Project Proposal

Student: Fengyu Chen

Student Number: 45982862

Supervisor: Guangdong Bai

Abstract

As IoT becomes popular in many domains, 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, we aim to build a fuzzing framework to analyse implementations of IoT protocols. Fuzzing has shown its effectiveness in analysing binaries, OS kernels, and many other programs, but has not been widely used in analysing 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. To help our fuzzer explore the state space efficiently, we propose a guided strategy to evaluate the inputs and combine fuzzers with different heuristics. To evaluate our methodology, we propose an IoT-specific evaluation that compares our methodology with general fuzzers and other IoT fuzzers. We have applied our methodology to analyse several implementations of CoAP.

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], smarthome [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 that 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, in-

cluding 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 is 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. Third, under the IoT context, only applying existing evaluations are not enough. IoT-specific evaluation is required to prove the advantages of our methodology over general fuzzers and other IoT fuzzers. To address these challenges, we propose three Research Questions (RQs).

• RQ1: How to design the oracle to fit into security protocol analysis?

• RQ2: How to guide the fuzzing process to improve efficiency?

• RQ3: How to evaluate our method in terms of metrics specific to IoT protocol?

Currently, we are working on fuzzing CoAP as a proof of concept of our methodology. We have built a prototype of fuzzer and used it to test implementations of CoAP. We obtain some preliminary findings, which confirm that our method is feasible. This work is going to submit to ISSTA 2022.

Literature Review

Our literature review consists of three parts, corresponding to three methods of analysing the security of IoT protocols and their implementations: formal analysis, static analysis, and fuzzing. We introduce the general concepts of each method and review some works applying each method to IoT protocols analysis.

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 researches, 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 can not 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.

Static Analysis

Static analysis aims to automatically analyse the behaviour 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 researches 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.

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 of them are important methods used in this thesis. We will also introduce some works applying fuzzing to analyse implementations of IoT protocols.

Differential Testing

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 behaviours 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 per-

formance 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.

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 similar to 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.

Motivation and Problem Statement

As IoT become popular in many areas, IoT security and IoT protocol security have become essential research topics. There mainly exists three methods to analyse the security of IoT protocols and their implementations, which are formal analysis, static analysis, and fuzzing. We use fuzzing to address the analysis of IoT protocol implementation for three reasons. First, fuzzing can find problems in run time. In this project, we focus on the implementations of IoT protocols. Finding run-time problems is more relevant to our objective than proving the correctness of protocols’ specifications or security properties. Second, fuzzing gives nearly no false-positive cases. Through analysing the results, we can efficiently identify security problems. Finally, with guided fuzzing strategies and differential testing, we are confident to build a well-structured fuzzing framework that can be applied to various IoT protocols.

Based on the above discussion, the objective of this thesis is to analyse implementations of IoT protocols based on fuzzing. We will build a fuzzing framework to analyse implementations of IoT protocols. The research questions (RQs) considered in this thesis are listed below. We discuss our methodology for each research question in the next section.

图示

描述已自动生成

Methodology

In this section, we discuss the methodology to solve each research question. These research questions determine how we build our fuzzing framework for IoT protocol implementations.

How to design the oracle to fit into security protocol analysis?

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 behaviours. Through analysing these discrepancies, we can find some security problems and non-  
compliance with the protocol specification. To process huge amount 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 behaviours, we may find some interesting cases of potential security issues.

How to guide the fuzzing process to improve efficiency?

During our fuzzing process, the state explosion happens. How to efficiently find more security issues in the huge state space? In order to solve this problem, we design a solution containing three steps.

First, we propose the formula below to score the input. I represents the input. An input can get a higher score for finding discrepancies or increasing coverage. If an input has been mutated to create some new inputs, its score will be divided by the number of inputs it generates. With this feedback evaluation formula, we can identify the input that tends to find discrepancies or increase coverage. α, β and γ will be determined by our experiment results. This formula is inspired by [51].

Second, when we start a new iteration of fuzzing, our fuzzer tends to select the inputs with higher scores. This preference can be expressed by a probability distribution function, which will be determined by experiment results.

Third, after we choose the input as a seed for mutation, we combine different fuzzers with various heuristics to mutate the seed. Heuristics means that a fuzzer has its own blind areas because they are designed with different strategies. Through sharing the seed among different fuzzers, fuzzers can reach some paths that can not be covered by all of them when they work alone. As a result, the coverage is increased. This method has been proved to be effective by [49]

How to evaluate our method in terms of metrics specific to IoT protocol?

Evaluation of this thesis should consider two points. One is the comprehensiveness of our analysis of IoT protocols. IoT protocols can be long and complex. How to prove we perform a deep and comprehensive analysis of them can be a challenge. Another one is the advantages of our approach beyond other fuzzing approaches. Currently, there exist many effective general fuzzing tools and other approaches targeting fuzzing IoT protocols. We need to show the advantages of our methodology beyond other fuzzing approaches to prove the significance of our work. We plan to prove these two points with two evaluation methods.

