Informatics

Sicherheitstests für das Bluetooth Low Energy Protokoll mit Kombinatorischen Methoden

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Dominik Schreiber, BSc.
Matrikelnummer 01326110

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Dimitrios Simos
Mitwirkung: Priv.-Doz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Manuel Leithner, MSc.

Wien, 25. Juni 2025

Dominik Schreiber   Dimitrios Simos

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Informatics

Security Testing for the Bluetooth Low Energy Protocol using Combinatorial Methods

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Dominik Schreiber, BSc.
Registration Number 01326110

to the Faculty of Informatics

at the TU Wien

Advisor: Priv.-Doz. Dr. Dimitrios Simos
Assistance: Priv.-Doz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Manuel Leithner, MSc.

Vienna, June 25, 2025

Dominik Schreiber   Dimitrios Simos

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Dominik Schreiber, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. Juni 2025

Dominik Schreiber

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Zuallererst möchte ich meinen Betreuern Dimitris Simos, Manuel Leithner und Edgar Weippl für ihre Beratung, ihr wertvolles Feedback und ihre Geduld während dieser Arbeit meinen tiefsten Dank aussprechen. Ihre Unterstützung und ihr Fachwissen haben diese Arbeit maßgeblich geprägt und ich bin sehr dankbar für die Möglichkeit, von ihnen zu lernen.

Mein Dank gilt auch meiner Mutter Gabriele Schreiber, die mich stets ermutigt und mich mein Leben lang bedingungslos unterstützt hat.

Darüber hinaus möchte ich meinen Freunden und Kollegen bei SBA Research für ihre Gesellschaft, ihren Rat und die interessanten Diskussionen danken. Diese Arbeit wäre ohne die Beiträge und die Unterstützung dieser bemerkenswerten Menschen nicht möglich gewesen und dafür werde ich ihnen immer dankbar sein.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisors Dimitris Simos, Manuel Leithner, and Edgar Weippl, for their guidance, invaluable feedback, and patience throughout the course of this thesis. Their support and expertise have been instrumental in shaping this work, and I am truly grateful for the opportunity to learn from them.

I would also like to extend my heartfelt thanks to my mother, Gabriele Schreiber, who constantly encouraged me to continue and unconditionally supported me throughout my life.

Furthermore, I would like to acknowledge my friends and colleagues at SBA Research for their company, advice and interesting discussions. This thesis would not have been possible without the contributions and support of these remarkable people and I will always be grateful for that.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Der Bluetooth Low Energy (BLE) Standard ist eine weit verbreitete drahtlose Kommunikationstechnologie für Geräte des Internets der Dinge (IoT) und ermöglicht energiesparende Datenübertragung für verschiedene Anwendungen. Da die Verbreitung von BLE rasant zunimmt, wird die Gewährleistung der Sicherheit zum Schutz vor potenziellen Schwachstellen immer wichtiger. Anbieter BLE fähiger Mikrocontroller müssen BLE Protokolle in ihren Geräten entsprechend der Bluetooth Core Specification implementieren. Trotz der Standardisierungsbemühungen wurden in den BLE Implementierungen verschiedener Anbieter mithilfe manueller und automatisierter Methoden Schwachstellen entdeckt, die potentiell Millionen von Geräten betreffen. Dies ist teilweise auf die Komplexität der Protokolle zurückzuführen, die sich aus der überwältigenden Anzahl möglicher Konfigurationen ergibt und darauf, dass ein gründliches Testen der Implementierung aufgrund der Host-Controller-Schnittstelle (HCI) schwierig ist. In den letzten Jahren wurde das GreyHound Fuzzing Framework entwickelt, das mithilfe kostengünstiger Hardware beliebige BLE Pakete bis zur Linklayer Schicht senden kann. Da Fuzzing von Natur aus probabilistisch ist, ersetzen wir die oben genannte Fuzzing Methode durch einen kombinatorischen Sicherheits Test (CST) Ansatz, der eine garantierte Abdeckung des modellierten Eingabebereichs bietet. Durch die Generierung von Testfällen, die mehrere Kombinationen von Eingabeparametern abdecken, wollen wir Schwachstellen identifizieren, die mit herkömmlichen Testmethoden möglicherweise nicht entdeckt werden. Wir evaluieren unseren Ansatz anhand von Tests mit zehn verschiedenen BLE-Geräten mit unterschiedlichen Firmware Versionen. Insgesamt identifizieren wir 19 verschiedene Probleme, reproduzieren Ergebnisse früherer Arbeiten und decken zusätzliche Fehler auf. Um die Wirksamkeit unserer Methode zu überprüfen, vergleichen wir zusätzlich die Leistung unseres CST-Tools mit der des ursprünglichen Fuzzers und vergleichen deren Ausführungszeit und Fehlererkennungsfähigkeiten.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

The Bluetooth Low Energy (BLE) standard is a widely used wireless communication technology for Internet of Things (IoT) devices, enabling low-power data transmission for various applications. As the adoption of BLE continues to grow rapidly, ensuring their security becomes more and more important to protect against potential vulnerabilities. Vendors of BLE capable micro controllers are required to implement BLE protocols in their manufactured devices compliant to the Bluetooth Core Specification. Despite the efforts of standardization, several vulnerabilities were discovered in the BLE protocol implementations of multiple vendors using manual and automated methods, potentially affecting millions of devices. This can partially be attributed to the protocol's complexity, stemming from an overwhelming number of possible configurations and the fact that it is difficult to test implementations thoroughly due to the Host Controller Interface (HCI). In recent years, the GreyHound fuzzing framework was developed, which is able to send arbitrary BLE packets down to the link layer, using inexpensive consumer hardware. Since fuzzing is inherently probabilistic, we replace the aforementioned fuzzing method with a Combinatorial Security Testing (CST) approach that provides a guaranteed degree of input space coverage over the parameter model. By generating test cases that cover multiple combinations of input parameters, we aim to identify vulnerabilities that may not be uncovered through traditional testing methods. We evaluate our approach by testing 10 different BLE devices with a variety of firmware versions. In total we identify 19 distinct issues, replicating findings of the previous work and uncovering additional faults. To examine the effectiveness of our method, we additionally provide a performance comparison of our CST tool against the original fuzzer, contrasting their execution time and fault detection capabilities.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Publications arisen from this Thesis

During the work conducted as part of this Master Thesis, the following scientific publication has been published in the field of Software Security Testing:

1. Dominik Schreiber, Manuel Leithner, Jovan Zivanovic, Dimitris E. Simos, *"Bluetooth Low Energy Security Testing with Combinatorial Methods"*, in **2025 USENIX Annual Technical Conference**, to appear

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xvi**

1 Introduction **1**
1.1 Problem Statement 2
1.2 Aim of the Work 2

2 Related Work **5**
2.1 Security Protocol Testing 5
2.2 Bluetooth Testing 7

3 Preliminaries **11**
3.1 Combinatorial Security Testing 11
3.2 The Bluetooth Protocol 14
3.3 GreyHound Fuzzer 20

4 Methodology **23**
4.1 Testing Strategy 23
4.2 Input Modeling 27
4.3 Covering Array Generation 29
4.4 Oracles 30
4.5 Walkthrough Example 31

5 Test Execution **37**
5.1 Setup 37
5.2 Target Devices 39
5.3 Challenges 40

6 Evaluation **43**
6.1 Results 43
6.2 Responsible Disclosure 47

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.3 Comparison 48

7 Summary **53**
7.1 Conclusion 53
7.2 Future Work 55

List of Figures **59**

List of Tables **61**

List of Algorithms **63**

Bibliography **65**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

With the introduction of the Internet of Things (IoT), the amount of internet connected devices started to increase dramatically. To reduce the required effort of installing IoT devices, most products provide some kind of wireless connectivity like Bluetooth Low Energy (BLE) or ZigBee with automated discovery features, with BLE being one of the most popular ones. The domain of IoT usually requires devices to be small with low power consumption, so that sensors, actuators and switches can operate on batteries over long periods of time across wide areas. Because of these constraints, IoT devices tend to have limited processing power, which increases the difficulty of implementing secure encryption and validation mechanisms. Due to the rapid growth of the IoT sector and ever increasing amount of different interconnected devices, the NIST issued a report, mentioning the lack of standardization, testing and certification in this area [VKLA18]. All of this lead to attacks against various Internet-of-Things systems, networks, servers, devices, and applications witnessing a sharp increase, especially with the presence of 35.82 billion IoT devices since 2021; a number that could reach up to 75.44 billion by 2025[YNSC23].

The complexity and rapid development of such devices poses many challenges for testing and quality assurance. To tackle these issues, it is important to develop automated testing and certification processes, that allow for quick and resource efficient testing of IoT devices. For a long time now, the field of information security testing tools is dominated by a testing method called fuzzing, which refers to the process of mutating input for a Software Under Test (SUT) in various ways, until a crash or other unspecified behavior is found or the process is stopped. In the past, this technique has been applied with a lot of success in all kinds of software testing scenarios.

One problem of fuzzing is that it relies on random mutation of input, which often leads to insufficient input coverage during testing. As an alternative, combinatorial testing (CT) has been proposed by the National Institute of Standards and Technology (NIST), which is based on the empirical observation that most software faults are triggered by

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

parameter interactions of at most 6 different parameters [KKL13]. Because of this, it is possible to create relatively small covering arrays (CAs) from input parameter models (IPMs), that provide mathematical coverage guarantees for parameter interactions of the input model for a specified interaction strength. In these CAs, every row represents a test case, with 4-way to 6-way coverage CAs having been suggested to provide so-called pseudo exhaustive coverage of parameter interactions. There are many tools available with optimized algorithms, which are able to efficiently produce small CAs in a short amount of time, even for large IPMs.

While traditional CT focuses on determining whether the System Under Test (SUT) processes a broad range of inputs correctly, combinatorial security testing (CST) was introduced to apply CT methods with the aim to uncover vulnerabilities. This difference primarily affects how parameter values for the input model are chosen and how test results are evaluated.´

1.1 Problem Statement

BLE is a popular protocol used in many IoT devices, often having different stack implementations depending on the vendor of the Bluetooth chip. Currently, there exist specialized testing tools, focusing on specific areas like the pairing, encryption or the application layer. Additionally there are fuzzer projects that try to find errors by mutating input fields of BLE packets on various layers.

Due to the fact, that the Bluetooth stack is separated by the Host Controller Interface (HCI), it is notoriously difficult to test lower level layers of the BLE stack, i.e. Link Layer. Time and time again, researchers were able to find software vulnerabilities in different parts of the BLE protocol, which shows that the current testing and validation methods for BLE stack implementations are insufficient. Since none of the current testing methods provide any coverage guarantee, it would be worthwhile to investigate methods arising from the field of combinatorial testing to validate various BLE protocols with respect to their software implementations.

1.2 Aim of the Work

The aim of this work is to develop a combinatorial security testing tool for the BLE protocol, that can be used to test and validate different implementations of the BLE stack, providing a combinatorial coverage guarantee over the input parameters.

First, the layers and fields of the individual layers of the BLE protocol will be analysed, by looking at the official BLE specification and other sources found on the internet. It has to be explored how to model the individual layers, parameters and interactions between them. Additionally, the handling of invalid values and the impact on the generated tests has to be checked as well. Depending on the structure of the layers, the data type of the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

fields and the ranges of the parameters, multiple IPMs will be devised, which can later be used for the generation of CA test sets.

Since there already exists a tool, which is capable of sending arbitrary BLE packets using the NRF52840 dongle together with a custom firmware (i.e. Greyhound Fuzzer), the tool is going to be extended and modified to structurally go through the states of a BLE connection and generate CAs from the IPMs depending on the currently tested state and packet. To reduce the effort of result analysis, a test oracle will be developed, which can classify an executed test as passed or failed, depending on the test results.

To test the capabilities of the developed approach, a suite of BLE test devices will be gathered to serve as devices under test. If the developed approach is capable of finding errors or security vulnerabilities, they will be further analyzed by counting the individual errors quantitatively and also qualitatively by debugging the problem and using the JTAG interface. Additionally we will test our suite of test devices with the original fuzzer framework, to use it as a baseline for a performance comparison against the developed CST tool.

This work aims to answer the following research question:

- **RQ1:** Can CST be applied to test devices using the BLE protocol?
- **RQ2:** Is it possible to find vulnerabilities in BLE using CST?
- **RQ3:** How many errors can we find with CST compared to fuzzing with the GreyHound fuzzer?
- **RQ4:** How efficient is the CST approach compared to the GreyHound fuzzer when it comes to exploitation rate (found vulnerabilities / executed tests)?

The remainder of this work is structured as follows. At first, Chapter 2 presents the most important related work in the areas of BLE security testing and combinatorial testing. Following that, Chapter 3 provides fundamental information on BLE, combinatorial security testing, and the GreyHound fuzzer. In Chapter 4, we provide a detailed description of our methodology, including the overall testing strategy, input modeling considerations, and oracles. Chapter 5 describes the testing setup, target devices and their firmware versions as well as practical challenges encountered throughout our experiments, followed by a discussion of evaluation results, identified faults, and the differences between the results of GreyHound and our approach in Chapter 6. Finally in Chapter 7, we summarize the work and outline potential avenues for future work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

CHAPTER 2

Related Work

In this Chapter we present and review the current state of the art of security testing for IoT protocols. First in Section 2.1, we discuss the current state of the art of security testing for protocols. Following that, Section 2.2 is about testing of the Bluetooth and Bluetooth Low Energy protocols, which are often used by IoT devices.

2.1 Security Protocol Testing

The landscape of protocol security testing has evolved rapidly to address vulnerabilities in both traditional and modern network protocols, with model-based testing and fuzzing techniques remaining among the most widely adopted strategies. Due to the large number of different protocols in use, recent advances have pushed toward automated protocol reverse engineering and input analysis methods.

A recently proposed work is U-FUZZ [SGC24], a stateful fuzzing approach, that has proven effective in discovering semantic flaws in various protocol implementations. Using a network capture file, that contains packet traces of a normal communication between two devices implementing a protocol, U-FUZZ is capable of automatically constructing a protocol state machine, to guide the testing process. Every packet from the trace is associated with a state with the sending of packets leading to state transitions. The fuzzer acts as a man-in-the-middle, observing packets before they are sent and systematically altering them to maximize a cost function specified by the user. By working this way, the fuzzing process can be used for different protocols, without the need to change the underlying fuzzing engine, which the authors show by testing devices employing three different protocols (i.e. 5G NR, Zigbee and CoAP).

Another tool that has become popular for testing protocol implementations is AFLNet [PBR20], which is an extension of the successful and well known coverage guided AFL fuzzer [afl] developed by Google. While AFL is very good at fuzzing file-based or stdin-driven

applications, it lacks the ability to handle the interactive, message-based communication of network protocols. AFLNet on the other hand is designed specifically for fuzzing stateful network protocols like HTTP, MQTT, and DNS. It addresses the limitations of AFL by introducing a state-machine learning component that infers the protocol state transitions dynamically. AFLNet uses server response codes to determine which server states are triggered by a given sequence of messages. This feedback allows AFLNet to find new areas within the protocol's state space and strategically guide the fuzzing process toward those regions. Additionally, AFLNet provides a range of protocol aware mutation operators to alter message subsequences. It constructs a message pool from a set of message sequences, consisting of both real messages captured via network sniffing and synthetically generated ones. From this pool, messages can be taken to be inserted into or used to replace parts of existing sequences to mutate them, enabling diverse and effective protocol specific transformations. By integrating structured message mutation with coverage-guided fuzzing, AFLNet greatly improves the detection of vulnerabilities in stateful services, achieving better results compared to general-purpose fuzzers in protocol-aware contexts.

Although fuzzing can be adapted relatively easily to various kinds of software testing scenarios, it inherently suffers from the fact that its input generation relies on random mutation of the input, which does not offer any coverage guarantees. Because of this, combinatorial testing has become more and more popular as an alternative method to fuzzing, due to its coverage guarantee regarding the parameter interaction of the input model. The application of combinatorial testing to the field of software security is well established [SKVK16] and lead to the new research field of combinatorial security testing (CST).

Regarding protocol testing, previous research using CT appears to have primarily focused on testing different aspects of the TLS protocol. In [SBD+17], the authors are using CT to test TLS library implementations, by targeting the client side of the handshake procedure of the TLS protocol. They start with constructing three different input models, one for each client-side TLS event. After that they use the CA generation program ACTS [YLKK13] to generate a CA for each of the parameter models. Finally, the tool TLS-Attacker [tlsa] is utilized to establish TLS connections with the target and replace values of TLS messages with the ones from the generated CAs. As oracle the miTLS library is taken as reference implementation and execution traces of test results are compared against it to decide if a test was executed correctly.

A different approach focusing on message sequence testing was applied in [GSD+19]. Here the notion of weighted t-way sequences is used to generate sequence covering arrays (SCAs), where the entries are from a finite set of symbols S, such that every t-way permutation of symbols from S occurs in at least one row and each row is a permutation of the symbols in S. As symbols, the authors define the individual messages sent by the client during a TLS handshake. The weights are gathered by analysing a bug database of previously discovered TLS vulnerabilities. TLS messages that appear more often in vulnerability triggering sequences are given a higher weight, so that they get tested more

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

extensively. The oracle works similar to the approach by [SBD+17], comparing execution traces of the target with a reference implementation.

