**文本

描述已自动生成**

**Analysing IoT protocol implementation based on Fuzzing**

by

Fengyu Chen

School of Information Technology and Electrical Engineering,

University of Queensland.

Submitted for the degree of

Master of Information Technology.

13th June 2022

Fengyu Chen

fengyu.chen@uqconnect.edu.au

13th June 2022

Prof Amin Abbosh

Acting Head of School

School of Information Technology and Electrical Engineering

The University of Queensland

St Lucia QLD 4072

Dear Professor Abbosh,

In accordance with the requirements of the Degree of Master of Information Technology in the School of Information Technology and Electrical Engineering, I submit the following thesis entitled

“Analyze IoT protocol implementation based on Fuzzing”

The thesis was performed under the supervision of Dr. Guangdong Bai. I declare that the work submitted in the thesis is my own, except as acknowledged in the text and footnotes, and that it has not previously been submitted for a degree at the University of Queensland or any other institution.

Yours sincerely

\_\_\_ {Your signature here once document is printed} \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fengyu Chen

To...

# Acknowledgments

I would like to thank my supervisor, Dr. Guangdong Bai, for his guidance through each stage of the process. His expertise was invaluable in formulating the research questions and methodology, insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to thank Dr. Yanjun Zhang, for her valuable guidance throughout my studies. She provided me with the tools that I needed to choose the right direction and successfully complete my project.

Finally, I would particularly like to acknowledge all UQ Trust Lab members, for their wonderful collaboration and patient support.

# Abstract

As IoT becomes popular in many domains, the security of IoT protocol has become an important research topic. IoT protocols include application-layer protocols like CoAP and MQTT and communication-layer protocols like Bluetooth and Zigbee. These protocols may be implemented differently by various manufacturers, who may realize their own requirements in the implementations. Nonetheless, failing to follow the protocols’ official specifications may introduce bugs and potential security problems. In this project, our goal is to analyze IoT protocol implementations using a fuzzing framework.

Fuzzing has shown its effectiveness in analyzing binaries, OS kernels, and many other programs, but has not been widely used in analyzing IoT protocols, due to some non-trivial IoT-specific challenges. Our fuzzing framework is proposed to bridge this gap. We take discrepancies and rules from RFCs as the oracles of our testing to address the ineffectiveness of crash-based oracle that current fuzzing techniques heavily rely on. We will apply our methodology to analyze several implementations of MQTT.

**Contents**

[Acknowledgments vi](#_Toc106638895)

[Abstract vii](#_Toc106638896)

[List of Figures ix](#_Toc106638897)

[List of Tables x](#_Toc106638898)

[Introduction 1](#_Toc106638899)

[Literature review 3](#_Toc106638900)

[2.1 Formal analysis 3](#_Toc106638901)

[2.2 Static analysis 4](#_Toc106638902)

[2.3 Fuzzing 4](#_Toc106638903)

[2.4 Fuzzing in IoT 6](#_Toc106638904)

[2.5 MQTT Security 6](#_Toc106638905)

[Theory 7](#_Toc106638906)

[Methodology 8](#_Toc106638907)

[4.1 IoT Protocol Analysis 8](#_Toc106638908)

[4.2 Test case generation 10](#_Toc106638909)

[4.3 Fuzzing 13](#_Toc106638910)

[4.4 Result analysis 15](#_Toc106638911)

[Results and discussions 17](#_Toc106638912)

[5.1 Protocol analysis 17](#_Toc106638913)

[5.2 Test case generation 17](#_Toc106638914)

[5.3 Fuzzing 17](#_Toc106638915)

[5.2 Result analysis 18](#_Toc106638916)

[Conclusions 19](#_Toc106638917)

[6.1 Summary and conclusions 19](#_Toc106638918)

[6.2 Possible future work 19](#_Toc106638919)

[Bibliography 20](#_Toc106638920)

# List of Figures

# List of Tables

**Chapter 1**

# Introduction

Internet of Things (IoT) has become an essential technology in many areas. It allows various devices to connect to the network and exchange data, such that people can better control, monitor, and utilize environments and devices. IoT has been applied to agriculture [1], industrial control [2], smart home [3, 4], wearable devices [5], and many other areas. However, as IoT becomes popular, security has become one of the main concerns for IoT deployment. In these years, IoT security incidents happen in all trades and professions. For example, Colonial Pipeline paid $4.4 million ransom to the attackers in 2021 [6, 7]. This company operates the main gasoline pipeline along the American east coast. Hackers’ attacks can also target personal properties. In Black Hat USA 2015, researchers showed how they remotely hacked a Jeep and controlled its speed [8]. IoT hacking can also target national defense security. Stuxnet worm took out the key Iranian nuclear facility in 2010 [9]. This is believed to be the first worm that can spy and reprogram industrial systems [10]. From these incidents, we can see those attacks on IoT systems will cause huge influence and loss, for infrastructure, industry, and personal devices.

IoT security greatly relies on the protocols used for secure communications and operations, including applications layer protocols such as MQTT [11], CoAP [12], and AMQP [13] and communication protocols such as Bluetooth [14], Zigbee [15], and 6LoWPAN [16]. Security of IoT protocols relies on implementations in practice. However, protecting implementations of IoT protocols is a non-trivial task. One reason is that IoT protocols work in a special environment. Many IoT devices have limited resources, including power and computation capability. Some security operations like encryption require much energy, which may be simplified or removed by manufacturers. This may result in weak security protection. Another reason is that although most manufacturers operate according to official protocol specification, many of them may add their own requirements. This can be found in their implementations, which cause some security problems [17, 18, 19, 20, 21]. Compared to the rapid development of IoT, the security of IoT protocol implementations has not gotten enough attention. Systematic analysing and guaranteeing are highly desirable.

There mainly exists three methods to analyse the security of IoT protocols and their implementations. The first one is formal analysis [22]. It extracts an abstract model from protocol specification and then uses some mathematical methods to verify security properties. Formal analysis can prove the correctness and security properties of protocols, but it is hard to build the formal link between implementations and abstract models. The second method is static analysis [23]. Its main concept is automatically analysing the programs without executing them. It can locate the vulnerabilities and bugs quickly at the development stage, but it has a high percentage of false positives and ignores runtime interaction with other software components of external environment. The third method is fuzzing [24], which delivers malformed or unexpected inputs to the target program to see if failure happens. Although fuzzing can be blind and random, it can detect problems in run time with nearly no false-positive cases. Programs passing fuzzing tests are convinced to have higher robustness. With proper guidance, fuzzing can be highly efficient.

Fuzzing has shown its effectiveness in analysing binaries [25, 26], OS kernels [27, 28], and many other programs, but it has not been widely used in analysing IoT protocols. In this thesis, we aim to analyse implementations of IoT protocols based on fuzzing. We will build a fuzzing framework to analyse implementations of IoT protocols. Currently, there exist some challenges in IoT protocol analysis and fuzzing. First, many current fuzzing approaches take crash as the oracle of testing, but they may not be suitable in the IoT context. Second, during fuzzing, the state space and possible fuzzing inputs explosively grow. It is a non-trivial task to find bugs in huge state space within limited time and computation power.

**Chapter 2**

# Literature review

## 2.1 Formal analysis

Formal analysis was first proposed to prove the security properties of protocols in cryptography [22]. It has been widely used to prove the correctness of general protocols. There exist some well-developed formal analysis tools, such as PRISM [29], Tamarin [30], ProvVerif [31]. General formal analysis consists of four steps: (1) analysing the protocol’s specification, (2) constructing an abstract model based on the protocol’s specification, (3) translating the abstract model into the model checker input language, (4) checking security properties and giving suggestions for standard changes.

Some researchers have used formal analysis to analyse IoT protocols. Duflot et al. [32] apply formal analysis to the device discovery phase of Bluetooth. Richard et al. [33] perform formal analysis on authentication properties of Bluetooth device pairing based on ProVerif. Kim et al. [34] formally analyse some popular IoT protocols including MQTT and CoAP. Coman et al. [35] analyse some Proof-of-Concept (PoC) attacks targeting LoRaWAN [36], Sigfox [37], and NB-IoT [38] and find some vulnerabilities in protocol specification.

Among these research, formal analysis improves IoT protocols security by proof of functional correctness and security properties in IoT protocols. This proof is strong and reliable because it depends on mathematics. Formal analysis can detect weaknesses and vulnerabilities at the protocol design stage. Finding weaknesses earlier can reduce the impact and give proper instructions to implementations. However, it also has some drawbacks. It is hard to build the formal link between implementation and abstract model because the programming language and library details cannot be fully represented by the abstract model. In addition, problems in compilation and runtime are not covered by it. Another drawback is it requires researchers to have deep professional knowledge, especially in mathematics, which can be difficult for many researchers without related background to apply this method.

## 2.2 Static analysis

Static analysis aims to automatically analyse the behaviors of computer programs without executing them. Its most common usage is source code analysis in the development stage. Many successful tools have been proposed for different programming languages, such as PC-lint Plus [39], Jlint [40], and Coverity [41].

Concerning IoT protocols security, there exist some research using static analysis to analyse the implementations of IoT protocols. Ferrara et al. [42] present an extension based on static analyzer Julia [43] and show its feasibility to detect some of the vulnerabilities mentioned in OWASP IoT Top 10 2018. Celik et al. [44] propose a static analysis system called SOTERIA. It can check properties violation in the model extracted from IoT implementations. Sachidananda et al. [45] propose a static analysis framework for IoT software.

Through analysing the source code of IoT implementations, these works try to find some security issues or properties violations. Using static analysis, detection of bugs and vulnerabilities at the development stage becomes feasible. In addition, it can locate the weaknesses in the code at the exact location. However, large amounts of false positives are inevitable in static analysis. These false positives require manual inspections, which can be a lot of effort. Static analysis usually focuses on one program, without analysing the interactions in runtime between different software. IoT is a complex system with interactions among multiparty. Ignoring these interactions leads to a restricted analysis of IoT protocols.

## 2.3 Fuzzing

Fuzzing is a popular automatic testing method. It generates different inputs and delivers them to the target program to see if failure happens. There exist many general fuzzing tools that can be used to test different programs, such as American Fuzzy Lop (AFL) [46], AFLFast [47], and FairFuzz [48]. Fuzzing has many research directions, and in this work, we mainly focus on guided fuzzing and differential testing. Guided fuzzing determines how to fuzz efficiently. Differential testing gives a new oracle for fuzzing. Both are important methods used in this thesis. We will also introduce some works applying fuzzing to analyse implementations of IoT protocols.

Different from general fuzzing which takes failures like crashes as the oracle, differential testing takes discrepancies as the oracle. It generates inputs and sends them to different programs with similar functions, to see their difference in behaviors and outputs, which is called discrepancies. Here we review some impressive works using differential testing. Chen et al. [52] propose a guided differential testing method called Mucert. They convert the state exploration problem into an optimization problem. They set code coverage as an optimization goal and use the Metropolis-Hastings algorithm [53] to search for better inputs that can increase the code coverage. This approach is proved to achieve higher code coverage with fewer inputs. Tian et al. [54] propose a novel differential testing approach based on the Request for Comments (RFC). They extract rules from the RFC and generate malformed certificates according to the rules. They send these malformed certificates to various SSL/TLS implementations to see if these programs accept them. Their approach shows a better performance combined with Mucert. Petsios et al. [55] propose a differential testing framework called NEZHA. NEZHA proposes a new measure called δ -diversity to represent the path coverage among different PUTs. By keeping all inputs that can improve δ -diversity as seeds to generate new inputs, NEZHA tends to have more significant seeds that can improve fuzzing efficiency. NEZHA finds 26 times more discrepancies than Mucert on average. It also finds some vulnerabilities confirmed by developers. Currently, differential testing is mainly used to test SSL/TLS certificate verification, but it is capable of testing other programs. The discrepancy is not only a novel oracle for fuzzing, but also gives a new view for guiding fuzzing. In this thesis, we will adopt differential testing as one of our main methodologies for IoT protocol analysis.

## 2.4 Fuzzing in IoT

Currently, some works have used fuzzing to test implementations of IoT protocols. Araujo et al. [56] and Hern ́andez et al. [57] propose fuzzing frameworks to analyse implementations of MQTT. Aichernig et al. [58] use automata learning to guide the fuzzing process, which can be another guided fuzzing method. They also target MQTT implementations. They find inconsistencies between five different MQTT brokers and point out the potential vulnerabilities behind them. This finding is like the discrepancies in differential testing. Kwon et al. [59] analyse RabbitMQ with fuzzing. RabbitMQ support AMQP, MQTT, and other IoT protocols. Zeng et al. [60] propose MultiFuzz, a coverage-based multiparty-protocol fuzzer. IoT protocols often involve interactions among multi-party. Ignoring this fact will miss many paths and details in fuzzing. MultiFuzz solves this challenge by collecting information of multiple connections. They apply MultiFuzz to some libraries of MQTT and CoAP. To summarize, most of these works apply fuzzing to implementations of MQTT. CoAP, AMQP, and many other IoT protocols have not gotten enough attention.

## 2.5 MQTT Security

MQTT provides multiple layers of security features. At the network layer, it is possible to connect the device and the MQTT broker by pulling a dedicated line or using a VPN to improve the security of network transmission. At the transport layer, the use of TLS encryption at the transport layer is a good way to ensure security and prevent Man-In-The-Middle Attacks. The client certificate can not only be used as the identity certificate of the device but also can be used to authenticate the device. At the application layer, MQTT also provides the Client Identifier and username and password to authenticate the device at the application layer.

**Chapter 3**

# Theory

Currently, most fuzzers use crash-based oracle, which may ignore some information in IoT context. Crashes can show problems in software engineering, but it is not directly related to the functionalities defined in protocols. Some implementations work well without crashes, but they do not follow the protocols. In addition, errors may be caught by a program without crashes, but it probably shows differences in behavior. These differences can be information to analyse whether the implementations follow the protocol. Ignoring them reduces the effectiveness of fuzzing. To address the drawback of crash-based oracle, in this thesis, we use discrepancies and RFC rules as oracles. Different implementations of one protocol may cause security problems. Finding the discrepancies among implementations of the same protocols helps detect security problems. For example, certificate validation defined by SSL/TLS is realized by different programs such as OpenSSL [61] and GnuTLS [62]. It has been proved that discrepancies found in these implementations lead to potential security problems [54, 52, 55] To find discrepancies and the security problems behind implementations, we generate inputs such as messages defined by protocols and send these inputs to different implementations of the same protocols. These implementations are expected to show discrepancies in behaviors. Through analysing these discrepancies, we can find some security problems and non-compliance with the protocol specification. To process huge number of discrepancies, we use rules from RFCs to automatically analyse discrepancies. RFCs or other protocol specification have been proved to be secure by formal analysis. Non-compliance with the protocol specification may lead to security problems. It is proved to be feasible to use keyword matching and regular expression [54] to extract rules from RFCs. By comparing these discrepancies with the rules, we extracted from RFC, we can judge which implementations do not follow the protocols. Analysing these behaviors, we may find some interesting cases of potential security issues.

**Chapter 4**

# Methodology

We divide our approach into 4 phases. In the first phase, we extract functionalities and rules defined in MQTT Version 5.0 OASIS Standard. In the second phase, we map the functionalities to modules in MQTT libraries, and write test cases for each functionality if it is provided by the library. We also capture the MQTT messages transmitted in test cases as the seeds for mutation and fuzzing. In the third phase, we generate mutants from seeds as the input of fuzzing and we implement fuzz testing of different MQTT implementations. In the final phase, we analyse the discrepancies based on the rules we extracted from RFC.

图示

描述已自动生成

Figure 4.1: workflow

## 4.1 IoT Protocol Analysis

In this phase, we focus on analyzing functionalities and rules defined in MQTT Version 5.0 OASIS Standard which defines the functionalities of MQTT. It also provides many rules and suggestions for implementation. Most of them are adopted by the MQTT libraries, while some of them may be ignored or violated. Before we analyze the libraries of MQTT, we must know the requirements in RFC.

The modules and behavior of the library are implemented based on the functionality described in the RFC. To define the scope of analyzing the MQTT library, we first extract the functionality from the RFC. We manually read the RFC and extract the functionality that should be implemented by the library. We have two reasons to do this manually. One is that learning the RFC is necessary for anyone who wants to analyze the protocol and its implementation. Another reason is that currently, we do not have a reasonable automated tool to help us extract relevant information from RFCs. With functional knowledge, we can write test cases for each library.

The MQTT protocol defines a client and a server. Clients can be divided into publishers and subscribers. A client can be both a publisher and a subscriber, or only one of the two. The MQTT server is more like a broker. Its basic job is to forward the messages sent by the publisher to the relevant subscribers to the subscribers.

The MQTT protocol has a lot of information, and we must read it according to certain guidelines and steps. As a summary of the information, the table of contents for this protocol is very helpful for us to parse the protocol. The table of contents includes seven parts: Introduction, MQTT Control Packet format, MQTT Control Packets, Operational behavior, Security (non-normative), Using WebSocket as a network transport, and Conformance. After we have read this protocol completely, we confirm that the MQTT control package mainly describes the basic functions that an MQTT implementation should meet.

In the part of the MQTT control package, the protocol respectively describes CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT, and AUTH. The MQTT protocol specifies that MQTT clients and MQTT servers each have their own functions.

## 4.2 Test case generation

To perform fuzzing testing, we implement the functionalities in clients and servers with the libraries, so that we can use CoAP messages as input to test the functionalities. Based on the functionalities we extracted from RFC, we map the functionalities to modules in the CoAP libraries' API document and write test cases for them.

Because our test cases are written by humans, it is difficult to control the quality. To realize functionality, different people's realizations will be different. To produce for each library, we follow the principles listed below.

We just call the corresponding functions and provide minimum necessary configurations, ignoring the complex configurations that are not reflected in the captured MQTT messages.

We do not realize the functionalities by ourselves if they are not provided by the library.

We define the same simple application-layer data for each library, to reduce the difference caused by data that is unrelated to MQTT.

By following these principles, the code executed by the clients and servers during fuzzing are mostly the code of the libraries. We just provide the content to be transmitted by the MQTT messages. In this way, we minimize our influence on the execution of libraries and produce fair test cases for fuzzing.

We deploy multiple IoT protocol implementations. The principle of our library selection is that the library is active, has more users, and has higher ratings. According to the above principles, we choose the library in the following list.

Table 4.1: MQTT Implementations chosen

|  |  |  |
| --- | --- | --- |
| Libraries | language | Github start |
| mosquitto | C | 6174 |
| MQTTX | Javascipt | 1749 |
| MQTT.fx | Java |  |
| Vert.x MQTT | Java | 143 |
| vernemq | Erlang | 2741 |
| rabbitmq-server | C | 9501 |
| MQTTnet | C | 2800 |

Some of these libraries provide both client and server. Some offer just one or the other. We implemented all the functions on the MQTT libraries and selected the following test cases.

Table 4.2: Test Case Category

|  |  |  |
| --- | --- | --- |
| Functionality | Corresponding RFC Section | Test case |
| CONNECT | MQTT Version 5 (Section 3.1) | 4 |
| CONNACK | MQTT Version 5 (Section 3.2) | 1 |
| PUBLISH | MQTT Version 5 (Section 3.3) | 3 |
| PUBACK | MQTT Version 5 (Section 3.4) | 1 |
| PUBREC | MQTT Version 5 (Section 3.5) | 1 |
| PUBREL | MQTT Version 5 (Section 3.6) | 1 |
| PUBCOMP | MQTT Version 5 (Section 3.7) | 1 |
| SUBSCRIBE | MQTT Version 5 (Section 3.8) | 1 |
| SUBACK | MQTT Version 5 (Section 3.9) | 1 |
| UNSUBSCRIBE | MQTT Version 5 (Section 3.10) | 1 |
| UNSUBACK | MQTT Version 5 (Section 3.11) | 1 |
| PINGREQ | MQTT Version 5 (Section 3.12) | 1 |
| PINGRESP | MQTT Version 5 (Section 3.13) | 1 |
| DISCONNECT | MQTT Version 5 (Section 3.14) | 1 |
| AUTH | MQTT Version 5 (Section 3.15) | 1 |

For CONNECT control package, we have 4 test cases for CONNECT because, CONNECT will have different version versions, resulting in different package structures, and connect has the Properties option. There are Property Length, Session Expiry Interval, Receive Maximum, Maximum Packet Size, Topic Alias Maximum, Request Response Information, Request Problem Information, User Property, Authentication Method, Authentication Data in the properties. Our testcases have version 3 with and without property, version 5 with and without property. If the package owns the property, make every field of the property filled.

For CONNACK control package, we only have one test case. Because the packet structure of CONNACK is fixed, it will not change its own data structure because of the different structure of connect.

For PUBLISH control package, we have three test case. Because there is QoS in the control header of the publish packet, when the QoS is different, the MQTT protocol stipulates that the ACK returned by the broker should be different. Therefore, we generate 3 test cases and set the QoS options to 0, 1, and 2 respectively.

For PUBACK control package, the sender client will receive this packet when QoS is set to 1. This packet will be captured along with the Connect packet.

For PUBREC, PUBREL, PUBCOMP control package, they correspond to QoS 2 delivery part 1, part 2, part 3, respectively. These packets will be captured along with the Connect packet.

For SUBSCRIBE control package, we only have one test case. Although the MQTT protocol specifies many ways to subscribe to topics, they are all implemented by using different text formats of the Topic field. The package structure remains unchanged. The same is true for UNSUBSCRIBE.

For SUBACK and UNSUBACK control package, their package structure is also unique because the package structure of SUBSCRIBE and UNSUBSCRIBE is unique.

For PINGREQ and PINGRESP control package, they serve as a function to verify whether the client and server are alive, and the structure is simple. So there is only one test case here.

For DISCONNECT control package, it will only be sent by the client, and the server will not return any control packets after receiving it.

Because usually the client initiates the communication first and then the client responds after receiving the information. Therefore, when creating a test case, we generally issue the method from the client side and expect the server side to make a normal response. We call this process a full communication and we replenish the full communication as seeds. The following figure shows a simple subscription-publish model.

图表

描述已自动生成

Figure 4.2: MQTT subscription-publish communication model

## 4.3 Fuzzing

After we realize the test cases, we can capture the MQTT messages transmitted between the clients and servers in test cases as the seeds for our fuzzing. Our program will take a normal pcap packet as input, mutate it once and send it directly to the server. The program ends until every field of the original package has been mutated.

At the same time because except CONNECT, all other functions need to be used after a successful connection. When testing these functions, we will first make sure that we have successfully connected.

For CONNECT control package, we send a CONNECT packet and expect the server to return a CONACK.

图示

描述已自动生成

Figure 4.3: Connect communication

For PUBLISH control package, we first send a CONNECT packet to make sure the connection is successful and then send our publish packet. During this part of the experiment, the connection we created earlier was interrupted. We write the program so that every time the link is disconnected, the reconnection is performed and then the fuzzing is performed. Because there are many reconnections when mutate some fields, if we execute too many reconnections when mutating a file, it will skip mutating the field. We fuzz each QoS setting individually.

图示

描述已自动生成

Figure 4.4: Publish QoS:0 communication

图示

描述已自动生成

Figure 4.5: Publish QoS:1 communication

图示

描述已自动生成

Figure 4.6: Publish QoS:2 communication

For SUBSCRIBE control package, we first test whether the subscriber and the broker can connect normally. We also design a publish-subscribe model. We define that first the publisher is connected to the broker normally. The subscriber then connects with the broker to subscribe to the topic. Then the publisher sends the message topic filed as the topic subscribed by the subscriber. We expect subscribers to receive this message. Subscribers should not receive any messages for different topics. We also tried multiple publishers and multiple subscribers.

图示

描述已自动生成

Figure 4.7: Subscription mode

图示

描述已自动生成

Figure 4.8: Subscribe-publish mode (same topic)

图示

描述已自动生成

Figure 4.9: Subscribe-publish mode (different topic)

## 4.4 Result analysis

The results obtained from fuzzing are analyzed using the rule information obtained from the RFC. In addition to comparing the differences between them and the RFC, compare the differences between the different implementations.

At this stage, we will use the packet format proposed by MQTT. The package structure can help us locate which field has an exception. The rule information helps us know if this exception is already mentioned in the protocol.

For CONNECT control package,

表格

描述已自动生成

Figure 4.2: CONNECT packet Fixed Header

表格

描述已自动生成

Figure 4.3: Protocol Name bytes

表格

描述已自动生成

表格

描述已自动生成

**Chapter 5**

# Results and discussions

## 5.1 Protocol analysis

When we used the functional information parsed from RFC to generate test cases on our deployed IoT protocol implementation, we found that the information we parsed was not complete enough. For the same function, different libraries give different APIs. Some API required details are ignored when we parse.

During the analysis of the results, the potential problems of manual RFC parsing are also exposed. There are some exceptions that we think are not mentioned in the RFC but are just our human parsing mistakes and misunderstandings.

## 5.2 Test case generation

Because we generate test cases based on the results of human parsing of RFCs, we cannot guarantee that our test cases cover all cases. Especially for complex functions, our test cases may not cover all possible combinations. For example, for the CONNECT package, it contains a lot of information, especially in Properties, including Property Length, Session Expiry Interval, Receive Maximum, Maximum Packet Size, Topic Alias Maximum, Request Response Information, Request Problem Information, User Property, Authentication Method, Authentication Data. Not each of these parameters must be included, but their direct mutual combination requires more testing.

## 5.3 Fuzzing

Because we're using a mutation-by-test approach, it's hard to be sure that that approach is very efficient. We may need a scoring mechanism to help us determine the quality of seeds, so that we can get effective feedback from each test and strengthen our test efficiency.

## 5.2 Result analysis

Because we perform the result analysis manually, and our result analysis uses the information obtained by our manual analysis of RFC, it will inevitably lead to some errors.

**Chapter 6**

# Conclusions

## 6.1 Summary and conclusions

We analyse multiple implementations of MQTT. Some unusual behavior was found. Some of the challenges faced in this project. Because we read the RFCs manually, we cannot guarantee that our test cases cover all functionality, and when analyzing test results, there may be misunderstandings. Even so, the abnormal behavior of the MQTT implementation can still be found using this method.

## 6.2 Possible future work

There are many IoT protocols, corresponding to many RFCs. As IoT develops, we cannot deny that more IoT protocols may appear in the future. To better use fuzzing to analyze IoT protocol implementations, we need an automated tool to help us read RFC. At present, NLP technology is expected to solve this problem.

# Bibliography

[1] J. Lee, “The top 5 iot applications for agriculture,” https://blog.particle.io/the-top-5-iot-applications-for-agriculture/, 2021, accessed: 2021-10-22.  
[2] S. Ranger, “What is the iiot? everything you need to know about the industrial internet of things,” https://www.zdnet.com/article/what-is-the-iiot-everything-you-need-to-know-about-the-industrial-internet-of-things/, 2019, accessed: 2021-10-22.  
[3] “Alexa,” https://developer.amazon.com/en-AU/alexa, accessed: 2021-10-06.  
[4] “Device access — google developers,” https://developers.google.com/nest/device-access, accessed: 2021-10-06.  
[5] B. Marr, “The biggest wearable technology trends in 2021,” https://www.forbes.com/sites/bernardmarr/2021/03/05/the-biggest-wearable-technology-trends-in-2021/?sh=1e7382a73092,2021, accessed: 2021-10-22.  
[6] “Colonial pipeline hit by communications outage,” https://www.cbc.ca/news/business/colonial-pipeline-outage-1.6031067, 2021, accessed: 2021-10-22.  
[7] S. Morrison, “How a major oil pipeline got held for ransom,” <https://www.vox.com/recode/22428774/ransomeware-pipeline-colonial> darkside-gas-prices, 2021, accessed: 2021-10-22.  
[8] A. Drozhzhin, “Black hat usa 2015: The full story of how that jeep was hacked,” https://www.kaspersky.com.au/blog/blackhat-jeep-cherokee-hack-explained/9493/, 2015, accessed: 2021-10-22.  
[9] M. Saunokonoko, “How stuxnet worm took out key iranian nuclear facility in 2010,” https://www.9news.com.au/world/how-stuxnet-cyberattack-took-out-natanz-nuclear-facility-in-iran/37694f90-c2e1-454c-9597-e7de8f54e495, 2021, accessed: 2021-10-22.  
[10] R. McMillan, “Siemens: Stuxnet worm hit industrial systems,” https://www.computerworld.com/article/2515570/siemens--stuxnet-worm-hit-industrial-systems.html, 2010, accessed: 2021-10-22.  
[11] “Mqtt: The standard for iot messaging,” https://mqtt.org, 2021, accessed: 2021-10-06.  
[12] “The constrained application protocol (coap),” <https://datatracker.ietf.org/doc/html/rfc7252>, 2014, accessed: 2021-10-06.  
[13] “Oasis advanced message queuing protocol (amqp) version 1.0,” http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf, 2012, accessed: 2021-10-06.  
[14] “Bluetooth technology website,” https://www.bluetooth.com/, accessed: 2021-10-06.  
[15] “Zigbee - connectivity standards alliance,” https://zigbeealliance.org/solution/zigbee/, accessed: 2021-10-06.

[16] “Ipv6 over low power wpan (6lowpan) - documents,” https://datatracker.ietf.org/wg/6lowpan/documents, accessed: 2021-10-06.  
[17] “CVE-2020-12883.” Available from MITRE, CVE-ID CVE-2020-12883., May 15 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-12883  
[18] “CVE-2020-12884.” Available from MITRE, CVE-ID CVE-2020-12884., May 15 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-12884  
[19] “CVE-2020-12885.” Available from MITRE, CVE-ID CVE-2020-12885., May 15 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-12885  
[20] “CVE-2020-12886.” Available from MITRE, CVE-ID CVE-2020-12886., May 15 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-12886  
[21] “CVE-2020-12887.” Available from MITRE, CVE-ID CVE-2020-12887., May 15 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-12887  
[22] D. Dolev and A. Yao, “On the security of public key protocols,” IEEE Transactions on Information Theory, vol. 29, no. 2, pp. 198–208, 1983.  
[23] B. Chess and G. McGraw, “Static analysis for security,” IEEE Security Privacy, vol. 2, no. 6, pp. 76–79, 2004.  
[24] M. Sutton, A. Greene, and P. Amini, Fuzzing: brute force vulnerability discovery. Pearson Education, 2007.  
[25] Y. Li, B. Chen, M. Chandramohan, S.-W. Lin, Y. Liu, and A. Tiu, “Steelix: Program-state based binary fuzzing,” in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 627–637. [Online]. Available: https://doi.org/10.1145/3106237.3106295  
[26] S. Dinesh, N. Burow, D. Xu, and M. Payer, “Retrowrite: Statically instrumenting cots binaries for fuzzing and sanitization,” in 2020 IEEE Symposium on Security and Privacy (SP), 2020, pp.1497–1511.  
[27] S. Schumilo, C. Aschermann, R. Gawlik, S. Schinzel, and T. Holz, “kafl: Hardware-assisted feedback fuzzing for {OS} kernels,” in 26th {USENIX} Security Symposium ({USENIX} Security 17), 2017, pp. 167–182.  
[28] D. R. Jeong, K. Kim, B. Shivakumar, B. Lee, and I. Shin, “Razzer: Finding kernel race bugs through fuzzing,” in 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 754–768.  
[29] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in Proc. 23rd International Conference on Computer Aided Verification (CAV’11), ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.  
[30] S. Meier, B. Schmidt, C. Cremers, and D. Basin, “The tamarin prover for the symbolic analysis of security protocols,” in Computer Aided Verification, N. Sharygina and H. Veith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 696–701.  
[31] B. Blanchet, Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif. Cham: Springer International Publishing, 2014, pp. 54–87. [Online]. Available: https://doi.org/10.1007/978-3-319-10082-1 3

[32] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker, “A formal analysis of bluetooth device discovery,” International Journal on Software Tools for Technology Transfer, vol. 8, no. 6, pp. 621–632, Nov 2006. [Online]. Available: https://doi.org/10.1007/s10009-006-0014-x  
[33] C. Richard and S. Vitaly, “Formal analysis of authentication in bluetooth device pairing,” in Proceedings of LICS/ICALP workshop on foundations of computer security and automated reasoning for security protocol analysis (FCS-ARSPA ’07), 2007.  
[34] J. Y. Kim, R. Holz, W. Hu, and S. Jha, “Automated analysis of secure internet of things protocols,” in Proceedings of the 33rd Annual Computer Security Applications Conference, ser. ACSAC 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 238–249. [Online]. Available: https://doi.org/10.1145/3134600.3134624  
[35] F. L. Coman, K. M. Malarski, M. N. Petersen, and S. Ruepp, “Security issues in internet of things: Vulnerability analysis of lorawan, sigfox and nb-iot,” in 2019 Global IoT Summit (GIoTS), 2019, pp. 1–6.  
[36] “Lorawan technology,” https://www.lora-alliance.org, accessed: 2021-10-06.  
[37] “Sigfox technology,” https://www.sigfox.com/en, accessed: 2021-10-06.  
[38] “Nb-iot technology,” https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot, accessed: 2021-10-06.  
[39] “Gimpel software - the leader in static analysis for c and c++ with pc-lint plus,” https://gimpel.com, accessed: 2021-10-06.  
[40] “Artho software - jlint,” http://artho.com/jlint/, accessed: 2021-10-06.  
[41] “Coverity scan - static analysis,” https://scan.coverity.com, accessed: 2021-10-06.  
[42] P. Ferrara, A. K. Mandal, A. Cortesi, and F. Spoto, “Static analysis for discovering iot vulnerabilities,” International Journal on Software Tools for Technology Transfer, vol. 23, no. 1, pp. 71–88, Feb 2021. [Online]. Available: https://doi.org/10.1007/s10009-020-00592-x  
[43] “Julia static analyzer,” https://juliasoft.com, accessed: 2021-10-06.  
[44] Z. B. Celik, P. McDaniel, and G. Tan, “Soteria: Automated iot safety and security analysis,” in 2018 USENIX Annual Technical Conference (USENIX 18), 2018, pp. 147–158.  
[45] V. Sachidananda, S. Bhairav, and Y. Elovici, “Over: Overhauling vulnerability detection for iot through an adaptable and automated static analysis framework,” in Proceedings of the 35th Annual ACM Symposium on Applied Computing, ser. SAC ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 729–738. [Online]. Available: https://doi.org/10.1145/3341105.3373930  
[46] “american fuzzy lop (2.52b),” https://lcamtuf.coredump.cx/afl/, accessed: 2021-10-06.  
[47] M. B ̈ohme, V.-T. Pham, and A. Roychoudhury, “Coverage-based greybox fuzzing as Markov chain,” IEEE Transactions on Software Engineering, vol. 45, no. 5, pp. 489–506, 2019.  
[48] C. Lemieux and K. Sen, “Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage,” in Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ser. ASE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 475–485. [Online]. Available: https://doi.org/10.1145/3238147.3238176

[49] Y. Chen, Y. Jiang, F. Ma, J. Liang, M. Wang, C. Zhou, X. Jiao, and Z. Su, “Enfuzz: Ensemble fuzzing with seed synchronization among diverse fuzzers,” in 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 1967–1983.  
[50] M. Rajpal, W. Blum, and R. Singh, “Not all bytes are equal: Neural byte sieve for fuzzing,” arXiv preprint arXiv:1711.04596, 2017.  
[51] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, “Guided, stochastic model-based gui testing of android apps,” in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 245–256. [Online]. Available: https://doi.org/10.1145/3106237.3106298  
[52] Y. Chen and Z. Su, “Guided differential testing of certificate validation in ssl/tls implementations,” in Proceedings of the 2015 10th Joint Meeting on Foundations of Software  
Engineering, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 793–804. [Online]. Available: https://doi.org/10.1145/2786805.2786835  
[53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” The journal of chemical physics, vol. 21, no. 6, pp. 1087–1092, 1953.  
[54] C. Tian, C. Chen, Z. Duan, and L. Zhao, “Differential testing of certificate validation in ssl/tls implementations: An rfc-guided approach,” ACM Trans. Softw. Eng. Methodol., vol. 28, no. 4, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3355048  
[55] T. Petsios, A. Tang, S. Stolfo, A. D. Keromytis, and S. Jana, “Nezha: Efficient domain-independent differential testing,” in 2017 IEEE Symposium on Security and Privacy (SP), 2017, pp. 615–632.  
[56] L. G. Araujo Rodriguez and D. Macˆedo Batista, “Program-aware fuzzing for mqtt applications,” in Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, ser. ISSTA 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 582–586. [Online]. Available: https://doi.org/10.1145/3395363.3402645  
[57] S. Hern ́andez Ramos, M. T. Villalba, and R. Lacuesta, “Mqtt security: A novel fuzzing approach,” Wireless Communications and Mobile Computing, vol. 2018, p. 8261746, Feb 2018. [Online]. Available: https://doi.org/10.1155/2018/8261746  
[58] B. K. Aichernig, E. Muˇskardin, and A. Pferscher, “Learning-based fuzzing of iot message brokers,” in 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), 2021, pp. 47–58.  
[59] S. Kwon, S.-J. Son, Y. Choi, and J.-H. Lee, “Protocol fuzzing to find security vulnerabilities of rabbitmq,” Concurrency and Computation: Practice and Experience, vol. 33, no. 23, p. e6012, 2021, e6012 CPE-20-0718.R1. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6012  
[60] Y. Zeng, M. Lin, S. Guo, Y. Shen, T. Cui, T. Wu, Q. Zheng, and Q. Wang, “Multifuzz: A coverage-based multiparty-protocol fuzzer for iot publish/subscribe protocols,” Sensors, vol. 20, no. 18, 2020. [Online]. Available: https://www.mdpi.com/14248220/20/18/5194