We use functionality coverage to evaluate the comprehensiveness of our work. IoT protocols contain many functionalities and definitions. For example, CoAP [12] defines functionalities like message transmission. It also defines a message format, which determines message parsing. To provide a deep analysis of IoT protocols, we have to cover as many functionalities as possible. Otherwise, the comprehensiveness of our analysis will be reduced. In contrast, general fuzzing approaches usually focus on one functionality of the program. Sometimes they may cover multiple functionalities, but what they care about is code or path coverage, rather than functionality coverage. By calculating functionality coverage, we can show how deep and complete our work is. This evaluation helps to prove  
the significance of our work. In addition, in this evaluation, we count and classify the functionalities defined in IoT protocols. These results can also be useful for other works in this area.

To show the advantages of our approach beyond other fuzzing approaches in IoT protocols analysis, we plan to evaluate our methodology comprehensively by comparison. We choose to compare with some state-of-the-art general fuzzing approaches and IoT protocols fuzzers. First, we compare the number of vulnerabilities and bugs found by us and others. We use known vulnerabilities and bugs as a benchmark. The number of unknown vulnerabilities and bugs found by us and other approaches can also be compared. Second, we compare the code coverage of both methodologies, to show the effectiveness of our fuzzing strategy. This is a fair evaluation because nearly all fuzzers take code coverage as an important evaluation metric. Finally, we show some case studies on real-world protocols to prove that we can find some issues ignored by other fuzzing approaches. With these comparisons, we perform a comprehensive evaluation of the advantages of our approach beyond other fuzzing approaches in IoT protocols analysis.

Preliminary Work

We have started to fuzz implementations of CoAP. We choose CoAP as the first target of this thesis because CoAP is widely used in IoT systems, especially in wireless sensor networks (WSN). We have built a prototype of fuzzer and found some discrepancies and interesting cases.

In our current experiment, we have tested 6 of the most popular and active libraries [63, 64, 65, 66, 67, 68] of CoAP. We build a pair of client and server for each library and record the messages of their communication. The origin messages are seeds for mutation. We mutate the seeds to create more messages to fuzz the servers. We expect to see different behaviours of those servers under the same mutation

We find that for malformed CoAP messages, the 6 libraries show some discrepancies. For example, Token Length (TKL) is a field in the CoAP message. It is a 4-bit unsigned integer that indicates the length of the variable-length Token field. When we change the value of Token Length (TKL) to different values that do not match the real length of Token, we expect that the libraries can detect that data alignment is broken. However, the libraries show different behaviours, including responding with errors and giving no response messages. One of the libraries parses malformed messages normally, making the data after Token corrupted. This finding confirms that discrepancies really exist in implementations of CoAP. Through analysing these discrepancies, we can find some potential security issues.

Another finding is that the same request may result in different responses when we test Californium [63]. We replay the original message we captured in normal communication, and we get the normal response. However, if we send some malformed messages before we replay the original message, we get a different response with different content, which is copyright information of Californium. We think there exists two potential problems behind this behavior. One problem is whether the same inputs causing different outputs are valid in functionality. Another problem is that this may lead to exposing the framework of the server. This may be not secure especially when there exist zero-day vulnerabilities. After we discuss with the developers of Californium, we identify that this is an ambiguity in RFC [12], which means that there is no unique correct definition of how to deal with message duplication. Although this behavior is not identified as a vulnerability or bug, it reminds us  
to pay attention to noise and duplication in fuzzing.

Based on the above findings and fuzzer prototype, we will further improve our fuzzer and test for more functionalities of CoAP. We will also include more libraries of CoAP into our analysis. We expect to submit this work to the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022) by the end of January in 2022

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. We propose 3 research questions (RQs) and corresponding methods. First, we choose discrepancy and rules from RFCs as our oracles, rather than crash and coverage. Second, we propose a formula to evaluate the inputs and combine fuzzers with different heuristics. Third, we evaluate our methodology comprehensively by function coverage and comparison between our approach and other state-of-the-art fuzzing approaches. We have started to analyse the implementations of CoAP as a proof of concept of our thesis. We will further improve our fuzzer framework and solve the RQs in the following years of my Ph.D.

In the first year of enrollment, I did the literature review and formulate the research problems. Currently, I am working on the proof of concept work, which is about fuzzing CoAP. After the confirmation milestone, I will submit a conference paper at the beginning of my second year of enrollment. After that, I will focus on solving RQ1 and RQ2. In the 4th research quarter, I will submit my second conference paper and prepare for the mid-candidature milestone. In my third year of enrollment, I will focus on solving RQ2 and RQ3. My third conference paper will be submitted by the end of the 4th research quarter. In my last two research quarters, I will submit a journal paper and write my final thesis report. The thesis review and submission will be the end of this thesis.