In [KS17], CST is applied to test the validation logic for X.509 certificates in TLS implementations. In this work, the authors provide insight into different aspects of the modeling process, to better encapsulate the structure and semantics of the input. The authors propose a block design, to divide TLS certificates into a block of mandatory parameters and several blocks of extension parameters, to better fit the structure of the input model to the semantics of the targeted validation logic. They combined the IPMs in different ways, which they refer to as flat, inter and intra models. More details about this work is given in Section 4.2, where also some other methods for structure modeling are discussed.

Recently TLS-Anvil was presented in [MNH+22], a CST testing approach for TLS build upon TLS-Attacker [tlsa] and TLS-Scanner [tlsb]. They start by deriving test templates for TLS client and server implementations from TLS related RFCs and known bugs, with each template checking for a specific requirement. These templates can be used to instantiate test cases, while also containing definitions for the desired outcome of a test generated from this template. For every template an IPM is devised, where all parameters of an IPM have the same semantic meaning (i.e. invalid or invalid in the context of the template). Because of that, a template also represents a test oracle for all tests that can be derived from it. Additionally, the authors added constraints to the IPMs in cases where the interaction of specific values are changing their semantics.

To evaluate their approach they created a library of docker images with 700 versions of 23 different implementations, that can be started and stopped using a Java interface. While the previous CST methods for TLS were only able to find discrepancies in the error handling behavior of various TLS implementations, TLS-Anvil could find five issues that affect cryptographic computations and three immediately exploitable vulnerabilities in the newest TLS library versions.

2.2 Bluetooth Testing

Given their widespread use, the security and privacy properties of Bluetooth protocols have been the subject of extensive research [BAAHH22, CPST22]. Over the years, many different issues and vulnerabilities have been discovered in Bluetooth classic and BLE, ranging from simple denial of service attacks to remote code execution or issues in the encryption/authentication mechanisms [WWX+24, BS23b, BS23a]. This is partially due to the complexity of the protocols and the abstraction layers in the stack, which pose challenges for testing and implementing them correctly. Many different testing methods have been proposed, some of which we will discuss below in more detail.

One of the most relevant studies for this work is an automated BLE fuzz testing method named GreyHound Fuzzer [GWC+20], which enables comprehensive testing across all BLE layers, including the link layer for peripheral devices. This approach involves custom

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

firmware for the NRF52840 dongle and the creation of a BLE state machine integrated into a fuzzer guided by probability optimization. This setup allows for automated testing of peripheral devices, whether they use closed-source firmware or BLE development boards that can be programmed with vendor-provided example applications or other custom implementations. Through this method, the researchers uncovered 17 previously unknown security vulnerabilities, collectively referred to as *SweynTooth*, in the firmware of various BLE controllers. The significant number of flaws discovered highlights the efficacy of this approach, particularly since it does not require access to the peripheral's source code for testing. More details about the inner workings of the GreyHound fuzzer framework are given in Section 3.3.

In a subsequent study, the same research group focused on Bluetooth classic peripheral devices [GBC+22]. This work presents an automated method to infer the state machine during test execution by observing the responses of the target device, together with a fuzzer and testing framework similar to their earlier work. Instead of the NRF52840 dongle, which is limited to BLE, the authors chose an ESP32 controller, again using a custom Bluetooth firmware to achieve full control over all packet layers, this time for the Bluetooth classic protocol. To avoid developing the Bluetooth firmware for the ESP32 from scratch, they developed a patching framework to patch the proprietary binary of the ESP32 to add the required functionality. The authors identified 16 vulnerabilities across 13 Bluetooth chips from major vendors like Qualcomm, Intel, and Texas Instruments.

Another framework that allows for the modification of low-level Bluetooth classic layers and also firmware patching is InternalBlue [MCSH19]. By reverse engineering and patching the firmware of the BCM4339 Bluetooth chip from Broadcom, InternalBlue is capable to modify read only memory (ROM) regions of the chip, inject custom code into the running firmware, send arbitrary HCI commands and monitor the link manager protocol (LMP) to sniff or alter Bluetooth packets on lower layers. In their initial publication [MCSH19], the authors found a new critical vulnerability affecting multiple Broadcom chips and showed how the framework can be used to add or fix features in closed source Bluetooth firmware. Additionally, the framework was used by other projects [ATR19, ATR20] to implement and validate their attacks. Compared to the GreyHound[GWC+20] framework, InternalBlue can only alter packets, after a connection with the peripheral is already established and it is also limited in the number of fields and layers that are accessible.

In [PA22], the authors implement a stateful black-box fuzzing method, that uses active automata learning to automatically infer a behavioral model of BLE peripherals. The learned state machine is then used to generate sequences of BLE packets, whose fields can be altered by the fuzzing component. To send manipulated BLE packets to the peripheral, the firmware and Scapy extension developed by [GWC+20] is used. If a peripheral reacts differently than defined by the state machine, it is marked as suspicious for further analysis. The authors validate their approach through a case study on BLE devices, where the framework successfully identified serious security and reliability flaws, including crashes in four out of six tested devices.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

The authors of [ATR20] present impersonation attacks they found by carefully analysing the Bluetooth specification of the Bluetooth classic protocol, that allow attackers to impersonate previously bonded devices without knowledge of the secret link key. By exploiting flaws in the Bluetooth specification, particularly in the authentication procedures, an attacker can initiate or respond to authentication requests using only the target device's Bluetooth address. Due to the absence of strict mutual authentication enforcement, the attacker can successfully complete a connection handshake, pretending to be a trusted device without requiring physical access or user interaction. To show the validity of their attacks and that they can also be exploited in practice, a total of 31 devices with 28 unique Bluetooth chips have been tested, with all of them being vulnerable.

Another successful approach using formal analysis was presented in [WNK+20]. In this work the authors also uncovered a group of impersonation attacks, that can be used to spoof the identity of a previously bonded device, similar to the vulnerabilities described in [ATR20]. In contrast to [ATR20] where the issues were discovered by carefully analysing the official specification, in this work the authors took advantage of ProVerif [pro], an automatic cryptographic protocol verifier. After building a formal model of the authentication process as two communicating state machines and specifying security properties, the tool is used to analyse the model and provide counterexamples, if violations of the properties are detected.

As of now, many issues in the Bluetooth classic and BLE protocols have been found by formal verification methods or by analysis of the official specification. Most have been focused on the authentication and encryption mechanisms, since they are the most critical and complex functions. Because of the abstraction layers of the protocol stacks, it was also very difficult to test all layers of the protocols, which was only made possible recently by the GreyHound framework.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 3

Preliminaries

This Chapter provides background information on three particular areas that will be the focus of this work. The first Section 3.1 provides an introduction about Combinatorial Security Testing. Following that, Section 3.2 explains the Bluetooth classic and BLE protocol stack, details about the differences between Bluetooth classic and BLE and the connection flow between two BLE devices. After that, Section 3.3 presents the GreyHound fuzzer framework, which our project is built on.

3.1 Combinatorial Security Testing

Combinatorial Testing is a well-known black-box testing technique that generates tests by modeling the input parameters of a SUT. Although the foundational idea was initially investigated by Burr and Young in 1998 [BY98], it gained significant attention following a comprehensive empirical study by the NIST.

Between 1999 and 2004, NIST carried out multiple studies focusing on software testing and the role of parameter interactions in software failures. The observations from these studies are illustrated in Figure 3.1. The key insights derived from this research were encapsulated in what is known as the *interaction rule*, as outlined in [KKL13] as follows:

Hypothesis 1 *Most failures are induced by single factor faults or by the joint combinatorial effect (interaction) of two factors, with progressively fewer failures induced by interactions between three or more factors.*

These studies revealed that all examined software faults were caused by interactions involving no more than six parameters [KBD+15]. As a result, it is generally unnecessary to test every possible combination of input parameter values to achieve a high degree of test quality assurance.

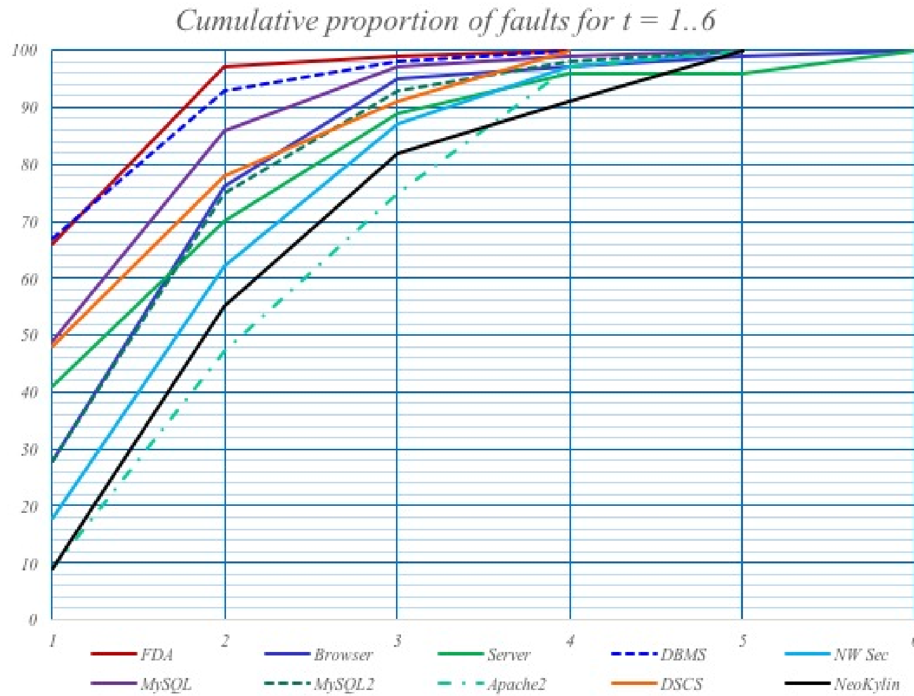

Figure 3.1: NIST study of fault inducing parameter interactions [Weba].

More specifically, CT uses an input parameter model (IPM) containing all parameters that can be externally supplied to a SUT, together with a list of their possible values. Additionally, models can be enhanced with constraints, that restrict which combinations of distinct parameter values are allowed to be contained in a single test case. For example, in a model involving as parameters a person's age and age group, a constraint might specify that if the age group is "child", the age must not exceed 17. A test case selects one value from the IPM for every input parameter. The coverage guarantees provided by CT are based on its use of a discrete mathematical structure known as a covering array (CA).

Definition 1 (Covering array as defined by [Col10]) *A covering array $CA_\lambda(N; t, k, v)$ is an $N \times k$ array. In every $N \times t$ subarray, each t-tuple occurs at least λ times. Then t is the strength of the coverage of interactions, k is the number of components (degree), and v is the number of symbols for each component (order). Only the case when $\lambda = 1$ is treated; the subscript is then omitted in the notation. The size N is omitted when inessential in the context.*

Each row of a CA represents a test case, with each column referring to a parameter from the IPM.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Test ↓ / Var →	A	B	C	D	E	F	G	H	I	J
1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	0	1	0	0	0	0	1
4	1	0	1	1	0	1	0	1	0	0
5	1	0	0	0	1	1	1	0	0	0
6	0	1	1	0	0	1	0	0	1	0
7	0	0	1	0	1	0	1	1	1	0
8	1	1	0	1	0	0	1	0	1	0
9	0	0	0	1	1	1	0	0	1	1
10	0	0	1	1	0	0	1	0	0	1
11	0	1	0	1	1	0	0	1	0	0
12	1	0	0	0	0	0	0	1	1	1
13	0	1	0	0	0	1	1	1	0	1

Figure 3.2: Covering array example [HWKK15].

Compared to exhaustive testing, the main advantage of CT is the reduction of test set size and therefore execution time. Consider, for example, an SUT with 10 on/off switches. We can model this as an IPM with 10 parameters, each of them having two values. It would take $2^{10} = 1024$ test cases to cover all input possibilities, but if we know that flaws are just triggered by 3-way parameter interactions or lower, we can find all issues using only 13 test cases. The reduction in the number of test cases is possible because a lot of interaction triples can be packed into one test case. The result is a covering array (CA) of strength t, where t is the degree of parameter interaction we want to cover (i.e. $t = 3$ for our example). In the CA, every row represents a test case with different configuration states for the SUT, where the `on` state is represented by 1 and the `off` state by 0.

Figure 3.2 shows an example covering array for the SUT with ten binary switches described above. Each color represents an example for full triple coverage of the corresponding 3-way parameter value selections. This array covers all 3-way parameter interactions with only 13 test cases, which is a reduction of 98.65%, compared to an exhaustive test set with 1024 tests.

In practice, applying CT usually entails the following steps:

1. Creating an IPM from a description of the SUT, source code or other type of analysis.

2. Generating one or more CAs from the model.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3. Translating resulting CAs to concrete test sets, as some values may be encoded symbolically and must be transformed to discrete data types.
4. Executing the test cases and recording the responses of the SUT. This can simply be the output of the SUT or more complex information such as execution traces.
5. Evaluating the SUT's behavior using an oracle to identify whether the device behaved as expected.

While traditional CT is a type of conformance testing, which tests whether the SUT processes a wide range of inputs correctly and adheres to its specified functionality, combinatorial security testing (CST) tries to find vulnerabilities and unspecified behavior. This primarily influences the first and last step from above. In conformance testing, the IPM mostly contains valid and well formed values, while in CST, the model consists of valid values and crafted values meant to trigger vulnerabilities (commonly based on expert knowledge on the particular type of attack). For example, some specifically crafted inputs might consist of very long sequences, empty values, or particular strings with a special meaning in languages such as SQL (for SQL injections), shell scripts (for command injections) or JavaScript (for cross-site scripting, XSS)[SKG+16].

CST also relies on different types of oracles than traditional CT. Whereas a CT oracle checks whether the SUT's output matches an expected result, a CST oracle focuses on identifying signs of a triggered vulnerability. This is typically assessed by detecting crashes, analyzing log files for unusual error messages, or observing unexpected behavior that deviates from the system's specification.

3.2 The Bluetooth Protocol

The Bluetooth protocol was initially developed in the 1990s, to facilitate short-range Point-to-Point wireless communication between devices and is also referred to as Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) or Bluetooth classic. In the beginning, it was mostly used for wireless audio streaming in headsets, wireless speakers and entertainment systems in cars, but it can also be used for file sharing and other services that require data transfer, as well as the creation of PAN ad-hoc networks. It operates in the license free 2.4GHz industrial, scientific and medical (ISM) band and has a relatively low power consumption compared to wireless LAN. To avoid congestion and collisions, Bluetooth uses Adaptive Frequency Hopping (AFH) to hop between different channels. Bluetooth classic can hop between 79 channels with 1 MHz spacing.

Later, Bluetooth Low Energy [blu14] was added in Bluetooth core specification 4.0 in 2010 as a means of enabling resource constrained entities such as Internet of Things (IoT) to communicate wireless. Compared to Bluetooth classic, BLE as the name suggests, consumes even less energy. BLE is one of the most widely used communication protocols for IoT devices [BAAHH22] and is currently managed by the Bluetooth Special Interest Group (SIG), that consists of more than 20,000 members, in version 5.4. It operates

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

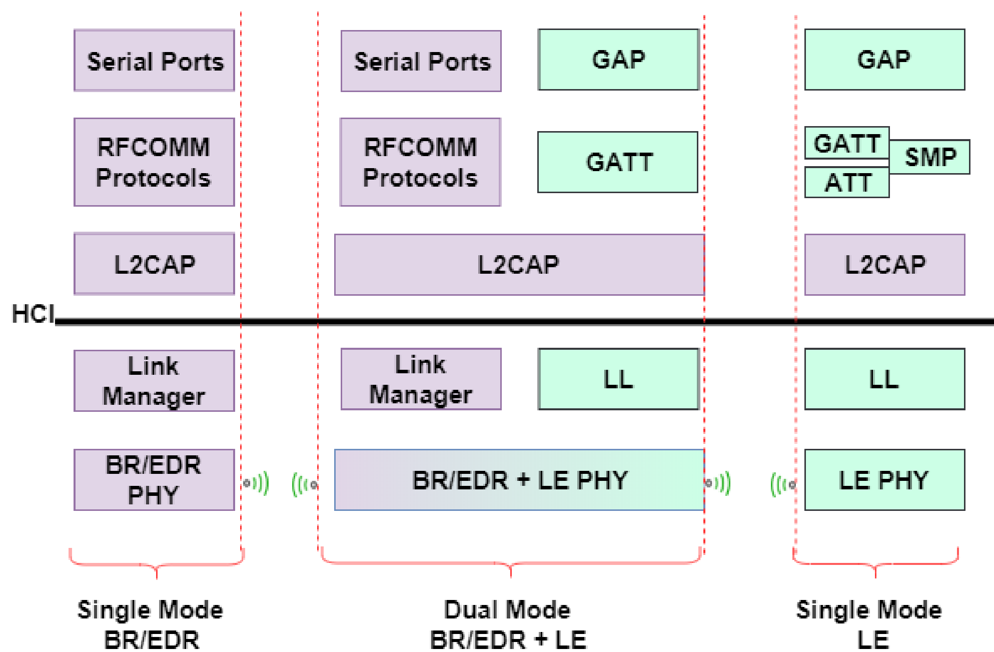

Figure 3.3: Bluetooth and BLE stack layers [TM24]

in the same 2.4GHz band but is a nearly completely separate protocol from Bluetooth classic. BLE can hop between 40 channels with 2MHz spacing, where 3 channels are so-called advertising channels and 37 are data channels. Compared to Bluetooth classic, BLE supports multiple communication topologies like broadcasting or Many-to-Many (mesh), that enable the creation of large scale device networks.

3.2.1 The BLE Stack

The Bluetooth classic stack depicted on the left and the BLE stack on the right in Figure 3.3, consist of several protocol layers that interact with each other, each responsible for a different set of functionalities. While both protocols have the `L2CAP` and `HCI` layer in common, they are inherently different protocols. Often a single chip provides functionality for both protocols, which is referred to as dual mode and is especially common in modern smartphones.

Since this work is about BLE testing, we will discuss the BLE protocol layers in more detail. At the bottom of the BLE stack in Figure 3.3, there is the phyiscal layer (`PHY`). It is responsible for operating the radio for physical data transmission over the air and manages the AFH between the different channels. For modulation it uses Gaussian Frequency-Shift Keying (GFSK), with a frequency deviation of +-185 kHz for the logical 1 and logical 0 states.

Next there is the Link Layer (`LL`), which manages a logical channel or "link" that is established between devices. The term "logical" refers to a virtual connection rather than

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

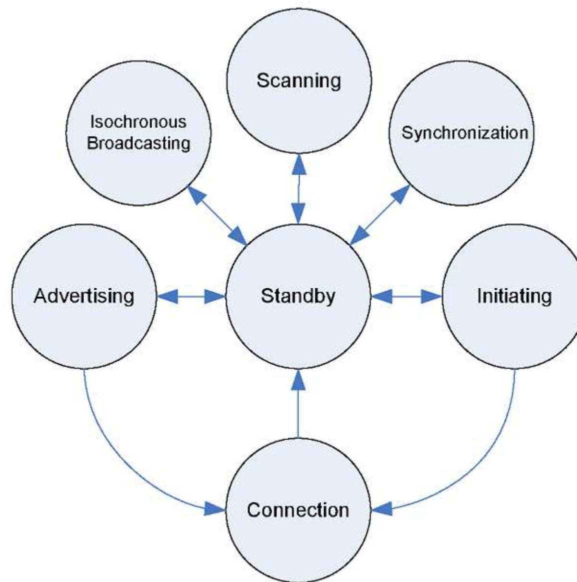

Figure 3.4: Link Layer State Machine [SIG]

a physical one, that represents an abstraction over the physical layer below. The LL can operate in one of seven different states:

- Standby: The initial state in which the radio is idle.
- Advertising: In the advertising state, the radio uses channels 37, 38, and 39 to broadcast advertising packets to announce its presence and provide information to nearby devices.
- Scanning: In the scanning state, the radio listens for advertising packets from other devices. This enables the device to discover nearby BLE devices and gather information about their services.
- Initiating: When receiving a connectable advertising packet, the radio sends a connection request to the target.
- Synchronization: In this state, the radio listens for periodic advertising packets and isochronous data packets.
- Isochronous Broadcasting: In the Isochronous Broadcasting State a device will transmit isochronous data packets on an isochronous physical channel.
- Connection: When this state is reached from the initiating state, the device becomes the "central" device, taking charge of managing the connection. If it comes from the advertising state, it takes on the "peripheral" role.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Packet format for Uncoded LE data packets

<table>
<tr><th>Preamble</th><th>Access Address</th><th colspan="6">PDU (2-257 bytes)</th><th>CRC</th></tr>
<tr><td rowspan="5">1 byte (1M PHY)
2 bytes (2M PHY)</td><td rowspan="5">4 bytes</td><td>LL Header</td><td colspan="4">Payload (0-251 bytes)</td><td rowspan="2">MIC (Optional)</td><td rowspan="5">3 bytes</td></tr>
<tr><td rowspan="4">2 bytes</td><td>L2CAP Header</td><td colspan="3">ATT Data (0-247 bytes)</td></tr>
<tr><td rowspan="3">4 bytes</td><td colspan="2">ATT Header</td><td>ATT Payload</td><td rowspan="3">4 bytes</td></tr>
<tr><td>Op Code</td><td>Attribute Handle</td><td rowspan="2">Up to 244 bytes</td></tr>
<tr><td>1 byte</td><td>2 bytes</td></tr>
</table>

Figure 3.5: Structure of a BLE data packet. [TPHB21]

The state machine and possible transitions of the link layer are depicted in Figure 3.4.

The Host Controller Interface (HCI) represents an isolation layer between the lower-layers (PHY, LL) and higher-layer protocols (L2CAP, ATT, GATT, GAP, etc.). The higher-layer protocols run on the host operating system, while the lower-layer protocols run on a separate hardware called controller, which receives HCI commands from the host that are then transformed into appropriate Link Layer (LL) packets. This isolation between components and the resulting inability to access all aspects of the protocol is one of the reasons why the Bluetooth protocol stack is difficult to test.

The next layer on the BLE stack is the Bluetooth Logical Link Control and Adaptation Protocol (L2CAP) The functions of the L2CAP include protocol/channel multiplexing, segmentation and reassembly (SAR), per-channel flow control, and error control. The L2CAP interfaces with the upper layer protocols of the stack.

On top of L2CAP, there are the Generic Attribute Profile (GATT) and Attribute Protocol (ATT). The ATT Protocol acts as a high level transport protocol, that describes how a device referred to as the server can expose a set of attributes and their associated values to a peer device referred to as the client. The GATT protocol defines a service framework using the ATT protocol and defines procedures and formats of services and their characteristics. The defined procedures include discovering, reading, writing, notifying and indicating characteristics, as well as configuring the broadcast of characteristics [gat]. The structure of a BLE data packet with all the transport layers is depicted in Figure 3.5.

To ensure safe transmission of data, there is the Security Manager Protocol (SMP), which defines methods of pairing and key distribution, a protocol for those methods and a security toolbox to be used by those methods and other parts of a BLE device. Each device generates and controls the keys it distributes and no other device affects the generation of these keys. The strength of a key is as strong as the algorithms implemented inside the distributing device. The Pairing procedure is performed to establish keys which can then be used to encrypt a link. A transport specific key distribution is then performed to share the keys which can be used to encrypt a link in future reconnections, verify signed data and random address resolution.

At last there is the Generic Access Profile (GAP) layer. It controls connections and advertising in BLE devices and determines how two devices can interact with each other. GAP defines various roles for devices, with the key concepts being the definition of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

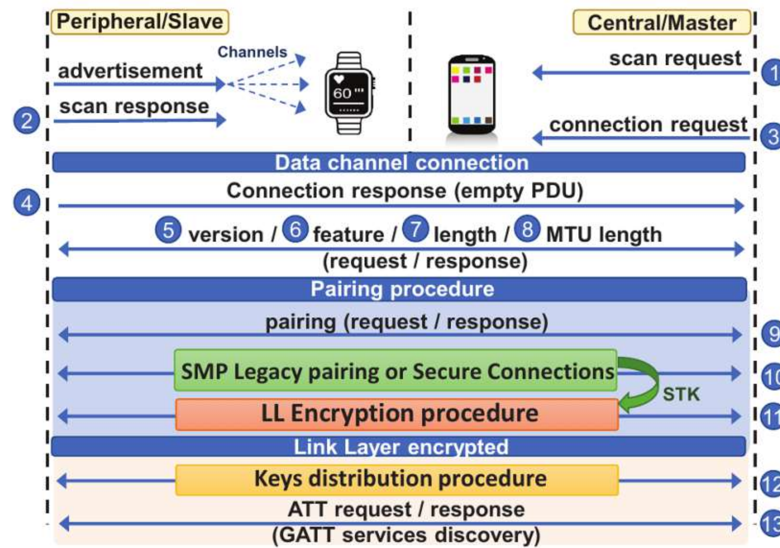

Figure 3.6: Message exchange between BLE Devices. [GWC+20]

Central and Peripheral devices. Additionally, the GAP can also operate as an observer or broadcaster.

- Peripheral devices are most of the time small, low power, resource constrained devices that advertise their services to a central device. Peripheral devices are things like a heart rate monitor, a BLE enabled proximity tag, etc.
- Central devices are usually devices like mobile phones or tablets. They initiate the connection and usually have far more processing power and memory.
- Broadcasters are devices that only send advertising events.
- Observer are devices that only receive advertising events.

3.2.2 The BLE Connection Flow

Now that we have discussed the BLE stack layers, we will explain the BLE connection and data transmission process in more detail, using the simplified model presented in [GWC+20]. A diagram showing the message exchange between a central and a peripheral device is given in Figure 3.6. Additionally, Figure 3.7 is showing a simplified model of the states of the BLE protocol.

A connection between a master and a peripheral device starts with the peripheral regularly broadcasting advertisements, while a central device starts in the scanning state, listening for such advertisements. If a central device wants to initiate a connection, it sends a `scan request` to get further information from the peripheral such as its name. The peripheral

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

then responds with a `scan response`, after which the central can send a `connection request` to start establishing a connection. The `connection request` contains connection parameters relevant to the synchronization and communication timing between central and peripheral. After that, the central is in the connection state. If the peripheral responds in time with a `connection response` packet, the central proceeds to the `initial_setup` state.

After the transition to the `initial_setup` state, the central requests information from the peripheral by sending a `version request`, `feature request`, `length request` and `MTU length request` in any order, which is shown on the top-left in Figure 3.7. The exchanged properties describe the supported LL features and capabilities of a BLE device and during the same exchange, the peripheral also receives the capabilities of the central.

Following that, the initial setup is complete and the central continues to the `list_pri_services` state. In this state, the central scans for the peripheral's main services using the Generic Attribute Profile (GATT) Service Discovery procedure and stores their attributes in local memory.

Next, the central proceeds to the `pairing_req` state and starts a procedure to establish an encrypted communication with the peripheral. The pairing procedure in BLE is performed in three steps:

1. Exchange of pairing information
2. Authentication of the link
3. Distribution of the keys

It begins with the central sending a `pairing request` packet to the peripheral, which contains information about the preferred pairing mode to be used in the next state. If the peripheral accepts the proposed pairing mode, it sends a response to the central and both proceed to the `smp_pairing` state.

For pairing, the devices can choose between two different modes (i.e. Legacy pairing or Secure Connection (SC) pairing via SMP exchanges). As is depicted in Figure 3.7, the devices go through the pairing procedure starting from either the `legacy_pairing` or `sc_pairing` state. At first a Temporary Key (TK) is used for initial authentication of the devices, which can be 0, a six digit zero padded number or a 128bit key, depending on the chosen authentication method. After the TK exchange, the TK is used by the devices to generate a Short Term Key (STK).

In the third step of the pairing procedure, the devices distribute the derived keys using specific SMP packets. The keys are encrypted with the STK and then stored in a secure database.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Next, it continues to the ll_encryption state and sends an encryption_req to start a Long Term Key (LTK) exchange challenge with the peripheral. Using the peripheral's response a LTK is derived by the central, which then sends an encrypted encryption_res packet by using the obtained sessionKey. If the peripheral successfully authenticates and decrypts the encryption_res from the central, it sends a new encrypted encryption_res back to the central, signaling that the connection has been successfully encrypted.

In sc_pairing mode, the STK is used as the LTK. However, if the devices are limited to legacy_pairing, the central and peripheral can optionally follow the key distribution process (step 12 in Figure 3.6) to exchange an LTK.

The LTK allows the central to skip the pairing process in future connections and proceed directly to step 11 in Figure 3.6. During subsequent stages, the central and peripheral exchange the LTK based on the parameters negotiated in the pairing request, and the central retrieves additional services from the peripheral in the list_sec_services state.

After establishing an LL connection and completing pairing, the central discovers all available attributes (i.e., information) on the peripheral by conducting a GATT Primary Service Discovery. This involves exchanging multiple ATT requests and ATT responses (step 13 in Figure 3.6) to retrieve predefined ATT attributes. In the gatt_read/write state, reading and writing is simulated, using locally stored ATT attributes, which were discovered during the list_pri_services and list_sec_services states. This step is needed to emulate the injection of malformed ATT attributes as part of the testing approach. Although the gatt_read/write state at the bottom of Figure 3.7 is not part of the official BLE protocol specification, it is necessary to evaluate how a peripheral behaves when exposed to malformed ATT attributes.

3.3 GreyHound Fuzzer

Since our methodology is built upon the GreyHound fuzzer framework [GWC+20], the following section provides a brief introduction of the hardware and software elements of this tool.

The developers of the GreyHound Fuzzer implemented a systematic and comprehensive testing framework for the WIFI and BLE protocols, which is able to find vulnerabilities in protocol implementations [blu14] and to manipulate and test all their layers, down to the lowest levels.

The parts of the GreyHound fuzzer framework that run on the host system consists of many Python 2.7 modules, which include a guided fuzzing component (i.e. GreyHound) and two models, one for WIFI and one for BLE, which represent the states and transitions of the respective protocols. Additionally, the authors extended the Scapy Python library with models for all the packets of both protocols and implemented a validator, which checks the validity of packets and if they are received in the correct state. Besides that,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

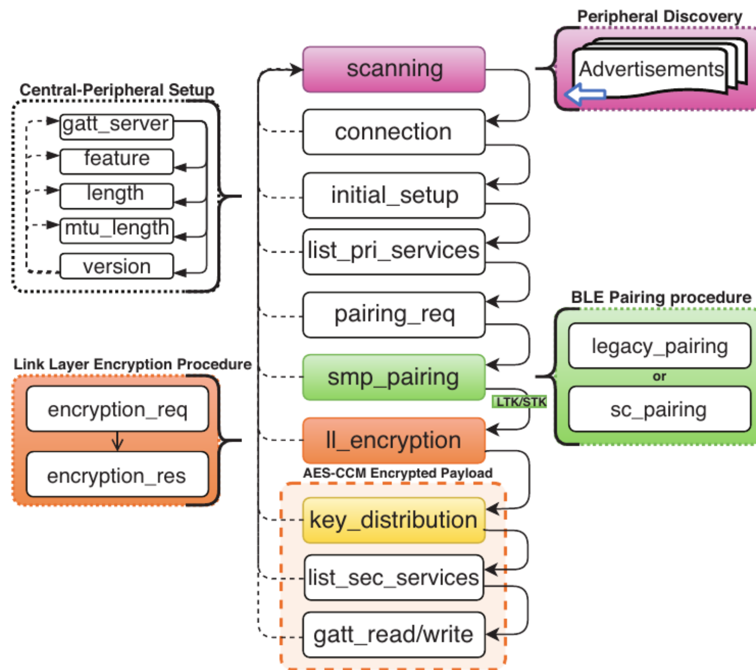

Figure 3.7: Simplified overview of BLE protocol model [GWC+20]

there is also a Python implementation of a command transceiver module, that uses a USB serial connection to transfer commands and BLE packet data between the host and a custom firmware, running on an NRF52840 USB dongle.

Usually, Firmware implementations for adapters that comply with the Bluetooth standard offer only restricted access to the internal protocol layers, as they can only be accessed through the HCI. To address this limitation, the GreyHound fuzzer framework uses the NRF52840 USB dongle running custom firmware developed in C++. Communication with the dongle is established via a USB serial connection in the command transceiver module, which enables us to control the dongle using Python functions. Together with the extended version of the Scapy framework, it is possible to create and dissect custom BLE packets, which can then be sent to the dongle. The custom firmware handles timing requirements and CRC calculations, ultimately sending the finalized packet over the air.

The BLE state machine model of the framework is capable of initiating packet transmission and responding to feedback from BLE slave devices. To ensure compatibility with various BLE implementations, the state machine supports multiple potential pathways, as the BLE specification allows flexibility in the sequence of certain interactions. A simplified representation of the BLE protocol model is illustrated in Figure 3.7.

Additionally to the components for sending and receiving custom BLE packets, GrayHound also features a fuzzing module that uses the particle swarm optimization (PSO) algorithm of the pyGMO library to guide the packet modification behavior of the fuzzer.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

At the beginning, the fuzzer gets a set of initial mutation probabilities, one probability for each layer occurring in the current state's packet, from the PSO module. The state machine is set to the `scanning` state and waits until an advertisement of the target device is encountered. On reception, the fuzzer receives a valid BLE packet from the state machine model, whose layers are then mutated according to the probabilities and sent to the target peripheral. If a layer is selected to be fuzzed, the fuzzer needs to decide the set of fields in the layer to be mutated. The fuzzer processes each field in the layer sequentially and applies a mutation probability of 50% to each individual field of the layer. All fields within a layer are assigned the same mutation probability, which helps minimize the number of parameters involved in the iterative optimization process.

For mutating the fields, the fuzzing module implements three Mutation Operators:

1. **Random bytes**: replaces the value of a field with random bytes
2. **Zero filling**: sets the value of a field to zero
3. **Bit setting**: sets the most significant bit of a single-byte field value to one

The mutated packet is also saved to a packet history set, from which a packet might be chosen and sent out of order, depending on a probability which is also optimized by the PSO algorithm.

If the peripheral responds, the received packet is checked by the validation module which detects deviations from the Bluetooth Core Specification. It checks the internal packet structure for correctness by comparing it against a set of expected layers or rejection layers of the current state in accordance with the protocol model.

After each iteration through the state machine, the PSO algorithm adjusts the probabilities depending on a cost function and which parameters have been fuzzed before, to systematically guide the fuzzing process. The goal of the fuzzer is to maximize the discovery of potential anomalies and crashes. Because of this, the GreyHound fuzzer uses the number of unique anomalies discovered during the fuzzing session as the cost function which is measured for each individual set of mutation probabilities.

The general goal of a PSO algorithm is to optimize the value of a chosen cost function by varying the position of the particles in the swarm (i.e., the population). In the GreyHound fuzzer, each particle within the swarm represents a different set of mutation probabilities.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 4

Methodology

In this Chapter, we present the general methodology of our work. The first Section 4.1 starts with a description of the overall testing strategy. Following that, Section 4.2 provides information and details about aspects of the input modeling process. Section 4.3 then briefly describes the CA generation process, after which Section 4.4 presents the inner workings of our developed test oracles. At the end in Section 4.5, we explain the execution of an individual test case in more detail, using a walk-through example.

4.1 Testing Strategy

Because of the strong performance shown by the GreyHound fuzzer and the inherent limitations of fuzzing due to its probabilistic nature, we want to investigate the applicability of Combinatorial Sequence Testing (CST) for generating test cases targeting BLE implementations. Although CST has been previously applied to the security testing of TLS implementations [SBG+19], a more sophisticated testing strategy is needed to handle the added complexity and multi-layered structure of the BLE protocol. Our approach builds on the GreyHound fuzzing framework that was introduced in Section 3.3, but replaces the guided fuzzing component entirely with combinatorial test generation. This modification involves the development of input models, test oracles and the assembly of a suite of test devices. Additionally, we want to assess whether the systematic input space coverage offered by CST can uncover previously undetected bugs or vulnerabilities.

The original GreyHound fuzzer chooses transitions through the state machine visualized in Figure 4.2 guided by probability and therefore in a nondeterministic way. In CST, we generally want to traverse the state machine in a more deterministic way to ensure that all valid paths are covered equally. A simplified overview of our testing strategy is provided in Figure 4.3, the details of which are discussed in the following paragraphs. Additionally, Algorithm 4.1 provides a description of the algorithm in pseudo-code.

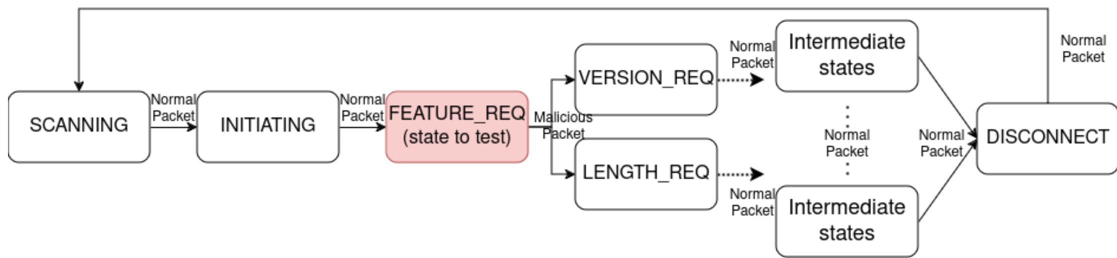

Figure 4.1: Example testing flow.

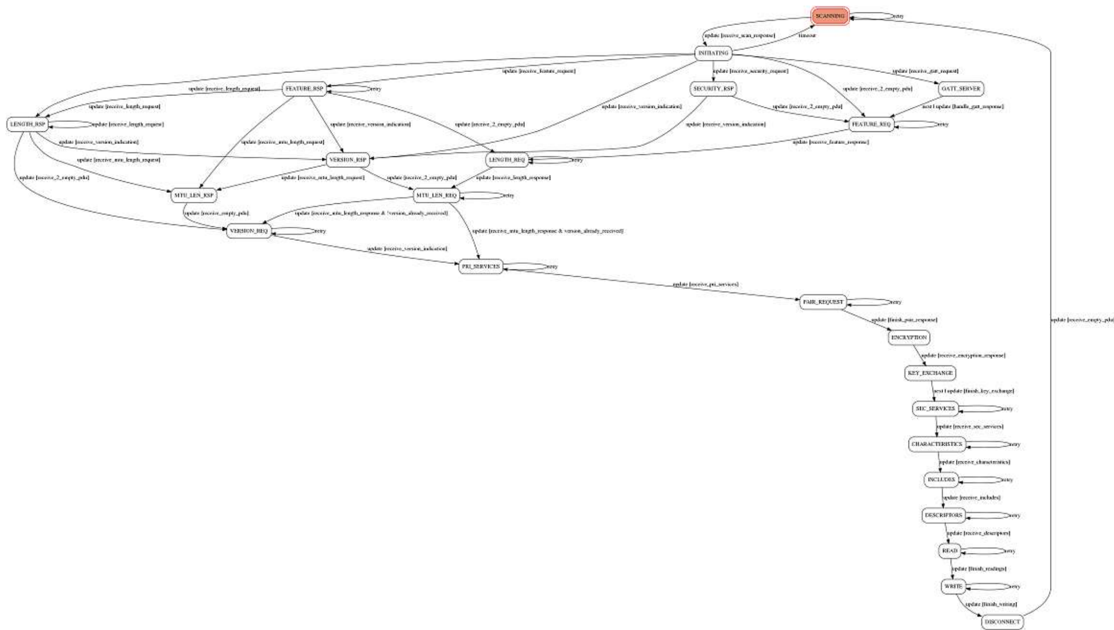

Figure 4.2: State machine as implemented by the GreyHound Fuzzer.

We begin by generating a set of all possible paths to every state of the state machine. Following that, a path from the set is chosen. The final state of the chosen path is the current state to test. While the selection order does not impact our approach, our implementation prioritizes paths leading to states closer to the initial state. We initialize the state machine of the GreyHound framework with all transitions of the selected path and all transitions that lead to retransmissions and those triggered by timeouts. Additionally, all transitions of paths that originate from the current state to test and lead to the *DISCONNECT* state are also added to the state machine.

If the current state to test is reachable via the selected path (not all paths are necessarily supported by the specific SUT), the path is marked as valid. Otherwise the paths are considered invalid and cannot be tested. If the path is valid, the next BLE packet to be transmitted according to the state machine of the GreyHound framework is examined to determine the protocol layers it consists of. This packet is referred to as the current

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 4.1: CST Test Execution as pseudo-code.

```
states_to_test = getAllStates()
paths = getAllPaths()

# Part 1: Preparation
# Initialize all paths to states
for state in states_to_test:
  state_paths[state] = getAllPathsToState(state)

# Part 2: Path validation
for state in states_to_test:
  for path in state_paths[state]:
    state_machine.setState("SCANNING")
    state_to_test = state
    current_state = state_machine.getState()
    tested_packets = []
    test_cases = []

    # Send normal packets until state_to_test is reached
    while current_state != state_to_test and tries < MAX_RETRIES:
      send(state_machine.getNextPacket())
      current_state = state_machine.getState()

      if current_state == state_to_test:
        markPathAsValid(path)

      # Part 3: CA generation
      # if the path is valid, check for new packets
      if isPathValid(path):
        packet_to_test = state_machine.getNextPacket()

        # if new packet is found, generate new CAs
        if packet_to_test not in tested_packets:
          test_cases = generateNestedCA(packet_to_test)
      else:
        continue

      # Part 4: Test Execution
      while test_cases not empty:
        state_machine.setState("SCANNING")
        while current_state != state_to_test:
          send(state_machine.getNextPacket())
          current_state = state_machine.getState()
          if current_state == state_to_test:
            new_packet = state_machine.getNextPacket()
            while new_packet != packet_to_test:
              send(new_packet)
              new_packet = state_machine.getNextPacket()
            replacePacketValuesWithCaValues(new_packet, test_cases.pop())
        repeat:
          send(state_machine.getNextPacket())
        until: timeout or state_machine.getState() == "SCANNING"

        # Save test data with oracle results
        saveTestData()

      tested_packets.append(packet_to_test)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

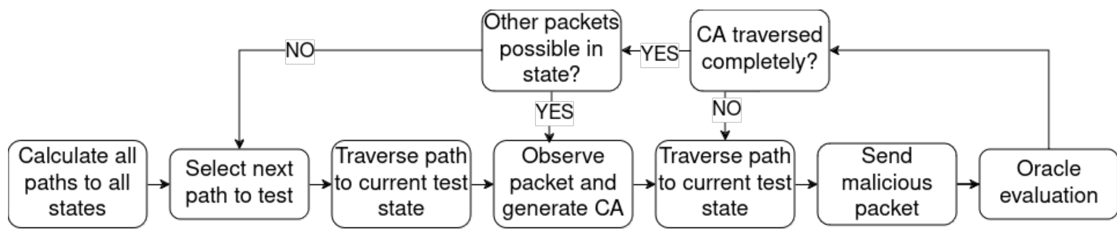

Figure 4.3: Overview of combinatorial testing strategy.

packet to test. The CST component then generates one CA for each individual layer of this packet, along with a meta CA that combines all layers. This meta CA is then used for further testing, with each row in the CA representing a distinct test case, specifying values for the various fields within the current packet. Further details on the CA generation process are provided in Section 4.3.

From this point onward, whenever we reach the current state to test, we select a previously unused row in the CA and modify the layers values of the current packet according to the values in the row of the CA, to create a modified packet (i.e. a malicious packet). This malicious packet is then sent to the SUT device. We then continue transmitting unmodified packets until either an error occurs or the system returns to the initial state (the `SCANNING` state), which indicates the end of the test. Once a test is finished, the log data and oracle results are saved to a database.

Figure 4.1 illustrates an example where the `FEATURE_REQ` state is selected as the current state to test. To traverse through the state machine unmodified packets (denoted as *Normal Packet*) are transmitted, except for the case in which the current state to test is reached. For the transition from the state to test to the next we send a malicious packet, which contains invalid values and is denoted as *Malicious Packet*.

When we reach the end of the current CA, we check whether there are additional packets sent in the same state, following the one we just tested. If such packets exist, we generate a new CA for the next packet and follow the same procedure. If not, we proceed along a different path until all valid paths to all states have been explored and tested.

Since some of the issues discovered by the GreyHound fuzzer require the sending of packets out of order or multiple malicious packets in one test, we also tried to extend our testing strategy to support such features. One attempt was to use tests that lead to a valid response from the peripheral as additional paths to test, including the malicious packet that was sent at the end. Initial test runs have shown that this approach leads to intractable testing times of one week or more per device. Because of this, we decided to use the simpler approach described before and do not support the sending of multiple malicious packets or out of order sending for now. To tackle this problem, more advanced weighted sequence modeling methods are needed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.2 Input Modeling

Creating CAs for testing starts with modeling the input space, which is often one of the more difficult and time consuming aspects of CT. This process typically involves manual effort to determine all relevant parameters and their possible values. A common way to identify parameters is by looking for variables that may influence the behavior of the SUT. Each of these variables can then be considered a parameter in the IPM. After that, existing methods such as boundary value analysis and classification tree can be used to identify the values of each parameter of the model [BGL+13]. Most of the time, IPMs are given as a grammar in Backus-Naur-Form (BNF). In our case, we represent the IPMs as Python dictionaries, with the keys corresponding to IPM parameters and the values being lists containing possible values.

Additionally to the IPM, the structure of the input and the relationship of components also often play a role and have to be considered when modeling the input space. In the past, there have been attempts to tackle this issue in various ways. The authors of [BGL+13] considered two different structure models for their approach, i.e. flat and graph. The flat structure has no hierarchy and treats all parameters equally. This simple structure can be used to represent a program's command-line options, where all options have equal priority and consist of single components, while a graph structure captures compositional relationships between components, allowing one component to consist of multiple sub-components. This approach is useful for modeling hierarchical data, such as elements within an XML file, where an element can contain multiple sub elements.

Another work that used structure modeling is [KS17]. Here the authors compare three different structure models, `flat`, `inter` and `intra` for testing TLS certificates. TLS certificate components are separated into a block of mandatory parameters and blocks of extension parameters. The `inter` modeling strategy is concerned with testing combinations of one parameter from one block with all parameters of a different block. For each parameter and each block which does not contain the parameter, a new relation containing the parameter and all parameters of the selected block with strength t is created. This means that all interactions between parameters from this relation with strength t are covered. The `intra` strategy on the other hand tests interactions between parameters inside of one block. Every block has a specified strength $t > 1$, while the interaction between blocks is set to 1, which means no interactions have to be covered. This is useful to investigate if triggering errors in certificates can be localized within a specific block and treats blocks as independent from each other, which leads to a reduction of the total number of test cases.

In this work, we regard the protocol with its multiple layers as a composite system. Therefore we model the individual layers as blocks of a packet and use their respective fields as our parameters, as they are encoded in the BLE scapy extension contained in the GreyHound framework implementation [GWC+20]. For each such field/parameter, we define a set of possible values. To this end, we examine both the source code of the GreyHound framework and the BLE specification [blu14], taking into account boundary

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
BTLE_DATA_IPM = {
    "LLID": ["~0", "~1", "~2", "~3", "UNCHANGED"],
    "SN": ["~0", "~1", "UNCHANGED"],
    "NESN": ["~0", "~1", "UNCHANGED"],
    "MD": ["~0", "~1", "UNCHANGED"],
    "RFU": ["~0", "~1", "UNCHANGED"],
    "len": ["~0x00", "RECALCULATE", "~LARGER", "~SMALLER",
     ↪  "~0xff"]
}
```

Listing 1: Example of an IPM for a BLE data layer.

values across different ranges as discussed in [Rei97]. We also introduce the symbolic values "LARGER", "SMALLER" or "UNCHANGED" to the parameter values which, when applied, instruct the testing tool to increment, decrement, or retain the original value of a field, respectively. Additionally, for length-related parameters, we include the symbolic value "RECALCULATE", which tells our tool to recalculate the length value based on the current packet content.

A separate IPM is constructed for each layer of the BLE protocol similar to the block modeling approach by [KS17]. This layered strategy helps to mitigate the combinatorial explosion that occurs, when individual parameters have an excessive number of possible values. Additionally, this also prevents the creation of packets with too many invalid values, which could mask some errors due to early test rejection.

It is important to take care when adding invalid values for a particular parameter, such as values that are out of range or not permitted for a specific field. The concept of negative value testing, as described by the authors of PICT [Cor22], recommends, that only a maximum of one such value be present in a row in the CA. Following this approach can prevent the masking of faults, as the presence of multiple invalid values in a single test case can obscure the effects of the others due to early rejection in the validation logic. In PICT, such invalid values can be denoted by prefixing them with a "∼" symbol. Before using our current models with marked negative values, we ran into the problem of not finding some known vulnerabilities that the SweynTooth project reported, which most likely happened because the invalid packets got rejected, before an error could be triggered. The marking of negative values not only helps to avoid the creation of messages that are altered too much, but also to reduce the amount of generated tests.

An example of an IPM for a BLE data layer is given in Listing 1. In this case, where nearly all values are marked as invalid, the benefits of CAs are not apparent, but there are other IPMs, e.g. for the `BTLE_CONNECT_REQ` layer, which have parameters with multiple valid values as can be seen in 2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
BTLE_CONNECT_REQ_IPM = {
    "InitA": ["UNCHANGED", "~__SLAVE_ADDR__"],
    "AdvA": ["UNCHANGED", "~__MASTER_ADDR__"],
    "AA": ["UNCHANGED", "0x8e89bed6", "~0xffffffff",
     ↪  "~0x00000000"],
    "crc_init": ["UNCHANGED", "0", "0xffffff"],
    "win_size": ["UNCHANGED", "0", "2", "0xf", "~0xff",],
    "win_offset": ["UNCHANGED", "0", "1", "2", "~0xffff"],
    "interval": ["UNCHANGED", "0", "16", "32", "~0xffff"],
    "latency": ["UNCHANGED", "0", "1", "10", "~500", "~501",
     ↪  "~0xffff"],
    "timeout": ["UNCHANGED", "~0", "16", "32", "~0xffff"],
    "chM": ["UNCHANGED", "~0x0", "0x11FFFFFFFF",
     ↪  "~0xFFFFFFFFFF"],
    "hop": ["UNCHANGED", "~0", "~4", "7", "~17"],
    "SCA": ["UNCHANGED", "~1"]
}
```

Listing 2: Example of an IPM for a BTLE_CONNECT_REQ layer.

4.3 Covering Array Generation

Combining CAs can be done in various different ways as we have seen from the works on modeling input structure [KS17, BGL+13]. A naive approach would be to exhaustively combine every row of one CA with every other row of another CA. This can be done without any specialized tools, but most often leads to large test sets. CA sizes are an important factor, since our current testing strategy already took roughly one day per device to execute all tests and previous modeling strategies we tried lead to even larger CAs, which were impractical for testing, since it would take an unreasonable amount of time. Other approaches, which we are following here are using CAs to generate IPMs to model sub-components, which are then used to generate a new CA representing the composite system.

In our approach, the CA generation occurs dynamically during the execution of our CST testing tool. For each new packet, where the layer structure differs from the structure of previously seen packets, we are using the PICT tool to generate individual CAs of strength $t = 2$ for each layer of the packet. As input for PICT, we utilize the layer IPMs described in Section 4.2.

At this stage, we have a collection of layer-specific CAs that need to be merged into a single, unified array. To achieve this, we adopt a method proposed by Kampel et al. [KGS17], which is designed for modeling composed software systems. First, we determine the number of rows in each of the layer CAs. Then we construct an IPM (referred to as *meta IPM*), where each parameter corresponds to a layer CA and takes

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

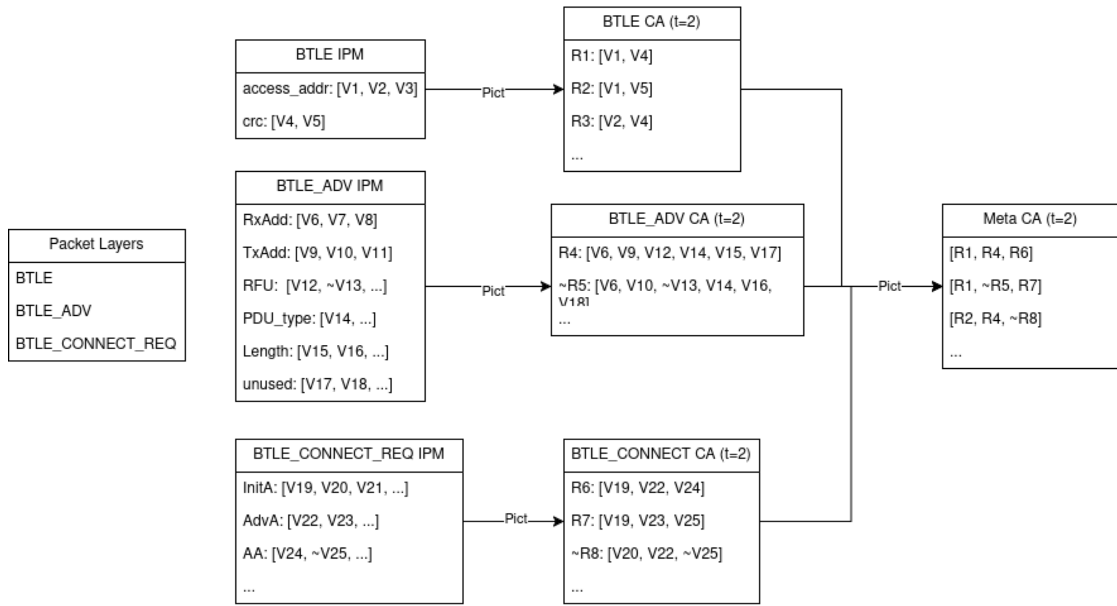

Figure 4.4: Example of CA composition for BLE connection requests.

values between 1 and the total number of rows. To avoid masking effects across layers, any index referring to a row containing negative values is prefixed with the "$\sim$" symbol. The resulting meta IPM is then used to generate a meta CA of strength $t = 2$, where each row is referencing one specific row from each layer CA. By merging the contents of the referenced rows, we get a single, combined meta CA that can be used in our previously described testing workflow. Figure 4.4 illustrates this process and the structure of the different CAs and IPMs. The final meta CA ensures t-way coverage of the input space both within individual layers and across multiple layers, with the level of interaction being adjustable by increasing the strength parameter t at the cost of potentially larger test sets and increased execution time.

4.4 Oracles

To check whether the malicious packets generated by our method caused the SUT to crash, we developed two oracles. The first oracle uses a secondary NRF52840 dongle running the same custom firmware from GreyHound, to monitor packets transmitted by the slave device (the SUT). If no packets are received for more than three seconds, the oracle considers the SUT as unresponsive and marks the test as failed. Conversely, if the SUT returns to its initial state without experiencing this timeout, the test is marked as successful. This oracle is designed to identify cases where the slave device becomes unresponsive for an unusual amount of time. In such scenarios, the device might reboot after an error was triggered and resume normal operation after a few seconds, or it could enter a state requiring a manual reboot, such as power-cycling via USB, to become

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

responsive again. Further details about the setup are provided in Section 5.1.

Since most development boards we tested include a serial port, we implemented an additional oracle using this interface. When the device under test has a serial port, we connect it to the host PC (either directly over USB or using a USB to UART adapter) and listen for output from the slave device. The captured serial output is scanned for specific keywords such as "trace", "crash" and "dump". If one of those keywords is detected, the oracle marks the test as failed.

The first oracle is generally applicable to all BLE devices. However, the generality of the approach can also pose a potential problem, since slave devices sometimes behave unreliable, which can lead to false positives if the device simply takes longer than usual to enable advertisements again after it disconnects.

The second oracle is less general, as it requires both the presence of a serial port on the test device and debug output that includes specific keywords. All tool chains of the devices we tested provide options to enable debug output, which provides more detailed information over the serial connection allowing for more detailed logging. In our experiments, we only encountered one device that did not produce serial output (i.e. NRF52840).

4.5 Walkthrough Example

To explain the inner workings of our CST approach in more detail, we will present an example of a test case execution and go through the individual steps of finding a path, generating CAs and executing the test. For our example we assume that we have finished testing the *FEATURE_RSP* state and continue from there on.

We start in *Part 2* of the algorithm from Algorithm 4.1 by selecting the next state to test from the set of all states, which in this case would be *LENGTH_REQ*. The algorithm checks for available paths to *LENGTH_REQ* in the set of all paths that was generated in the beginning when the tool was started and selects one. For the sake of simplicity, we assume that one of the shortest paths was chosen first, which for our example is:

SCANNING → *INITIATING* → *FEATURE_RSP* → *LENGTH_REQ*.

The tool now tries to reach the *LENGTH_REQ* state using the given path, which would show that the path is valid. At first, all states and transitions are removed from the state machine and we add the states and transitions from the new path. Additionally, we add all paths and states that can be taken from *LENGTH_REQ* to the state *DISCONNECT*. After that, the state machine is set to the *SCANNING* state and the lists of already tested packets and test cases are cleared.

The tool now traverses from one state to the next, sending and receiving normal packets trying to reach the *LENGTH_REQ* state. If the path is valid and the *LENGTH_REQ* state is reached, *Part 3* of the algorithm begins and the next packet generated by the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

state machine is observed. In this case, it is a *ll_length_request*, with the following layer structure:

[BTLE, BTLE_DATA, CtrlPDU, LL_LENGTH_REQ]

Since a packet of this structure is not in the list of already tested packets, it is set as the *packet to test* and the CA generation process starts. First the tool checks if CAs for the individual layers already exist. If not, the IPMs for the layers (i.e. BTLE [Listing 4], BTLE_DATA [Listing 1], CtrlPDU [Listing 5], LL_LENGTH_REQ [Listing 3]) are processed one by one by PICT and a CA of strength 2 is generated for each of them. The number of rows of the resulting CAs are:

- BTLE: 3
- BTLE_DATA: 17
- CtrlPDU: 33
- LL_LENGTH_REQ: 38
- Meta CA: 289

Following that, the CAs of the layers are used as sub-components for a meta CA as described in Section 4.3. The first eight rows of the resulting meta CA is given in 6. This meta CA is used as a test set for the packet we just discovered, with every row of the CA representing a test case. The meta CA only contains 289 rows, which is a reduction of 99.97% compared to exhaustively testing all parameter combinations, which would lead to $3 * 1 * 5 * 3 * 3 * 3 * 3 * 5 * 33 * 5 * 5 * 5 * 5 = 125,296,875$ rows.

Since the CA generation process can take a couple of seconds and the BLE protocol has strict time constraints, in *Part 4* of the algorithm, the state machine is set back to the *SCANNING* state and the path is traversed again, before any tests are executed. When we reach the *LENGTH_REQ* state again, it is finally time to execute the first test. The next packet from the state machine is observed and its structure is compared to the one of the current packet to test. If they are the same, the next row of the CA is used to replace the respective values in the packet with the values from the CA row.

An example of a CA row of this structure is given in Listing 7. Like most rows in our CAs, there are a lot of values set to *UNCHANGED*. This is due to the fact that our models have many values marked as invalid and only one invalid value is allowed to occur per test. As already mentioned, this is important in the case of BLE testing, since too many invalid values lead to early test rejection, which can lead to errors not being discovered.

The altered packet, which is depicted in Listing 8, is then sent to the peripheral instead of the original. As we can see, the values have been replaced with the values listed in the CA row in Listing 7, with *max_rx_bytes* and *max_tx_bytes* being set to 27 or

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
LL_LENGTH_REQ_IPM = {
    "max_rx_bytes": ["UNCHANGED", "~0", "27", "~252", "251"],
    "max_rx_time": ["UNCHANGED", "~0", "216", "~2121", "2120"],
    "max_tx_bytes": ["UNCHANGED", "~0", "27", "~252", "251"],
    "max_tx_time": ["UNCHANGED", "~0", "216", "~2121", "2120"],
}
```

Listing 3: IPM for the *length_request* layer.

```
BTLE_IPM = {
    "access_addr": ["UNCHANGED", "~0xffffffff", "~0x00000000"],
    "crc": ["UNCHANGED"]
}
```

Listing 4: IPM for the *BTLE* layer.

```
CtrlPDU_IPM = {'optcode': ['~0x0C', '~0x0B', '~0x0A', '~0x0F',
↪   '~0x0E', '~0x0D', '~0x1D', '~0x1E', '~0x18', '~0x1A',
↪   '~0x1B', '~0x1C', '~0x16', '~0xFF', '~0x17', '~0x19',
↪   'UNCHANGED', '~0x03', '~0x02', '~0x01', '~0x00', '~0x07',
↪   '~0x06', '~0x05', '~0x04', '~0x14', '~0x15', '~0x09',
↪   '~0x08', '~0x10', '~0x11', '~0x12', '~0x13']}
```

Listing 5: IPM for the *Ctrl_PDU* layer.

0x1b in hex, *max_rx_time* and *max_tx_time* taking the value 2120 or 0x848 in hex and the parameters *crc* and *len* being set to *None*, which means that their values are being calculated by Scapy before sending it. The execution then proceeds as normal by sending and receiving normal packets until we either reach the *DISCONNECT* state or the peripheral stops responding, which marks the end of the execution of this test.

Finally, the tool waits for up to five seconds to see if the peripheral starts to send advertisements again, to see if it is still operational. If advertisements of the target are detected or the five seconds have passed, the details of the test case are logged to the database and we continue with the next test, until all rows of the CA have been used to generate a packet. If the end of the CA has been reached, the packet is added to the list of tested packets and the tool checks for other packets that are sent in this state.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
BTLE; BTLE_DATA; CtrlPDU; LL_LENGTH_REQ
[crc:UNCHANGED, access_addr:UNCHANGED]; [MD:UNCHANGED,
↪ RFU:UNCHANGED, len:RECALCULATE, NESN:UNCHANGED,
↪ LLID:UNCHANGED, SN:UNCHANGED]; [optcode:UNCHANGED];
↪ [max_tx_time:2120, max_rx_time:2120, max_tx_bytes:27,
↪ max_rx_bytes:27]
[crc:UNCHANGED, access_addr:UNCHANGED]; [MD:UNCHANGED,
↪ RFU:UNCHANGED, len:RECALCULATE, NESN:UNCHANGED,
↪ LLID:UNCHANGED, SN:UNCHANGED]; [optcode:UNCHANGED];
↪ [max_tx_time:216, max_rx_time:216, max_tx_bytes:UNCHANGED,
↪ max_rx_bytes:251]
[crc:UNCHANGED, access_addr:UNCHANGED]; [MD:UNCHANGED,
↪ RFU:UNCHANGED, len:RECALCULATE, NESN:UNCHANGED,
↪ LLID:UNCHANGED, SN:UNCHANGED]; [optcode:UNCHANGED];
↪ [max_tx_time:2120, max_rx_time:216, max_tx_bytes:251,
↪ max_rx_bytes:UNCHANGED]
[crc:UNCHANGED, access_addr:UNCHANGED]; [MD:UNCHANGED,
↪ RFU:UNCHANGED, len:RECALCULATE, NESN:UNCHANGED,
↪ LLID:UNCHANGED, SN:UNCHANGED]; [optcode:UNCHANGED];
↪ [max_tx_time:UNCHANGED, max_rx_time:UNCHANGED,
↪ max_tx_bytes:251, max_rx_bytes:27]
[crc:UNCHANGED, access_addr:UNCHANGED]; [MD:UNCHANGED,
↪ RFU:UNCHANGED, len:RECALCULATE, NESN:UNCHANGED,
↪ LLID:UNCHANGED, SN:UNCHANGED]; [optcode:UNCHANGED];
↪ [max_tx_time:UNCHANGED, max_rx_time:216,
↪ max_tx_bytes:UNCHANGED, max_rx_bytes:27]
[crc:UNCHANGED, access_addr:UNCHANGED]; [MD:UNCHANGED,
↪ RFU:UNCHANGED, len:RECALCULATE, NESN:UNCHANGED,
↪ LLID:UNCHANGED, SN:UNCHANGED]; [optcode:UNCHANGED];
↪ [max_tx_time:2120, max_rx_time:216, max_tx_bytes:27,
↪ max_rx_bytes:251]
[crc:UNCHANGED, access_addr:UNCHANGED]; [MD:UNCHANGED,
↪ RFU:UNCHANGED, len:RECALCULATE, NESN:UNCHANGED,
↪ LLID:UNCHANGED, SN:UNCHANGED]; [optcode:UNCHANGED];
↪ [max_tx_time:216, max_rx_time:2120, max_tx_bytes:27,
↪ max_rx_bytes:27]
[crc:UNCHANGED, access_addr:UNCHANGED]; [MD:UNCHANGED,
↪ RFU:UNCHANGED, len:RECALCULATE, NESN:UNCHANGED,
↪ LLID:UNCHANGED, SN:UNCHANGED]; [optcode:UNCHANGED];
↪ [max_tx_time:UNCHANGED, max_rx_time:2120, max_tx_bytes:251,
↪ max_rx_bytes:251]
...
```

Listing 6: First 8 lines from the meta CA.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

```
"BTLE": [crc:UNCHANGED, access_addr:UNCHANGED]
"BTLE_DATA": [MD:UNCHANGED, RFU:UNCHANGED, len:RECALCULATE,
↪ NESN:UNCHANGED, LLID:UNCHANGED, SN:UNCHANGED]
"CtrlPDU": [optcode:UNCHANGED]
"LL\_LENGTH\_REQ": [max_tx_time:2120, max_rx_time:2120,
↪ max_tx_bytes:27, max_rx_bytes:27]
```

Listing 7: Example of a CA row.

```
###[ BT4LE ]###
  access_addr= 0x9a328370
  crc        = None
###[ BTLE data header ]###
     RFU        = 0
     MD         = 0
     SN         = 0
     NESN       = 0
     LLID       = control
     len        = None
###[ CtrlPDU ]###
        optcode    = LL_LENGTH_REQ
###[  LL_LENGTH_REQ ]###
           max_rx_bytes= 0x1b
           max_rx_time= 0x848
           max_tx_bytes= 0x1b
           max_tx_time= 0x848
```

Listing 8: Example of an altered BLE *length request* packet.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Test Execution

This Chapter provides a detailed description of the test setup and devices used for the execution of the tests. The first Section 5.1 presents details about the experimental setup and explains how the devices interact with each other. After that, Section 5.2 describes the SUTs and the software that have been used as targets for the tests. Finally in Section 5.3, we discuss some problems we experienced during the testing process.

5.1 Setup

The experimental test setup includes two NRF52840 dongles flashed with the custom firmware of the GreyHound framework and a laptop running Ubuntu 22.0.4 as the host device. The dongles are connected to the host via USB. One is used to send and receive BLE packets from the slave device being tested, while the other passively listens for packets from the slave, serving as one of the test oracles. Our setup is illustrated in a simplified form in Figure 5.1.

Details about executed test cases and their results are stored in a MongoDB instance. Each database entry consists of the CA row used for the test, the packet history recorded during the test, the first received response after the test, the results of both oracles, as well as the current state and path to test. Additionally, we store messages received via the serial port of the BLE slave device for post-analysis.

During testing, we encountered various issues with hardware becoming unresponsive. At times, some of the Bluetooth peripheral devices under test as well as the NRF52840 dongle used for BLE communication crashed or became unresponsive. This may have been caused by the number of invalid values in transmitted packets. Because of this, we needed a way to automatically reset both of the devices. There are a couple of tools like *usbreset* from ubuntu's *usbutils* package and scripts mentioned on the internet, which supposedly have this capability. During initial tests, we found that most solutions either

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

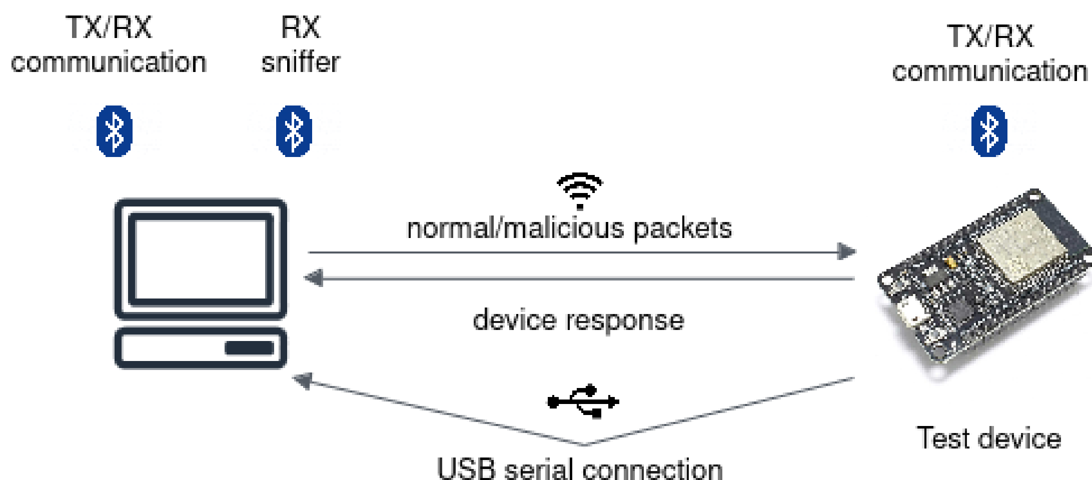

Figure 5.1: Overview of the test setup.

did not work at all or only reset the USB connection to the device but not the device itself.

We briefly considered using the reset pins on the peripheral devices to restart them, but some chips like the NRF52840 did not provide one on their breakout board. In the end, we chose a more generic way by interrupting the power to the devices based on relays that interrupt the positive power line of USB extension cables for a few seconds. The relays are controlled by an ESP32 micro-controller, that is connected to WIFI and exposes a REST API. The testing tool can communicate with the ESP32 using this REST API and is therefore able to control the power supply of the connected devices. This method of power cycling the USB devices allows us to reset them reliably to a working and responsive condition so that we can execute the test again.

Specifically, we used the ESP32 NodeMCU development board, together with a PCF8574 module and a 8-channel relay module. The ESP32 is connected to the PCF8574 module via the SDA and SCL pins, which are used to communicate via I2C, which is a serial communication protocol used to allow multiple integrated circuits (ICs) to exchange messages. Using a library [pcf], we can easily control the state of the digital output pins of the PCF8574 module, which are connected to the input pins of the 8-channel relay module. To get access to the positive wire of the USB connection we modified USB extension cables, by cutting the positive wire and connecting one side in the normally closed (NC) and one side in the middle port of one of the relays. This way, the positive wire is connected until we trigger the relay to switch, which interrupts the power to the USB device. A wiring diagram of the described system is depicted in Figure 5.2.

The devices we tested show significant differences in performance, often taking up to a day to complete all tests. Because of this, we parallelized our setup by replicating it four times in total (i.e. using four host computers, each hosting a SUT device and two

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

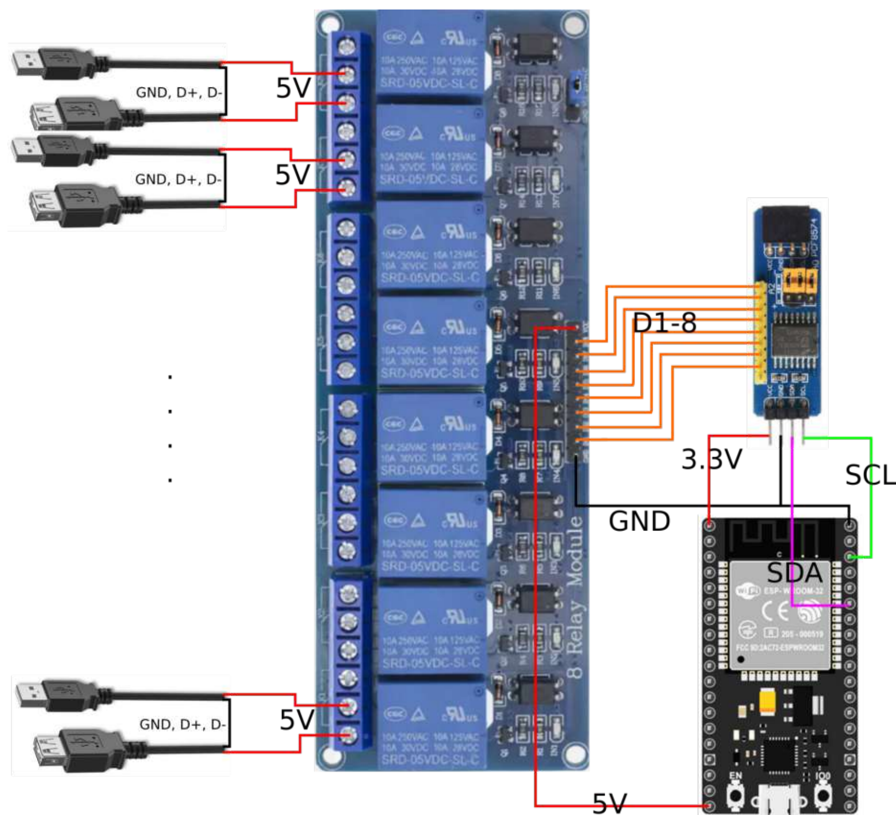

Figure 5.2: A picture of the real world setup during testing.

NRF52840 dongles). An image of the setup during execution is shown in Figure 5.3.

5.2 Target Devices

To test the BLE stack implementation of various BLE chips, it was necessary to acquire a range of BLE chips, ideally integrated into development boards. Development boards usually come in a plug-and-play format, providing easy access to the chip's input and output pins and integrating necessary components like power supply and USB to UART chips, for serial connections. Some devices such as the Texas Instruments CC2640R2 board even provide an integrated debug bridge, which can be used for debugging, instead of a generic JTAG adapter.

For our suite of test devices, we selected ten different BLE devices from different vendors. Table 5.1 shows details of the SDKs and sample applications of the chips we tested. To enable a better performance comparison, our suite also includes chips that were tested before with the GreyHound fuzzer. For these devices, we also examined firmware versions that were more recent than those analyzed in the earlier work, in order to identify whether the issues that were published as part of the SweynTooth set of vulnerabilities had been

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

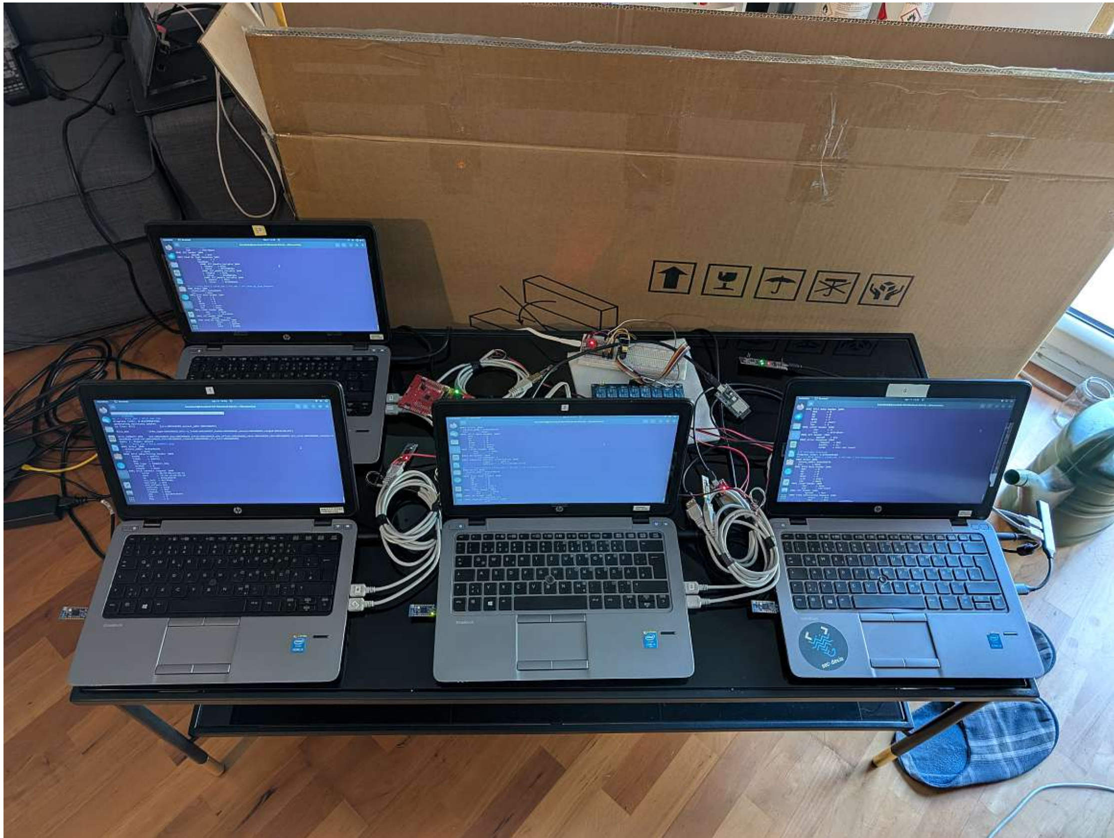

Figure 5.3: A picture of the real world setup during testing.

successfully fixed.

5.3 Challenges

Besides the previously mentioned issue of SUTs and dongles becoming unresponsive or crashing, we faced additional challenges when testing specific devices.

First, we discovered that the Arduino SDKs of the Apollo3 and MG126 chips were still under development at the time we conducted the tests, with their BLE implementations being incomplete. Because of the missing functionality, these devices did not traverse the state machine completely, which led to the low number of executed tests as shown in Table 6.1.

Another issue we encountered was, that for some devices (i.e. ch582m and w801), the onboard USB connectivity was only usable for flashing new firmware to the chip. To get access to the serial output an additional adapter is required. In these cases, we utilized a FT232RL USB to UART adapter to capture the serial output that is required for our second oracle.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

SoC Vendor	SoC Model	SDK Versions	Sample App
Texas Instruments	CC2640R2	3.30.00.20 (old fw), 5.30.00.03 (new fw)	project zero
Espressif Systems	ESP32	4.1 (old fw), 5.0 (new fw)	nimble/bleprphrl
Nordic Semiconductors	nRF52	15.3.0 (old fw), 17.0.1 (new fw)	ble_app_gatts_c
Bouffalolab	bl602	AI-Thinker WB2 beta v1.1.8	ble_slave
WCH	CH582M	MounRiver Studio community v1.50	Peripheral
Hi-Link	W801	a93b517	wm_ble_client_api-_multi_conn_demo
Realtek	RTL8720DN	Realtek Ameba Boards 3.1.6	BLEBatteryService
Silicon Labs	BG22	simplicity studio SV5.7.1.1	bluetooth_controlling-_led_from_smartphone
Ambiq	Apollo3	sparkfun apollo3 boards v2.2.1	LED_Button
MacroGiga Electronics	MG126	seeed SAMD boards 1.8.4	analog_output

Table 5.1: Tested devices and software versions.

In some instances, the testing process was significantly slowed down due to some states not being reliably reachable via certain paths. This issue could be observed while testing the MG126, CC2640R2 and ESP32 chips. Because of this, we implemented a routine that monitors how much time has passed, since the state to test has been reached the last time. If the state to test was not reached in the last 400 seconds, the current path is excluded from further testing and we proceed with the next available path.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Evaluation

In this Chapter we evaluate our test results in depth and compare the performance of our CST tool against the GreyHound Fuzzer. First, Section 6.1 presents the test results of our tool and explains them in more detail. In Section 6.2, we discuss the disclosure process we followed and the experience we had with the different companies. At the end in Section 6.3, we provide a detailed comparison of the amount of issues discovered by the two tools and how effective they were in terms of exploitation rate.

6.1 Results

Due to the instability of some of the tested devices and the relatively high rate of false positives detected by our oracles, we validated test cases that were marked as failed by either of our oracles using a simple script that replays the same inputs and records the results. For further analysis and for the result statistics, only those test cases considered reproducible were included.

Besides the already mentioned instability issues during testing, we also observed reproducible crashes and availability issues in the majority of the devices we evaluated. Table 6.1 summarizes the unique faults identified across the various chips and firmware versions. More details about the SDK versions and example applications are given in Table 5.1. For our analysis, we consider a fault to be unique if the parameters that trigger it have not been identified as part of the trigger of another fault on the same device or if overlapping fields have semantically distinct values according to the BLE specification or an expert security analyst's opinion (e.g. a change in a length field's value from 2453 to 2454, would typically not be considered to be semantically different, while a difference in the packet's `TYPE` value would indicate a separate fault).

The *TO* (Timeout) column of Table 6.1 lists the number of test cases where the SUT became unresponsive for at least four seconds but recovered on its own within at most

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Target	TO	URWF	URN	Dump	Tests
CC2640R2 (old fw)	1	4	0	0	1,309
CC2640R2 (new fw)	0	3	0	0	1,188
ESP32 (old fw)	0	1	0	1	2,115
ESP32 (new fw)	0	2	0	0	1,910
nRF52 (old fw)	0	0	0	0	1,309
nRF52 (new fw)	0	0	0	0	1,309
bl602	0	0	1	1	2,985
CH582M	0	0	0	0	1,184
W801	0	0	1	0	1,132
RTL8720DN	4	0	1	0	2,664
BG22	0	0	0	0	1,282
Apollo3	0	0	1	0	594
MG126	2	0	0	0	373

Table 6.1: Identified unique faults per device and software version.
Legend: TO - Timeout, URWF - Unavailable but Recoverable With Code Fix, URN - Unavailable Reset Needed, Dump - Core dump detected , Tests - Total number of executed tests

two minutes without external intervention. The *URWF* (Unavailable but Recoverable With Code Fix) column captures instances in which the SUT became unresponsive and required a restart when running unmodified example code. In these cases, the BLE stack remained responsive to GAP and HCI commands and it was possible to modify the application code to re-enable advertisements. The *URN* (Unavailable Reset Needed) column reflects scenarios where the SUT became unresponsive after receiving a malicious packet and could not recover without a reset. Attempts to re-enable advertisements by modifying the example code were unsuccessful, as BLE functions returned errors or behaved abnormally. The *Dump* column shows the number of times that a SUT sent a core dump to the host PC via the serial interface, followed by a reboot. At last, the column *Tests*, shows how many tests were executed against each SUT.

A more detailed overview of the errors is given in Table 6.2, which lists the individual errors together with the parameters and values that were involved. Column *Issue* contains abbreviations of the type of the error as they were described above. The next column *State* lists the state in which the error was triggered. Finally, column *parameter* lists the parameter and the value that was responsible for triggering the fault.

As can be seen in Table 6.2, most timeouts and unavailability issues occurred in the *initiating* state, after a connection request with an invalid parameter was received by the peripheral device. This could be due to the fact that we send a *connection request* in this state, which contains many important parameters that influence the properties of the *link layer* of a connection. The core dump issued by the bl602 chip was the only error we found, that was triggered in the *length request* state, which is part of the initial setup procedure between the central and the peripheral. Similarly, the Apollo3 was the only

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Target	Issue	State	Parameter
CC2640R2 new fw	URWF	initiating	interval: 0xffff
	URWF	initiating	timeout: 0
	URWF	initiating	latency: 0xffff
CC2640R2 old fw	URWF	initiating	interval: 0xffff
	URWF	initiating	timeout: 0
	URWF	initiating	latency: 0xffff
	URWF	initiating	interval: 0
	URWF	initiating	latency: 501
ESP32 new fw	URWF	initiating	AA: 0x00000000
	URWF	initiating	AA: 0xffffffff
ESP32 old fw	core dump	initiating	chM: 0x0
	URWF	initiating	AA: 0x00000000
bl602	URN	initiating	chM: 0
	core dump	length_req	max_rx_bytes: 0
W801	URN	initiating	AA: 0x00000000
RTL8720DN	URN	initiating	latency: 0xffff
	TO	initiating	interval: 0xffff
	TO	initiating	win_offset: 0xffff
	TO	initiating	latency: 500
	TO	initiating	interval: 0
Apollo3	URN	pair_request	authentication: 255
MG126	TO	initiating	win_offset: 0xffff
	TO	initiating	latency: 501

Table 6.2: Individual issues per device.
Legend: TO - Timeout, URWF - Unavailable but Recoverable With Code Fix, URN - Unavailable Reset Needed, core dump - Core dump detected

device that became unresponsive in the *pairing request* state, after receiving a malicious pairing request containing a code for an invalid authentication method. Interestingly, all issues we discovered were triggered by a single invalid value.

In addition to the quantitative assessment above, we also used the available JTAG pins on the individual SUTs together with openOCD [ope], GDB [gdb] and the FT232H USB to JTAG adapter, to attach a debugger to the software running on the micro controller and gather more insight into the identified errors. The only micro controller that came with a debug controller integrated into the development board was the CC2640R2 from Texas Instruments, which means it supported JTAG debugging using the provided IDE without an additional adapter.

Debugging software running on micro controllers is a challenging task, especially when it comes to timing sensitive protocols like BLE. To debug software with a debugger like GDB, it is necessary to set breakpoints or at least pause the execution in a convenient

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

moment, so that the currently running code can be stepped through and the memory content analysed. The problem with most protocols and also BLE is, that they require a constant exchange of packets to keep a connection established. In a debug scenario, when we hit a breakpoint or otherwise pause the running software, the connection gets interrupted, which ultimately alters the execution path of the binary being paused.

To get the debug process started we had to connect the JTAG pins of the microcontrollers to the FT232H adapter and start a JTAG connection using openOCD. The appropriate openOCD configuration files for the JTAG adapter and the different microcontrollers can be found in the config directory of openOCD and are supplied via command line arguments. If the JTAG connection is established, we can use the GDB implementation shipped by the vendor of the microcontroller to start the firmware binary and attach the debugger to the process.

We started our analysis by starting a binary as described with GDB and executed one of the failure inducing tests. When the peripheral became unresponsive, we paused the execution and noted down the address of the currently executed code for later use. It turned out that most peripherals were stuck in an event processing loop, waiting for BLE events that would trigger further actions. Attempts to step further through the code lead into a section of pure assembly code, which turned out to be the proprietary binary blob of the BLE firmware provided by the vendor. This part is stripped from any debug information, which makes it especially difficult to debug. If we step through the code in single steps, we trigger a lot of system timeouts, which alter the execution path and often lead from one timeout handler to the next. Another thing we tried was continuing the execution until it reached the point it was stuck in again. When doing this, no additional events were processed and we ended up at the same address as before, waiting for BLE events. The Apollo3 chip was the only one behaving a bit differently. Instead of being stuck in the event loop, this time the application tried to execute the authentication method of the binary blob, which never returns.

Using the address where the peripheral stopped, we used the reverse engineering tool Ghidra [webb], to get further insight into the inner workings of the firmware. Unfortunately, Ghidra was not able to decode the stripped part of the binary, Initial approaches of finding suitable breakpoints using binary analysis and stopping the process before it gets stuck in the event loop turned out unsuccessful by either failing to trigger the error or leading to the same situation as before, without gaining new information.

Our analysis shows that most errors lead to similar behavior of the tested devices. When a slave device receives a malicious connection request, it fails to properly validate the connection parameters and incorrectly processes the packet as valid. However, since at least one parameter is invalid, the BLE central terminates the connection and stops communicating with the slave device, while the slave device remains stuck in a state where it waits indefinitely for events from the BLE stack.

In this state, the slave devices become effectively unusable until a reset is performed. No valid connection is established and other devices cannot discover them because

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

advertisements cannot be re-enabled. The specific impact varies by SUT, depending on the manipulated field and value in the malicious packet. The various effects are reflected in different BLE function error codes when attempting to restart advertisements, as well as in the sequence of GAP events. Our findings were independently confirmed by Espressif for the ESP32 during the disclosure procedure.

According to the BLE specification, a device should terminate a connection based on the following formula:

$EffectiveConnectionInterval = (ConnectionInterval) * (1 + [SlaveLatency])$ [Ins16].

Additionally, the `EffectiveConnectionInterval` must not exceed a maximum of 16 seconds if no packets are received anymore. In our tests, the majority of devices did not drop the connection and were not able to reactivate advertisements after receiving a malicious packet. At the same time, applications received a lower number of GAP events from the BLE stack compared to normal connections. This might suggest a problem with the parameter validation and state transition in the GAP implementation, which could cause further corruption of the BLE stack when invalid parameters are handled. Only the MG126 and RTL8720DN were able to automatically recover in some of these cases. However, they were often unreachable for more than a minute, resulting in denial of service (DoS) conditions.

The majority of identified faults lead to the target becoming unresponsive which can be exploited to carry out denial of service (DoS) attacks. The source of these issues is either the BLE firmware or the sample application running on the SUT. Inputs that cause devices to send core dumps suggest memory access violations, which could be exploited to gain control of the SUT or expose cryptographic keys or other confidential information. To enable a coordinated responsible disclosure process, we informed the vendors of all affected devices, as detailed in the next Section. Since the 90 day non-disclosure period already passed, we decided to provide all details about the issues in Table 6.2.

With all the information presented so far, we can already answer our first research question (**RQ1**): "Can CST be applied to test devices using the BLE protocol?" Given all the results in this section gathered by the execution of our tool, together with the modeling and testing strategy discussed in Chapter 4, we can definitely say that CST can be applied to test devices using the BLE protocol.

The second question we posed (**RQ2**): "Is it possible to find vulnerabilities in BLE using CST?", can also be answered with a definite yes. Our CST approach was not only able to find vulnerabilities previously discovered by the GreyHound fuzzer, but could also find new issues that have not been found before, which clearly shows that CST can find vulnerabilities in the BLE protocol.

6.2 Responsible Disclosure

To guarantee a responsible and coordinated disclosure process, we notified the vendors of all affected devices 90 days in advance. As of now, only Espressif and Realtek have

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

replied. Espressif acknowledged the issues and provided fixes for all problems within the affected software versions, with plans to assign CVEs soon. Notably, Espressif was the only vendor we contacted that maintained a dedicated bug bounty program for software vulnerabilities. For the disclosure of the vulnerability affecting the access address, we were awarded a bounty of 2,229$.

After Realtek requested more information about the reported issues, we have provided them with all the necessary information to reproduce the issues, but have not received any response since.

Some vendors proposed that we post our test results to the public support forum. We pointed out that publicly disclosing potential security issues is not a good idea and that this would go against our 90-day non disclosure policy, but did not get any response after that.

Since the 90-day disclosure period has ended, we have made the proof-of-concept code available to demonstrate how the vulnerabilities can be replicated. The example code, along with MongoDB database exports of the test results, can be downloaded at `https://zenodo.org/records/14647110`.

6.3 Comparison

To compare our combinatorial testing method against the performance of the original tool, we executed the unaltered fuzzer as it is used in the original work [GWC+20], against our set of test devices. Since the effectiveness of fuzzing is partly influenced by execution time and our method required in the worst case just under one day to complete testing for a single SUT, we let the fuzzer run for 24 hours against each device. After that, we compared the reported results to those of our CST method. A summary of the results is given in Table 6.3.

In contrast to our oracles, the oracle used by the GreyHound fuzzer is more complex, checking not only for timeouts and crashes (listed in column *Issues* of Table 6.3), but also for deviations from the BLE specification, such as BLE devices responding to packets in states in which they should not or accepting invalid packets. Such deviations are referred to by the authors as anomalies and are listed separately in column *Anomalies*.

Upon manual analysis of the fuzzer results, we discovered that many of the issues found were not reproducible and were therefore considered false positives. This was also the reason why we decided to develop our own oracles, which we described in detail in Section 4.4 and did not check for anomalies with our approach. Because of this, there are also no anomalies reported for our approach in Table 6.3. Due to the amount of false positives that were detected, we created a script to replay test cases marked as failed by the GreyHound fuzzer's oracle and only included those that were reproducible, as we have done with the results of our CST tool. We also provide a measurement of the exploitation rate (ER) in column *ER*, which is defined as the amount of failed test

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

	Issues		Anomalies		Tests		ER	
Target	GH	CST	GH	CST	GH	CST	GH	CST
CC2640R2 (new fw)	0	**3**	0	-	2,129	**1,188**	0	**0.002525**
ESP32 (new fw)	0	**2**	1	-	5,979	**1,910**	0.000167	**0.001047**
nRF52 (new fw)	0	0	1	-	8,790	**1,309**	**0.000113**	0
bl602	1	**2**	2	-	7,233	**2,985**	0.000415	**0.000670**
CH582M	0	0	2	-	4,237	**1,184**	**0.000472**	0
W801	0	**1**	1	-	**879**	1,132	**0.001138**	0.000883
RTL8720DN	0	**5**	2	-	10,153	**2,664**	0.000197	**0.001877**
BG22	0	0	0	-	8,016	**1,282**	0	0
Apollo3	**1**	**1**	0	-	7,935	**594**	0.000126	**0.001684**
MG126	0	**2**	0	-	9,993	**373**	0	**0.005362**

Table 6.3: Comparison of combinatorial security testing (CST) versus GreyHound (GH) fuzzer.

cases divided by the number of total executed tests. This metric was first introduced in [BGK+15] and is since commonly used in the literature.

The column *Tests* contains the total number of test cases (i.e. BLE sessions) executed against a specific target. As we can see, the fuzzer most of the time executed a lot more tests than our tool in the same amount of time, except when testing the W801 where it was acting unreliable, leading to a lower number of test iterations. The low execution rate of our CST tool is predominantly caused by the time required to identify unreliable paths as we already mentioned in Section 5.3). Additionally, the NRF52 dongle and peripheral devices became unresponsive sometimes, requiring a reset. While our approach implements a reset method, that can interrupt the power of the NRF52 dongle and the SUT (see Section 5.1), the original work only includes a software reset, which might not work if a device becomes completely unresponsive.

Method	Total Tests	Total Unique Issues	ER
GreyHound	68,344	17	0.000249
CST	**19,354**	**19**	**0.000982**

Table 6.4: Comparison of total identified faults.

Overall, the CST approach presented in this work was able to find more unique faults than the GreyHound fuzzer in the different firmware versions we tested, which is also summarized in Table 6.4, but did not find any anomalies. In total, the fuzzer executed more than three times the amount of tests in the same time as our CST tool, but found 2 unique errors less. For the ER this means that our tool was nearly four times more efficient, when it comes to executing tests that actually trigger a fault. Interestingly, three

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Target	False Positives (FP) GH	CST
CC2640R2 (new fw)	27	**20**
ESP32 (new fw)	28	**10**
nRF52 (new fw)	19	**0**
bl602	18	**1**
CH582M	31	**7**
W801	**7**	15
RTL8720DN	26	**8**
BG22	9	**3**
Apollo3	**0**	8
MG126	28	**7**

Table 6.5: Comparison of false positives (FP) of combinatorial security testing (CST) versus GreyHound (GH) fuzzer oracles.

issues identified in the old version of the CC2640R2 were still present in the later version. For the ESP32, one issue was fixed in version 5.0, but a new URN was introduced.

When looking a bit closer at the results of the fuzzer, we noticed that all issues were found within the first 600 iterations. This means that it would have found the same amount of errors, even if it would have been running for only 1-2 hours. This in turn would have improved the ER a lot and probably beat our CST approach in that aspect. On the other hand, since most of the issues found by our tool were found in the initiating state, a similar argument could be made for our tool, with an earlier test termination leading to an improved ER. This is a classic issue of fuzzers in general, that there is no clear point where testing can be considered finished, while our CST approach is considered finished, when all tests are executed and at least provides a mathematical coverage guarantee over the input models.

The information provided in this section provides us with an answer to the last two research questions. First (**RQ3**): "How many errors can we find with CST compared to fuzzing with the GreyHound fuzzer?". As can be seen in Table 6.4, our CST approach managed to find 19 unique faults, while the GreyHound fuzzer only managed to find a total of 17.

The last question (**RQ4**): "How efficient is the CST approach compared to the GreyHound fuzzer when it comes to exploitation rate?", can also be answered by looking at Table 6.4. Considering only unique faults, our approach reached an ER of 0.000982, while the original fuzzer only reached an ER of 0.000249, when being executed for 24 hours, which means, that our CST tool was better equipped to generate fault inducing tests.

To illustrate the difficulties of developing reliable oracles, Table 6.5 provides an overview of the amount of false positives detected during testing. Since the oracle of the original fuzzer is a lot more complex than our oracles, which simply check for crashes and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

availability, it is also more likely to detect false positives in case of unforeseen delays during execution. This can be seen by the high number of false positives, where the oracle marked the tests incorrectly as failing. In many of those cases, the oracle detected a message as received in an incorrect state, but since we were not able to reproduce this behavior, it was most likely just an unexpected execution delay in the host or peripheral. Despite the simplicity of our oracles, they also detected many false positives, even though in lesser amounts.

Re-testing

The authors of the original work uncovered several issues, which they subsequently reported to the affected vendors. As the results from Table 6.3 and Table 6.6 show, most of the old bugs have been fixed in current firmware versions of the CC2640R2, ESP32 and nRF52. In order to provide a more direct comparison to the fuzzer, we reevaluated the three devices from our set of SUTs that were also previously tested by the GreyHound fuzzer with our CST method, using the outdated firmware versions as reported in the original publication.

	Issues		Anomalies		Tests		ER	
Target	GH	CST	GH	CST	GH	CST	GH	CST
CC2640R2 (old fw)	2	**5**	1	-	**1,000**	1,309	0.003	**0.00382**
ESP32 (old fw)	**2**	**2**	0	-	**1,000**	2,115	**0.002**	0.000946
nRF52 (old fw)	0	0	1	-	**1,000**	1,309	**0.001**	0

Table 6.6: Comparison of CST versus GreyHound fuzzer applied to old firmware versions.

The results of the original work compared to those of our CST tool executed against the old firmware versions are summarized in Table 6.6. Although our method detected more faults in total, results differ across individual devices. One significant limitation of our current method is that it does not alter the order of protocol messages. A feature that the GreyHound fuzzer supports and that led to the discovery of various issues. Because of this, we were unable to find all of the faults reported in the original work.

For the ESP32 chip with old firmware, one of the issues found by the CST method overlaps with the original work, which is that the device crashes when receiving a connection request with the `chM` parameter set to `0`. However, the second fault discovered by GreyHound relies on reordering protocol messages, while the second issue the CST approach detected is triggered simply by altering message parameters.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Summary

This Chapter summarizes the key findings and contributions presented in this work, highlighting the core insights. Section 7.1 begins with the conclusion that summarizes the work, results and their implications. Following that, Section 7.2 outlines potential directions for future work, identifying opportunities to extend and build upon the current implementation.

7.1 Conclusion

As we have seen, the complexity (i.e. many states, many layers, many parameters, strict timing requirements, abstraction layers) of the BLE protocol make it a difficult target to test. In this work, we presented a comprehensive CST approach for peripheral implementations of the BLE protocol, to see if CST is applicable to the testing of the BLE protocol and how well it performs. To this end, we discussed different strategies for input modeling, with their benefits and drawbacks and how it can lead to impractical test set sizes in some cases.

During the modeling procedure, we encountered some problems that we had to address by adapting our modeling approach. Our first modeling attempts of creating one large CA from all IPMs lead to very large CAs and the individual tests had too many invalid values, which got most of them to be rejected very early in initial tests. Next, we tried generating CAs for the individual IPMs, but we also wanted to combine them in some way, so they would cover possible interactions between the protocol layers. In the end, we decided on a composite system approach, to generate meta CAs from the individual layer IPMs, but had to settle with strength $t = 2$ for the meta CAs, since they would otherwise again become too large.

Another issue we encountered was that initial tests were not able to reproduce any issues of the GreyHound fuzzer. That was mostly due to the fact that generated packets were

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

rejected very early by the validation logic, because they contained too many invalid values. To solve this issue, we marked invalid values in our model with a "$\sim$" symbol and used the CA generation tool PICT, which makes sure to only include a maximum of one invalid value per CA row.

To show that CST can be applied to the testing of the BLE protocol, we created IPMs of the input space and extended the GreyHound fuzzing framework with a CST component. This component is able to generate test sets (CAs) from the IPMs and further transform them into BLE packets, which represent executable test cases, which are then sent to a SUT over the air. Compared to the probabilistic behavior of fuzzing, our method provides combinatorial coverage guarantees over the input models, while minimizing the required number of test cases, which lead to the discovery of 19 distinct issues across 10 device/firmware combinations. These findings underscore the insufficiency of current BLE testing practices and emphasize the need for more systematic and comprehensive testing approaches in the development and validation of BLE devices.

We also want to point out that only one of the vendors we contacted (i.e. Espressif Systems) actually acknowledged the reported issues and informed us, when patches were available. Others either did not respond or stopped responding after an initial request for more information. Some proposed that we post our results to the public support forum, which we could not do due to our non-disclosure policy. It was surprising that most vendors to this day do not have a proper bug bounty program in place and often do not respond at all.

To evaluate the effectiveness of our approach, we compared the results of our CST approach with the original GreyHound fuzzer implementation. A comparison against the earlier work shows that both methods were able to find errors that the other one could not. In general, the evaluation indicates increased fault detection capabilities of our approach, which is also confirmed by testing the older firmware versions that contain bugs that were previously uncovered by GreyHound. However, our current work does not detect issues that depend on packets arriving in unexpected order or multiple malicious packets to be triggered, since our naive approach of extending paths that have led to valid responses, required too much time to execute. Furthermore, the performance of our test execution is limited by the slow detection of unreliable paths, which needs to be optimized or replaced by a better method. We believe that these issues can be handled with more engineering and modeling effort, which is left for future work.

Finally we want to highlight some difficulties that come with testing of firmware on real microcontrollers. Even though this approach makes the testing tool easily applicable to BLE firmware on all kinds of different CPU architectures, it also has some drawbacks. One issue we often mentioned is the unreliability of the micro controllers, with some of them experiencing processing delays from time to time, which can lead to the oracles incorrectly classifying test results. In the case of testing wireless protocols, the unreliability of wireless transmissions can additionally be problematic.

Another issue with using external devices is, that if they stop responding, which we

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

experienced many times during test execution, the host needs some way of restarting the devices under test. Interestingly, this was not mentioned anywhere in the original work presenting the GreyHound fuzzer. There are many ways this can be achieved, but our method of interrupting the power line of USB cables using relays can be applied to any device using a USB power supply. Even if a device uses a different kind of power source, our method can be used by interrupting the power line of any cable.

When analysing the issues we found, we needed a way to debug the code running on the micro controllers. In the case of the TI Launchpad CC2640R2 from Texas Instruments, this was relatively easy since the development board came with an integrated debug probe. Using the IDE provided by the vendor, we could analyse the running code in the GUI. For the other devices this was more cumbersome, using openOCD together with a JTAG adapter. Every microcontroller needs different configuration files, which are provided by openOCD or the vendor. It is important to make sure that the wire connection between the JTAG adapter and the micro controller is reliable and does not get interrupted. Otherwise the debugger gets detached from the process and we have to start again from the beginning.

Most of the problems that come with using real hardware to test can be alleviated if the firmware could be emulated. This has the drawback, that a tool developed for a specific emulator is restricted to it's supported CPU architectures. Also in the case of BLE or WIFI testing, where the CPU has to interact with peripheral hardware (i.e. RF Module), emulation is often very difficult.

7.2 Future Work

Since our current BLE testing approach does not cover the sending of packets out of order, a natural extension of this work would be the addition of combinatorial sequence modeling and Sequence Covering Arrays (SCAs) [CCHZ13]. Such an approach has already been successfully used to test protocols [GSD+19] and would offer additional coverage guarantees. To deal with the problem of combinatorial explosion that we experienced with some test generation strategies, adding a guiding heuristic with adjustable weights might be advantageous. This would allow testers to focus their resources and prioritize testing of the most interesting components.

Considering that this was the first attempt of applying CST to test the BLE protocol and the issues we described regarding the large sizes of test sets, the modeling and CA combination methods could be refined. Instead of creating one IPM per layer, a template base approach like it is used by TLS-Anvil [MNH+22] could be used. That way testing can be better focused on specific behavior and functions of the protocol and therefore less but more specific tests.

Another future line of research would be the automation of the initial modeling process of the CT pipeline of input formats in general. As already mentioned in Section 4.2, the input modeling process of CT can be difficult and time consuming. In the future, we aim

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

to develop a fully automated approach for reverse-engineering protocol message formats directly from binary implementations, both client-side and server-side. Using the extracted information, we want to generate IPMs for the input space of the analysed message format, automatically inferring parameters, appropriate value ranges and constraints.

To achieve this goal, we want to build upon advanced binary analysis techniques such as those found in [CPC+08]. The authors developed a tool called `Tupni`, which uses dynamic analysis techniques like taint tracing, execution trace recording and control flow graph (CFG) reconstruction to infer message structures, field boundaries and identify semantic data types of protocol messages sent and received by the target binary. Additionally, the signatures of well known functions from common libraries like `libc` are used to get additional information about field boundaries and values types. All this information is then used to output a protocol specification in enhanced BNF form, which can be used to generate, parse or validate messages of the protocol being analyzed. Additionally the output contains constraints for individual fields, to specify properties like length, inter-message dependencies and symbolic predicate constraints.

Since the `Tupni` tool is proprietary closed source software developed by Microsoft and is restricted to the analysis of binaries using x86 instructions, there is still a lot that could be improved. Many protocols (e.g. BLE, ZigBee), especially in the world of IoT, are running on microcontrollers using CPUs with the ARM or RISC-V instruction sets. Therefore it would be beneficial to build a tool around a taint tracing engine having a wide range of support for different instruction sets.

To unify the taint tracing and analysis process, these algorithms could be implemented for an intermediate representation (IR) language like PCode used by the reverse engineering tool Ghidra [webb]. An emulator like Quiling [Webc], which supports a wide variety of CPU architectures could be used to emulate the execution of binaries built for many different systems. The instructions recorded during execution could then be lifted to PCode, which then could be emulated using Ghidras internal PCode emulator for further taint tracing and analysis using similar methods as the `Tupni` tool.

A persistent challenge in this space is the absence of a standardized machine-readable, structured format to describe extracted protocol message formats with all necessary constraints. To address this issue, we want to experiment with the Scapy [webd] framework and its internal protocol and packet representation as classes and subclasses, which was also heavily used by the GreyHound fuzzer framework. Since Scapy already contains implementations for many protocols, we first want to assess what constraints and dependencies can already be modeled with the current functionality of Scapy. Additionally, we want to explore how well the representation format is suited as an output format for our reverse engineering approach described before. A benefit of using Python classes and functions from Scapy to specify protocol structures is that we can utilize Scapy's parsing, constructing and transceiving capabilities that come with it, which are important for test generation and execution and avoid having to implement complementary functionality ourselves.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Some challenges in this line of research, besides the implementation effort, will be to ensure scalability for large or complex protocol implementations. This will be mostly achieved by algorithm optimizations and reducing the collected data to only contain strictly necessary information. Additionally, analyzing obfuscated binaries may pose further difficulties and necessitate hybrid static-dynamic methods to maintain analysis robustness.

Ultimately, this research seeks to advance the state of the art in input format analysis using reverse engineering methods by tightly integrating low-level analysis with formal specification and structured test generation, offering a robust foundation for future automated analysis and security validation of networked systems.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

3.1 NIST study of fault inducing parameter interactions [Weba]. 12
3.2 Covering array example [HWKK15]. 13
3.3 Bluetooth and BLE stack layers [TM24] 15
3.4 Link Layer State Machine [SIG] 16
3.5 Structure of a BLE data packet. [TPHB21] 17
3.6 Message exchange between BLE Devices. [GWC+20] 18
3.7 Simplified overview of BLE protocol model [GWC+20] 21

4.1 Example testing flow. 24
4.2 State machine as implemented by the GreyHound Fuzzer. 24
4.3 Overview of combinatorial testing strategy. 26
4.4 Example of CA composition for BLE connection requests. 30

5.1 Overview of the test setup. 38
5.2 A picture of the real world setup during testing. 39
5.3 A picture of the real world setup during testing. 40

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

5.1 Tested devices and software versions. 41

6.1 Identified unique faults per device and software version. **Legend:** TO - Timeout, URWF - Unavailable but Recoverable With Code Fix, URN - Unavailable Reset Needed, Dump - Core dump detected , Tests - Total number of executed tests . 44
6.2 Individual issues per device. **Legend:** TO - Timeout, URWF - Unavailable but Recoverable With Code Fix, URN - Unavailable Reset Needed, core dump - Core dump detected 45
6.3 Comparison of combinatorial security testing (CST) versus GreyHound (GH) fuzzer. 49
6.4 Comparison of total identified faults. 49
6.5 Comparison of false positives (FP) of combinatorial security testing (CST) versus GreyHound (GH) fuzzer oracles. 50
6.6 Comparison of CST versus GreyHound fuzzer applied to old firmware versions. 51

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Algorithms

4.1 CST Test Execution as pseudo-code. 25

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[afl] Afl fuzzer. https://github.com/google/AFL. [Online; accessed 10-April-2025].

[ATR19] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B. Rasmussen. The KNOB is broken: Exploiting low entropy in the encryption key negotiation of bluetooth BR/EDR. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1047–1061, Santa Clara, CA, August 2019. USENIX Association.

[ATR20] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Bias: bluetooth impersonation attacks. In *2020 IEEE symposium on security and privacy (SP)*, pages 549–562. IEEE, 2020.

[BAAHH22] Arup Barua, Md Abdullah Al Alamin, Md Shohrab Hossain, and Ekram Hossain. Security and privacy threats for bluetooth low energy in iot and wearable devices: A comprehensive survey. *IEEE Open Journal of the Communications Society*, 3:251–281, 2022.

[BGK+15] Josip Bozic, Bernhard Garn, Ioannis Kapsalis, Dimitris Simos, Severin Winkler, and Franz Wotawa. Attack pattern-based combinatorial testing with constraints for web security testing. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 207–212, 2015.

[BGL+13] Mehra N Borazjany, Laleh Sh Ghandehari, Yu Lei, Raghu Kacker, and Rick Kuhn. An input space modeling methodology for combinatorial testing. In *2013 IEEE sixth international conference on software testing, verification and validation workshops*, pages 372–381. IEEE, 2013.

[blu14] Bluetooth specifications and documents. https://www.bluetooth.com/specifications/specs/?keyword=core+specification, 2014. [Online; accessed Jan. 15th, 2024].

[BS23a] Dor Zusman Ben Seri, Gregory Vishnepolsky. Bleedingbit, the hidden attack surface within ble chips. https://media.armis.com/PDFs/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

wp-bleedingbit-ble-chips-en.pdf, 2023. [Online; accessed Jan. 15th, 2024].

[BS23b] Gregory Vishnepolsky Ben Seri. The dangers of bluetooth implementations: Unveiling zero day vulnerabilities and security flaws in modern bluetooth stacks. https://media.armis.com/PDFs/wp-blueborne-bluetooth-vulnerabilities-en.pdf, 2023. [Online; accessed Jan. 15th, 2024].

[BY98] Kevin Burr and William Young. Combinatorial test techniques: Table-based automation, test generation and code coverage. In *Proc. of the Intl. Conf. on Software Testing Analysis & Review*. Citeseer, 1998.

[CCHZ13] Yeow Meng Chee, Charles J Colbourn, Daniel Horsley, and Junling Zhou. Sequence covering arrays. *SIAM Journal on Discrete Mathematics*, 27(4):1844–1861, 2013.

[Col10] Charles J Colbourn. *CRC handbook of combinatorial designs*. CRC press, 2010.

[Cor22] Microsoft Corporation. Pairwise independent combinatorial testing. https://github.com/microsoft/pict, 2022. [Online; accessed April. 17th, 2024].

[CPC+08] Weidong Cui, Marcus Peinado, Karl Chen, Helen J. Wang, and Luis Irun-Briz. Tupni: automatic reverse engineering of input formats. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, page 391–402, New York, NY, USA, 2008. Association for Computing Machinery.

[CPST22] Matthias Cäsar, Tobias Pawelke, Jan Steffan, and Gabriel Terhorst. A survey on bluetooth low energy security and privacy. *Computer Networks*, 205:108712, 2022.

[gat] Gatt server and client roles | gatt protocol | bluetooth le | v5.0 | silicon labs. https://docs.silabs.com/bluetooth/5.0/bluetooth-general-gatt-protocol/. [Online; accessed 10-April-2025].

[GBC+22] Matheus E Garbelini, Vaibhav Bedi, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. {BrakTooth}: Causing havoc on bluetooth link manager via directed fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1025–1042, 2022.

[gdb] Gdb: The gnu project debugger. https://www.gnu.org/savannah-checkouts/gnu/gdb/index.html. [Online; accessed 20-March-2025].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[GSD+19] Bernhard Garn, Dimitris E. Simos, Feng Duan, Yu Lei, Josip Bozic, and Franz Wotawa. Weighted combinatorial sequence testing for the tls protocol. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 46–51, 2019.

[GWC+20] Matheus E. Garbelini, Chundong Wang, Sudipta Chattopadhyay, Sun Sumei, and Ernest Kurniawan. SweynTooth: Unleashing mayhem over bluetooth low energy. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 911–925. USENIX Association, July 2020.

[HWKK15] J. D. Hagar, T. L. Wissink, D. R. Kuhn, and R. N. Kacker. Introducing combinatorial testing in a large organization. *Computer*, 48(4):64–72, 2015.

[Ins16] Texas Instruments. Overview - ble-stack user's guide for bluetooth 4.2 3.01.01.00 documentation. `https://software-dl.ti.com/lprf/simplelink_cc2640r2_latest/docs/blestack/ble_user_guide/html/ble-stack-3.x/gap.html`, 2016. [Online; accessed April. 17th, 2024].

[KBD+15] D Richard Kuhn, Renee Bryce, Feng Duan, Laleh Sh Ghandehari, Yu Lei, and Raghu N Kacker. Combinatorial testing: Theory and practice. *Advances in computers*, 99:1–66, 2015.

[KGS17] Ludwig Kampel, Bernhard Garn, and Dimitris E Simos. Combinatorial methods for modelling composed software systems. In *2017 IEEE international conference on software testing, verification and validation workshops (ICSTW)*, pages 229–238. IEEE, 2017.

[KKL13] D.R. Kuhn, R.N. Kacker, and Y. Lei. *Introduction to Combinatorial Testing*. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series. Taylor & Francis, 2013.

[KS17] Kristoffer Kleine and Dimitris E Simos. Coveringcerts: Combinatorial methods for x. 509 certificate testing. In *2017 IEEE International conference on software testing, verification and validation (ICST)*, pages 69–79. IEEE, 2017.

[MCSH19] Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. Internalblue - bluetooth binary patching and experimentation framework. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, page 79–90, New York, NY, USA, 2019. Association for Computing Machinery.

[MNH+22] Marcel Maehren, Philipp Nieting, Sven Hebrok, Robert Merget, Juraj Somorovsky, and Jörg Schwenk. {TLS-Anvil}: Adapting combinatorial testing for {TLS} libraries. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 215–232, 2022.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[ope] Open on-chip debugger. https://openocd.org/. [Online; accessed 20-March-2025].

[PA22] Andrea Pferscher and Bernhard K. Aichernig. Stateful black-box fuzzing of bluetooth devices using automata learning. In Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez, editors, *NASA Formal Methods*, pages 373–392, Cham, 2022. Springer International Publishing.

[PBR20] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. Aflnet: A greybox fuzzer for network protocols. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 460–465, 2020.

[pcf] Pcf8574 library. https://docs.arduino.cc/libraries/pcf8574-library/. [Online; accessed 20-April-2025].

[pro] Proverif. https://bblanche.gitlabpages.inria.fr/proverif/. [Online; accessed 15-April-2025].

[Rei97] Stuart C Reid. An empirical analysis of equivalence partitioning, boundary value analysis and random testing. In *Proceedings fourth international software metrics symposium*, pages 64–73. IEEE, 1997.

[SBD+17] Dimitris E. Simos, Josip Bozic, Feng Duan, Bernhard Garn, Kristoffer Kleine, Yu Lei, and Franz Wotawa. Testing tls using combinatorial methods and execution framework. In Nina Yevtushenko, Ana Rosa Cavalli, and Hüsnü Yenigün, editors, *Testing Software and Systems*, pages 162–177, Cham, 2017. Springer International Publishing.

[SBG+19] Dimitris E. Simos, Josip Bozic, Bernhard Garn, Manuel Leithner, Feng Duan, Kristoffer Kleine, Yu Lei, and Franz Wotawa. Testing tls using planning-based combinatorial methods and execution framework. *Software Quality Journal*, 27(2):703–729, June 2019.

[SGC24] Zewen Shang, Matheus E. Garbelini, and Sudipta Chattopadhyay. U-fuzz: Stateful fuzzing of iot protocols on cots devices. In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 209–220, 2024.

[SIG] Bluetooth SIG. Part b link layer specification. https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/low-energy-controller/link-layer-specification.html. [Online; accessed Nov. 28th, 2024].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[SKG+16] Dimitris E Simos, Kristoffer Kleine, Laleh Shikh Gholamhossein Ghandehari, Bernhard Garn, and Yu Lei. A combinatorial approach to analyzing cross-site scripting (xss) vulnerabilities in web application security testing. In *Testing Software and Systems: 28th IFIP WG 6.1 International Conference, ICTSS 2016, Graz, Austria, October 17-19, 2016, Proceedings 28*, pages 70–85. Springer, 2016.

[SKVK16] D. E. Simos, R. Kuhn, A. G. Voyiatzis, and R. Kacker. Combinatorial methods in security testing. *Computer*, 49(10):80–83, 2016.

[tlsa] Tls-attacker. https://github.com/tls-attacker/TLS-Attacker. [Online; accessed 12-April-2025].

[tlsb] Tls-scanner. https://github.com/tls-attacker/TLS-Scanner. [Online; accessed 15-April-2025].

[TM24] Inc. The MathWorks. Bluetooth protocol stack. https://de.mathworks.com/help/bluetooth/gs/bluetooth-protocol-stack.html, 2024. [Online; accessed Nov. 20th, 2024].

[TPHB21] Kristof T'Jonck, Bozheng Pang, Hans Hallez, and Jeroen Boydens. Optimizing the bluetooth low energy service discovery process. *Sensors*, 21(11), 2021.

[VKLA18] Jeffrey Voas, Richard Kuhn, Phillip Laplante, and Sophia Applebaum. Internet of things (iot) trust concerns. *NIST Tech. Rep*, 1:1–50, 2018.

[Weba] Automated combinatorial testing for software | csrc. https://csrc.nist.gov/Projects/Automated-Combinatorial-Testing-for-Software. [Online; accessed 28-February-2020].

[webb] Ghidra. https://ghidra-sre.org/. [Online; accessed 28-February-2025].

[Webc] Qiling framework. https://qiling.io/. [Online; accessed 28-February-2025].

[webd] Scapy. https://scapy.net/. [Online; accessed 20-March-2025].

[WNK+20] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave Jing Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. {BLESA}: Spoofing attacks against reconnections in bluetooth low energy. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020.

[WWX+24] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave Jing Tian, and Antonio Bianchi. Sok: The long journey of exploiting and defending the legacy of king harald bluetooth. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 2847–228066, 2024.

[YLKK13] Linbin Yu, Yu Lei, R.N. Kacker, and D.R. Kuhn. ACTS: A combinatorial test generation tool. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pages 370–375, 2013.

[YNSC23] Jean-Paul A. Yaacoub, Hassan N. Noura, Ola Salman, and Ali Chehab. Ethical hacking for iot: Security issues, challenges, solutions and recommendations. *Internet of Things and Cyber-Physical Systems*, 3:280–308, 2023.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub