper is to assist IoT developers in better understanding the security risks associated with the Zigbee protocol. By providing insights into these risks, developers can make informed decisions when utilizing Zigbee for constructing IoT applications. To mitigate these security risks, programmers should take necessary actions either before or during the development process.

Inourframework,weassumethattheownerofthe plannedprotocolimplementation's source code is the user utilizing our framework. We also presume that the user possesses all essential information about the embedded device they are using, including details such as the type of CPU, peripheral interrupts, lines of code for peripherals, and processor model. This information is crucial for implementing the stack protocols tested in our simulator.

We are currently focused on generating high-quality Zigbee Cluster Library (ZCL) messages for fuzzing the Zigbee protocol, given their importance for device functionality. Our approach aims to assist Internet of Things (IoT) developers in identifyingimproper ZCLmessages thatcouldleadto protocolstackexecution issues.

Ourthreatmodelassumesthattheuser'sZigbee-basednetworkisatargetforpotential adversaries. It is assumed that attackers have the capability to access the victim's Zigbee network or possess the network key, enabling them to gain unauthorized access through false ZCL messages. This implies that ZCL messages sent to the victim's IoT device may be tampered with or encrypted by the attacker.

4. PROPOSED MODEL AND DESIGN

Wewilltalkabouttheintricaciesof Z-Fuzzer'screationandapplicationinthissection. This article's first section will go through the problems with the Z-Fuzzer idea andtheir fixes. Then, we present the protocol fuzzing algorithm of Z-Fuzzer. We then present the details of how to implement the key elements in Z-Fuzzer to tackle the issues.

Z-FuzzerDesign:

Z-Fuzzer seeks to identify vulnerabilities in the Zigbee protocol without the need for real embedded devices. In other words, to mimic how the Zigbee protocol functions in anappropriatesoftware-basedenvironment.Duetodifferentdeviceandsystemsetups, most existing IoT software for software simulation has problems executing the Zigbee protocol. Different embedded devices operate with different peripheral disruptions to trigger different actions in the Zigbee protocol. Regretfully, embedded simulators lack the necessary expertise to simulate every kind of peripheral interrupt.Additionally,the Zigbee protocol typicallyruns on a hardware-embedded device. It can be modified to suit specific embedded devices that manufacturers require that aren't compatible with all simulators available. So, it is necessary to build a suitable environment for executing software to mimic peripheral interrupts to simulate peripheral interrupts, without taking into account the hardware used in specific embedded devices.

To meet these requirements, we have developed Z-Fuzzer comprising two main elements: a test harness and a mutation engine. The test harness includes an analysis tool for generating coverage reports to calculate cumulative coverage data. It also features an execution engine responsible for running applications that utilize Zigbee protocol stacks using the generated test cases within an emulator.

Utilizing the coverage input, the test harness maintains interesting test cases, leading to an increasein coverageand enabling more effectivefuzzing. Additionally, wehave implemented proxy servers within the execution engine to bridge the gap between simulators and the mutation engine, allowing for testing without the need to join a complete Zigbee network.

Algorithm:

```
AlgorithmforZ-Fuzzer

Input:ProgrammebeingtestedP,inputformscriptS

Output:Thecurrentcumulativeseedsthatcausetheprogramtocrash    code    coverage
current_coverage

1:crash<-0

2:block<-Start(S) 3: high_rated <- 0

4:current_coverage<-0

5: repeat

6:if high_rated is not 0 then sorted<-select(high_rated) seed<- stack(sorted)

if sorted.was_fuzzed then high_rated <- high_rated \ sorted else

test_case<-Choose(block) seed <- stack(test_case)

7: end
```

ImplementationDetails

The fuzzing procedure is a core component of the Z-Fuzzer framework, whichconsistsoffourfundamentalelements: anoffline parser, atestcase generationengine, an execution engine, and a coverage report parser. Additionally, it includes an online proxy server and a stack driver that facilitates the execution of protocols.

The message fields used to create the test case will be mutated depending on their chosen sequence. If a test case that is preferred is selected, the engine is able to skip the transformation of interesting values when there is a skip_mutation flag. In addition, the other basic data representing the spaces below are changed sequentially in sequence. Z-Fuzzer can fuzzy initial information by piecing together the standard test scenario based on the selection order of the basic data. Z-Fuzzer may choose the valuesinasequence tobechangedif theuserhasspecifiedthemessagefieldcontains numbers inside the formatting script. The mutation engine modifies it in additional situations by using the pre-defined fuzzing library.

Once each message field has been altered entirely, the test case that has garnered a great deal of popularity is removed from the corpus. If all of the tests in the text are fuzzed and no favourite test cases remain unfinished, Z-Fuzzer can finish the fuzzing operation entirely. This mutated input then gets transmitted directly to an execution engine to test during runtime.

PerformingaProtocol.

Z-Fuzzer consists of two main components: a stack driver, responsible for executing the Zigbee protocol, and a proxy server. The proxy server serves to facilitate communication between the mutation engine and the simulator via socket connection. It locally hosts the proxy server to enable transmission of mutations to the simulator, while also saving received messages to a file for later processing through the protocol stack.

Additionally, we have developed an engine for stack drivers to configure the system properly by analyzing the source code of the protocol stack intended for implementation.

EVALUATION

The Z-Fuzzer protocol fuzzing technique aims to provide high-quality testing inputs that adhere to the protocol's frames. By comparing it with two widely used advanced protocol fuzzers, BooFuzz and Peach, we demonstrate the uniqueness of Z-Fuzzer. Both BooFuzz and Peach have been extensively utilized in previous research studies.

One significant distinction is that BooFuzz and Peach do not support Zigbee or other IoT wireless protocols, such as Bluetooth. To address this limitation, we integrated Zigbee and Bluetooth protocols into our simulator platform, allowing Z-Fuzzer to interface with both protocols.

When it comes to the evaluation criteria, we carried out our research based on variables suggested by Klees and co. [32]. Particularly, we assessed the vulnerability count and the effectiveness of fuzzing during 24 hours of experimentation with fuzzing. Regarding vulnerabilities, We assessed the efficacyof fuzzers on four fronts: distinct vulnerabilities, The relation between the amount of tests, as well as theamount of vulnerabilities found and also the connection between the level of protection for line as well as the amount of vulnerabilities identified and the vulnerabilities found on real IoT devices. Four variables wereassessed to evaluate the effectiveness of fuzzers. These include the number of tests with a system-specific nature as well as the time of edge coverage of the test, and also the duration of time- to-execute coverage.

Experiment Setup. All of our tests were conducted on a Windows 10 Pro system that has a 32 GB RAM and an Intel(r) CoreTM i7-6700 CPU running at 3.40 GHz. Additionally, version 8.3 of IAR Embedded Workspace for ARM was installed[29]. Texas Instruments created Z-Stack using many project codebases. There is also the source code accessible.

Table1.Totalnumberof crashesandunique vulnerabilities.

Fuzzer	TotalCrashes	UniqueVulnerabilities
BooFuzz	65	3
Peach	8	4
Z-Fuzzer	122	8

UniqueVulnerabilities.

We used the information in the call stack to deduplicate the crashes observed. The simulator provides a trace of the call stack for a crash in memory. This includes informationabouttheroutinesthatwereexecuted,linenumbers,particularstatements in the functions, as well as statements in the functions and. For locating a crash, we analyzed the name of the function the line number and memory address. Stack hashing may thus result in bug overcounting. To prevent overcounting, we closely inspecttheoriginalcodeofeveryvulnerability. Inoursituation,wepersonallyreview the function call trail in the source code of each vulnerability is to ensure that there are no instances of the overcount of vulnerabilities. The experiment's findings are shown in Table 1. It demonstrates that compared to the other two fuzzers, the Z- Fuzzer can identifya greater number of distinct vulnerabilities and crashes. Bycross- checking, all vulnerabilities have also been verified. BooFuzz is limited to reproducing a single vulnerability. Peach Fuzzer and Z-Fuzzer test scenarios can replicate any vulnerability. Every vulnerability was reported to both the vendors and CVE. CVE IDs and CVSS ratings (7.5-8.2) are high for three of them.

```
Psudeo code typedefstruct
{
unit8discComplete; unit8 cmdtype;4 unit8 numCmd; unit8 *pCmdID;
}zclDiscoverCmdsCmdRsp_t;
saticvoid
{
if9pDiscoverRSPcmd!=Null)
{
unit8 i;
pDiscovercmd<-discComplete=*pBuff++ for(i=0,i<numcmds; i++)
{
PDiscover->pCmdID[i]=*pBuf++;
}
}
return((void*)
}
```

Most protocol suppliers substitute common functions of the C library with their own customised APIs. It is because embedded devices have limited computing and memorypower.ThismakesitdifficulttorunallC-standardAPIlibraries,likePC software. This customization can also pose security threats, in addition to the bugsthat may exist within the implementation of the protocol. The Zigbee vendors are responsible for anyvulnerabilities in the protocol. Securityproblems can be mitigated or prevented entirely by the vendor's response to reported security issues. IoT developers might not know about these potential problems until after they have completed the production. We were also motivated by this observation to suggest Z- Fuzzer to developers so they could be aware of the potential Zigbee stack issues earlier in the development process. This way, they would have the opportunityto take the necessary actions and avoid these problems before waiting for feedback from the protocol vendor.

Fuzzing Performance

Each fuzzer was subjected to a series of experiments in order to determine the variation of the test case number, execution time, edge and line coverage over time. All fuzzers were given the same format of protocol frames. The fuzzing processes were therefore initialized using the same protocol frame. The fuzzers use the script to format the frames and generate the test cases using the predefined or user-specific fuzzing dictionary.

Fuzzer	Total # of Unique Test Cases	Test Case Exec. Time (ms)	LineCoverage	EdgeCoverage
Z-Fuzzer	62458	541	98776.21%	78184.23%
BooFuzz	17542	645	85472.31%	68772.31%
Peach	19845	684	83069.12%	58564.21%

Weinitiallyexaminedtheuniquenessproducedbyeachfuzzer.Table3illustratesthat Z-Fuzzer can generate six times as many distinct test cases as the other two fuzzers. Testcases arecategorizedbasedonthe ZCLheader'sfield commandidentification,as specified in the Zigbee Protocol Standard, enabling differentiation of various fuzzers in terms of test case creation. While BooFuzz or Peach can generate only 35 of the 308 distinct test case types generated by Z-Fuzzer.

Manytest cases also provide incremental coverage, making it desirable to retain them for further mutations. For example, if the field framework controls are kept at 0x08,Z-Fuzzer may generate the message seen in Figure 3.

Additionally, test case execution times were assessed on average. Z-Fuzzer's overall execution time is 541 milliseconds per test, as indicated in Table 3's third column. This represents a 14.9% and 13.4% increase in execution speed compared to Peach and BooFuzz, respectively.

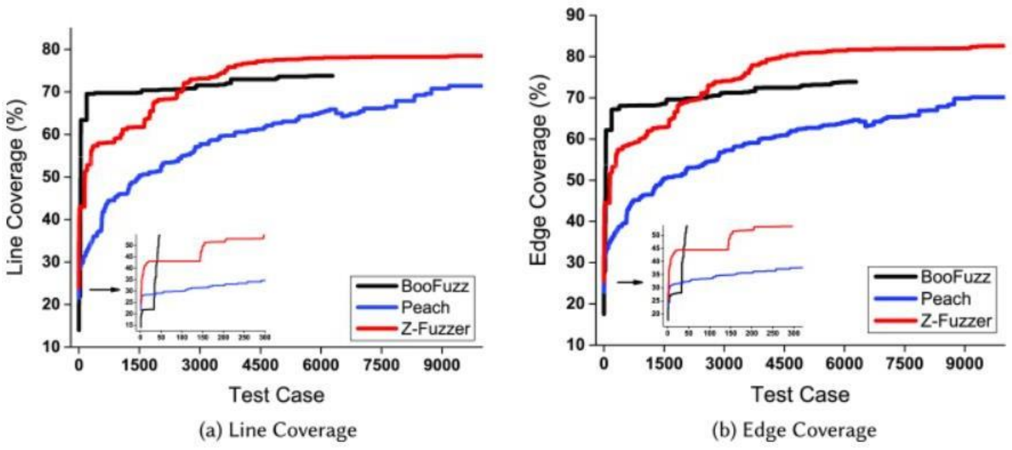

Fig. 9. Fuzzers achieved line coverage and edge cover over 10 runs. The median test case number is represented by the x-axis.

5. Conclusion

Fuzzer has better code coverage than BooFuzz or Peach, thanks to the test cases generated based on coverage. With the feedback on coverage, the interesting values can be recorded and used to guide the test case generation process. This allows for more detailed code to be accessed. ZCL has a number of functions that handle processing the upper-level application object's payload. It's possible that in order to execute more intricate code inside those methods, they need that the test case pass certain condition checks. BooFuzz and Peach ignore the message values that maymeet such dependency constraints during their mutation processes. Z-Fuzzer, on the other hand, can deduce such a relationship from runtime feedback. All of the previous fields and mutant primitives are kept for further fuzzing, satisfying these conditions. This covers more code and edges.

We have already implemented the BooFuzz fuzzing engines on our simulation platform. In the future, In our effort, we want to include additional simulation settings. This also applies to the proxy servers that other fuzzers make use of. To implementthe Zigbee protocol, they may transmit their test cases through the Internet. We can also apply our test-case generating engine to existing embedded fuzzers, as HALucinator[16]. After that, they may test Zigbee protocols and get acquainted with the Zigbee session format.

References

1. Orfanos, V.A.; Kaminaris, S.D.; Papageorgas, P.; Piromalis, D.; Kandris, D. A Comprehensive Review of IoT Networking Technologies for Smart HomeAutomation Applications. J. Sens. Actuator Netw. 2023, 12, 30. [CrossRef]
2. D.SPA.16 Rev. 1.0, IST-2002-507932 ECRYPT; YearlyReport on Algorithms and Keysizes (2005). European Network of Excellence in Cryptology. 2006.
3. Traore, M. Analyse des biais de RNG pour les mécanismes cryptographiques et applications

industrielles. In Cryptographie et Sécurité [cs.CR]; Université Grenoble Alpes: Saint-Martin-d'Hères, France, 2022; p. 190.
4. Cryptographic Key Length Recommendations. Available online: http://www.keylength.com (accessed on 1 January 2020).
5. FIPS 197; Advanced Encryption Standard (AES). National Institute of Standards and Technologies: St. Andrews, SC, USA, 2001.
6. NIST Special Publication 800-57 Draft; Recommendation for KeyManagement. National Institute of Standards and Technologies: St. Andrews, SC, USA, 2006.
7. NIST Special Publication 800-67 Version1; Recommendation for the Triple DataEncryption Algorithm (TDEA) Block Cipher. National Institute of Standards and Technologies: St. Andrews, SC, USA, 2004.
8. Lee, J.Y.; Lin, W.C.; Huang, Y.H. A lightweight authentication protocol for internet of things. In Proceedings of the 2014 International Symposium on Next- Generation Electronics (ISNE), Tao-Yuan, Taiwan, 7–10 May 2014; pp. 1–2.
9. Kulkarni, S.; Ghosh, U.; Pasupuleti, H. Considering security for ZigBee protocol using message authentication code. In Proceedings of the 2015 Annual IEEE India Conference (INDICON), New Delhi, India, 17–20 December 2015; pp. 1–6.
10. Zhao, G.; Wang, X.; Si, J.; Long, X.; Hu, T. A novel mutual authentication scheme for internet of things. In Proceedings of the 2011 International Conference on Modelling,IdentificationandControl(ICMIC),Shanghai,China,26–29June2011;pp. 563–566.
11. Chu, F.; Zhang, R.; Ni, R.; Dai, W. An improved identity authentication scheme for internet of things in heterogeneous networking environments. In Proceedings of the 2013 Sixteenth International Conference on Network-Based Information Systems, Gwangju,RepublicofKorea,4–6September2013;pp. 589–593.
12. Gaikwad, P.P.; Gabhane, J.P.; Golait, S.S. 3-level secure Kerberos authentication for smart home systems using IoT. In Proceedings of the 2015 First International Conference on Next Generation Computing Technologies (NGCT), Dehradun, India, 4–5 September 2015; pp. 262–268.
13. Ashibani, Y.; Kauling, D.; Mahmoud, Q.H. A context-aware authentication framework for smart homes. In Proceedings of the 2017 IEEE Thirtieth Canadian ConferenceonElectricalandComputerEngineering(CCECE),Windsor,ON,Canada, 30 April–3 May 2017; pp. 1–5.
14. Mishra, D.; Vijayakumar, P.; Sureshkumar, V.; Amin, R.; Islam, S.H.; Gope, P. Efficient authentication protocol for secure multimedia communications in IoT- enabled wireless sensor networks. Multimed. Tools Appl. 2018, 77, 18295–18325. [CrossRef]
15. Alshahrani, M.; Traore, I. Secure mutual authentication and automated access control for IoT smart home using cumulative Keyed-hash chain. J. Inf. Secur. Appl. 2019, 45, 156–175. [CrossRef]
16. Chang,C.-C.;Le,H.-D.AProvablySecure,Efficient,andFlexibleAuthentication Scheme for Ad hoc Wireless Sensor Networks. IEEE Trans. Wirel. Commun. 2015, 15, 357–366. [CrossRef]
17. Alalak, S.; Ahmed, Z.; Abdullah, A.; Subramiam, S. Aes and ecc mixed for zigBee wireless sensor security. Int. J. Electron. Commun. Eng. 2011, 5, 1224–1228.
18. Mirsaraei, A.G.; Barati, A.; Barati, H. Asecure three factorauthentication scheme for IoT environments. J. Parallel Distrib. Comput.2022, 169, 87–105. [CrossRef]
19. Gong, B.; Zheng, G.; Waqas, M.; Tu, S.; Chen, S. LCDMA: Lightweight Cross- domain Mutual Identity Authentication Scheme for Internet of Things. IEEE Internet Things J. 2023. [CrossRef]
20. Amor, A.B.; Jebri, S.; Abid, M.; Meddeb, A. A secure lightweight mutual authentication scheme in social industrial IoT environment. J. Supercomput. 2022. [CrossRef]
21. Yang, B. Studyon security of wireless sensor network based on ZigBee standard. In Proceedings of the International Conference on Computational Intelligence and Security, Beijing, China, 11–14 December 2009; pp. 426–430.

22. Qianqian, M.; Kejin, B. Security analysis for wireless networks based on ZigBee. In Proceedings of the 2009 International Forum on Information Technology and Applications, Chengdu, China, 15–17 May 2009; pp. 158–160.
23. Misic, J.; Misic, V. Wireless Personal Area Networks: Performance, Interconnections and Security with IEEE, 2008, 802.15.4; John Wiley & Sons Ltd.: Hoboken, NJ, USA, 2008.
24. Varghese, J.M.; Rao, N.; Varghese, V.T. Asurveyof thestateofthe art in ZigBee. Int. J. Cybern. Inform. 2015, 4, 145–155. [CrossRef]
25. Haque, K.F.; Abdelgawad, A.; Yelamarthi, K. Comprehensive Performance Analysis of ZigBee Communication: An Experimental Approach with XBee S2C Module. Sensors 2022, 22, 3245. [CrossRef] [PubMed]
26. Amin, R.; Islam, S.H.; Biswas, G.P.; Khan, M.K.; Leng, L.; Kumar, N. Design of an anonymity-preserving three factor authenticated key exchange protocol forwireless sensor networks. Comput. Netw. 2016, 101, 42–62. [CrossRef]
27. Gope, P.; Hwang, T. A realistic lightweight anonymous authen-tication protocol forsecuringreal-timeapplicationdataaccessinwirelesssensornetworks. IEEETrans. Ind. Electron. 2016, 63, 7124–7132. [CrossRef]
28. Li, X.; Ibrahim, M.H.; Kumari, S.; Sangaiah, A.K.; Gupta, V.; Choo, K.K.R. Anonymous mutual authentication and keyagreement scheme for wearable sensors in wireless body area net-works. Comput. Netw. 2017, 129, 429–443. [CrossRef]
29. Wu, F.; Li, X.; Xu, L.; Kumari, S.; Karuppiah, M.; Shen, J. A lightweight and privacy-preserving mutual authentication scheme for wearable devices assisted by cloud server. Comput. Electr. Eng. 2017, 63, 168–181. [CrossRef]
30. Ankur, G.; Meenakshi, T.; Jamil, S.T.; Aakar, S. A lightweight anonymous user authentication and key establishment scheme for wearable devices. Comput Netw. 2019, 149, 29–42.
31. Fotouhi, M.; Bayat, M.; Das, A.K.; Far, H.A.N.; Pournaghi, S.M.; Doostari, M. A lightweight and secure two-factor authentication scheme for wireless body area networks in health-care IoT. Comput. Netw. 2020, 177, 107333. [CrossRef]
32. Dave Aitel. 2002. The advantages of block-based protocol analysis for security testing. Immunity Inc., 105 (February2002), 106.
33. Dimitrios-Georgios Akestoridis, Madhumitha Harishankar, Michael Weber, and Patrick Tague. 2020. Zigator: Analyzing the security of Zigbee-enabled smart homes. In Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and MobileNetworks(WiSec’20).AssociationforComputingMachinery,NewYork,NY, 77–88. https://doi.org/10.1145/3395351.3399363
34. Zigbee Alliance. August 5, 2015. Zigbee Specification. https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee- specification.pdf.
35. ZigbeeAlliance.January14,2016.ZigbeeClusterLibrarySpecification. https://zigbeealliance.org/wp-content/uploads/2019/12/07-5123-06-zigbee-cluster- library-specification.pdf.
36. Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein,JaimeCochran,ZakirDurumeric,J.AlexHalderman,LucaInvernizzi,Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher,Chad Seaman, Nick Sullivan, Kurt 37. Thomas, and Yi Zhou. 2017. Understandingthe Mirai botnet. In Proceedings of the 26th USENIX Security Symposium (USENIX Security’17). USENIX Association, 1093–1110. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.
38. Vaggelis Atlidakis, Roxana Geambasu, Patrice Godefroid, Marina Polishchuk, and Baishakhi Ray. 2020. Pythia: Grammar-Based Fuzzing of REST APIs with Coverage-guided Feedback and Learning-based Mutations. arxiv:2005.11498 [cs.SE].
39. Greg Banks, Marco Cova, Viktoria Felmetsger, Kevin Almeroth, Richard Kemmerer,and

Giovanni Vigna.2006. SNOOZE: Toward astatefulnetworkprotocol fuzzer. In Proceedings of the 9th International Conference on Information Security (ISC'06). Springer, Berlin, 343–358.
40. Fabrice Bellard.2005.QEMU,a fastand portabledynamictranslator. InUSENIX Annual Technical Conference, FREENIX Track, Vol. 41. USENIX Association, 46.
41. Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, and Géraldine Marconato. 2021. WazaBee: Attacking Zigbee networks by diverting Bluetooth low energy chips. In Proceedings of the 51st IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'21). IEEE, Piscataway, NJ.
42. Peng Chen and Hao Chen. 2018. Angora: Efficient fuzzing by principled search. In Proceedings of the 39th IEEE Symposium on Security and Privacy (S&P'18). IEEE, Piscataway, NJ, 711–725. DOI: https://doi.org/10.1109/SP.2018.00046
43. Zicheng Chi, Yan Li, Xin Liu, Wei Wang, Yao Yao, Ting Zhu, and Yanchao Zhang. 2020. Countering cross-technology jamming attack. In Proceedings of the13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'20). Association for Computing Machinery, New York, NY, 99–110.
44. Catalin Cimpanu. April 12, 2018. Over 65,000 Home Routers Are Proxying Bad Traffic for Botnets, APTs. https://www.bleepingcomputer.com/news/security/over-65-000-home-routers-are-proxying-bad-traffic-for-botnets-apts/.
45. Abraham A. Clements, Eric Gustafson, Tobias Scharnowski, Paul Grosen, David Fritz, Christopher Kruegel, Giovanni Vigna, Saurabh Bagchi, and Mathias Payer. 2020. HALucinator: Firmware re-hosting through abstraction layer emulation. In Proceedings of the 29th USENIX
46. Security Symposium (USENIX Security'20). USENIX Association, Berkeley, CA, 1201–1218.
https://www.usenix.org/conference/usenixsecurity20/presentation/clements. IEEE C/LM LAN/MAN Standards Committee. 2020. IEEE Standard for Low-RateWireless Networks.
47. Baojiang Cui, Shurui Liang, Shilei Chen, Bing Zhao, and Xiaobing Liang. 2014.A novel fuzzing method for Zigbee based on finite state machine. International Journal of Distributed Sensor Networks 10, 1 (2014), 762891.
48. Baojiang Cui, Ziyue Wang, Bing Zhao, and Xiaobing Liang. 2016. CG-Fuzzing:A comprehensive fuzzy algorithm for ZigBee. International Journal of Ad Hoc and Ubiquitous Computing 23, 3–4 (2016), 203–215.
49. National Vulnerability Database. 2021. NVD Vulnerability Severity Ratings. https://nvd.nist.gov/vuln-metrics/cvss.
50. Ganesh Devarajan. 2007. Unraveling SCADA Protocols: Using Sulley Fuzzer. Defon 15 Hacking Conference.
51. Bo Feng, Alejandro Mera, and Long Lu. 2020. P2IM: Scalable and hardware-independentfirmwaretestingviaautomaticperipheralinterfacemodeling.In Proceedings of the 29th USENIX Security Symposium (USENIX Security'20). USENIX Association, 1237–1254.
52. Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. 2020. AFL++: Combiningincrementalstepsoffuzzingresearch.InProceedingsofthe14thUSENIX Workshop on Offensive Technologies (WOOT'20). USENIX Association.
53. Patrice Godefroid, Adam Kiezun, and Michael Y. Levin. 2008. Grammar-based whitebox fuzzing. In Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'08). Association for Computing Machinery, New York, NY, 206–215.
54. Serge Gorbunov and Arnold Rosenbloom. 2010. Autofuzz: Automated network protocol fuzzing framework. International Journal of Computer Science and Network Security (IJCSNS) 10, 8 (2010), 239.
55. Fortune Business Insights. July 2019. Internet of Things (IoT) Market Analysis.https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-

market-100307.

56. Texas Instruments. 2006. Z-Stack 3.0 Developer's Guide. https://software-dl.ti.com/simplelink/esd/plugins/simplelink_zigbee_sdk_plugin/1.60.00.14/docs/zigbee_user_guide/html/zigbee/developing_zigbee_applications/z_stack_developers_guide/ z-stack-overview.html.
57. TexasInstruments.2013.CC2538.http://www.ti.com/product/CC2538.
58. George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. 2018. Evaluating fuzz testing. In Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS'18). Association for Computing Machinery, New York, NY, 2123–2138.
59. David Lodge. 2016. Steal Your Wi-Fi Key from Your Doorbell? IoT WTF!https://www.pentestpartners.com/security-blog/steal-your-wi-fi-key-from-your- doorbell-iot-wtf/.
60. Zhengxiong Luo, Feilong Zuo, Yuheng Shen, Xun Jiao, Wanli Chang, and Yu Jiang. 2020. ICS protocol fuzzing: Coverage guided packet crack and generation. In Proceedings of the 57th Annual Design Automation Conference (DAC'20). ACM/IEEE, Piscataway, NJ, 1–6. DOI: