Samuel Nogueira Carvalho

Anomaly Based Network Intrusion Detection

Deteção de Intrusões de Rede Baseada em Anomalias

outubro de 2022

Universidade do Minho
Escola de Engenharia

Samuel Nogueira Carvalho

Anomaly Based Network Intrusion Detection

Deteção de Intrusões de Rede Baseada em Anomalias

Dissertação de Mestrado
Mestrado Integrado em Eletrónica Industrial e Computadores

Trabalho efetuado sob orientação do

Professor Doutor Paulo Cardoso

outubro de 2022

DECLARAÇÃO DE INTEGRIDADE

Universidade do Minho, 31 de outubro de 2022.

Nome: Samuel Nogueira Carvalho

Assinatura: ____

Acronyms

ACL Asynchronous Connection-Less. i, 12, 28

API Application Programming Interface. i, 6, 33, 36, 37, 44, 67, 71

ATT Attribute Protocol. i, 34, 37

BD_ADDR Bluetooth Device Address. i, 15, 17, 19, 21, 23, 28

BLE Bluetooth Low Energy. i, 9

CAN Controller Area Network. i, ix, 1, 4, 5, 6, 7

CID Channel Identifier. i, 12

CRC Cyclic Redundancy Check. i, 26

DoS Denial of Service. i, ix, 5, 12, 19, 20, 27, 28, 29, 30, 31, 48, 67, 78, 82

ECU Electronic Control Unit. i, 1, 4, 6, 7, 8

FHS Frequency Hopping Sequence. i, 21, 25, 26

FHSS Frequency-Hopping Spread Spectrum. i, 11

GAP Generic Access Profile. i, 32, 34, 37

GATT Generic Attribute Profile. i, 32, 33, 34, 36

GSM Global System for Mobile communications. i, 21

HCI Host Controller Interface. i, vi, vii, xi, 10, 11, 32, 37, 38, 39, 50, 53, 55, 56, 57, 62, 66, 69, 70, 73, 79, 80, 81, 82, 83, 85

HTTP Hypertext Transfer Protocol. i, 12

IDS Intrusion Detection System. i, vii, ix, 40, 41, 42, 43, 44, 45, 47, 48, 63

IoT Internet of Things. i, 9, 67

IP Internet Protocol. i, 13

ISO International Organization for Standardization. i

L2CAP Logical Link and Control Adaptation Protocol. i, 12, 13, 15, 28, 32

LC Link Controller. i, 11

LM Link Manager. i, 11, 12, 28, 38

LMP Link Manager Protocol. i, viii, ix, 11, 30, 31, 68, 78, 79, 80, 83, 86

LSTM Long Short-Term Memory. i, vi, vii, viii, ix, xii, 47, 48, 49, 58, 60, 71, 72, 73, 84, 85

MAC Media Access Control. i, 7, 8, 27

MITM Man in the Middle. i, ix, 17, 23, 24, 25, 28, 29

ML Machine Learning. i, vii, ix, 43, 45, 46, 47, 54, 58, 74

NIST National Institute of Standard and Technology. i, 20

OBD-II On-Board Diagnostic II. i, 6, 8

OBEX Object Exchange Protocol. i, 12, 20, 36

PAN Personal Area Network. i, 9

PDU Protocol Data Units. i, 11

PHY Physical Layer. i, 18, 27, 28, 29

PIN Personal Identification Number. i, ix, 14, 18, 19, 22, 28, 29

PoC Proof of Concept. i, viii, 66, 68, 69, 78

PPP Point-to-Point Protocol. i, 13

RF Radio Frequency. i, 11, 18, 19

RFCOMM Radio Frequency Communications. i, 12, 13, 15

RFID Radio Frequency Identification. i, 7

RNN Recurrent Neural Network. i, ix, 48, 49

SCO Synchronous Connection-Oriented. i, 12, 32, 38, 79

SSP Secure Simple Pairing. i, 16, 17, 18

TPMS Tyre Pressure Monitoring System. i

WMA Windows Media Audio. i, 6, 8

Resumo

Ao longo dos últimos anos, a segurança de *hardware* e *software* tornou-se uma grande preocupação. À medida que a complexidade dos sistemas aumenta, as suas vulnerabilidades a sofisticadas técnicas de ataque têm proporcionalmente escalado. Frequentemente o problema reside na heterogenidade de dispositivos conectados ao veículo, tornando difícil a convergência da monitorização de todos os protocolos num único produto de segurança. Por esse motivo, o mercado requer ferramentas mais avançadas para a monitorizar ambientes críticos à vida humana, tais como os nossos automóveis.

Considerando que existem várias formas de interagir com os sistemas de entretenimento do automóvel como o Bluetooth, o Wi-fi ou CDs multimédia, a necessidade de auditar as suas interfaces tornou-se uma prioridade, uma vez que elas representam um sério meio de aceeso à rede interna do carro. Atualmente, os mecanismos de segurança de um carro focam-se na monitotização da rede CAN, deixando para trás as tecnologias referidas e não contemplando os sistemas não críticos. Como exemplo disso, o Bluetooth traz desafios diferentes da rede CAN, uma vez que interage diretamente com o utilizador e está exposto a ataques externos.

Uma abordagem alternativa para tornar o automóvel num sistema mais robusto é manter sob supervisão as comunicações que com este são estabelecidas. Ao implementar uma detecção de intrusão baseada em anomalias, esta dissertação visa analisar o protocolo Bluetooth no sentido de identificar interações anormais que possam alertar para uma situação fora dos padrões de utilização. Em última análise, este produto de software embebido incorpora uma grande margem de auto-aprendizagem, que é vital para enfrentar quaisquer ameaças desconhecidas e aumentar os níveis de segurança globais. Ao longo deste documento, apresentamos o estudo do problema seguido de uma metodologia alternativa que implementa um algoritmo baseado numa LSTM para prever a sequência de comandos HCI correspondentes a tráfego Bluetooth normal. Os resultados mostram a forma como esta abordagem pode impactar a deteção de intrusões nestes ambientes ao demonstrar uma grande capacidade para identificar padrões anómalos no conjunto de dados considerado.

Palavras-chave: Segurança Automóvel, Sistemas de Entretenimento, Bluetooth, Deteção de Intrusões.

Abstract

In the last few years, hardware and software security have become a major concern. As the systems' complexity increases, its vulnerabilities to several sophisticated attack techniques have escalated likewise. Quite often, the problem lies in the heterogeneity of the devices connected to the vehicle, making it difficult to converge the monitoring systems of all existing protocols into one security product. Thereby, the market requires more refined tools to monitor life-risky environments such as personal vehicles.

Considering that there are several ways to interact with the car's infotainment system, such as Wi-fi, Bluetooth, or CD player, the need to audit these interfaces has become a priority as they represent a serious channel to reach the internal car network. Nowadays, security in car networks focuses on CAN bus monitoring, leaving behind the aforementioned technologies and not contemplating other non-critical systems. As an example of these concerns, Bluetooth brings different challenges compared to CAN as it interacts directly with the user, being exposed to external attacks.

An alternative approach to converting modern vehicles and their set of computers into more robust systems is to keep track of established communications with them. By enforcing anomaly-based intrusion detection this dissertation aims to analyze the Bluetooth protocol to identify abnormal user interactions that may alert for a non-conforming pattern. Ultimately, such embedded software product incorporates a self-learning edge, which is vital to face newly developed threats and increasing global security levels. Throughout this document, we present the study case followed by an alternative methodology that implements an LSTM based algorithm to predict a sequence of HCI commands corresponding to normal Bluetooth traffic. The results show how this approach can impact intrusion detection in such environments by expressing a high capability of identifying abnormal patterns in the considered data.

Keywords: Automotive Security, Infotainment Systems, Bluetooth, Intrusion Detection.

Contents

List of Figures

List of Tables

Listings

Introduction

This chapter introduces the problem we have at hand, exposing its root cause, the essential motivation for its study, and the approach followed to solve it. Through the different sections, it is presented an overview of the challenges about protecting the car in a connected environment and what particularities to consider when committing to a task of this demand. Additionally, we provide an overview of this documentation structure in order to facilitate its readability and comprehension.

1.1 Background

Modern vehicles are no longer mere mechanical devices but contain a multitude of computers. These computers coordinate and monitor sensors, components, and the car's occupants. According to a recent study, a typical luxury sedan is estimated to contain over 100MB of binary code across 50-70 independent computers—Electronic Control Unit (ECU) in automotive vernacular—in turn communicating over one or more shared internal network buses [1], being CAN the inner car network that controls the critical devices. While the automotive industry has always considered safety a critical engineering concern, the increasing degree of computing control also brings with it a corresponding multitude of potential threats, especially when opening the network to external connections. As an example of the connectivity basis, when exposed to external threats, Controller Area Network (CAN) can compromise the entire vehicle and, consequently, put the passengers life's in danger. Amplifying this issue, the attack surface for modern vehicles is growing promptly as different network technologies interact with the user and the environment of the car.

As mentioned, vehicles now interact either directly with the driver or the associated devices through a diverse set of networks and protocols, being the infotainment systems a highly representative group. On many occasions, the functionalities they provide are insufficiently protected, since they are not directly connected to the system's critical components (belonging to the low-speed CAN bus), and because of that their physical or remote unauthorized access may cause a major impact on safety. In order to protect the various networks such as Bluetooth and Wi-fi, it is important to employ exhaustive security systems in vehicles, including the supervision of the such communication protocols. As Bluetooth technology is present in almost every vehicle and is a potential target to an attacker, it is necessary to implement measures to become this protocol safer from an attack surface perspective, thus preventing this channel from sending arbitrary messages to the car's network [2].

1.2 Problem Statement

In a world with an enormous need for communication enablers, the diversity of communication mechanisms grows likewise. To create safer environments where these technologies are used, many techniques for intrusion detection have been developed, but only a few are Bluetooth standard oriented. The Bluetooth approach to communication is quite different from others as it involves very specific protocols and patterns. Consequently, the market demands equivalent approaches for this technology.

By addressing such an issue, several challenges arise. Since the scope of communication is distinct from the remaining, it imposes its own challenges, starting by discovering how to best capture the protocol features. According to the communication level we consider, different perspectives of data and attacks can be obtained. From the embedded device's point of view, the operative system acts in accordance to the adopted Bluetooth protocol stack, in which one of the primary questions is whether its execution can indicate any anomaly or not. On the other hand, at a different abstraction level, the interaction between the host and the controller may also be a good reference for this characterization process, as it allows the correct establishment of a connection and possible threats verification.

Once this resolution has been accomplished, complex challenges of different natures arise. It is then necessary to gather the identified properties and design a system capable of detecting potential threats. Such a product must be efficient both in assembling Bluetooth data and pinpointing abnormal patterns. Ultimately, the main objective is to present a complementary security mechanism capable of recognizing hidden threats.

1.3 Approach

This work comprises three main phases distributed in a specific order, where each one can be broken down into several challenges. They are Bluetooth technology deployment, intrusion detection systems acknowledgment and, finally, design and implementation of a problem solution.

Characterizing Bluetooth communication is a task that requires a great level of expertise. Firstly, it is necessary to be aware of the technology protocol stack. Then, understand how such an environment is protected and dive into its security mechanisms. Authentication, pairing, authorization, and encryption, among others, are all processes of high complexity that directly impact the methodology followed. Being aware of the main vulnerabilities and gathering a survey of the known attacks is the baseline for a sustained solution.

Afterward, selecting an intrusion detection technique must be made based on a literature exploration, including emerging methods that fit the constraints of our problem. There are different types of intrusion detection as well as detection methodologies. Considering all that, we aim to converge on a machine-learning technique capable of responding to the available data. In order to gather that information, we also have to surpass another barrier as any self-learning system requires data to learn. Regarding that, it is necessary to move from the theory of a healthy connection to its materialistic representation. Only after defining the right feeding strategy, it's possible to build a proper model.

To achieve a final solution with a certain level of confidence, we approach the problem in two main stages. First, we develop an algorithm to handle the communication specifications and detect possible intruders. And once it is fully implemented, we try to evaluate its performance with actual captures.

1.4 Our Contribution

Integrated solutions for secure vehicle architecture are based on interlocking and complementing protective measures on all relevant system levels. These lines of defense may consist of securing control units, onboard networks, gateways isolation, or protecting the interfaces. Secure communication protocols protect the connection to the cloud, a firewall shields the vehicle network, and vehicle-specific certificates protect firmware updates.

However, simple threats such as a compromised driver's mobile phone are not considered. Certainly, they are further away from becoming an actual danger to the running system. Nonetheless, it is important to go through questions that address the probability of penetrating the current security measures. A device with undue access to the car's Bluetooth can perfectly pretend to be benign and inject malicious software into the network.

In this regard, we propose a tool that does not limit itself to characterizing benign Bluetooth communications but also detects network intruders. This feature is a considerable complement to the current vehicle security products. Moreover, this IDS can be enforced in different environments as long as it relies upon this communication protocol.

1.5 Dissertation Outline

This dissertation is organized as follows: Chapter 2 contemplates a literature review of Bluetooth technology as well as a showcase of the various intrusion detection systems. When diving into Bluetooth, this work also exposes a sum up of the known threats converted into actual attacks. As we explore the different topics, a reference to its state-of-the-art is consistently made. Based on all that, chapter 3 suggests a proof of concept design. It goes through the consecutive implementation steps explaining each one of the selected features. Chapter 4 materializes that implementation showing the adopted environment and the experimental results of the tool. The proof of concept is evaluated according to the established test cases, and the response given to each confrontation is also discussed. Last but not least, chapter 5 presents the main conclusions about this approach and proposes some future work related to the matter.

State-of-the-art

2.1 Car Vulnerabilities

Modern vehicles are controlled by a heterogeneous combination of Electronic Control Unit (ECU). Usually interconnected by common wired network called Controller Area Network (CAN), these components oversee a broad range of functionality, including the drivetrain, brakes, lighting, and entertainment. If, on the one hand, this interconnection provides complex and diversified safety features, on the other hand, its architecture permits a broad internal attack surface. Since each element of a given bus has implicit access to many others, an attacker can exploit this architecture by simply sending messages from a compromised component. This point leads this investigation line to the main question, which resides on how can an adversary get access to the car's internal bus in order to compromise its connected ECUs.

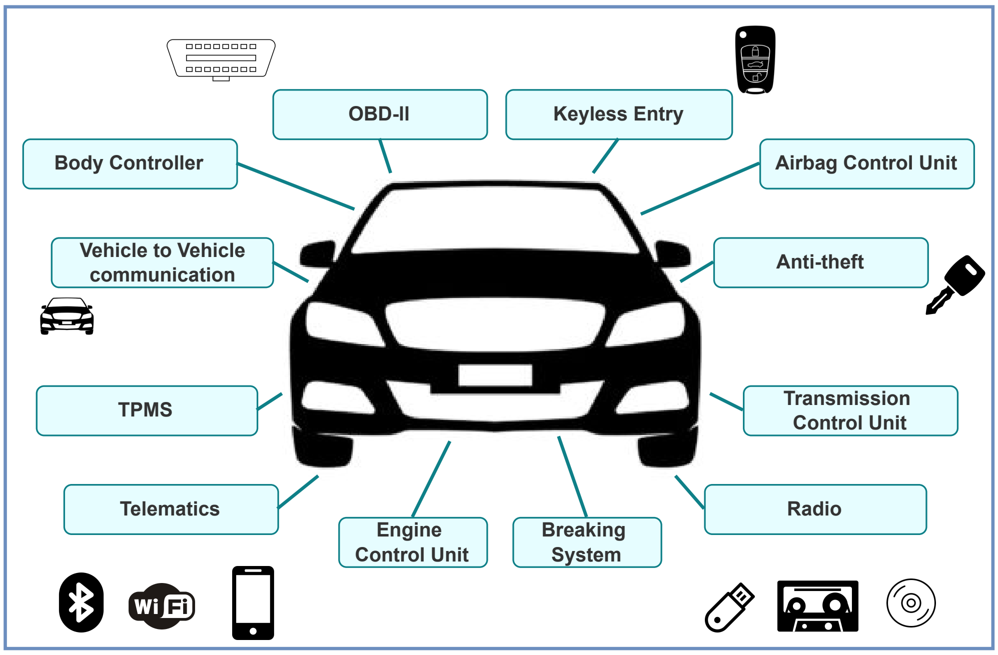


Figure 2.1: Digital I/O channels appearing on a modern car.

In the past, with traditional automobiles, one would require physical access to compromise it. Today, however, computerization has allowed remote accessibility. Prior work in this field has defined threats to automotive systems by distinguishing between technical and operational capabilities [3]. Technical capabilities address the assumptions made when describing the adversary's knowledge about the target vehicle, as well as his ability to develop malicious inputs for diverse channels. In turn, operational capabilities illustrate the adversary's requirements in delivering the alleged malicious input to a specific access entry in the field.

Considering the spectrum of I/O capabilities present in a modern vehicle there are different challenges to accessing an entry point. These can be categorized as indirect physical access, short-range wireless access, and long-range wireless access.

Remote compromise is feasible by utilizing a vast range of attack vectors such as mechanics tools, automotive infotainment systems [4], Bluetooth and cellular radios, etc. With the current demand for the development of new features offered by infotainment systems, they are now allowed to communicate with other devices in the car through CAN. Hence, devices interconnected by CAN buses are divided into convenience and vehicle-critical communication devices. Components such as the infotainment system, air conditioning, door locks, and windshield wipers are considered to be convenience devices, while devices such as the engine and braking system are considered vehicle-critical communication devices. Such classification is illustrated in figure 2.2.

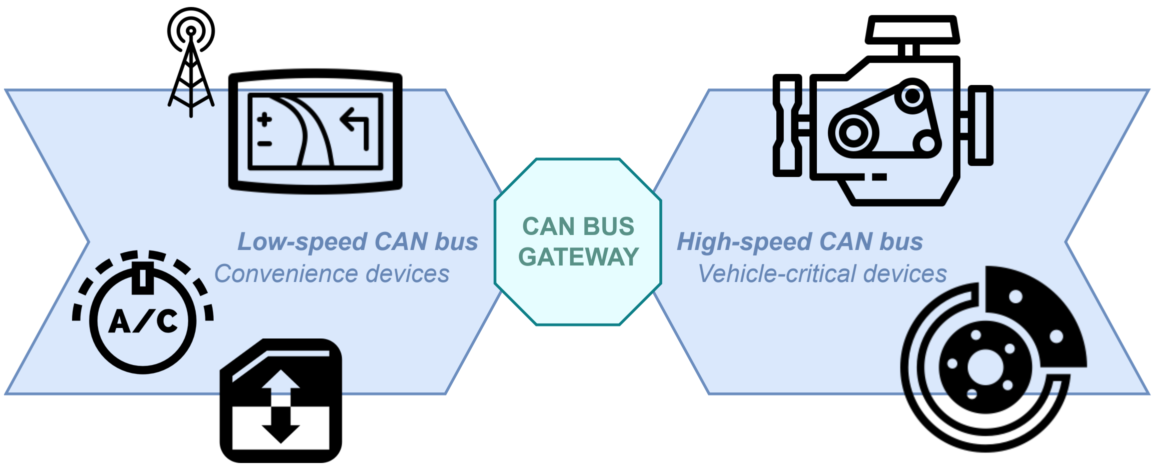


Figure 2.2: Schematic overview of CAN bus system [5].

While the vehicle-critical communication devices are interconnected by high-speed CAN buses, comfort devices communicate via a low-speed CAN bus. The low-speed CAN buses and high-speed CAN buses are connected to each other through a gateway called the CAN bus gateway. This connection exists because there are specific circumstances in which data must flow between low-speed and high-speed CAN buses. The CAN bus gateway is supposed to act as a firewall to the traffic that passes between the two types of CAN buses. However, there is always the possibility that malicious traffic could cross through this gateway unnoticed if the gateway firewall is not familiar with such malicious content [6].

In any case, before looking over the security of individual car components, it is relevant to analyze the security properties of the CAN bus in general. CAN packets (as represented in figure 2.3) do not contain addresses in the traditional sense, instead supporting a publish-and-subscribe communication model. The CAN-ID header is used to indicate the type of packet, each packet is physically and logically sent to every node, and the nodes independently decide whether to process the packet or not. The underlying CAN protocol has some inherent weaknesses common to all implementations. The most relevant of these are broadcast nature, fragility to DoS, no authenticator fields, or even weak access control [1].

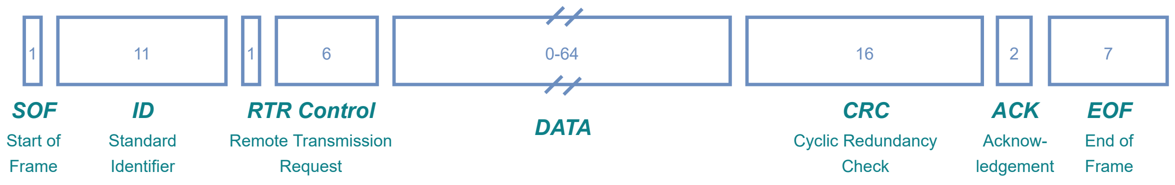


Figure 2.3: CAN packet structure.

2.1.1 Indirect Physical Access

Modern vehicles contain several physical interfaces from where one can either directly or indirectly access the car's internal network. Among them, On-Board Diagnostic II (OBD-II), CD player and USB ports can be stated as the most significant ones. *OBD-II* gives direct access to the automobile's key CAN buses and might allow an attacker to compromise the entire automotive system [1]. Commonly accessed by service personnel during routine maintenance for both diagnostics and ECU programming, OBD-II can be directly controlled and its port can access all CAN bus in the vehicle. This process is intermediated by hardware tools (sold by automobile manufacturers and third parties) that plug into the OBD-II port and can then be used to upgrade ECUs' firmware or to perform multiple diagnostic tasks such as checking the diagnostic trouble codes. Since 2004, all new cars support a windows API that provides a standard, programmatic interface to communicate with a car's internal buses, which is called *PassThru*. Regarding this standard device, there were found two classes of vulnerabilities. First, an attacker on the same network as the *PassThru* device can easily connect to it and, if the *PassThru* device is also connected to a car, obtain control over the car's reprogramming. Second, it is possible to compromise the *PassThru* device itself, implant malicious code, and thereby affect a far greater number of vehicles.

The other important class of physical interfaces contemplates infotainment systems. Nowadays, almost every car is equipped with a *CD player* able to interpret a wide range of audio formats (audio, MP3, WMA, and so on). In addition, vehicle manufacturers also provide interfaces such as *USB ports* or iPhone docking ports for allowing users to control their car's media system. As a consequence, an attacker might deliver malicious input by encoding it onto a CD or as a song file and using social engineering to convince the user to play it. Alternatively, he can compromise the user's smartphone and install software onto it that attacks the car's media system when connected.

The media player unit itself is manufactured by a major supplier of entertainment systems. Software running on the CPU handles audio parsing and playback requests, UI functions, and directly controls the connection to the CAN bus. Associated with these systems there is a well-known vulnerability. Acknowledging that the media player can parse complex files, it is possible to examine the firmware for input vulnerabilities that would allow the construction of a file that, if played, gives the ability to execute arbitrary code. For instance, it is possible to modify a WMA audio file such that, when burned onto a CD, plays perfectly on a PC but sends arbitrary CAN packets when played by the car's media player. This functionality would add only a small space overhead to the WMA file.

2.1.2 Short-range Wireless Access

With the appearance of the connected car environment, in-vehicle networks (e.g., CAN) are now connected to external networks (e.g., 3G/4G mobile networks), enabling an adversary to perform a long-range wireless attack using CAN vulnerabilities [7]. Compared to indirect physical access, when considering short-range attack surfaces, we devalue the operational requirements of the attacker. For technologies such as Bluetooth, remote keyless entry, RFIDs, or WiFi, we admit that an attacker is able to place a wireless transmitter near the car's receiver, where the distance depends on the targeted channel [3].

Bluetooth has been established as the standard for supporting hands-free calling and mainstreams, among other features to provide the driver and the remaining occupants a comfortable experience. In normal usage, the Bluetooth devices built into our car's telematics unit have a range of 10 meters, although it can be extended through amplifiers. Via reverse engineering, it is feasible to access the telematics ECU's Unix-like operating system and identify the particular program responsible for handling Bluetooth functionality. Thus, any paired Bluetooth device can exploit its vulnerabilities to execute arbitrary code on the telematics unit.

It may be challenging for an attacker to pair his own device with the car's Bluetooth system - a challenge addressed further on in this document. At this point, we assume that if an attacker can independently compromise one of the smartphones connected to the vehicle, then he can leverage the smartphone as a stepping-stone for jeopardizing the car's telematics unit, and thus all the critical ECUs on the car. To assess an attack of this nature, a simple Trojan Horse application can be implemented on an android phone. Such an operation checks whether the connected party is a telematic unit (from its MAC address) and, if so, sends an attack payload.

RFID-based vehicle immobilizers are nearly ubiquitous in modern automobiles. These systems incorporate an RFID reader in, or close to, the steering column of the car, and an RFID tag into the key or key fob. Such systems prevent the car from operating unless the valid key is displayed. However, there are several threats to these wireless communication channels. As new locking systems are launched in the market (physical keys, remote keyless entry, remote keyless ignition, etc), new vulnerabilities and attack opportunities arise with it.

Recent studies [8] have put radio jamming, replay attack, roll jam wireless attack, and relay attack on the list of main threats in this field. For instance, a radio jam attack will not allow the attacker to steal any kind of possession. The functionality of the fob could be stolen or controlled more dangerously during a roll jam attack, though. In summary, different attacks are created for various technologies (operating on the fob), some of them may be hard to detect, and a powerful attack will undoubtedly cause more victim loss.

WiFi allows infotainment systems to communicate with external devices on the same network. Therefore, data or commands can be sent to this system from other devices present on the same network. At the same time, other devices can get data in the opposite direction. A very convenient connection point, WiFi functionality can also create vulnerabilities in infotainment systems by acting as an attack vector. That is because one can connect his infotainment system to an external network, the outside world where possible enemies can sneak.

Present in the literature, there are some vulnerabilities associated with this channel. First, we have *Nmap port Scan*, which consists of a security scanner used to look for hosts and services. When connected to the same network, it is possible to find the IP and MAC address of the infotainment system. Another way to compromise it is by performing an attack on the operating system. This would require creating a malicious *APK* file to request and gain the special privilege of the Android services in the infotainment system upon installation. Moreover, a denial-of-service attack is also proven to be very effective against this technology.

In summary, automotive infotainment systems have a wide range of vulnerabilities within their WiFi capability [6]. Some of these vulnerabilities lead to the discovery of secret information about the infotainment system, simply by using a scanner. The WiFi vector also enables to compromise of the file system of the telematics unit and stealing of data such as photos, videos, music, files, etc. Considering all these channels, if there is a vulnerability in the ECU software responsible for parsing the channel messages, an attacker can compromise the ECU (and thus the entire vehicle) simply by sending malicious input near the vehicle.

Table 2.1: Attack surface capabilities.

Vulnerability class	**Channel**	**Capability**	**Visible to user**	**Scale**	**Cost**
Direct physical	OBD-II	Plug attack hardware directly into car OBD-II port	✓	Small	Low
Indirect physical	CD	CD-based firmware update	✓	Small	Medium
Indirect physical	CD	Adulterated song (WMA)	✓	Medium	Medium-high
Indirect physical	*PassThru*	WiFi or wired control connection	✗	Small	Low
Indirect physical	*PassThru*	WiFi or wired shell injection	✗	Viral	Low
Short-range wireless	Bluetooth	Buffer overflow with paired android phone and trojan app	✗	Large	Low-medium
Short-range wireless	Bluetooth	Sniff MAC address, brute force PIN, buffer overflow	✗	Small	Low-medium
Long-range wireless	Cellular	Call car, authentication exploit, buffer overflow	✗	Large	Medium-high

Table 2.1 [3] resumes how different vulnerabilities can be explored as well as the qualitative assessment of the cost (in effort) to discover and exploit those vulnerabilities. The Visible to User column indicates whether the driver or technician can detect the compromise process. The scale column records the approximate range of the attack. For example, CD firmware update attacks are considered small because they require a CD to be distributed to each targeted vehicle.

2.2 Bluetooth Technology

The Bluetooth standard [9] (or IEEE 802.15.1) is a radio-based cable replacement technology that allows a multitude of devices to interact and work together in dynamically changing topologies. It has become the de-facto standard in the market for Personal Area Network (PAN) and owes its popularity to the diversity of the products that support it, the minimum power consumption required, and the low price of Bluetooth components combined with their ability to fit into the small size of the required hardware [10]. It operates in short-range distances often within 10 meters and uses a frequency hop spread spectrum, working at either the physical or the protocol level.

Bluetooth links laptops, PDAs, phones, smartphones, PCs, printers, various peripherals (keyboards, mice, headsets), and a multitude of other equipment together providing them with services such as file transfer, voice communications, printing, MP3 downloads, GPS, and data synchronization (for phone books, calendars, messaging and note taking).

The ease of Bluetooth user interaction and seamless interoperability between devices supporting the multiple and diverse capabilities is only possible through the multitude of profiles and protocols involved. Beneath the simple user interaction lies a complex specification developed with a generic approach to applications so it can accommodate constantly emerging new features and applications.

Since its launch date, the technology has evolved and expanded into a globally accepted standard available in almost every mobile device in the current days. Bluetooth has classically been used to replace serial ports, creating a virtual serial port in both devices, allowing for backward compatibility with older devices [11]. That evolution has been motivated to meet the needs of the Internet of Things (IoT) market with the introduction of Bluetooth 5.0 and Bluetooth mesh networking. The broadcast topology available on BLE enables indoor positioning and location services. The mesh topology on BLE is optimized for creating large-scale device networks, making it particularly well suited for emerging markets in need of a reliable wireless solution to establish large-scale control, monitoring, and automation systems [12].

2.2.1 Bluetooth Networks

The Bluetooth architecture features a piconet as its basic unit of organization. A piconet starts with two connected devices, such as a PC and cellular phone, and may grow to eight connected devices. All Bluetooth devices are peer units and have identical implementations. However, when establishing a piconet, one unit will act as a master for synchronization purposes, and the other(s) as a slave(s) for the duration of the piconet connection.

Multiple piconets can exist and overlap; when communicating with each other, connected via a bridge node, the resulting internetwork is called a scatternet. A slave, as well as a master unit in one piconet, can establish this connection by becoming a slave in the other piconet. The master unit is the device in the piconet whose clock and hopping sequence are used to synchronize all other devices in the piconet, whereas slave units are all others.

In piconet, each Bluetooth device has a unique 48 bit hard wired device address for identity which allows for 248 devices. The master controls access to the Bluetooth network by constantly polling to discover all the devices near to it and then allowing them to participate. A graphical representation of such network can be observed in figure 2.4.

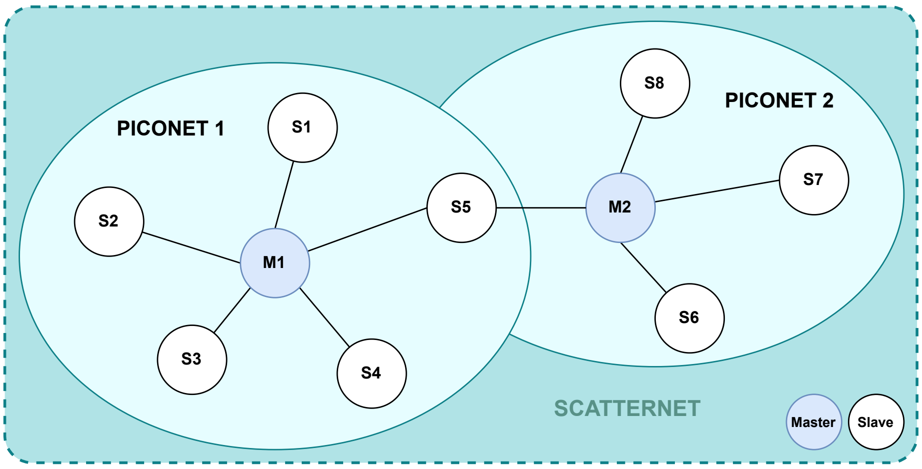


Figure 2.4: Bluetooth Personal Area Network.

All Bluetooth devices transmit their signals within the 2.402 to 2.480 GHz frequency range. This frequency is subdivided into 79 bands, the master device controls access to these bands and hops between the channels at the rate of 1600 times per second. The slave machines in the Bluetooth piconet synchronize with the master and so have access to the network. This random channel hopping across a spread frequency spectrum allows many devices and piconets to synchronize and share the radio band.

The small size of the packets (typically a packet is transmitted for every channel slot allowing for 1600 packets a second) with a payload of up to 2745 bits allows fast transmission and good interference protection. Therefore, if a frequency jam does occur with another piconet or via some other means, error correction ensures easily repeatable retransmission and subsequent acknowledgment.

2.2.2 Bluetooth Protocol Stack

A Bluetooth protocol stack is illustrated in figure 2.5. Protocols below the Host Controller Interface (HCI) are built-in to the Bluetooth microchip, and protocols above the HCI are located as a part of the host device's software package. An HCI is needed between the hardware and software protocols. The purpose of the HCI is to enable the manufacturer-independent combining of Bluetooth chips (Host Controller) and the actual host device. The HCI handles secure communication between the host and the Bluetooth module. The host sits right below the application, and is comprised of multiple (non real-time) network and transport protocols enabling applications to communicate with peer devices in a standard and interoperable way. In its turn, the controller implements the link layer, the low-level, real-time protocol which provides, in conjunction with the radio hardware, standard-interoperable over-the-air communication.

The BR radio [13] is the lowest defined layer of the Bluetooth specification. This layer defines the requirements of the Bluetooth transceiver device operating in the 2.4 GHz ISM frequency band. It implements a 1600 hops/sec Frequency-Hopping Spread Spectrum (FHSS) technique. The radio hops in a pseudo-random way on 79 designated Bluetooth channels, each one with a 1MHz bandwidth.

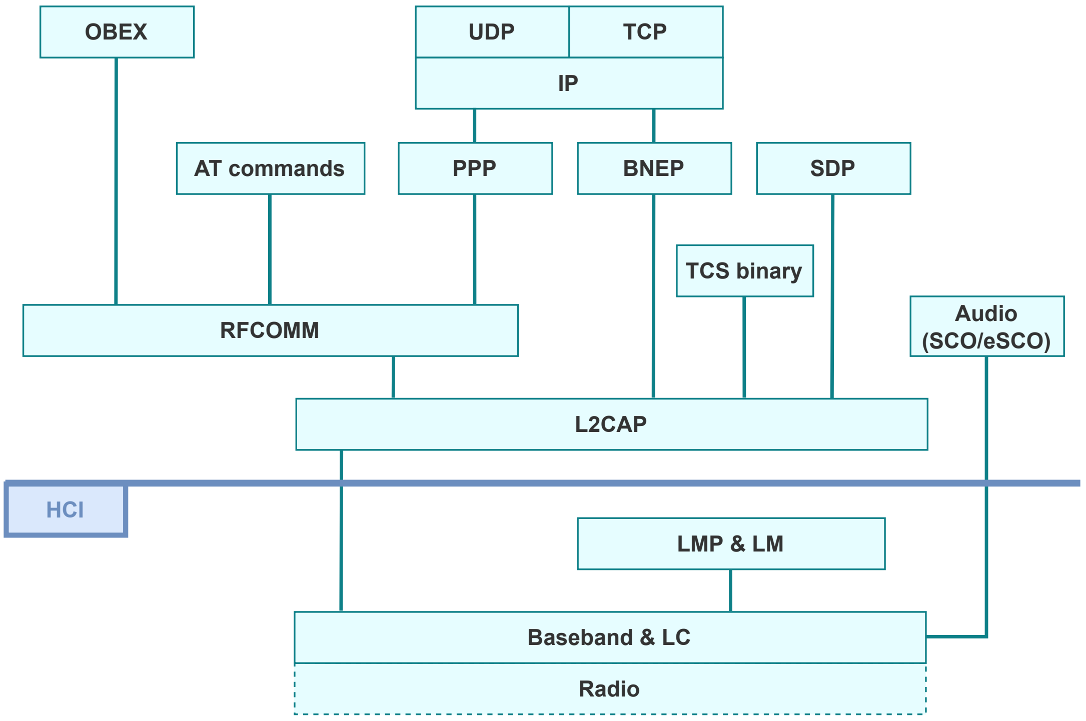


Figure 2.5: Bluetooth protocol stack.

Baseband and Link Manager Protocol (LMP) together enable the physical RF connection. The LMP essentially consists of a number of Protocol Data Units (PDU), which are sent from one device to another, determined by the machine address in the packet header. The processes of link setup and link configuration between different Bluetooth devices include establishing security functions such as authentication and encryption by generating, exchanging, and checking link and encryption keys. Furthermore, LMP controls the power modes and duty cycles of the Bluetooth radio device and the connection states of a Bluetooth unit in a piconet. The Link Controller (LC) is a state machine that defines the current state of the Bluetooth device. A Bluetooth device can be in low-power mode for saving batteries, in the connected state for normal piconet operation, or in the paging state for the master to bring new slaves to the piconet, for example. The LC has a pseudo-random number generation capability, methods for managing security keys, and the capability for providing the mathematical operations needed for authentication and encryption.

The Link Manager (LM) acts as a bridge between the application and the LC on the local device, and it also communicates with the remote LM via PDU, i.e. the LM communicates with three different entities during a Bluetooth session: the local host through the HCI, the local LC (local operations), and the remote LM (link configuration, link information, and link management operations). The PDU is acknowledged at the Baseband level, but it is acted upon by the LM. The local LM usually resides on the Bluetooth module as a complete host-module implementation. The remote LM can be defined as the LM at the other end of the Bluetooth link. The LM also has several commands for handling security issues.

Synchronous Connection-Oriented (SCO) and eSCO links are used for transferring real-time two-way voice. They are established directly from the Baseband level, so the overhead of upper layer protocols does not cause any delays for real-time two-way voice connections. Four packet types have been defined for SCO links, whereas eSCO links support seven packet types. One of these 11 packet types, SCO link's HV1 (High-quality Voice 1), is really interesting from the security point of view because one single HV1 SCO link reserves all Bluetooth piconet resources and therefore makes various Denial of Service (DoS) attacks possible.

The Logical Link and Control Adaptation Protocol (L2CAP) [14] is a software module that normally resides on the host. L2CAP adapts higher-layer protocols over the baseband. It shields the higher-layer protocols from the details of the lower-layer protocols. The L2CAP also offers CO (Connection-Oriented; from master to one slave and from slave to master) and CL (Connection-Less; from master to multiple slaves) services, and it is defined only for Asynchronous Connection-Less (ACL) links. The L2CAP provides user data services over the Bluetooth ACL channel. These services consist of protocol multiplexing, packet segmentation and reassembly, and group management. L2CAP assumes that the Bluetooth baseband provides a reliable channel with full duplex capability and delivers packets in the same order as they are generated at the remote device. L2CAP operates by setting up channels representing endpoints of each data pipe between devices. The signaling channel is used to designate a Channel Identifier (CID) at each end, and then to configure the channel to meet either a best effort or guaranteed quality of service. Communication between the L2CAP layer and other layers on both local and remote devices can occur through indications, confirmations, requests, and responses [10].

The Service Discovery Protocol (SDP) is used to find the services of Bluetooth devices in the range. Radio Frequency Communications (RFCOMM) emulates serial ports over the L2CAP, and therefore it is possible to use existing serial port applications via Bluetooth. The SDP is built on top of the L2CAP.

The Object Exchange Protocol (OBEX) [15] is used to exchange objects, such as calendar notes, business cards, and data files, between devices by using the client-server model. The OBEX protocol provides functionality similar to that of Hypertext Transfer Protocol (HTTP), but in a simpler manner. HTTP is an application layer protocol and layered above the Transmission Control Protocol/Internet Protocol (TPC/IP). The OBEX protocol provides the client with a reliable transport for connecting to a server. It also provides a model for representing objects and operations. The OBEX supports six simple and self-explanatory operations: Connect (choose your partner, negotiate capabilities, and establish a connection), Disconnect (terminate connection), Put (push objects to the server), Get (pull objects from the server), Abort (abort an object exchange that is in progress), and SetPath (set server's directory path to a new value).

The Telephony Control protocol Specification (TCS) binary defines the call control signaling for the establishment/release of speech and data calls between Bluetooth devices. It also provides functionality for exchanging signaling information that is unrelated to ongoing calls. Many AT commands are also supported for transmitting control signals for telephony control.

The Bluetooth Network Encapsulation Protocol (BNEP) is used to provide networking capabilities for Bluetooth devices. It allows Internet Protocol (IP) packets to be carried in the payload of L2CAP packets. The IP is a network layer protocol in the TCP/IP protocol suite. TCP and User Datagram Protocol (UDP) are transport layer core protocols used in the TCP/IP protocol suite. Point-to-Point Protocol (PPP) can also be used to provide TCP/IP networking capabilities for Bluetooth devices, but it is slower, i.e. it works over RFCOMM whereas BNEP works directly over the L2CAP, and therefore PPP is rarely used now.

2.2.3 Bluetooth Security

Security mechanisms [16] included in the Bluetooth specifications to consequently characterize their limitations by presenting a survey of how their vulnerabilities can be explored. There are three components specified as basic security services in the Bluetooth standard: authentication, authorization and encryption. Each one plays an important role in the plan of security, being activated or deactivated according to the application.

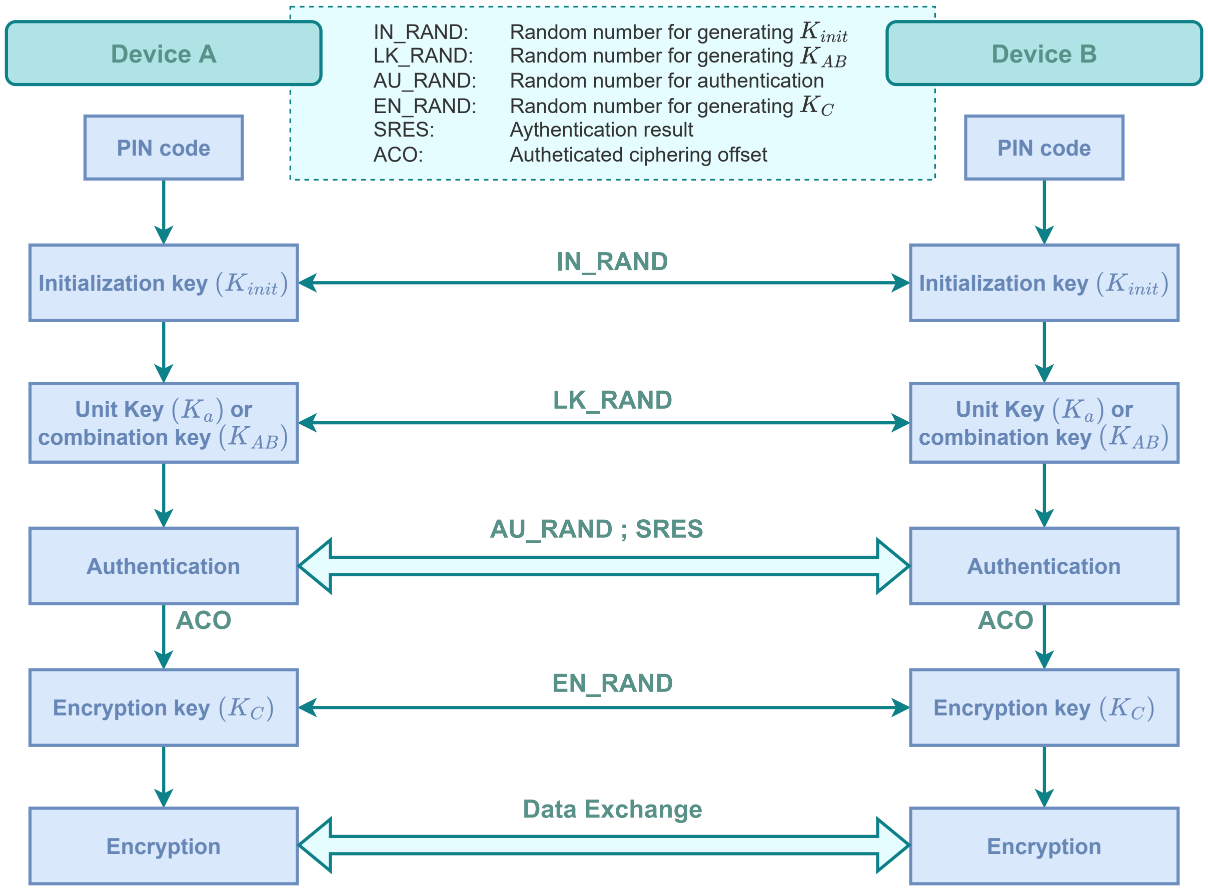

Device A
Device B
IN_RAND: Random number for generating K_{init}
LK_RAND: Random number for generating K_{AB}
AU_RAND: Random number for authentication
EN_RAND: Random number for generating K_C
SRES: Aythentication result
ACO: Autheticated ciphering offset
PIN code
PIN code
Initialization key (K_{init})
IN_RAND
Initialization key (K_{init})
Unit Key (K_a) or combination key (K_{AB})
LK_RAND
Unit Key (K_a) or combination key (K_{AB})
Authentication
AU_RAND ; SRES
Authentication
ACO
ACO
Encryption key (K_C)
EN_RAND
Encryption key (K_C)
Encryption
Data Exchange
Encryption

Figure 2.6: Summary of Bluetooth security operations.

Authentication

The Bluetooth authentication [16][17] procedure is based on the challenge-response scheme. Authentication addresses the identity of each communicating device, and is used for determining the client's authorization level to access certain services. The sender sends an authentication request frame to the receiver. The receiver sends an challenge frame back to the sender. Both perform a predefined algorithm. The sender sends its findings back to the receiver, which in turn either allows or denies the connection. Two devices interacting in an authentication procedure are referred to as the claimant and the verifier. The verifier is the Bluetooth device validating the identity of another device. The claimant is the device attempting to prove its identity. Basically, it is the basis of the security paradigm upon which both authorization and encryption depend, encompassing several procedures.

Bluetooth security is based on building a chain of events, none of which provides meaningful information to an eavesdropper, and all events must occur in a specific sequence for security to be set up successfully. Two Bluetooth devices begin pairing with the same Personal Identification Number (PIN) code that is used for generating several 128-bit keys, as figure 2.6 suggests.

Pairing

Pairing [18] for instance is a procedure invoked when a link key has not been created for the unique connection between devices. In order that devices can connect easily and quickly, a scheme known as Bluetooth pairing may be used. Once Bluetooth pairing has occurred two devices may communicate with each other. Bluetooth pairing is generally initiated manually by a device user. The Bluetooth link for the device is made visible to other devices. They may then be paired.

The steps of calculating the keys in the Bluetooth pairing process are as under:

1. Each device creates a random number and encrypts it together with its hardware address;
2. The random number is *XORed* with initialization key and sent away to the other unit;
3. Now the two units have the other's random number. The hardware address is public so each unit can calculate their counter part's encrypted random number together with hardware address;
4. Both units now do a bit-wise addition to combine the two units encrypted values;
5. The result of the addition is the combination key of the two units;
6. A mutual authentication is required in order to confirm that both units have the correct combination key.

After both devices create K_{init}, the pairing process continues through the creation of the link key and the use of it to perform an authentication. Once the link keys are generated by the two devices, the referred process is called link key exchange. Each device provides the other enough information that they will both generate the same link key, ensuring that they both began with the same PIN. During the authentication process, one of two keys can be used. The simpler one is called the unit key, which provides only a low level of security. The combination key is functionally identical, except that it is used for subsequent authentications and encryption key derivations.

Authorization

Authorization [16][19] requires a successful authentication and it is then triggered when the remote Bluetooth device makes an attempt to connect to a service. More accurately, this security procedure is invoked when a peer-to-peer protocol connection is requested at the Logical Link and Control Adaptation Protocol (L2CAP) or Radio Frequency Communications (RFCOMM) port layers. Authorization requires the remote device to be identified, and the service being requested to be reported to the service provider. Aware of this information, the user can choose to give access permission to the requested service, granting temporary trust. If a device has been previously marked as trusted, the authorization process completes successfully without user interaction.

Encryption

Bluetooth encryption [20] is performed to protect payloads of the packet being exchanged between the two Bluetooth devices. There are three steps needed to implement encryption. First, the same encryption key must be constructed at both ends of the link. Next, a key streamer generator is loaded with the encryption key (and other information as well), and a sequence of what appears to be random bits are produced as its output. In the end, a bit by bit *XOR* operation is performed between the key stream and the plain text payload of each transmitted Bluetooth packet to obtain cipher text. Neither the header or the access code are encrypted, only the payload of Bluetooth packets. As the encryption algorithm is symmetric, decryption occurs at the destination simply by performing another bit by but *XOR* operation between the cipher text and the key stream generator to produce plain text.

Bluetooth Security Modes

Basic Bluetooth security configuration is done by the user determining how Bluetooth devices implement connectivity and discoverability options. Its different combinations and capabilities result in three distinct categories, or security levels:

- *Silent*: the device will never accept any connections, limiting itself to monitor Bluetooth traffic;
- *Private*: the device remains occult, i.e. it is a so-called non-discoverable device. Connections are accepted only if the Bluetooth Device Address (BD_ADDR) of the device is known to the prospective master. A 48-bit BD_ADDR is normally unique and refers globally to only one individual Bluetooth device;

- *Public*: the device can both be discovered and connected to. It is therefore called a discoverable device.

In addition to that, the various versions of the specifications detail four Bluetooth security modes [21]. Each Bluetooth device must operate in one of four modes, which are described as follows:

- *Bluetooth Security Mode 1*: This mode is non-secure. The authentication and encryption functionality is bypassed and the device is susceptible to hacking. Bluetooth devices operation in Bluetooth Security Mode 1. Devices operating like this do not employ any mechanisms to prevent other Bluetooth-enabled devices from establishing connections. While it is easy to make connections, security is an issue. It may be applicable to short range devices operating in an area where other devices may not be present. Security Mode 1 is only supported up to Bluetooth 2.0 + EDR and not beyond.

- *Bluetooth Security Mode 2*: For this Bluetooth security mode, a centralized security manager controls access to specific services and devices. The Bluetooth security manager maintains policies for access control and interfaces with other protocols and device users.

  It is possible to apply varying trust levels and policies to restrict access for applications with different security requirements, even when they operate in parallel. It is possible to grant access to some services without providing access to other services. The concept of authorization is introduced in Bluetooth security mode 2. Using this it is possible to determine if a specific device is allowed to have access to a specific service.

  Although authentication and encryption mechanisms are applicable to Bluetooth Security Mode 2, they are implemented at the LMP layer (below L2CAP).

  All Bluetooth devices can support Bluetooth Security Mode 2. However, v2.1 + EDR devices can only support it for backward compatibility for earlier devices.

- *Bluetooth Security Mode 3*: In Bluetooth Security Mode 3, the Bluetooth device initiates security procedures before any physical link is established. In this mode, authentication and encryption are used for all connections to and from the device.

  The authentication and encryption processes use a separate secret link key that is shared by paired devices, once the pairing has been established.

  Bluetooth Security Mode 3 is only supported in devices that conform to Bluetooth 2.0 + EDR or earlier.

- *Bluetooth Security Mode 4*: Bluetooth Security Mode 4 was introduced at Bluetooth v2.1 + EDR.

  In Bluetooth Security Mode 4 the security procedures are initiated after link setup. Secure Simple Pairing uses what are termed Elliptic Curve Diffie Hellman (ECDH) techniques for key exchange and link key generation.

  The algorithms for device authentication and encryption algorithms are the same as those defined in Bluetooth v2.0 + EDR.

The security requirements for services protected by Security Mode 4 are as follows:

- Authenticated link key required;
- Unauthenticated link key required;
- No security required.

Whether or not a link key is authenticated depends on the Secure Simple Pairing association model used. Bluetooth Security Mode 4 is mandatory for communication between v2.1 + EDR devices.

Link Key Generation

As previously mentioned, there are two methods in which link key generation is performed. Bluetooth security modes 2 and 3 use one method, while security mode 4 makes use of a different one. When operating in security mode 2 or 3, two associated devices simultaneously drive link keys during the initialization phase as the user enters an identical PIN into one or both devices. If the PIN happens to be shorter than 16 bytes, then the BD_ADDR is used to supplement the PIN value used to generate the initialization key. After that process is complete. devices automatically authenticate and initiate the encryption procedure to secure the wireless link, if encryption is enabled. The PIN code used in Bluetooth devices can vary between 1 and 16 bytes. While the typical four-digit PIN may be sufficient for low-risk situations, a longer PIN should be used for devices that require a higher level of security.

Secure Simple Pairing (SSP) was introduced in Bluetooth v2.1 + EDR for use with Security Mode 4. SSP simplifies the pairing process by providing a number of association models that are flexible in terms of device input capability. SSP also improves security through the addition of ECDH public key cryptography for protection against passive eavesdropping and Man in the Middle (MITM) during pairing. There are four models offered in SSP:

- *Numeric comparison*: a user is shown a six-digit number on each display and provides a ”yes” or ”no” response;
- *Passkey entry*: one of the devices shows in its display a six-digit number entered by the user in the other device (which must contain input capability);
- *Just works*: It performs numeric comparison, except that a display is not available, therefore the user is required to accept a connection without verifying the calculated value on both devices;
- *Out of band*: this model allows devices to pair by simply “tapping” one device against the other, followed by the user accepting the pairing via a single button push.

Security Mode 4 requires Bluetooth services to mandate an authenticated link key, an unauthenticated link key, or no security at all. Of the association models mentioned, all but the Just Works model provide authenticated link keys.

2.2.4 Bluetooth Vulnerabilities

Vulnerabilities are often the result of poor Bluetooth implementation. Considering the enormous amount of Bluetooth devices constantly in use, malicious security violations can be expected to occur all the time. But where are the weaknesses? How can they be exploited by an intruder? These topics must be addressed because phenomena such as eavesdropping and disruption are quite present. Intruding on a wired network almost always requires actual contact with the Physical Layer (PHY), and the proximity of an unauthorized person in such a context is often easy to detect. When we consider a wireless network there are significant differences, even for short-range systems such as Bluetooth. An intruder can, for example, place a bug nearby to intercept and record Bluetooth activity over a period of time and, later on, analyze its content.

Overall security in Bluetooth networks is based on the security of the Bluetooth medium, the security of Bluetooth protocols and the security parameters used in Bluetooth communication. There are several issues that can significantly weaken the overall security of Bluetooth networks. Among them, we here present the following:

- Vulnerability to eavesdropping: Because Bluetooth is a wireless RF communication system using mainly omnidirectional antennas, an eavesdropper is often not detected. All the contents of unencrypted transmissions can be seen clearly. Even if Bluetooth data encryption is used, all intercepted packets can be recorded for later cryptographical analysis.

- Weaknesses in encryption mechanisms: When two Bluetooth devices negotiate the parameters for encryption, the length of the encryption key is restricted by the Bluetooth device that has the shorter maximum encryption key length. The average search time, when using a naive guess-and-try brute-force method, measured in seconds is $2^{L-1}/n$, where *L* is the length of the encryption key in bits and *n* is the number of key search trials per second.

- Weaknesses in PIN code selection: In older versions, the PIN code can be as long as 128 bits (16 bytes), so it can contain up to sixteen 8-bit characters. This makes an eavesdropper's work much easier, because he needs to go through only 10000 possible PIN values and witness the initial pairing process between the target devices in order to get all the required information for various attacks.

- Weaknesses in association models of SSP: SSP uses four association models, according to the device's IO capabilities. The most significant weakness occurs when at least one of the devices has neither input nor output capability and an *OOB* cannot be used. In this case, the *Just Works* association model is used, in which the user is simply asked to accept the connection.

- Weaknesses in device configuration: The default settings of Bluetooth devices usually provide no security at all: the device is set as discoverable (i.e. public security level) and nonsecure (i.e. nonsecure security mode).

Therefore, an attacker can discover the BD_ADDR of the target device in a few seconds and perform various attacks against it.

Since Bluetooth has been adopted by different manufacturers and used by the general public, a whole series of additional security flaws has come to light. Bluetooth's friendly and dynamic nature allows its basic security parameters to be circumvented; more complex security mechanisms demand more memory and power consumption, which many battery powered devices simply do not have. To make matters worse, many Bluetooth devices allow users to simply switch off security. The default security configuration on most devices offers no security at all. Following that, we will approach the three main threats in distributed networks: disclosure threat, integrity threat, and Denial of Service (DoS) threat.

A disclosure threat means that information can leak from the target system to an eavesdropper that is not authorized to access the information. This threat is especially serious. The presence of an eavesdropper is often not realized because of the physical separation of eavesdropping equipment from the communicating devices.

In contrast to eavesdropping, where an attacker simply listens for desirable information, the integrity threat involves unauthorized access by an attacker who actually connects to the piconet. Once access is granted, the attack can be conducted in different directions: sensitive files can be retrieved, deeper access into the network can be attempted, or misinformation can be injected into the network. Very often the goal of such an attack is the retrieval of information. However, since there is an actual connection to the target device rather than just listening, it is considered to be an integrity threat instead. The retrieval or possible introduction of damaging information or software into the piconet can be achieved at the expense of different known weaknesses: PIN and initialization, unit key, authentication, encryption, or even access via RF capture.

On the other hand, we have DoS threat, which involves blocking access to a service, making it either unavailable or severely limiting its availability to an authorized user. This threat can be accomplished in several different ways and at different levels of the Bluetooth protocol stack. At the physical layer, an intruder can either capture the channel from a legitimate piconet member or jam the piconet entirely. Attacks on higher protocol layers try to exploit some of their characteristics in an attempt to occupy the attention of one or more members of the piconet such as they're unavailable to service other devices in a timely manner.

2.2.5 Bluetooth Attacks Survey

The problems regarding Bluetooth security have been reported since its inception. But, it has not been considered as a significant problem until its adaptation into mobile devices. Researchers identified attacks as attempts to gain unauthorized access to a victim's device without the knowledge of the victim [22]. Attacks can be used to destroy, alter, disable, or steal data from a user. Attacks on Bluetooth devices may be active or passive. Attacker may directly breach the security system of the device and gain control of the victim device. Again attackers may manipulate the victim or apply different schemes to gain control of the victim device. In the glossary of key information security terms by National Institute of Standard and Technology (NIST), attack is defined as "An attempt to gain unauthorized access to system services, resources, or information, or an attempt to compromise system integrity." or "Any kind of malicious activity that attempts to collect, disrupt, deny, degrade, or destroy information system resources or the information itself" [23]. The following section exhibits attacks classified in three categories: Disclosure, Integrity and DoS.

2.2.5.1 Disclosure Attacks

BlueJacking

A BlueJacking [24][25] attack is a social engineering-based attack where the attacker must be authorized to pair with the victim's device. A large number of ready-made Bluetooth hacking tools are available for anyone to download from the Internet. An example is *Spooftooph* - a sniffing software that can monitor and log all Bluetooth devices and display their Bluetooth-friendly name. *Bluebrowse* is another hacking tool that is equivalent to a port scanner and displays all the available services on devices. These tools, among others, allow and incentive the use of BlueJacking – the sending of anonymous messages to other Bluetooth machines. A Bluetooth phone produces a message as a contact entry in the address book and then instructs the phone to send it via Bluetooth. The phone seeks out any other Bluetooth-enabled phone within range and the message pops up on the other phone's screen. These can be in the form of adverts, SPAM, or simply nuisance messages. Furthermore, the usage of these tools can violate users' locational privacy, so their movements can be tracked and recorded and subsequently sold commercially to potential advertisers [10].

BlueSnarfing

BlueSnarfing [26][25] consists of taking advantage of the Bluetooth connection to steal information from wireless devices, particularly common in smartphones and laptops. It is an OBEX Push Service pairing protocol in which the attacker hacks into and gains unauthorized access to the victim's device. OBEX can be used to exchange business cards between Bluetooth devices using the OBEX protocols' *object push* feature, i.e. pushing (sending) business cards (objects) to Bluetooth devices using the OBEX protocols' *Put operation*. In most cases, this service does not require authentication. BlueSnarfing is based on the exploitation of the OBEX protocol's *object pull* feature instead of the *object push* feature. This feature is used for pulling (receiving) objects from Bluetooth devices using the OBEX protocol's *Get operation*. BlueSnarfing attack conducts an OBEX *Get request* for known filenames such as telecom/pb.vcf (the phone book) or telecom/cal.vcs (the calendar file).

Once a device is compromised, hackers owns entrance to its content: contacts, emails, passwords, photos, and other useful data [27]. In order to testify how easily this attack can be performed, we tested it on an android device. Herein, the open-source *Bluesnarfer* tool built into Kali Linux was used. To accomplish that, all it took was to find and interpret the command: *bluesnarfer –r (read phonebook entry) 1-100 –b (Bluetooth device address)*, whose conversion to the real test can be resumed in the following sequence: *'bluesnarfer -r 1-10 -b 3C:7D:0A:80:6D:3F'*. As a result, the first ten contacts in the device's phonebook were promptly printed on the screen. This experiment only confirms how dangerous this attack category is and how seriously it must be considered.

BlueBugging

A BlueBugging [28] [29] attack means that an attacker connects to the target device (typically a Bluetooth mobile phone) without alerting its owner, steals some sensitive information, such as an entire phonebook, calendar notes and text messages, and has full access to the Global System for Mobile communications (GSM) AT command set. It means that the attacker can, in addition to stealing information, send text messages to premium numbers, initiate phone calls to premium numbers, write phonebook entries, connect to the Internet, set call forwards, try to slip a Bluetooth virus or worm to the target device, and many other things. This makes this kind of attack even more dangerous than BlueSnarfing attack, especially when the BD_ADDR of the target device is known to the attacker. To perform this attack, several public BlueBugging tools exist: for example, *btxml* and *Blooover*.

2.2.5.2 Integrity Attacks

Cracking the Bluetooth PIN

Using a Bluetooth frequency sniffer (or protocol analyzer) and acquisition of a Frequency Hopping Sequence (FHS) packet, attackers can attempt to intercept the pairing an authentication processes. Assume that the attacker was well succeed in its task, and saved all the messages (observe table 2.2). The attacker can now use a brute force algorithm to find the PIN used. The attacker enumerates all possible values of the PIN.

Table 2.2: List of messages sent during the pairing and authentication process.

#	Src	Dst	Data	Length	Notes
1	A	B	IN_RAND	128 bit	Plaintext
2	A	B	LK_RAND_A	128 bit	*XORed* w/ K_{init}
3	B	A	LK_RAND_B	128 bit	*XORed* w/ K_{init}
4	A	B	AU_RAND_A	128 bit	Plaintext
5	B	A	$SRES$	32 bit	Plaintext
6	B	A	AU_RAND_B	128 bit	Plaintext
7	A	B	$SRES$	32 bit	Plaintext

Knowing IN_RAND and the BD_ADDR, the attacker runs the key generation algorithm with those inputs and the guessed PIN, and finds a hypothesis for K_{init}. The attacker can now use this hypothesis of the initialization key, to decode messages 2 and 3. Messages 2 and 3 contain enough information to perform the calculation of the link key K_{AB}, giving the attacker a hypothesis of K_{AB}. The attacker now uses the data in the last 4 messages to test the hypothesis: Using K_{AB} and the transmitted AU_RAND_A (message 4), the attacker calculates $SRES$ and compares it to the data of message 5. If necessary, the attacker can use the value of messages 6 and 7 to re-verify the hypothesis K_{AB} until the correct PIN is found [30][31].

To sum up the attack procedure, we next present its ordered steps as well as its representative flowchart (as it can be observed in figure 2.7):

1. Calculate hypothesis for K_{init};
2. Decode the random numbers for generating the combination key (K_{AB});
3. Calculate hypothesis for the combination key;
4. Calculate provisory SRES out of AU_RAND_A;
5. Does the authentication result of device A match? If so, jump to step 7;
6. Try a consecutive PIN and go back to step 1;
7. Calculate provisory SRES out of AU_RAND_B;
8. Does the authentication result of device B match? If so, jump to step 10;
9. Move backward to step 6;
10. PIN guess corresponds to the correct value. PIN crack successfully performed.

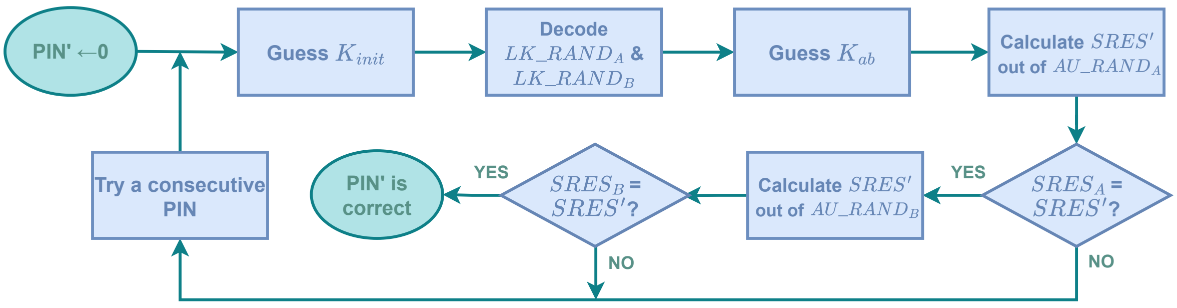


Figure 2.7: PIN cracking attack.

This PIN cracking can also be accomplished through packet modification. The attacker constructs a message which is inserted at a specific point in the protocol forcing the master and slave devices to repeat the pairing process. Thereafter the attacker, with additional specialized hardware, can spoof the Bluetooth address and crack the PIN.

This attack can be construed as an entry point for further experiments. Since the connection is fully established, an attacker has the freedom to do many other things, as the main barrier is already overstepped. The other enemy for those who fight this kind of abuse is the existence of several tools capable of automatizing this procedure. By the name of *BTCrack*, and available in different repositories, we can find a powerful tool to commit perjuries of this nature.

Backdoor Attack

The backdoor attack [31][29] involves establishing a trust relationship through the pairing mechanism, but ensuring that it no longer appears in the target's register of paired devices. In this way, unless the owner is actually monitoring their devices at that moment, a connection is established. The attacker may continue using the resources that a trusted relationship with that device grants access to until the users notice such attacks. The attacker can not only retrieve data from the phone but other services such as modems, Internet, WAP and GPRS gateways may be accessed without the owner's knowledge or consent.

When the backdoor is installed in the target device via a Backdoor attack, the attacker can continue the attack in many different ways: for example, trying to exploit the resources of the target device via a trusted relationship, trying to perform a *BlueSnarfing* attack, or trying to slip a virus or worm to the target device. A backdoor attack works only if the BD_ADDR of the target device is known. Moreover, the target device has to be vulnerable to a backdoor attack.

Man in the Middle (MITM) Attack

Every Bluetooth device is associated with a unique 48-bit device address (BD_ADDR) and a 28-bit clock (CLK) running at 3.2 kHz and thus wrapping every 23.3 hours. A time-division duplex communication scheme is used, where the slave is synchronized to the clock setting of the master. The master starts transmissions in even slots ($CLK0 = 0$ and $CLK1 = 0$) and the previously addressed slave may respond in odd slots ($CLK0 = 0$ and $CLK1 = 1$). To avoid interference of several piconets operating in the same area, a frequency hopping [32] mechanism is employed. The following protocols are used to establish a connection:

- *Inquiry* [32]: The inquiry protocol is used to discover unknown devices in the area. The discoverability of a device can be restricted by the user: a) A limited discoverable device is only temporarily discoverable. b) A non-discoverable device will never respond to an inquiry message.

- *Paging* [32]: The paging protocol is used to establish a connection to an already discovered device. The connectability of a device can be restricted by the user: A non-connectable device will never scan for page requests.

- *Authentication* [20]: The pairing protocol is used at the first interconnection of two devices to set up link keys for subsequently establishing authenticated connections. The pairability of a device can be restricted by the user.

Having these concepts as a fundamental basis for the MITM attack understanding, we will present different approaches to its implementation. As it can be observed in figure 2.8, an attacker poses as a fake access point by intercepting the connection of two Bluetooth devices connected on the piconet. From there, the attacker attempts to access the data that is transmitted between two Bluetooth devices by intercepting the submission. The victims here often believe that they are sending information on a private connection, however, the attacker is in control of the connection the whole time.

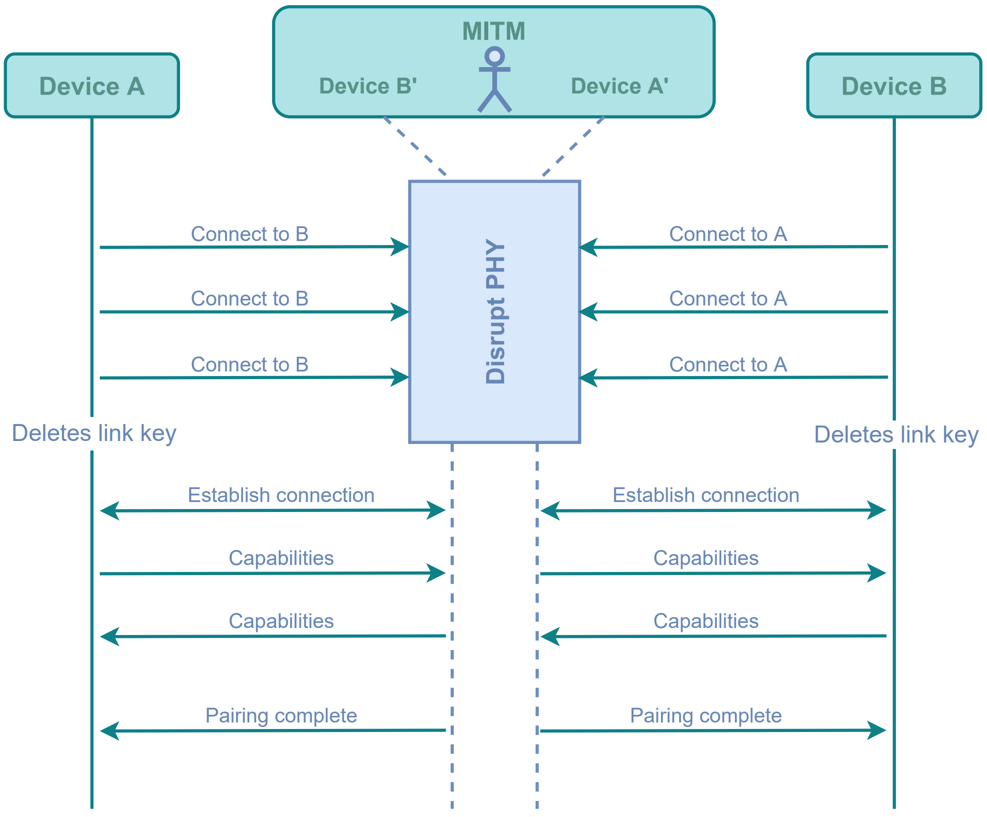


Figure 2.8: Idea of the attack.

One could think that frequency hopping increases security, as synchronization to the hopping sequence is not easy. Actually, frequency hopping decreases security, as its properties can be exploited for a MITM attack. This attack is induced by paging [33]. As the master does not know when the paged device starts the page scan procedure, the master has to sequentially send a number of page requests until the slave responds. While that, the attacker proceeds as follows:

1. The attacker responds faster than the slave to the page request. Thus, the master establishes a connection with the attacker instead of the slave. After the attacker is connected to the master, the channel hopping

sequence of the master will be used for communication.

2. Then the attacker restarts the paging procedure with the slave. After the slave has returned the slave response, the attacker will respond with an FHS packet containing a clock setting different from the master's. Thus, the master and slave will use the same channel hopping sequence, but a different offset in this sequence.

3. If the slave requests a role switch, the current channel hopping sequence is used until the switch is finally agreed upon and an FHS packet is sent from the new master to the old master.

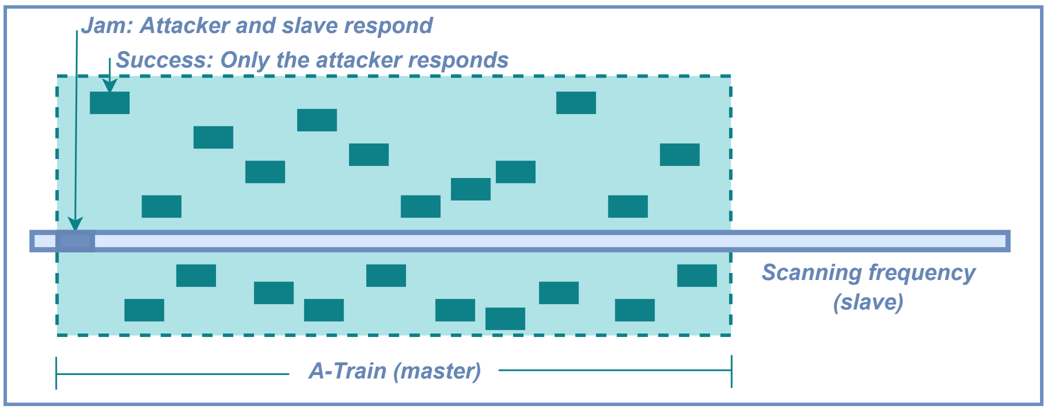

Jam: Attacker and slave respond
Success: Only the attacker responds
Scanning frequency (slave)
A-Train (master)

Figure 2.9: MITM attack with one jam step.

Thus, as the master and slave use the same channel hopping sequence, but always a different offset in this sequence, it is guaranteed that the attacker can intercept every communication between both devices. However, it is crucial to understand how the master can establish a connection with the attacker instead of the slave. In that regard, there are two distinct methods to prevent the slave from responding to a page request:

- **Jamming**: In most cases, the slave will not immediately receive the page request. Thus, an attacker scanning all 32-page hopping frequencies in parallel will be able to respond to the first-page message and set up the communication with the master (see figure 2.9. There is, however, a slight chance that the slave will respond to the first-page request. In this case both the attacker and the slave send out the slave response nearly in parallel. In this case, the synchronization will fail completely as the channel is jammed. The master will send out the next page request on the next frequency to which only the attacker is able to respond as the slave changes its scanning frequency. If the slave coincidentally also changes its scanning frequency, an additional "jam step" is required.

- **Busy Waiting**: Alternatively, the attacker starts establishing a connection to the slave before the master does. However, the attacker interrupts the connection establishment before sending the master response (the FHS packet). Then the connection is "half-open" and the slave continuously scans for an FHS packet until a timeout occurs. After this timeout, the slave, once again, scans for a certain time for an ID packet. If the slave does not receive an ID packet, the slave returns to normal operation. The attacker can exploit this behavior and hold the slave most of the time in this “half-open” state.

On the other hand, we can also perform these attacks by attacking an encrypted link [33]. Bluetooth uses a stream cipher for optional encryption. The cipher is rekeyed for every transmitted or received packet. If encryption is turned on, the integrity of the encrypted payload is protected by a Cyclic Redundancy Check (CRC) that is appended to the payload before encryption. It is a well-known result that such an integrity check does not protect against the intended manipulation of the ciphertext. It is possible to toggle single bits in the ciphertext, which will directly affect the corresponding bit in the plaintext. While the attacker still is not able to eavesdrop on the encrypted communication, he may manipulate every intercepted packet, before forwarding it to the recipient. Together with full or partial knowledge of the plaintext, this is a powerful attack.

If we are towards unidirectional communication, the attack is relatively simple. As Bluetooth offers reliable transport channels, acknowledge messages are sent on the link level, which is not encrypted. To exploit this, we assume that data is transmitted from master to slave and that the clock setting of the slave is at least one tick behind the master. For every intercepted data packet the attacker first responds to the master with an unencrypted ACK packet to acknowledge the reception of the data packet. Then, he manipulates the packet and forwards it to the correspondent slave. Finally, the attacker discards every ACK packet received from the slave. A graphical representation of such an attack can be found below:

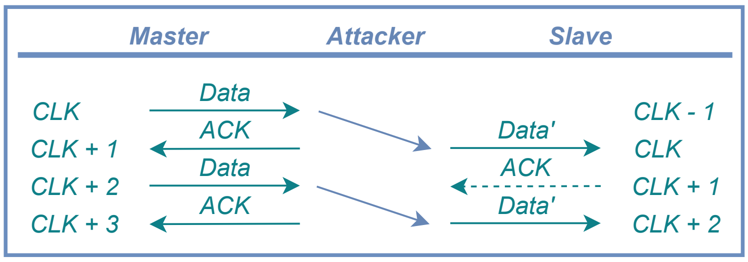


Figure 2.10: Attacking unidirectional communication.

If both devices send and receive data packets - so-called bidirectional communication - it is crucial that both devices use the same plaintext to encrypt and decrypt packets. However, the attacker lets the victim’s devices use different offsets in the same hopping sequence. To manipulate the plaintext and the channel hopping sequence, we can manage to flip a clock bit in the FHS packet at the connection establishment. This way both clocks drift so that master and slave use very different offsets in the channel hopping sequence but the same plaintext. Due to their synchronized clocks (at least for the lower bits), master and slave are tightly coupled and the attacker has not much time to manipulate the ciphertext. In fact, the attacker has to retransmit every single bit received. A slight delay is however possible as every slot has an uncertainty window of 10μs.

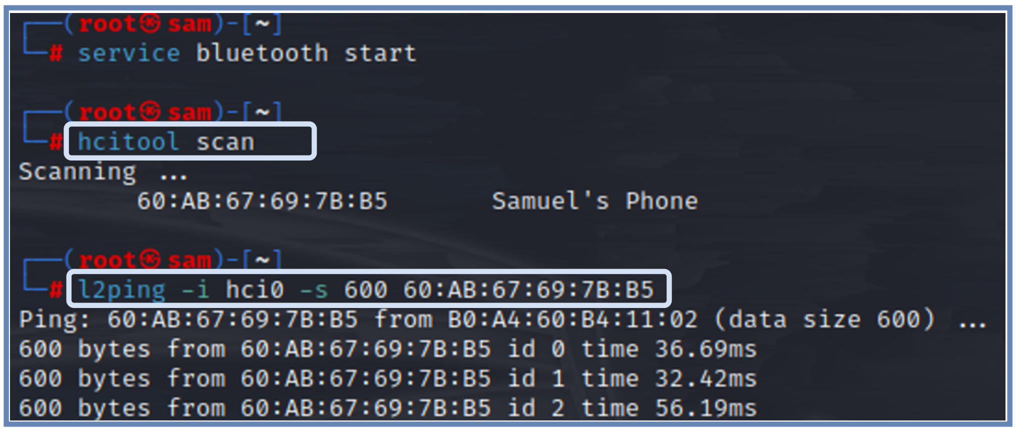
2.2.5.3 DoS Attacks

DoS attacks use artificial or natural factors to cause system resources to be over-committed, thereby failing to provide normal services for legitimate devices [34]. DoS threats can be roughly divided into two parts: attacks against the PHY and attacks against protocols above the PHY.

Attacks Against the PHY

At the PHY, an attacker can jam the piconet entirely or capture the channel from the legitimate piconet device. Bluetooth uses frequency hopping spread spectrum technology to eliminate interference and reduce fading, which divides the frequency band into several frequency hopping channels [35]. The baseband controller is in charge of the regular frequency hopping selection for communication parties. The spread spectrum technology expands the narrowband multiple times into a wide frequency band to reduce the influence of interference, thereby improving the performance of Bluetooth transmission. However, the frequency hopping sequence between different Bluetooth piconets is independent of each other, and there is no information exchange between them. The frequency selected by the other party for data packet transmission at a certain moment can't be known to each other. Therefore, in places where there are many Bluetooth piconets and other ISM frequency band networks, there will be multiple networks selecting the same frequency at the same time, causing the same frequency interference.

One can start a DDoS attack against a Bluetooth device in proximity with simple Linux command-line tools. For instance, an attacker in proximity could start reconnaissance with *hcitool* and can scan the environment for Bluetooth devices (i.e: *$ hcitool scan*). If there are any, the MAC address would be found out. With the knowledge of the target device's MAC, the next step would be to start ping with large sized request packets in flood mode. *l2ping* tool could be used for this step (i.e: *$ l2ping -i hci0 -s 600 -f <Bluetooth MAC>*). Observe figure 2.11. If the attacker has multiple Bluetooth interfaces in his attacking machine, with automated scripts, he can start a devastating DDoS attack on any target in proximity.



```
┌──(root㉿sam)-[~]
└─# service bluetooth start

┌──(root㉿sam)-[~]
└─# hcitool scan
Scanning ...
        60:AB:67:69:7B:B5       Samuel's Phone

┌──(root㉿sam)-[~]
└─# l2ping -i hci0 -s 600 60:AB:67:69:7B:B5
Ping: 60:AB:67:69:7B:B5 from B0:A4:60:B4:11:02 (data size 600) ...
600 bytes from 60:AB:67:69:7B:B5 id 0 time 36.69ms
600 bytes from 60:AB:67:69:7B:B5 id 1 time 32.42ms
600 bytes from 60:AB:67:69:7B:B5 id 2 time 56.19ms
```

Figure 2.11: DoS attack example.

Attacks Against Protocols Above the PHY

Attacks on higher levels of the Bluetooth protocol stack try to exploit some of the characteristics of higher level protocols in an attempt to occupy the attention of one or more devices of the piconet in such a way that they are unable to serve other legitimate devices within a reasonable time [29]. Here is a list of the possible attacks:

- *Big NAK attack:* An intruder can connect to the target and request some information. When the transfer begins, the intruder always sends a NAK in response to the baseband packet, possibly putting the target device into an endless transmission loop. The target's supervision timeout won't be triggered because the timer is reset each time the intruder sends a NACK.
- *BD_ADDR duplication:* An intruder could plant a bug that spoofs the BD_ADDR of a target device. Whenever the user tries to contact the target device, either the bug responds (if it hears the page first) or both units respond and jam each other, thus denying access to the legitimate device.
- *L2CAP guaranteed service attack:* An intruder can request that the target device set up a L2CAP link with guaranteed service and highest possible token (data) rate. The LM under these conditions is supposed to refuse any further ACL channels and disable periodic scans to support the requested data rate.

2.2.5.4 Summary and Perspective

Bluetooth vulnerabilities come in various forms and can do a great deal of damage. Our research work deals with weaknesses in the Bluetooth medium, Bluetooth protocols and Bluetooth security parameters. Currently, weaknesses in Bluetooth security parameters seem to be the biggest problem in Bluetooth security. Users need to mitigate these attacks on their own accord to stay secure. For this reason, it is important to understand the Bluetooth piconet and security modes so that we can understand how we can protect user data. To sum up the different attacks and their respective impact on a certain network, we next expose a gap analysis in the form of table 2.3.

From this table analysis, one can conclude that one of the major problems is to implement countermeasures against MITM and DoS. Another aspect that makes these attacks quite concerning is its presence in the literature and proofs of concept. While on the one hand MITM requires some expertise level and it is not so easy to implement, DoS can be very perturbing and anyone can execute it. Then, we have disclosure attacks, very well documented and whose counteraction depends mostly on the user. As data privacy has become a global issue, these attacks must be prevented. Last but not least, we want to emphasize the power of PIN cracking because there are many tools to perform it and, even if there is a chance to make it hard for the attacker by increasing the PIN length, only a few users take that into consideration. And once the attacker oversteps that barrier, he can move to dangerous actions.

Table 2.3: Snapshot of Bluetooth Security Threats.

Attack	**Category**	**Expertise level**	**Literary exhibition** (by number of papers)	**Danger effectiveness**	**Counteraction**
BlueJacking	Disclosure	Low	9	Low	Average
BlueSnarfing	Disclosure	Low	8	Medium	Average
BlueBugging	Disclosure	High	9	High	–
PIN cracking	Integrity	Low	2	High	Easy
Backdoor	Integrity	High	1	High	–
MITM	Integrity	Medium	6	Medium	Hard
DoS (PHY)	Denial of Service	Low	11	Medium-low	Hard
DoS (Protocols)	Denial of Service	Medium	1	Medium-low	Hard

2.2.6 Bluetooth Exploits

Repeated Host Connection

The implementation of Bluetooth Classic Audio in several products may not properly manage connection attempts from hosts with the same Bluetooth address as the currently connected BT host. It simply connects to the target device (BT speaker) using a fake BD address that matches the BD address of the first connected host (Device A – 60:ab:67:69:7b:b5).

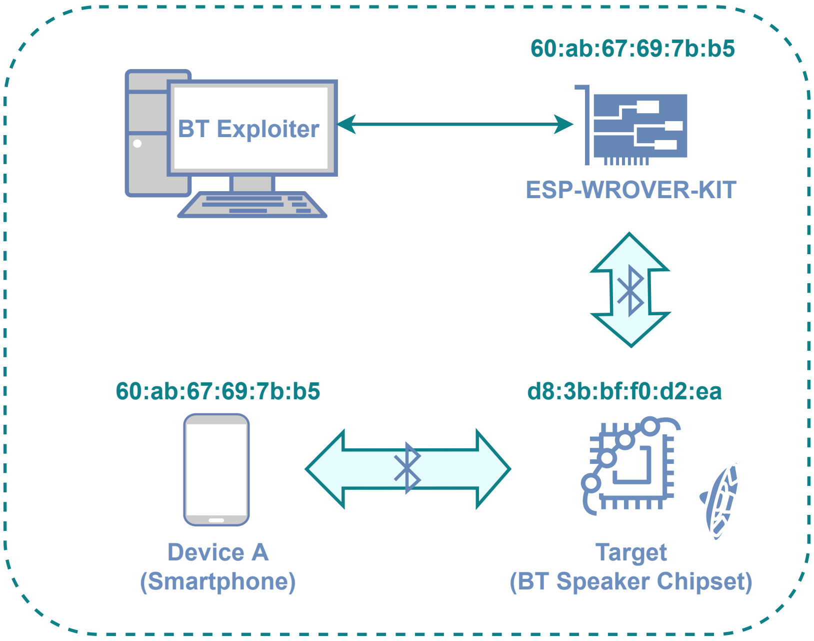


Figure 2.12: Repeated host connection.

As shown in figure 2.12, the target device will disconnect and final cause deadlocks. This will require the user to reboot the target device to restore the BT functionality. An attacker in radio range can exploit this vulnerable behavior to perform DoS and intentionally deadlock the target device.

Invalid Timing Accuracy

Bluetooth Classic implementations on numerous chipsets also do not rightly tolerate the reception of a malformed LMP timing accuracy response followed by multiple re-connections to the target link slave.

This particularity allows an attacker to exhaust the device's BT resources. Eventually, the attacker could make multiple attempts to send a crafted *LMP_timing_acc_response* (i.e., LMP timing accuracy response) before suddenly reconnecting to the target using a random Bluetooth address. Such an attack can result in a crash or can cause a BT interruption.

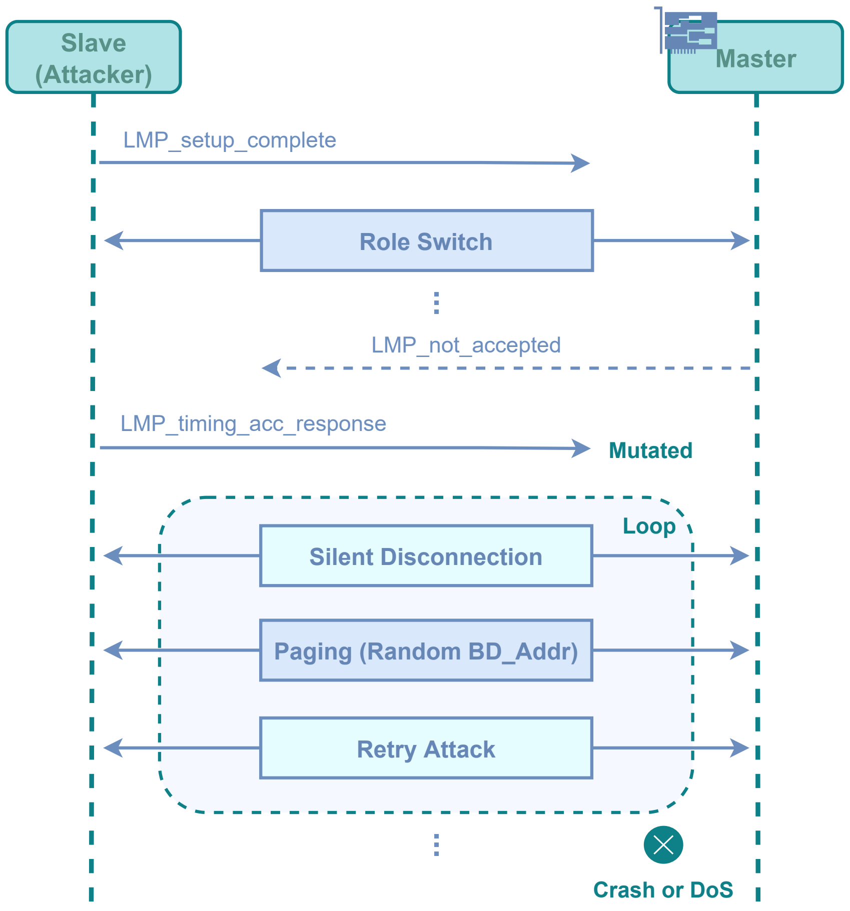


Figure 2.13: Invalid timing accuracy.

As shown in Figure 2.13 the attacker must run a loop of reconnection and injection of the faulty *LMP_timing_acc_response* until the target chipset becomes unstable. This will either crash the firmware or disconnect other active BT devices. The faster the reconnection is performed, the easier it is for the attack to interfere with other BT devices connected to the target's chipset. This offense causes a DoS attack.

During the attack, the targeted device is unable to use its Bluetooth service normally, and its effect does not persist after it ceases. This fact is because the target will typically attempt to reestablish a BT connection by reconnecting to a previously disconnected device. The device cannot be used normally during an attack as the BT connection can be interrupted constantly. The attacker only needs to know the Bluetooth address of the target device, and no authentication is required to boot.

LMP Random Authentication Number Flooding

Bluetooth classic implementations on multiple chipsets do not properly handle receiving continuous unsolicited LMP responses that trigger a heap overflow in the BT firmware. The such vulnerability allows an attacker within wireless range to cause a denial of service (reboot or deadlock the device) by flooding the device with *LMP_AU_rand* packets as shown in figure 2.14.

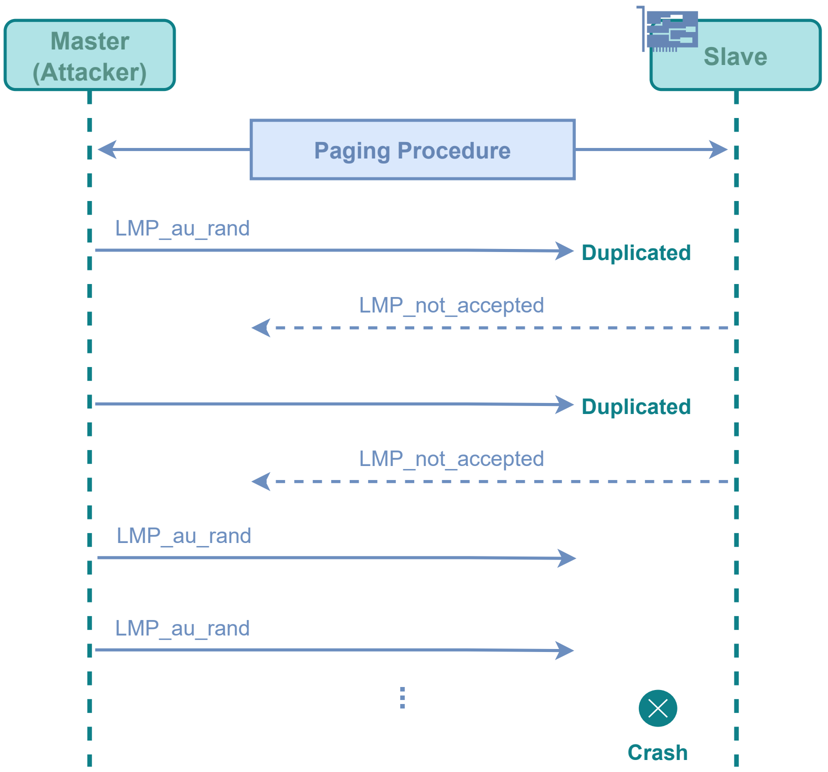


Figure 2.14: LMP AU Rand Flooding.

This attack indicates that flooding testing with packets from certain LMP procedures may not have been well tested. It may arise because the Core Specifications only allow a limited LMP testing mode [36] that restricts the System on Chip (SoC) to work with only a few LMP packets. This can limit the testing flexibility of BT providers. In particular, vendors may have a limited amount of testing that can be done on production BT firmware, such as flooding or out-of-order transmission of packets during normal LMP procedures. An attacker within radio range could exploit this vulnerable behavior to perform a DoS and intentionally crash or block the target device.

2.2.7 Bluetooth within Linux

The integration of the Bluetooth technology into the Linux kernel and the major Linux distributions has progressed over the last years. Such technology is present almost everywhere. The use of Bluetooth with a Linux-based system is easy and in most cases, it only needs a one-time setup. Linux computers typically use BlueZ - the official Linux Bluetooth Protocol Stack [37]. It provides, in its modular way, support for the core Bluetooth layers and protocols. It also maps Bluetooth protocol layers to kernel modules, kernel threads, user space daemons, configuration tools, utilities and libraries. Currently BlueZ consists of the following separate modules:

- Bluetooth kernel subsystem core;
- L2CAP and SCO audio kernel layers;
- RFCOMM, BNEP, CMTP and HIDP kernel implementations;
- HCI UART, USB, PCMCIA and virtual device drivers;
- General Bluetooth and SDP libraries and daemons;
- Configuration and testing utilities;
- Protocol decoding and analysis tools.

Bluetooth Protocol Stack vs the OSI model

The reason for comparing the Bluetooth protocol stack with the Open Systems Interconnection (OSI) model is to understand the former in a simpler way (observe figure 2.15). The radio, baseband and link manager layer are the hardware part of the Bluetooth architecture which is implemented on the Bluetooth chipset. These layers roughly map to the physical, data link and network layers in the OSI model.

The HCI maps to the transport layer, and transports data between the L2CAP layer and the Bluetooth chipset. The controller (also referred as adapter in the BlueZ documentation and code) typically resides within a chip that is either an integral part of the computer or is implemented within a peripheral device like a USB Bluetooth dongle. The L2CAP maps to the session layer. At the top of the stack reside various application environments called profiles. Since the radio, baseband and link manager are usually part of Bluetooth hardware, operating system support starts at the HCI layer.

Applications either implement profiles and/or services based on GAP and GATT or they implement Bluetooth mesh models and act as a node in a Bluetooth mesh network. Two of the main components are *bluetoothd* (the Bluez daemon process) and *dbus-daemon*, a key part of an Interprocess Communication (IPC) system on Linux called D-Bus. D-Bus has wider applicability than communication involving desktop GUI components and since version 5.52 of BlueZ, it has been the standard interface between Bluetooth applications running on Linux and BlueZ itself.

BlueZ provides a documented API in a series of text files that are part of its distribution. Applications do not make direct calls to BlueZ functions and do not receive direct callbacks from BlueZ either, instead using D-Bus inter-process communication centered around a message bus. Messages are placed on the bus by one process and travel along the bus to be delivered to one or more other processes connected to the same bus. A process that connects to a bus is called a client, whereas a process that listens for and accepts connections is called a server.

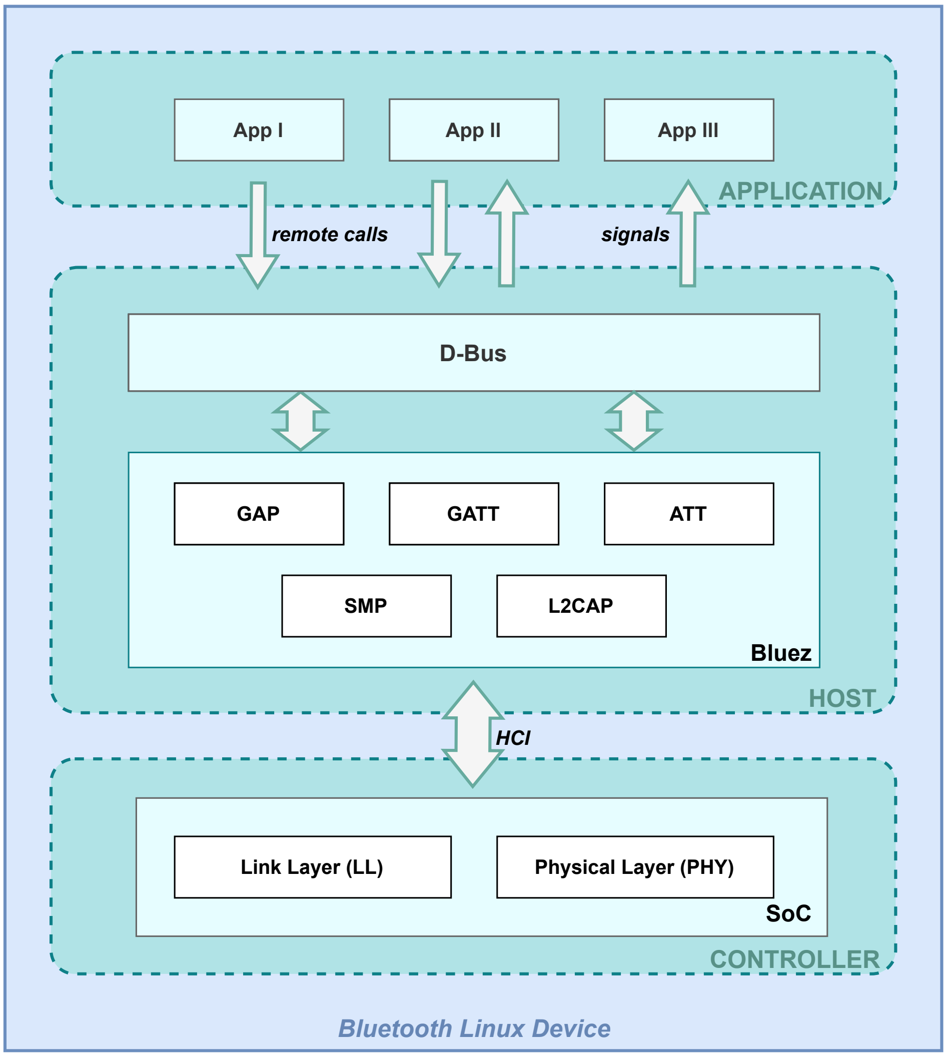


Figure 2.15: Bluetooth Protocol Stack.

Bluetooth profile

The Bluetooth profile is a specification that describes how devices must use Bluetooth protocols to implement a particular task, and this is the theoretical part of the Bluetooth architecture. Bluetooth services are the practical part of the Bluetooth protocol stack. There are various types of profiles available. Some are:

- Generic Attribute Profile (GATT) allows us to define a table of data which represents aspects of our device and their state at any given time and operations which can be carried out against that data. GATT allows us to apply a clean, hierarchical structure to our state data in the form of constructs called Services, Characteristics and Descriptors.

- Generic Access Profile (GAP) is another part of a Bluetooth stack. Its primary responsibilities concern how devices discover each other and how connections are then created or taken down. According to GAP, Bluetooth LE devices play one of 4 possible roles, as defined by that masterpiece, the Bluetooth Core Specification. The four roles are called Peripheral, Central, Broadcaster and Observer.

- The Attribute Protocol (ATT) is the layer of the Bluetooth LE stack which allows a connected GATT client to communicate with a GATT server and vice versa. For example, GATT allows clients to discover the GATT services that the remote server has in its attribute table, request the current value of a characteristic, change a characteristic value, turn on or off notifications and indications and much more.

2.2.7.1 Objects, Interfaces, Methods, Signals, and Properties

D-Bus uses an object-oriented paradigm for some of its concepts (as figure 2.16 suggests). Applications that use D-Bus are deemed to contain various *objects*. Objects implement *interfaces* that consist of a series of one or more functions or *methods*. An application can call a method of an object owned by another application by sending a special message over its connection to the bus. The message is passed along the bus and over a connection to the application which owns the target object.

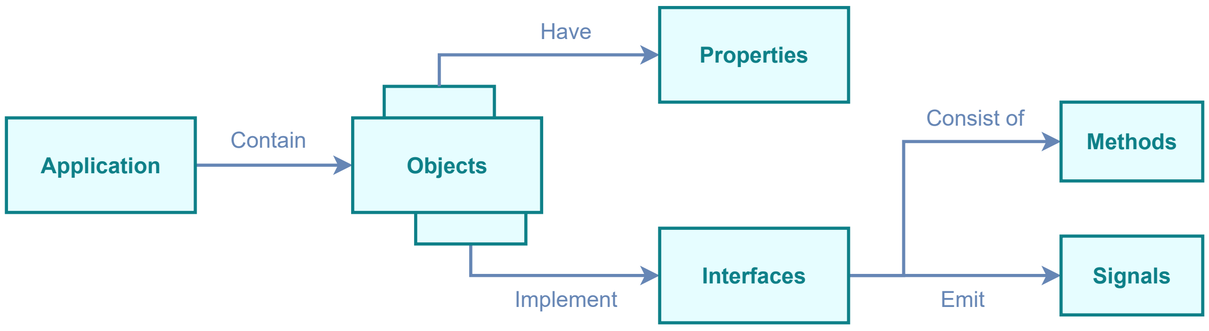


Figure 2.16: D-Bus elements.

Objects must be registered with the D-Bus daemon to allow their methods to be called by other applications. The act of registering an object with the bus is known as exporting. Each object has a unique identifier that takes the form of a *path*. The registration of an object with its path enables the D-Bus daemon to route messages addressed to the object's identifying path over the appropriate connection to the owning application. Servers often expose objects and are sometimes then referred to as D-Bus *services*.

An object or more precisely, an interface of an object may emit *signals*. A signal is a message that an object can send unprompted and can be likened to an event. Applications may subscribe to or register an interest in receiving signals. More than one application can register for a given signal and a copy of the signal will be delivered to each registered application. An object can have *properties*. A property is an attribute whose value can be retrieved using a get operation or changed using a set operation. Properties are referenced by name and are accessible via an interface that the object implements.

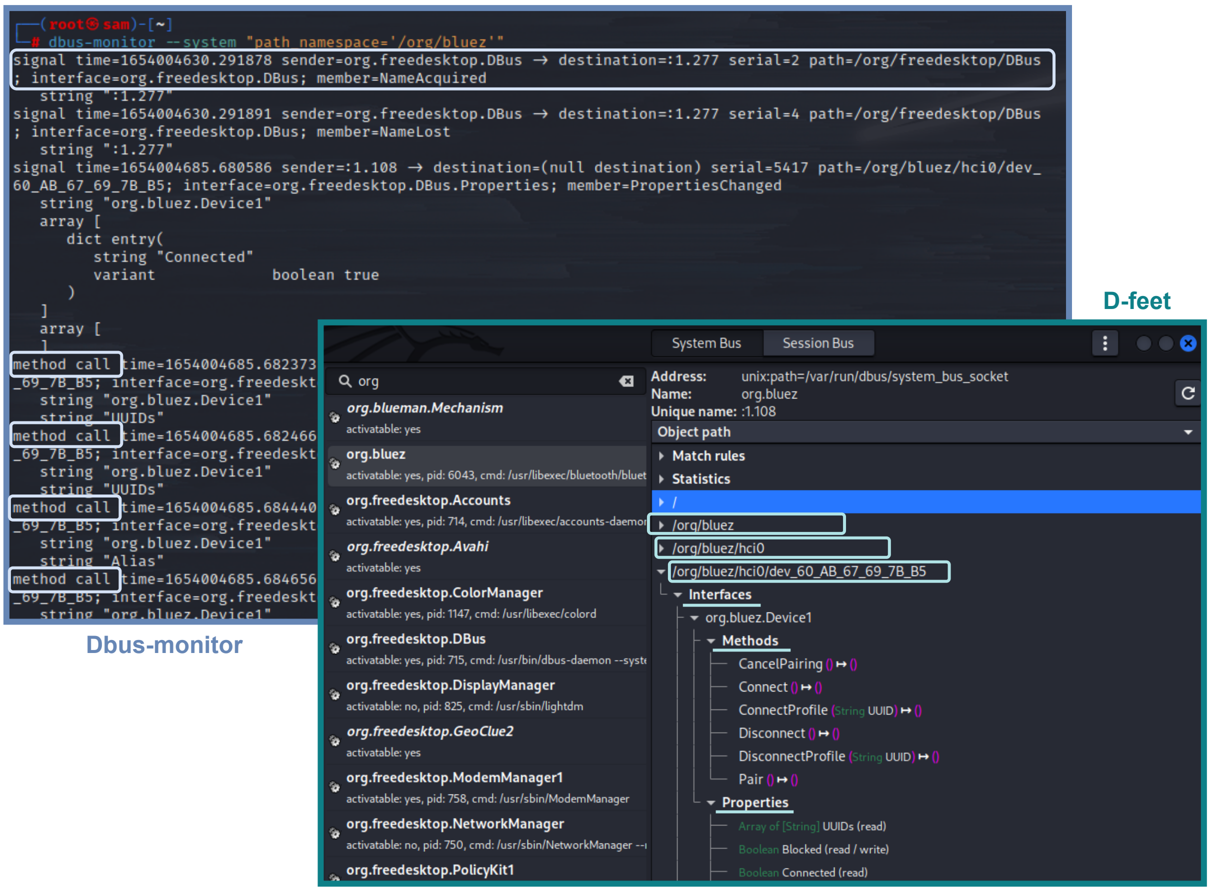
One can observe an introspection example illustrating the entire path and relating all the aforementioned elements in the following listing:

```
dbus-send --system --print-reply --dest=org.bluez /org/bluez/hci0
    org.freedesktop.DBus.Properties.GetAll string:"org.bluez.Adapter1"
```

Listing 2.1: Introspection example.

D-Bus Capture Tools

In order to capture the interaction between the applications and the services that they subscribe to, we can listen to the D-Bus communications. The various kernel threads and modules deployed from the stack protocols are in constant information exchange, and that can give the user a considerable awareness of what is happening and which services are being required given a specific moment.



```
┌──(root㉿sam)-[~]
└─# dbus-monitor --system "path_namespace='/org/bluez'"
signal time=1654004630.291878 sender=org.freedesktop.DBus -> destination=:1.277 serial=2 path=/org/freedesktop/DBus
; interface=org.freedesktop.DBus; member=NameAcquired
   string ":1.277"
signal time=1654004630.291891 sender=org.freedesktop.DBus -> destination=:1.277 serial=4 path=/org/freedesktop/DBus
; interface=org.freedesktop.DBus; member=NameLost
   string ":1.277"
signal time=1654004685.680586 sender=:1.108 -> destination=(null destination) serial=5417 path=/org/bluez/hci0/dev_
60_AB_67_69_7B_B5; interface=org.freedesktop.DBus.Properties; member=PropertiesChanged
   string "org.bluez.Device1"
   array [
      dict entry(
         string "Connected"
         variant             boolean true
      )
   ]
   array [
   ]
method call time=1654004685.682373
_69_7B_B5; interface=org.freedeskt
   string "org.bluez.Device1"
   string "UUIDs"
method call time=1654004685.682466
_69_7B_B5; interface=org.freedeskt
   string "org.bluez.Device1"
   string "UUIDs"
method call time=1654004685.684440
_69_7B_B5; interface=org.freedeskt
   string "org.bluez.Device1"
   string "Alias"
method call time=1654004685.684656
_69_7B_B5; interface=org.freedeskt
   string "org.bluez.Device1"
```

org.blueman.Mechanism
activatable: yes
org.bluez
activatable: yes, pid: 6043, cmd: /usr/libexec/bluetooth/bluet
org.freedesktop.Accounts
activatable: yes, pid: 714, cmd: /usr/libexec/accounts-daemor
org.freedesktop.Avahi
activatable: yes
org.freedesktop.ColorManager
activatable: yes, pid: 1147, cmd: /usr/libexec/colord
org.freedesktop.DBus
activatable: yes, pid: 715, cmd: /usr/bin/dbus-daemon --syste
org.freedesktop.DisplayManager
activatable: no, pid: 825, cmd: /usr/sbin/lightdm
org.freedesktop.GeoClue2
activatable: yes
org.freedesktop.ModemManager1
activatable: yes, pid: 758, cmd: /usr/sbin/ModemManager
org.freedesktop.NetworkManager
activatable: no, pid: 750, cmd: /usr/sbin/NetworkManager --
org.freedesktop.PolicyKit1

Address: unix:path=/var/run/dbus/system_bus_socket
Name: org.bluez
Unique name: :1.108

```
Object path
▸ Match rules
▸ Statistics
▸ /
▸ /org/bluez
▸ /org/bluez/hci0
▾ /org/bluez/hci0/dev_60_AB_67_69_7B_B5
  └ ▾ Interfaces
      ▾ org.bluez.Device1
        ▾ Methods
            CancelPairing () ↦ ()
            Connect () ↦ ()
            ConnectProfile (String UUID) ↦ ()
            Disconnect () ↦ ()
            DisconnectProfile (String UUID) ↦ ()
            Pair () ↦ ()
        ▾ Properties
            Array of [String] UUIDs (read)
            Boolean Blocked (read / write)
            Boolean Connected (read)
```

Figure 2.17: D-Bus capture tools.

D-feet (figure 2.17) is a GUI application which allows applications connected to the system or session bus to be viewed and the objects and interfaces they export to be browsed. It also allows methods to be executed which is useful for testing purposes. The *org.bluez* service has been selected in the left-hand pane and this caused the objects the service contains, each identified by a hierarchical path, to be listed. Objects can be expanded to show their supported interfaces and within interfaces we can see their methods.

Dbus-monitor (figure 2.17) is a command line tool that allows messages exchanged with the D-Bus daemon to be viewed in real time. When running *dbus-monitor*, either the system bus or session bus must be selected. Some system messages will only be visible if running as root and if eavesdropping has been enabled.

2.2.7.2 Probe Location Analysis

From D-Bus we are able to extract relevant information that allows us to infer some aspects of the Bluetooth communication established with the target device. The BlueZ stack contains several APIs, from the adapter, the device, GATT protocol, media, network, OBEX, or even profiles. Each of these has methods and properties. Calling these methods or querying these properties can be done through the command line. The screenshots above 2.17 show this functionality. Based on the captured content, we could profile normal usage and build a model capable of identifying moments of the attack on the Bluetooth system. Although it would be extremely advantageous to implement an intrusion detection system at this level (which has no precedent in the literature), this would be a method that we see as having the insufficient potential to identify with an interesting degree of certainty. Some of the reasons we dropped this approach are mentioned below:

- Limited capture options and poor documentation;
- Manual implementation of filters capable of selecting relevant information;
- Omission of important moments such as the connection; we are only aware of the end result and not the steps to achieve this, which turns out to be quite important because it is a critical moment when addressing Bluetooth security and possible attacks; wccording to a survey of the main threats, most of them can be identified during the authentication process;
- Regardless of the final indication, it is extremely difficult to correlate the data and justify an attack based on a set of system calls;
- When simulated with real traffic, the variety of system calls is very limited, with numerous repetitions but little information about what is being done by the user;
- Intrusion detection system limited to a Linux-based device.

2.2.8 Host Controller Interface

HCI [38] is a standardized Bluetooth interface for sending commands, receiving events, and sending and receiving data. This is typically implemented as a serial interface using an RS232 or USB communication device. As the name suggests, HCI is used to bridge host and controller devices. Commands and events can be specified or vendor-specific for extensibility.

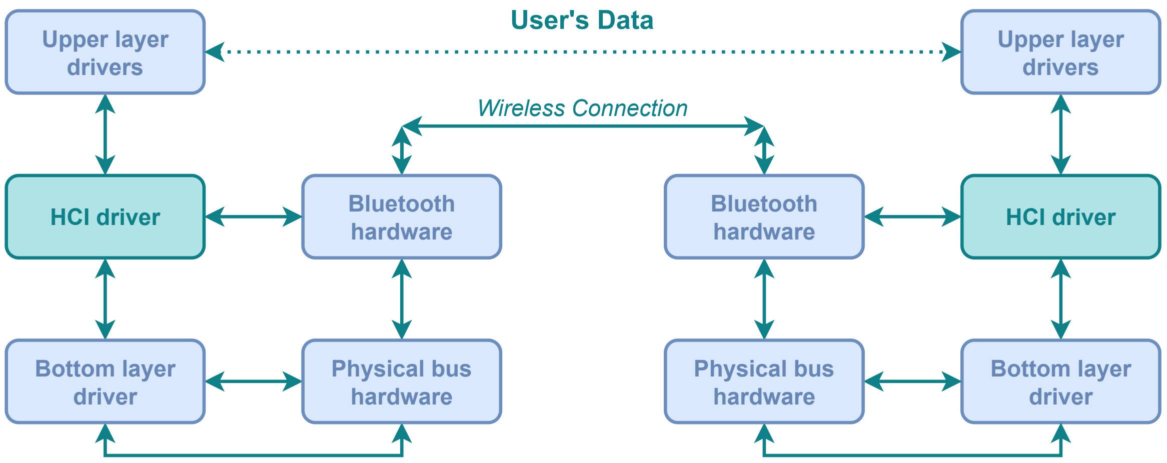


Figure 2.18: Bluetooth communication mode.

In embedded wireless MCU projects such as simple-peripheral projects, the HCI layer is implemented through function calls and callbacks within the wireless MCU. All of the commands and events discussed that communicate with the controller, such as ATT, GAP, etc, will eventually call an HCI API to reach the controller from the upper layers of the protocol stack through the HCI layer. Similarly, the controller sends received data and events to the host and higher layers via HCI.

Table 2.4: HCI packet types.

Packet	**Content**	**Packet type**
Command	Opcode, number of parameters, and the parameters themselves	0x01
Asynchronous Data	Connection handle, fragmentation bits, number of data bytes, and the data bytes themselves	0x02
Synchronous Data	Not used	0x03
Event	Event code, number of event parameters, and the event parameters themselves	0x04
Extended Command	Two-bytes payload	0x09

The HCI supports four types of packets: Command Packet, Asynchronous Data Packet, Synchronous Data Packet, and Event Packet. The packet type is a one byte value that precedes the HCI packet. The packet type can assume one of the types presented in table 2.4.

2.2.8.1 HCI Commands

HCI provides a uniform command method for accessing Bluetooth hardware capabilities. HCI Link commands allow the host to control link layer connections to other Bluetooth devices. These commands typically use the LM to exchange LMP commands with remote Bluetooth devices. HCI policy commands are used to affect the behavior of local and remote LMs. These policy commands provide hosts with a way to influence how LM manages the piconet. Host controller and baseband commands, information commands, and status commands allow the host to access various registers within the host controller. A closer description of these various standardized commands is presented below:

1. *HCI-Specific Information Exchange:* The host controller's transport layer ensures a transparent exchange of HCI-specific information. These transport mechanisms also provide the ability for the host to receive HCI events, ACL data, and SCO data from the host controller. Because the host controller's transport layer provides a transparent exchange of HCI-specific information, the HCI specification establishes formats for command, event, and data exchange between the host and the host controller.

2. *Link control commands* allow the host controller to control connections to other Bluetooth devices. Using link control commands the LM controls how Bluetooth piconets and scatternets are set up and maintained. These commands direct LM to create and modify link-layer connections with remote Bluetooth devices, query other Bluetooth devices within range, and other LMP commands.

3. *Link policy commands* allow hosts to influence how the piconet is managed by the link manager. Depending on tunable policy parameters, the LM controls how Bluetooth piconets and scatternets are established and maintained, even when link policy commands are used. These policy commands modify the behavior of the connection manager. This can change the link-layer connection with the remote Bluetooth device.

4. *Informational parameters* are set by the Bluetooth hardware manufacturer. These parameters provide information about the Bluetooth device and capabilities of the host controller, link manager, and baseband. The host device cannot change these parameters.

5. The host controller modifies all *status parameters*. These parameters provide information about the current status of the host controller, link manager, and baseband. The host device cannot change these parameters except by resetting certain specific parameters.

6. *Testing commands* are used to provide the ability to test various features of the Bluetooth hardware. These commands provide the ability to set various test conditions.

As shown in figure 2.19 HCI commands use a 16-bit opcode for identification. The opcode is subdivided into two parts: a 10-bit Opcode Command Field (OCF) and a 6-bit Opcode Group Field (OGF). The OGF values are defined by the Bluetooth Core specification. The LE specification has its own OGF value. Also, there is an escape OGF value so that vendor specific OCF codes can be used.

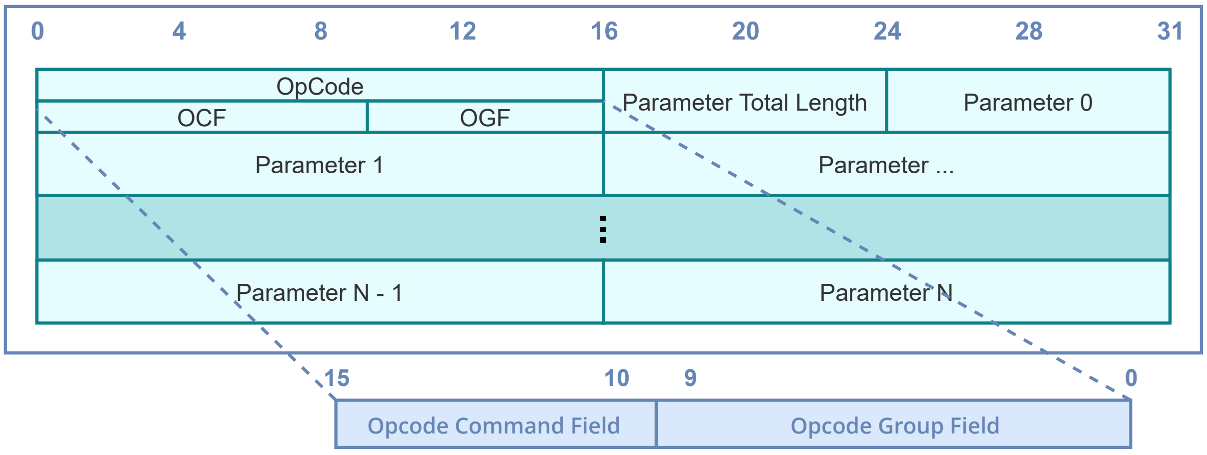


Figure 2.19: Command Packet.

2.2.8.2 HCI Capture Tools

TShark (*wireshark* homologous without GUI) is a network protocol analyzer that allows the capture of packet data from a live network or read packets from a previously saved capture file. It does so by printing the decoded form of these packets to standard output or by writing the packets to a file. *TShark*'s native capture file format is the *pcapng* format, which is also used by *Wireshark* and various other tools. Without any options set, *TShark* works like *tcpdump*, using the *pcap* library to capture traffic from the first available network interface and print a summary line for each packet received to standard output.

When displaying packets on the standard output, *TShark* writes, by default, a summary line containing the fields specified by the preferences file. Using *TShark*'s read filters to select which packets to decode or write to a file is very powerful. It lets us filter on more fields than other log analyzers and has a richer syntax for creating filters. As *TShark* progresses, it is expected that more protocol fields will be allowed in the read filter.

2.3 Intrusion Detection Systems

Intrusion can be defined as any set of actions that threatens the integrity, availability, or confidentiality of a network resource [39] [40]. Intrusions are caused by attackers accessing the systems, authorized users of the systems who attempt to gain additional privileges for which they are not authorized, and authorized users who misuse the privileges given to them. Hence, an Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues an alert when such activity is detected. A software application is implemented to scan a network or system for malicious activity or policy violations. Malicious activity and breaches are typically reported to administrators or tracked centrally using security information and event management systems.

Intrusion detection systems offer several benefits to business, starting with their ability to identify security incidents. An IDS can be used to analyze the number and types of attacks. Organizations can use this information to modify their security systems or implement more effective controls. Intrusion detection systems also help organizations identify network device configuration errors and problems. These metrics can be used to assess future risks. They can also help enterprises attain regulatory compliance.

An IDS allows companies greater visibility across their networks, making it easier to meet security regulations. Additionally, IDS can also improve security responses. Since its sensors can detect network hosts and devices, they can also work as data inspection within the network packets, as well as identify the OSes of services being used. Using an IDS to collect this information can be much more efficient than manual censuses of connected systems.

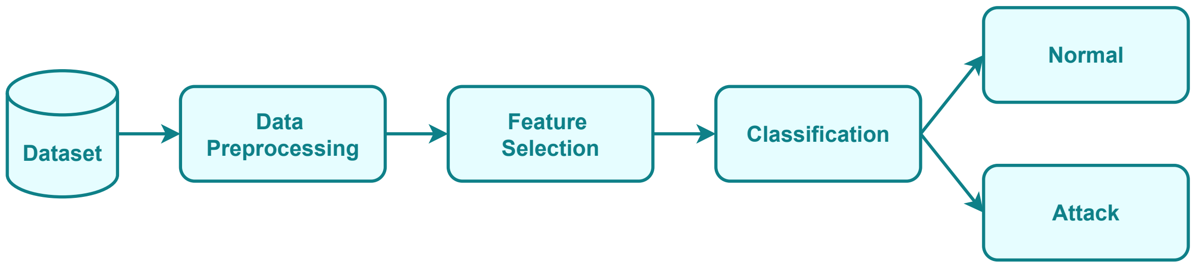


Figure 2.20: Block Diagram of Intrusion Detection System [41].

With increasing network and security threats the vehicle becomes more exposed, reason why the study of IDS has received a great deal of attention in all areas of computer science [42]. Even though various techniques of intrusion detection are performed, their accuracy is one of the main issues, as detection rate and false alarm rate still fall short of what is reasonable. A serious IDS mistake is a false negative, where the IDS misses a threat and misidentifies it as legitimate traffic. In false-negative scenarios, IT teams have no indication that an attack is taking place, and often discover it only after the network has been compromised in some way. Thus, intrusion detection needs to be enhanced to reduce false alarms and increase detection rates [41].

2.3.1 Common Types of Intrusion Detection

There is a wide spectrum of IDS, varying from complex software to hierarchical systems that monitor the traffic of an entire backbone network. Each implementation is characterized by different monitoring and analysis approaches, which have distinct advantages and disadvantages. The most common classifications are Network

Intrusion Detection Systems (NIDS) and Host-based Intrusion Detection Systems (HIDS), depending on if they were implemented as software applications running on customer hardware or as a network security appliance. Cloud-based intrusion detection systems are also available to protect data and systems in cloud deployments.

Network Based IDS (NIDS)

This class of IDS [40] are deployed at scheduled points in the network to inspect traffic from all devices on the network. Monitor the traffic that traverses the entire subnet and compare traffic forwarded on the subnet to a collection of known attacks. Alerts can be sent to administrators when attacks are identified or when anomalous behavior is observed. Network IDS uses either network taps, span ports, or hubs to collect packets traversing a particular network. Based on the data collected, the IDS system processes and flags suspicious traffic. Unlike intrusion prevention systems, intrusion detection systems do not actively block network traffic. The Network IDS role is passive, collecting, identifying, logging, and alerting only. An example of a NIDS is installing it on the subnet where firewalls are located in order to see if someone is trying to crack the firewall. *Snort* and *Suricata* are other good exampes of NIDS free to use and run on Windows, Linux, and Unix.

Host Based (HIDS)

A HIDS [40] runs on a separate host or device on the network. HIDS only monitors incoming and outgoing packets from devices and alerts administrators when suspicious or malicious activity is detected. It generally involves an agent installed on each system, monitoring and alerting on local OS and application activity. The installed agent uses a combination of signatures, rules, and heuristics to identify unauthorized activity. The role of a host IDS is passive, only gathering, identifying, logging, and alerting. A use case for HIDS is found in mission-critical machines where layout changes are not expected. *SolarWinds Security Event Manager (SEM)* is a great example of a HIDS with a robust lineup of automated threat remediation tools.

Physical (Physical IDS)

Physical intrusion detection [40] is the act of identifying threats to physical systems. Physical intrusion detection is most commonly thought of as the physical controls put in place to secure the CIA. In many cases, physical intrusion detection systems also act as defensive systems. There have been quite a number of cases that have gathered the limelight and shown the need for security in some places. Further to this, physical intrusion detection systems were also adopted to improve the overall security of some public places. Layered security reflecting utilization of multiple levels of security equips the system with redundancies that improve the system's ability to detect, assess and track intruders who are seeking unauthorized access to secure areas or facilities and its customization property.

Hybrid IDS

A hybrid intrusion detection system is created by combining two or more intrusion detection system approaches. A hybrid IDS combines host agent or system data with network information to create a complete view of your network system. A hybrid intruder alarm system is more effective compared to other intruder alarm systems. *Prelude* is an example of a hybrid IDS.

Table 2.5: Pros and cons of different intrusion detection types.

	Network Based IDS	**Host Based IDS**
Pros	Can look at data in the context of the protocol. Can qualify and quantify attacks. Make it easier to keep up with regulation. Can boost efficiency.	Capable of verifying the success of an attack. Can monitor all users' activities. Do not require any extra hardware. Cost effective for a small scale network.
Cons	Required expertise to administer them. Do not process encrypted packets. False positives are frequent.	Useless if the host server is compromised. Requires extra computing power from the host. Can be ineffective during DoS attacks.

2.3.2 Detection Approaches

Intrusion detection methodologies are classified into three major categories: Signature-based Detection (SD), Anomaly-based Detection (AD), and Stateful Protocol Analysis (SPA). Table 2.6 [42] shows the pros and cons of three detection methodologies. Most IDS technologies use multiple methodologies, either separately or integrated, to provide more broad and accurate detection. These methodologies are described in detail below.

Signature-based Detection

A signature-based [40] IDS recognizes attacks based on specific patterns such as the number of bytes in network traffic, the number of 1s, and the number of 0s. It also detects based on known malicious command sequences used by malware. Patterns recognized by IDS are called signatures. A signature [43] is a pattern that corresponds to a known attack or type of attack. A signature-based IDS can easily detect attacks that already have patterns (signatures) in the system, but it is very difficult to detect new malware attacks because the patterns (signatures) are unknown.

This technique is also known as misuse detection and is the simplest detection method. In misuse detection [44], a pattern of known malicious activity is stored in the dataset and identifies suspicious data by comparing the current unit of activity, such as a packet or a log entry, against a list of signatures using string comparison operations.

It is very effective at detecting known attacks but largely ineffective at detecting previously unknown attacks, attacks disguised by the use of evasion techniques, and many variants of known attacks.

Anomaly-based Detection

Anomaly-based [43] [40] IDS was introduced to detect unknown malware attacks when new malware rapidly escalated. Anomaly-based IDS uses Machine Learning (ML) to build a trustworthy activity model, compares everything that happens to that model, and flags it as suspicious if it's not found in the model. Machine-learning-based methods have more generalized properties compared to signature-based IDS, as models can be trained according to the application and hardware configuration. Here baseline of normal data in network, for example, load on network traffic, protocol, packet size, etc are defined by the system administrator and according to this baseline, the anomaly detector monitors new instances. An IDS that uses anomaly-based detection has profiles that represent the normal behavior of users, hosts, network connections, applications, and so on. Profiles are created by monitoring typical activity characteristics over a period of time.

The main advantage of anomaly-based detection methods is that they can detect unknown attacks very effectively. Suppose a computer is infected with a new type of malware. Malware can consume your computer's processing resources, send large amounts of e-mail, initiate numerous network connections, and perform other actions that differ significantly from the computer's configured profile.

An initial profile is created over a period of time, sometimes called the training period. Profiles are either static or dynamic. Static profiles that are created do not change unless the IDS is specifically instructed to create a new profile. Dynamic profiles are continuously adjusted as additional events are observed. As systems and networks change over time, so do the corresponding measures of normal behavior. Static profiles eventually become inaccurate and must be recreated periodically. Dynamic profiles do not have this issue but are vulnerable to evasion attempts by attackers. For example, an attacker might occasionally perform small amounts of malicious activity, and then gradually increase the frequency and amount of activity. If the rate of change is slow enough, the IDS may consider malicious activity normal behavior and include it in its profile.

Another problem with profiling is that the calculations are so complex that it can be very difficult to be precise in some cases. For example, if particular maintenance activity of performing a large file transfer happens to him only once a month, it might not be observed during the training period. If maintenance occurs, it may be considered a significant deviation from the profile.

Anomaly-based IDS products, especially in more diverse and dynamic environments, often generate many false positives due to benign activity that deviates significantly from their profile. Another notable problem with using anomaly-based detection techniques is that it is often difficult for analysts to determine what triggered a particular alert.

Stateful Protocol Analysis

Stateful protocol analysis [43] is the process of comparing observed events to a specific profile of commonly accepted definitions of benign log activity for each protocol state to identify deviations. Unlike anomaly-based detection, which uses host-specific or network-specific profiles, stateful log analysis relies on vendor-developed universal profiles that specify how a particular protocol should or should not be used. Stateful means that an IDS can understand and track the state of the network, transport, and application protocols that are aware of the state. The ”protocol analysis” performed by this method usually includes reasonableness checks for individual commands, such as minimum and maximum lengths for arguments. However, the main drawback of the stateful protocol analysis method is that the analysis is complex and resource-intensive due to the overhead of performing state tracking for a large number of concurrent sessions. Another issue is that these methods cannot detect attacks that do not violate the properties of generally accepted protocol behavior.

Table 2.6: Pros and cons of intrusion detection methodologies.

	Signature-based
Pros	Simplest and effective method to detect known attacks. Detail contextual analysis.
Cons	Ineffective to unknown attacks, evasion attacks, and variants of known attacks. Little understanding of states and protocols Hard to keep signatures/patterns up to date. Time consuming to maintain the knowledge.
	Anomaly-based
Pros	Effective to detect new and unforeseen vulnerabilities. Less dependent on OS. Facilitate detections of privilege abuse.
Cons	Weak profiles accuracy due to observed events being constantly changed. Unavailable during rebuilding of behavior profiles. Difficult to trigger alerts in right time.
	Stateful Protocol Analysis
Pros	Know trace and protocol states. Distinguish unexpected sequences of commands.
Cons	Resource consuming to protocol state tracing and examination. Unable to inspect attacks looking like benign protocol behaviors. Might incompatible to dedicated OSs or APIs

2.3.3 Machine Learning Techniques for Intrusion Detection

As a branch of Artificial Intelligence (AI), ML [45] can be defined as a technique in which computers are trained to have the ability to automatically learn and improve or optimize performance criteria without being explicitly programmed, using past experience or example data. Machine learning model focuses on a training set of data according to features of interest so that different classes can be predicted.

There are many data mining techniques for intrusion detection such as frequent pattern mining, classification, clustering, mining data streams, etc. For IDS, the ML algorithm works more accurately in detecting the attacks for a huge amount of data in less time. Typically, ML algorithms can be classified into three categories: supervised, unsupervised and semi-supervised. During the learning phase, these algorithms use training algorithms to build recognition or prediction models based on training data. This predictive model is tested against new data during the testing phase to distinguish between benign and attack data. Input data must be preprocessed so that it can be understood by machine learning algorithms. Input data contains examples, also called instances, observations, or datasets.

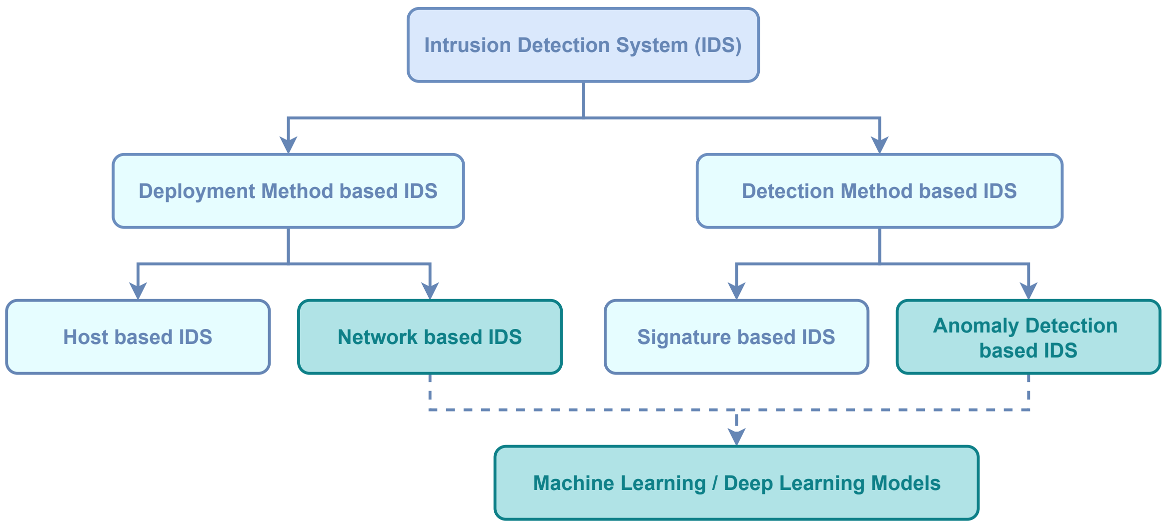

Intrusion Detection System (IDS)
Deployment Method based IDS
Detection Method based IDS
Host based IDS
Network based IDS
Signature based IDS
Anomaly Detection based IDS
Machine Learning / Deep Learning Models

Figure 2.21: Intrusion Detection System classification taxonomy.

2.3.3.1 Supervised algorithm

The supervised algorithm [46] deals with fully class-labeled data and finds the relationship between data and its class. This can be done either by classification or regression. The classification has two steps such as training and testing. The training data is done with the help of the response variable. The common algorithms under the classification category are Support Vector Machine (SVM), Discriminant Analysis, Naïve Bayes, Nearest Neighbour, Neural Network, and Logistic Regression. While some algorithms under the regression category are Linear Regression, Support Vector Regression (SVR), Ensemble Methods, Decision Tree, and Random Forest. In this essay, some of the most used methods are discussed.

- **Decision tree** [47] is a recursive and tree like structure for expressing classification rules. It uses divide and conquer method for splitting according to attribute values. Classification of the data proceeds from the root node to the leaf node, where each node represents the attribute and its value, and each leaf node represents the class label of data. Tree-based classifiers have the highest performance in the case of a large dataset.
- **Support Vector Machine** [48] is a method in which various types of data from different subjects get trained. In a high-dimensional space, SVM creates a hyperplane or multiple hyperplanes. The hyperplane optimally separates the given data into various classes with the major partition considered the best hyperplane. For evaluating margins between hyperplanes, a non-linear classifier applies several kernel functions. Due to the growing attention to SVMs, eminent applications have been established by developers and researchers. SVM deals the main role in image processing and pattern recognition applications. Usually, a classification task mainly involves dividing data into two sets namely training datasets and testing datasets. In that class labels will be defined as ”target variables” and attributes will be defined as features or ”observed variables”.
- **Naive Bayes** [48] classifiers are statistical classifiers. They are capable to forecast the probability that whether the given model fits to a particular class. It is based on Bayes’ theorem. It constructed on the hypothesis that, for a given class, the attribute value is independent to the values of the attributes. This theory is called class conditional independence.

2.3.3.2 Unsupervised ML algorithm

For intrusion detection, the unsupervised learning [46] algorithm will try to find out the hidden structure in unlabelled data. There is no training data for unsupervised learning. This can be done by clustering or association analysis or dimensionality reduction. The clustering algorithms such as K-Means, K- Mediods, and C-Means can be used. The dimensionality reduction algorithms such as Singular Value Decomposition (SVD) and Principle Component Analysis (PCA) can be used.

- **K-Nearest Neighbor** is one of the simplest classification techniques. It calculates the distance between different data points on the input vectors and assigns the unlabeled data point to its nearest neighbor class. K is an important parameter. If k=1, then the object is assigned to the class of its nearest neighbor. When the value of K is large, then it takes a large time for prediction and influences the accuracy by reducing the effect of noise.
- **K-means** is one of the unsupervised ML algorithms. This algorithm works based on the finding groups in the data, and the number of groups can be represented by the variable. K-means algorithm is highly used in time series data for pattern matching. The drawback of this algorithm is it is not applicable for a non-spherical form of data.

- **Long Short-Term Memory** [49] are RNNs capable of capturing long-term dependencies that connect consecutive tasks. The primary goal of LSTM is to achieve disappearing gradient descent which is an optimization algorithm to find artificial neural network weights to avoid long-term dependency problems.

2.3.3.3 Semi-supervised ML algorithm

Supervised learning requires a large amount of data to build a good classifier. The more training data we use, the better performance we get for the supervised machine learning classifier. Unfortunately obtaining a large amount of training data is a costly process. Semi-supervised learning [50] requires just a small set of labeled data in the training dataset. If we use only a small set of labeled data, we may obtain poor performance for the supervised learning classifier. In order to improve the performance, we consider using unlabeled data in the training data set in semi-supervised learning. Unlabeled data does not require costly human-labor tasks. We hope that we could improve the performance of the classifier in support of unlabeled data in the training dataset. This is a fundamental motivation for our use of semi-supervised learning.

2.3.4 Review of Intrusion Detection Systems

Intrusion detection can be built at several levels in a real computer network system. It will be the selection of features that best characterize the user or system usage patterns so that a clear distinction is made between anomalous and normal activity.

Currently, we have Bluetooth security products such as *Air Defense*, *Red-Detect*, *BlueAuditor*, or *AirMagnet*. These solutions mainly focus on detecting devices within a certain range and demand extensive human intervention to detect intrusions in the network. Even though they are capable of monitoring all Bluetooth enable devices and preventing intrusions in a network, it becomes very expensive and lacks critical configuration abilities. Hence, they fail to present important alerts to administrators.

On the other hand, one can consider tools like *SNORT* or *Suricata*. *SNORT* is a signature-based intrusion detection/prevention system, and it can perform real-time packet analysis on an IP network. *Suricata* is a fast real-time intrusion detection that is a signature-based, rule/policy-driven security, and anomaly-based approach for detecting intrusions. It inspects the network traffic using rules and signatures.

Early anomaly detection research focused on profiling systems or user behavior from monitored systems or accounting log data. Among the aforementioned traditional methods to classify patterns, we have distinct accuracies. Those methods have evolved, and new approaches have appeared in the literature. Pattern classification approaches are widely used in various IDS implementations. But since Bluetooth attacks exploit the vulnerabilities of the Bluetooth protocol, barely any systems have been designed based on data collected from Bluetooth mesh network flows.

In this field, Satam *et al.* [2][51] propose an anomaly behavior analysis approach to distinguish between normal and abnormal behavior in Bluetooth networks. Satam *et al.* use n-grams to characterize the regular behavior of the Bluetooth protocol. A machine learning model is used to distinguish between the normal behavior of the Bluetooth protocol and its behavior when under attack. Later on, they extended the Bluetooth IDS to operate in infrastructure mode as a Multi-Level Bluetooth Intrusion Detection System (ML-BIDS) [52]. The ML-BIDS could detect a wide range of Bluetooth attacks and prevent malicious devices from using the network by using whitelisting. Such a technique is intended to continuously authenticate and authorize Bluetooth network devices, and their operations. Although its results have proven to be quite a successful tool, false positives and the limitations of the traditional ML methods are still something to be researched and surpassed.

2.3.5 Proposed Approach: LSTM-based IDS

Network attacks have patterns according to their type. These patterns generally do not appear in a single package but may be distributed across multiple packages. However, most previous machine learning techniques in network-based IDS have been unable to account for such features and capture patterns that appear in multiple packets. For example, the Multi-Layer Perceptron (MLP) ignores time dependencies and performs intrusion detection with just one packet. In fact, if one wants to detect DoS attacks with MLP, it is very difficult because DoS is an attack that shuts down a service by sending a large number of packets not much different from normal packets. This issue is not limited to DoS attacks but also applies to the other types mentioned before in this document. Therefore, multiple packets should be processed instead of single packets for more accurate intrusion detection.

Basic Concept

Traditional feedforward neural networks perform well for classification tasks, but are limited to examining single instances rather than analyzing a set of inputs. Sequences can be of arbitrary length, have complex time dependencies and patterns, and have high dimensionality. Examples of sequential data sets include text, genomes, music, speech, text, handwriting, stock market price changes, and images. These can be split into a series of patches and treated as a sequence [53]. Recurrent neural networks are built on neurons like feedforward neural networks, but with additional connections between layers.

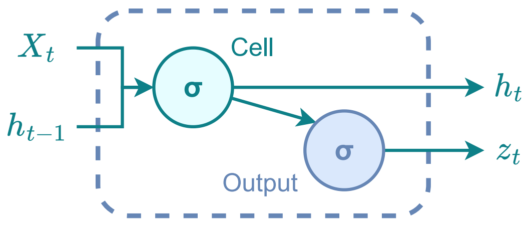


Figure 2.22: RNN cell.

LSTM is a Recurrent Neural Network (RNN) architecture that remembers values at arbitrary intervals. LSTMs are well suited for classifying, processing, and predicting lag time series of unknown duration. Due to its relative insensitivity to gap length, LSTM outperforms alternative RNNs, hidden Markov models, and other sequence learning methods. One of the advantages of LSTM is its insensitivity to gap length. RNN relies on the hidden state before emission/sequence. If one wants to predict the sequence after 1,000 intervals, the model forgot the starting point by then. In that case, the architecture which allows LSTM to remember is an RNN cell, as shown in figure 2.23.

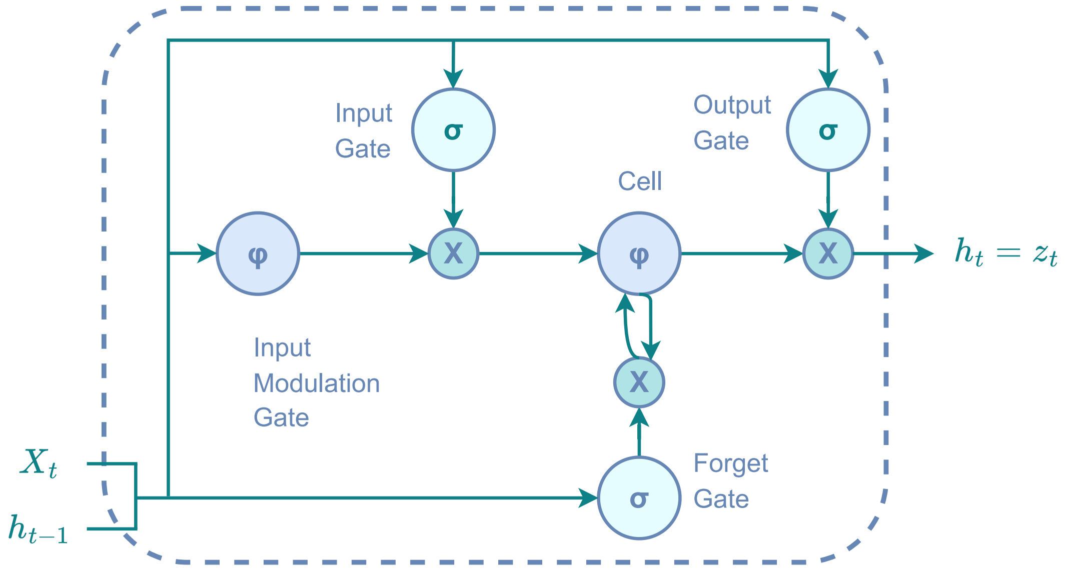


Figure 2.23: LSTM cell.

Long-term memory is commonly referred to as a cellular state. The loop arrow indicates the recursive nature of the cell. This allows information from previous intervals to be stored in the LSTM cells. The cell state is changed by a forget gate placed below the cell state and also set by an input modulation gate. From the equation 2.1, we forget the previous cell state by multiplying the forget gate and add new information by outputting the input gate.

$$c_t = f_t \times c_{t-1} + i_t \times c_t + \tilde{c}_t \tag{2.1}$$

Memory vectors are commonly called forget gates. The output of the forget gate tells the cell state which information to forget by multiplying the position in the matrix by 0. When the forget gate output is 1, the information is held in the cell state. Then, according to equation 2.2 a *sigmoid* function is applied to the weighted inputs/observations and the previous hidden state. Then, the save vector is usually called the input gate, whose formula is given in 2.3.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_t) \tag{2.2}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.3}$$

These gates determine which information should enter the cell state / long-term memory. The important parts are the activation functions for each gate. The input gate is a *sigmoid* function and has a range of [0,1]. Because the equation of the cell state is a summation of the previous cell state, the *sigmoid* function alone will only add memory and not be able to remove/forget the memory. Finally, the output gate (or focus vector) is expressed by 2.4, and the hidden state (or working memory) is given by 2.5.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.4}$$

$$h_t = o_t \times tanh(c_t) \tag{2.5}$$

W_f , W_i and W_c are weights, and b_f , b_i and b_c are bias. The *sigmoid* layer makes the decision, it is called the *"forget gate layer"* and it outputs between 0 and 1. C_t is the new cell state, it is obtained from the old cell state $C_{t\text{-}1}$ and is regulated by the input it and forget f_t gates.

In supervised learning, we have an input variable (X) and an output variable (Y). We use an LSTM model to learn a mapping function from inputs to outputs: $Y = f(X)$. The objective is to determine the mapping accurately; Therefore, given new input data (X), we can predict the output variable (Y). We give an input sequence $X = (X_1, X_2, X_3, ..., X_i)$ and $X_j = (x_{j1}, x_{j2}, x_{j3}, ..., x_{jn})$, $j[1, ..., i]$, where x_jz , $z[1, ..., n]$, is an element of the input variables (X), and we obtain an output sequence $Y = (y_1, y_2, y_3, ..., y_i)$, where y_q , $q[1, ..., i]$, is an element of the output variables (Y).

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ . \\ . \\ . \\ X_i \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2n} \\ x_{31} & x_{32} & x_{33} & \cdots & x_{3n} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ x_{i1} & x_{i2} & x_{i3} & \cdots & x_{in} \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_i \end{bmatrix}$$

For each time step, the input X_j is each row/HCI packet payloads of the corresponding Bluetooth communication dataset. Each row/HCI packet is an observation comprised of N features as input variables (X) and one output variable to be predicted (Y).

Proof of Concept's Design

As we progressed in the investigation, it was possible to start shaping the problem statement and, consequently, the best way to solve it. This chapter focuses essentially on explaining every design step in detail. Firstly, it is presented an overview of this proof of concept to get an abstract idea, and subsequently, the global problem is broken into small pieces. Each one of the particular steps requires decisionmaking based on the aforementioned literature. Therefore, the next segment will discuss these solutions and the implicit techniques and tools to implement them.

3.1 Background

A thorough understanding of the concepts laid out is necessary to extract the fundamental underlying issues. First, we must be sensitive to the ease with which an attacker can gain access to a car. With this concern as a basis, and interpreting Bluetooth as a harmful means of access, arises the urge to offer the market a way of counterattacking.

Bluetooth communication spectrums are so different that their behavior must be studied in great detail. In relation to this scope of research, the priority is to find out the best way to characterize this communication. Traces of typical behavior can be found at distinct levels. Thus, a prominent impact of building a proof of concept is to determine the reliability of a probing tip.

Additionally, proving that an interaction behavior can be characterized and lead to anomaly detection is an equally interesting challenge. For these reasons, this research will be a milestone in combating this vulnerability. Depending on the results, we will be capable of validating the security strategy and declare this branch of Artificial Intelligence as an important aid in solving this type of issue.

Considering all the concepts reviewed, from all the specifics of Bluetooth communication to systems capable of detecting intrusions in that communication, we present below the outline of a proof of concept addressing the underlying challenges.

We have seen that Bluetooth technology presents great complexity from the way the network devices are connected to all the security measures employed. Although communication technology is only one, it is hard to design a system that fits all communications. Depending on the Bluetooth version and the settings of each of the devices contained in the network, the connection properties will differ. However, security operations such as authentication, authorization, and encryption, when required, always present a similar set of procedures. This characteristic will allow us to analyze the communication spectrum and understand how a system behaves at the pairing stage. Once we are able to make this distinction between procedures, we can infer whether they were carried out normally or not.

When building a proof of concept such as this, it is important to address the vulnerabilities studied in order to overcome them. There are communication details that the user abstracts away. Many times responsible usage by the user could be enough to overcome those vulnerabilities, though. Given an attacker who is aware of how the system is exposed, if he can't get in by taking advantage of the user's sloppiness, then he will exploit further entries.

Within the range of attacks presented above, we can see that most are carried out by forging an insecure connection. Analyzing the integrity attacks, which are considered the most harmful to the system, they all take advantage of the knowledge of the Bluetooth stack and exploit the different security operations or protocols. This activity naturally occurs at the beginning of the connection. For this reason, we have designed an intrusion detection system that not only checks the entire spectrum of communication but gives particular relevance to the moment of connection between any two devices.

Thus, the objectives of this experiment are meant to meet the current security systems' lack mentioned in state-of-the-art. In that regard, we following propose a unique tool that employs unprecedented intrusion detection strategies for Bluetooth communication.

3.2 Overview

Whenever we want to design the solution to a problem, we must think about how to solve each of its specifics while never ceasing to imagine how it will fit into the overall context, and that is what we did by splitting the problem into parts. In this way, we start by presenting the solution as a whole (observe figure 3.1), to later go into detail. The implementation of an intrusion detection system is not linear and, like many others, a slow process that often requires going back and redefining steps. However, along the way, we can distinguish four main stages of development.

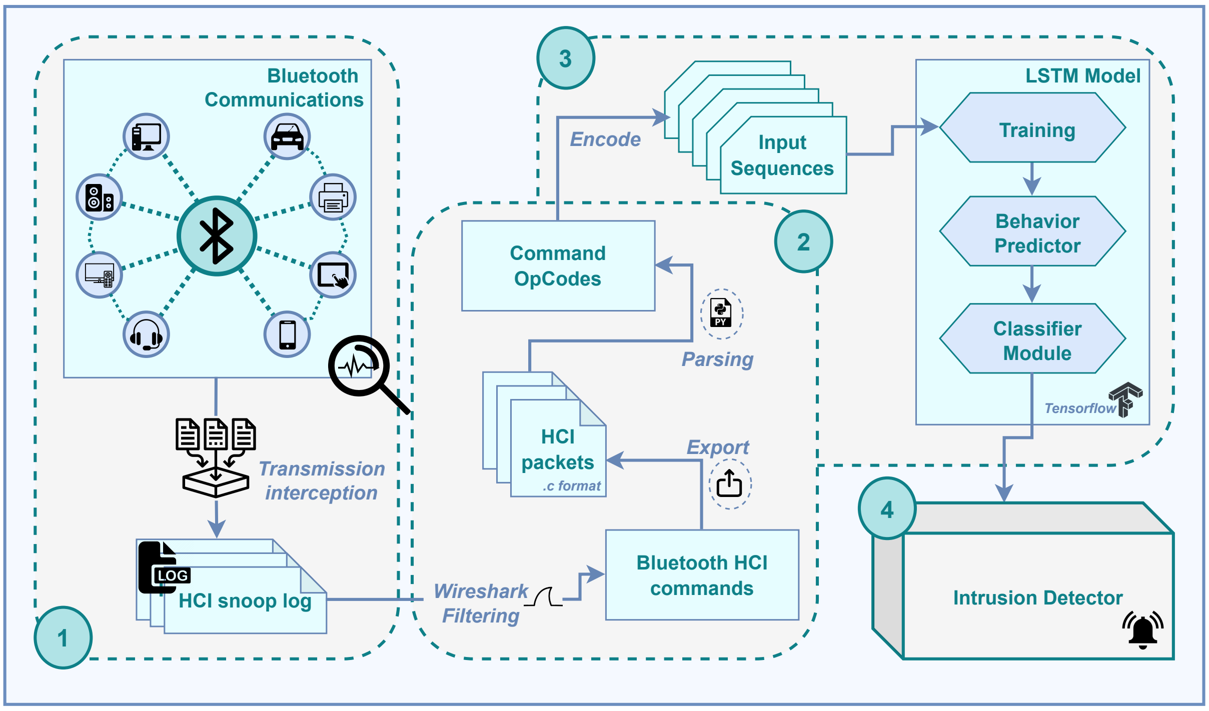


Figure 3.1: Overview of the Bluetooth Intrusion Detection System (BIDS).

The first stage, as any machine learning based solution demands, consists of gathering data to work with. More than that, regardless of the problem, it requires this step to be explored very carefully. The better it is researched and the more value the data collected, the better the end result. In the context of our problem, to characterize Bluetooth communication in a proper way, it is necessary to broaden the spectrum of collected data as much as possible. Complete coverage occurs at the HCI level, regardless of the device considered at both extremities of the communication. Consequently, the record of this capture is stored in the form of log files. The collection of a set of files with various communications represents the final step of this first stage.

Next, we must analyze and transform the collected data into a format that our machine learning model will understand in the future. Very briefly, we need to apply the right filters to restrict Bluetooth communication to the sequence of HCI commands in it. From there, using a splitting and indexing process, we get the *OpCode* sequence that translates those events. Having completed this, we are towards the data that will form the next stages' output and the input of the next one.

In regard to the field of deep learning, we must start by translating our inputs into normalized numerical sequences through a process of encoding. Once this is done, the most critical stage of the implementation initiates. To get started, it is necessary to study in depth the type of model that we want to implement. At that point, as feedback from the training is obtained, it is relevant to understand which parameters can be adapted to improve the training results. The process is completed when the model is able to predict the behavior of Bluetooth communication with satisfactory accuracy. In this case, we are referring to the correct prediction of the next HCI command considering the plot that is passed to it. Assuming this feature is captured by the system, the classification of the observed data is from the similarity of the data to the predictions made: constituting our classifier module.

Finally, it is necessary to wrap up all the modules previously developed so that we can look at the system as a black box. That box is our intrusion detection system. As input it should receive any log file recording a Bluetooth communication and as output determine whether or not it is anomalous behavior. If so, ultimately an alert should be generated for the user.

3.3 Data Collection

ML depends heavily on data. This is the most crucial aspect that makes algorithm training possible and explains why machine learning has become so widespread in recent years. Regardless of the actual amount of information and data science expertise, if we can't make sense of data records, a machine learning algorithm would be nearly useless. There are no perfect datasets, which is why data preparation is such an important step in the machine learning process. In short, as figure 3.2 suggests, data preparation is a series of steps that make a dataset more suitable for machine learning. Data preparation also includes setting up appropriate data collection mechanisms. And these techniques consume most of the time spent in machine learning.

Knowing what we want to predict is the major key to deciding which data may be more valuable to collect. Assuming a regression problem, we want an algorithm to yield some numeric value. Thus, we have to establish the data collection mechanisms accordingly. The basic concept of our system is to predict future Bluetooth traffic based on known communication patterns. Another thing to consider is the quality of those conventions. For instance, the same records can be duplicated due to some error when capturing the traffic. Those details can determine whether we possess a consistent dataset or not.

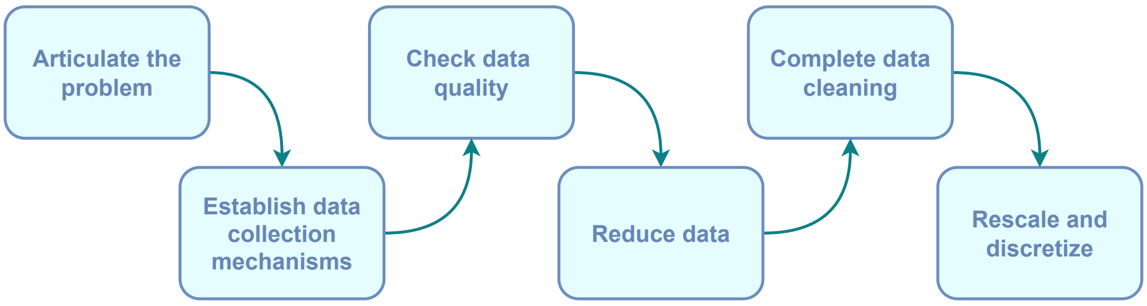


Figure 3.2: Preparation of a robust dataset for ML training.

Within the context of big data, we may be driven to include as much data as possible, which is wrong-headed. We definitely would like to enclose all data possible. However, when preparing a dataset with this particular task in mind, it is reasonable to reduce data. If, on the one hand, including all available Bluetooth traffic would give the algorithm more information to learn from, it might be challenging to converge on one accurate outcome. Figure 3.3 illustrates the general idea of this phase definition.

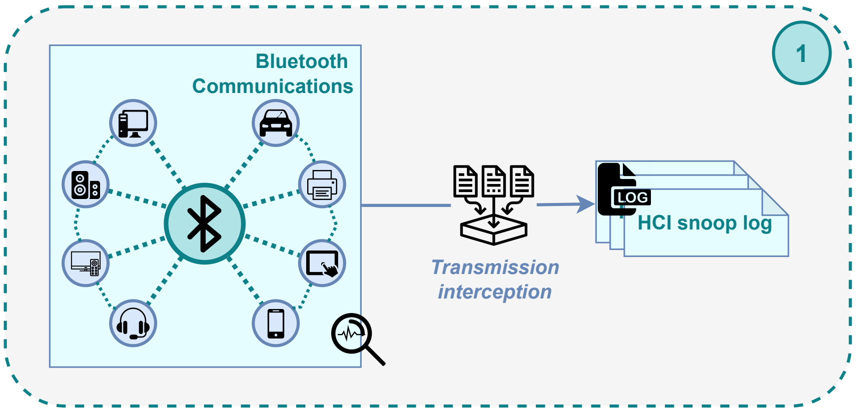


Figure 3.3: Proof of concept: step 1.

Based on the exhibited architecture, and abstracting from the necessary pre-processing of the data, the collection mechanism was defined as follows (see also figure 3.4):

1. Select any two devices capable of communicating via Bluetooth: This selection should be made as randomly and widely as possible so that the data is not limited to communication resulting from certain configurations. Indeed, the device configuration directly affects the steps sequence of a connection procedure. It is also

important to include devices with different Bluetooth versions because they have varying levels of security mechanisms.

2. Activate HCI traffic logging on one of these devices: After mentioning the various HCI capture mechanisms available, we must target a device containing such features to log it into a file. Examples are any paired computer or cell phone.

3. Activate Bluetooth and establish the connection: Once the data recording is running, we can start the experiment. The Bluetooth connection and its establishment should only take place at this point. An early start will stop us from getting the most meaningful raw material, which is the sequence of steps for a connection to be accepted by both ends.

4. Perform random operations according to the devices in question: For example, imagine that device A is a computer, and device B is a speaker. The capture is, logically, done on the master side (computer). In this case, the most evident interaction would be to send multimedia content to be played by the speaker. Involved in this experience is interesting from a behavioral point of view to explore the different operations associated with the respective profile. In the case of a connection between a cell phone and a car infotainment system, associated with the hands-free profile we have the start/end of calls or sending messages. Each of these operations will generate a distinct execution order.

5. Close the connection and extract the log file: It is also important for our algorithm to understand what a safe exit is. That is the reason why we should close the connection before we simply stop recording. From there, the process is to extract the file, either directly using tools such as *Wireshark* or by enabling debug mode in an android environment.

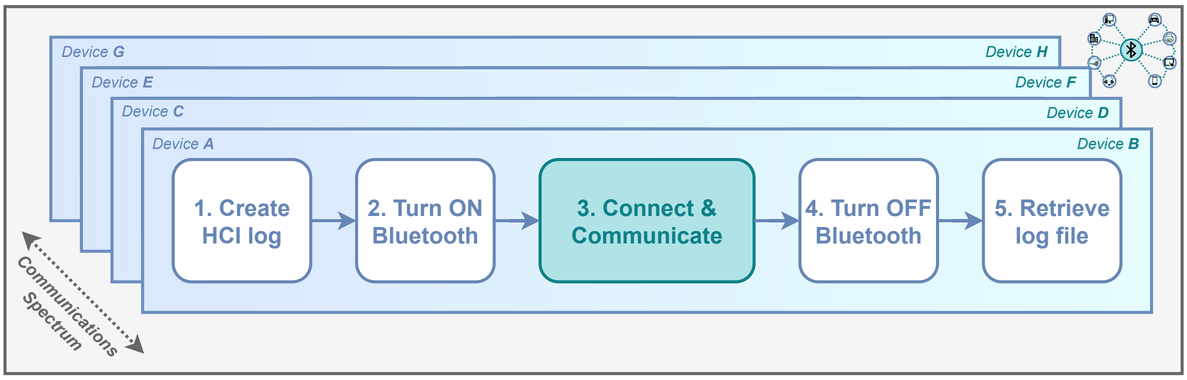


Figure 3.4: Schematic of data collection basic mechanism.

In conclusion, our dataset spectrum will be defined by the variety of devices and communication profiles that we can include, being its dimension proportional to that scope. On the other hand, according to different sources, this dataset is sufficient when its overall size exceeds its variability by at least ten times, which means that the more diverse the content of the captures, the more information the system needs to distinguish the relevance of each one.

3.4 Data Filtering and Processing

The second step of this project involves processing the data and formatting it in a way that serves the system. The processes of data reduction, cleaning, and discretization that follow its collection are, according to our design, converted into the progression scheme in figure 3.5. With it as a basis, the first content to be processed is the set of HCI packages using either a graphical tool like *Wireshark* or a more simplistic one like *Tshark*, so that it is possible to filter the recorded communication.

The data passing through the plot can be of different natures, but the ones we are interested in are the commands executed between the controller and the host, regardless of direction. For this primary selection, one can use the *bthci_cmd* input. Such an operation will dramatically reduce the size of the dataset, but it still contains unnecessary information. Data such as the packet size and the contents of its parameters, until proven to be extremely necessary, will be discarded. In this sense, the next challenge is to extract the *OpCode* of the command in question by decoding the second and third bytes of the packet.

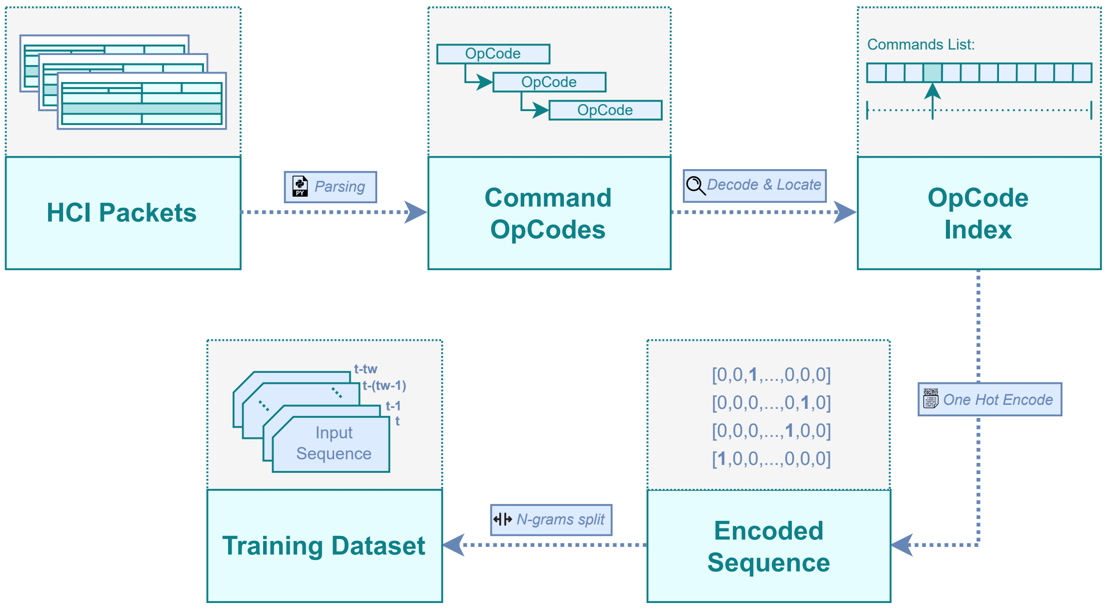


Figure 3.5: Data pre-processing sequence.

This time there is no tool that isolates this content for us. Therefore, we will start by doing the extraction of the entire packets into a data format that we can easily manipulate. One of the options that the capture tools offer us is to convert the information to C language format files. Although we are familiar with these types of files, implementing their decoding in the same language requires unnecessary effort. The simplest way to read the log files and separate the information in them is through a script written in a higher-level programming language such as Python. Using a library whose purpose is to perform this kind of parsing operation, we can model the sequences to be read and define the content to be extracted. Automating this procedure we then obtain the ordered list of command *OpCodes*.

When defining our model, one of the parameters that characterize it is the size of the data that feeds it. In this field, the size of the data will correspond to the diversity of the commands transmitted. To facilitate control over the number of parameters in our model, we must know beforehand the set of possible inputs. Responding to this requirement, the starting point will be to survey all commands. Once identified, we can index them in a database to improve their accessibility.

After taking these steps, instead of an HCI package, we then obtained a command, from which an opcode has been extracted and that boils down, at this point, to an indexing value.

3.4.1 Format Conversion

For categorical variables where no ordinal relationship among data exists, operations such as integer encoding are not enough. In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories). In this case, a one-hot encoding must be applied to the integer representation. This is where the integer encoded variable is removed, and a new binary variable is added for each unique integer value. Considering our context, there are N HCI commands (or categories) and therefore N binary variables are needed. A “1” value is placed in the binary variable for the command and “0” values for all others.

To complete the pre-processing of the data stage, we still need to divide the sequence using the n-grams model. The main idea of generating sequences using *n-grams* is to assume that the last input (x^n) of the n-gram can be inferred from the other inputs that appear in the same n-gram $(x^{n-1}, x^{n-2}, ..., x^1)$, which we refer as context. So the main simplification of the model is that we do not need to keep track of the whole sequence in order to predict the next command, we just need to look back for $n - 1$ tokens. Take a look at figure 3.6 for a better understanding of the concept.

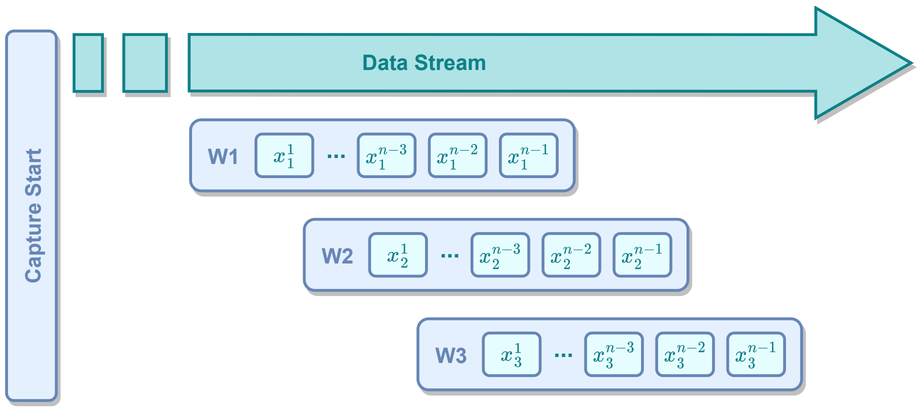


Figure 3.6: N-grams (or sliding window) methodology.

3.5 ML Model

At this stage of the project's development, it is anticipated that we will have at our disposal everything necessary to begin training the model. As stated in the literature, there is a great diversity of models aimed at different applications, being one of the most advanced to forecast time series the Long Short-Term Memory (LSTM) neural network. The LSTM cell adds long-term memory in an even more performant way because it allows even more parameters to be learned. This makes it the most powerful RNN to do forecasting, especially when we have a longer-term trend in our data, which is relevant for our problem. The training process also varies according to the chosen model, so let's consider the steps described across the following section in regard to a LSTM network.

The first thing to understand is how to link the data we have to the ML model concept to, ultimately, get a result. So notice in figure 3.7 that our data goes directly into an LSTM layer of the model. As highlighted, its configuration will be reviewed throughout the next section of this document, resulting in a dense layer that returns the output. The meaning of this value corresponds to the system's prediction for the consequent value of the input data.

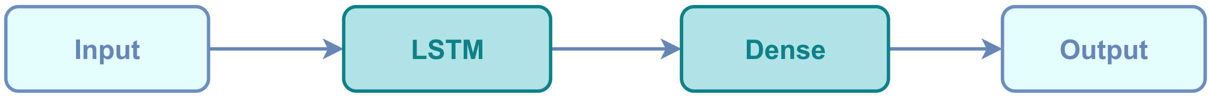


Figure 3.7: LSTM vanilla blocks diagram.

3.5.1 Hyperparameters

A hyperparameter is a specification whose value controls the learning process and determines the model parameter value that the learning algorithm ultimately learns. When designing the model, we select and set the hyperparameter values used by the learning algorithm before starting training the model. From this perspective, the hyperparameters are said to be external to the model because the model cannot change its value during learning/training.

Hidden States

When defining that layer, it is inherent the specification of the output space dimensionality, which corresponds to the number of hidden neurons. There is no strict rule for the number of hidden nodes we should use, as it is something that we have to figure out for each case by trial and error. However, some sources recommend the use of a formula that helps with supervised learning problems, preventing issues such as over-fitting. Let's observe equation 3.1, where:

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))} \tag{3.1}$$

- N_i = number of input neurons;
- N_o = number of output neurons;
- N_s = number of samples in training dataset;
- α = an arbitrary scaling factor, usually 2-10.

It is desirable to limit the number of free parameters (or the number of orders or non-zero weights) in the model to a small subset of the degrees of freedom in the data. The degrees of freedom for the data is the number of samples multiplied by the degrees of freedom for each sample (dimension), or $N_s \times (N_i + N_o)$ (assuming all are independent). Thus, α is a way of indicating how well the model generalizes or avoids overfitting. For the automated procedure, it is suggested to start with an alpha of 2 (twice as many degrees of freedom in our training data as in our model). Afterward, we can work the way up to 10 if the error (loss) of our training dataset is significantly smaller than the testing one.

Input and Output Shapes

Since our intention is to use *n-grams* as a feed-forward technique for the model, our input layer is automatically bounded by the dimensions of the predefined sequence. This requires a reshaping of the data prior to input. For better visualization of the previous hyperparameters definition, take a look at figure 3.8.

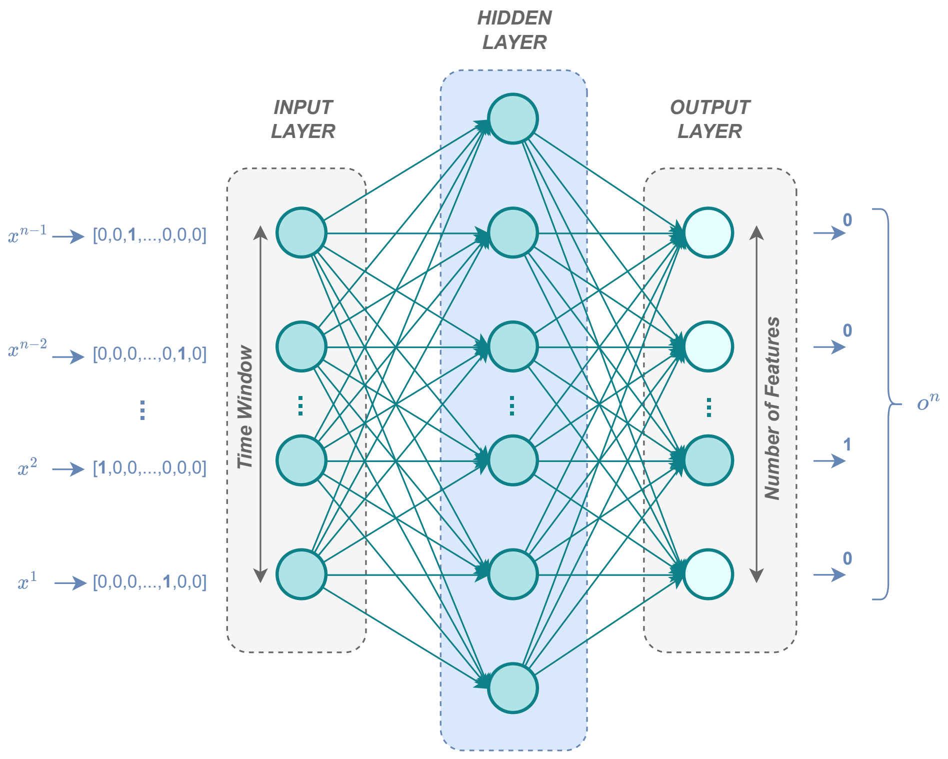


Figure 3.8: LSTM network.

The expected form is $1 \times$window_size $\times$ n_features, where *n_features* corresponds to the number of different values our input can take. In this particular problem, this characteristic can be understood as the total number of expected commands. At the other end, the output bed will be shaped only by the number of values that the system's prediction can take. The larger the number of possibilities (commands), the larger the size of the dense layer. At each training session, a prediction is made. It is dictated by the activation of one of the neurons present in the output layer. The activated neuron corresponds to one of the identified commands and is expected to be the next in the communication sequence.

There LSTM network structure illustrates the different layers with a representative size according to the key parameters. In addition, it is possible to deduce how layers and neurons are, in fact, interconnected. Although it is hard for the human brain to follow every path, we hereby present a global idea of the process.

Network Depth

As part of our model definition, the first aspect to consider is its depth, i.e. the number of LSTM layers. Extending the depth allows the network to learn a more powerful function from input to output. It should not be expected for a network's depth to increase its capability of learning long-term dependencies unless somehow the input-to-output transformation makes the dependency easier to remember.

Generally, increased depth will make the network harder and slower to train. If adding depth means increasing the number of parameters, then we might remember long-term dependencies better simply because we have a higher number of memory cells. But if we don't need that powerful transformation, we are better off adding those parameters to the first layer. Therefore, our model is composed of a single hidden layer.

Batch size

Neural networks are trained using gradient descent, where the estimate of the error used to update the weights is calculated based on a subset of the training dataset. The number of examples from the training dataset used in the estimate of the error gradient is called the batch size and is an important hyperparameter that influences the dynamics of the learning algorithm. In general, larger batch sizes result in faster progress in training. They don't always converge as fast, though. Smaller batch sizes train slower but can converge faster. It is definitely problem-dependent.

The error gradient is a statistical estimate. The more training samples used for estimation, the more accurate it will be. Moreover, it is more likely that the network weights will be adjusted, in a way that will improve the performance of the model. The improved estimate of the error gradient comes at the cost of having to use the model to make many more predictions before the estimate can be calculated, and in turn, the weights updated. Alternatively, using fewer samples results in a less accurate estimation of the error gradient, which is highly dependent on the specific training samples used. This leads to noisy estimations and noisy updates to the model weights. Nonetheless, these noisy updates can result in faster training and a more robust model [54].

Even though it is hard to know off the bat what the perfect batch size is for our needs, we will start with a preset value of 64. This decision permits a certain balance, since it is expected to allow our model a quick learning rate, while it guarantees a stable learning process.

3.5.2 Compilation Properties

Previously, we studied the basics of how to create a model using and this chapter explains how to compile the model. The compilation is the final step in creating a model. The compilation process encompasses concepts such as activation and loss functions, optimization, and evaluation metrics.

Activation function

Simply put, an activation function is a function that is added into an artificial neural network in order to help the network learn complex patterns in the data. The choice of hidden layer activation function controls how well the network model learns the training data set. The choice of output layer activation function defines the type of predictions the model can make. Typically, a differentiable nonlinear activation function is used in the hidden layers of a neural network. This allows the model to learn more complex functions than a network trained using a linear activation function.

In our model, we will make use of the hyperbolic tangent activation function, also referred to simply as the *Tanh* function. The function takes any real value as input and outputs values in the range -1 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

Loss Function

As part of the optimization algorithm, the error of the model's current state must be estimated repeatedly. This requires choosing an error function, commonly called the loss function, that can be used to estimate the loss of the model so that the weights can be updated to reduce the loss in the next evaluation.

The selection of such a function must be made according to the problem under consideration. Regression and classification require different loss functions. As we intend to build a model capable of predicting a real value, we step into the field of regression problems, whose default loss function is Mean Squared Error or MSE.

Mean squared error is calculated as the average of the squared differences between the predicted and actual values. The result is always positive regardless of the sign of the predicted and actual values and a perfect value is

0.0. The squaring means that larger mistakes result in more errors than smaller mistakes, meaning that the model is punished for making larger mistakes.

Optimizer

Optimizers in machine learning are used to tune the parameters of a neural network in order to minimize the cost function. They are divided into two families: gradient descent and adaptive optimizers. This division is exclusively based on the learning rate.

To address this field, we have chosen an adapted algorithm called Adam. The direction of the step is determined by a moving average of the gradients and the step size is approximately upper bounded by the global step size. Adam is well-known for achieving good performance with little hyperparameter tuning.

3.5.3 Training Phase

To perform machine learning, it's usually recommended to split a given dataset into three parts: a training set, a validation set, and a test set. The training set is used for training. The validation set is for estimating the prediction error of model selection. The test set serves to evaluate the generalization error of the final selected model. A general rule of thumb is to split the dataset into 50% for training and 25% for validation and testing [55].

The difference between the validation set and the test set is that the validation set is used to tune the network parameters based on the error rate. From validation, we select the model with the best performance. The test set is used only to evaluate the performance of the selected model, so no tuning is required during testing. In our case, the goal is to build a model that can predict the next HCI command index. We are towards a generative model that can create new traffic sequences by sampling output probabilities. So we have a training and validation set to refine the model, but no test set. Instead, we can input random batches of data from training to generate the output.

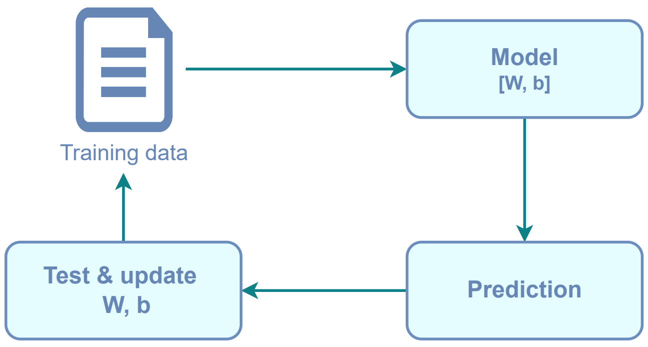


Figure 3.9: Training phase.

Now we move onto what is often considered the bulk of machine learning - the training. In this step, we will use our data to incrementally improve our model's ability to predict the next communication element. The training process involves initializing some random values for W (weights) and b (bias) and attempting to predict the output with those values. In the beginning, it is expected to perform quite poorly. However, we can compare our model's predictions with the alleged output, and adjust the values in W and b such that it will improve its predictions.

This process then repeats (as it can be observed in figure 3.9). Each iteration or cycle of updating the weights and biases is called one training "step". The number of steps required to train the model will depend mostly on the results obtained. In order to get a closer notion of the model performance during training, some metrics can be defined. One of the most objective metrics to evaluate our prediction results is accuracy. While training, the system automatically calculates the successful predictions per number of attempts. Other metrics, namely loss value are used, but their interpretation is not so straightforward. Resuming, once we find the predictions reasonable enough, no more training steps are required.

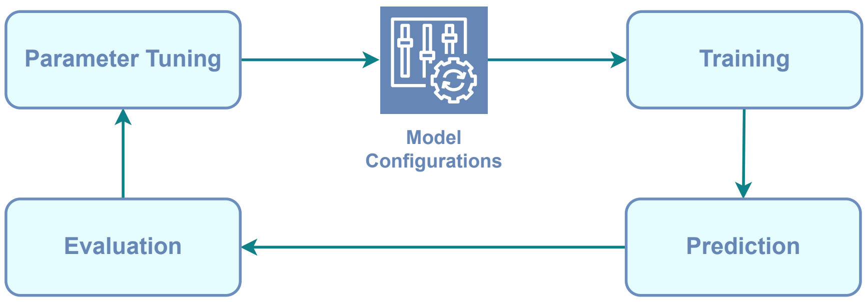


Figure 3.10: Parameter tuning.

Once we have done the evaluation, it's possible that we want to see if we can further improve the training in any way. We can do this by tuning our parameters. There are a few parameters we are implicitly assuming when defining the training, and then it will be an opportune time to come back and test these assumptions and try other values (cyclic procedure illustrated in figure 3.10).

3.5.4 Classification Module

After we are able to accurately predict the sequence of commands that make up the interaction between the host and the controller, we can test this implementation against real behaviors. Without the classification module, the values achieved are meaningless.

Once we get the behavioral prediction of a given communication thread we can calculate the accuracy of these results. In addition to this accuracy, we must also consider a certain tolerance. The more different the data is from actual communication, the lower the system accuracy will be. Such output means that the analyzed behavior differs from what the system is used to observing. And all the system knows is benign data, i.e. communication segments without any anomalous activity. Since an anomaly-based IDS has been defined, such situations are identified when this difference occurs. What is important to highlight is that this decision is composed of a decision boundary.

In order to place the decision boundary in a zone close to reality, we must assume that our model is not perfect. And since it is not perfect it doesn't know all the possible and imaginary data. In this case, it was not given as input all the possible scenarios for the combination of commands. Therefore, tolerance must be admitted for the new data to be analyzed. Even so, this tolerance can also not be too large. Otherwise, we face the risk of not identifying all anomalies.

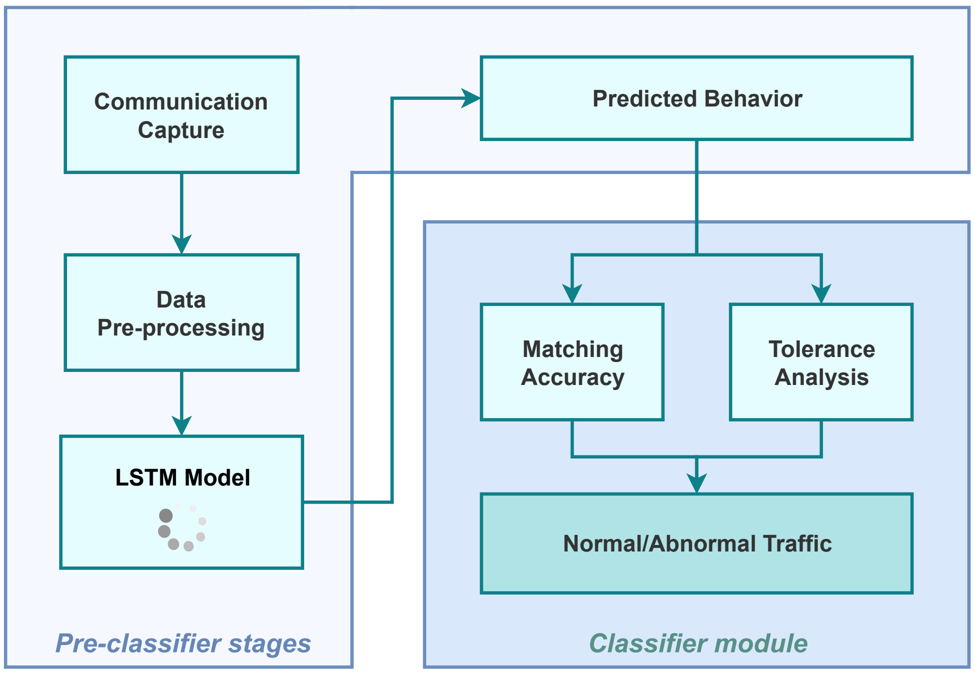


Figure 3.11: Classifier module schematic.

Only through a test situation will it be possible to define these values. At this design stage, it is, however, possible to foresee the need for this flexibility and to say regardless of this margin, the system cannot be ambiguous. As soon as the calculation formula takes real values, the decision will determine whether the observed traffic highlights an alert situation for the user or not.

3.6 Intrusion Detector

As a final step in the implementation of the project, we can isolate the challenge of wrapping all the modules. This stage is extremely important for validating this proof of concept in the sense that it allows, in a simple and universal way, its production and application in any situation.

Therefore, from an abstract point of view, our system is composed of two major modules: sniffer and behavior analysis modules. The first is responsible for collecting data from any Bluetooth host. The second encompasses all the transformation of these data up to their analysis and classification. Take a look at figure 3.12:

In practical terms, this stage includes automating the processes so that the data can flow between the different tools without the need for user intervention. The user only has to identify the communication to be analyzed and wait for the result.

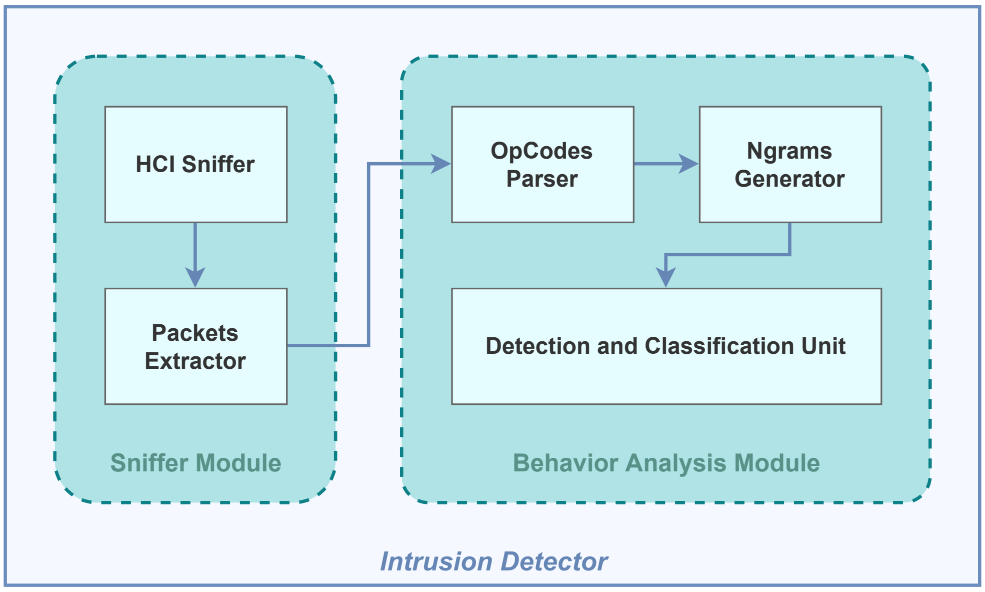


Figure 3.12: Architecture of the final Intrusion Detector.

Implementation and Results

In order to validate the proposed architecture, the following section presents a description of the Proof of Concept implementation process as well as its results. We start by introducing the tools used for implementation and testing, proceeding with the PoC development and, finally, the results under different circumstances. The main point of this stage in the global methodology is to confront each outcome with the expected value and infer whether or not our tool accomplishes to fulfill the challenges we proposed to solve.

4.1 Environment and Tools

Since the roots of this project are tied to a Linux environment and there is still much to be discovered in this field, it started by studying the OS behavior in relation to the Bluetooth operations performed. Even though it didn't end up as the chosen approach, there is one main idea that persists. We are towards a product designed for applications in Linux systems. Although equivalent operations can perfectly be performed in other operating systems, the effort will be made to optimize the processes aiming at a final application that runs in this environment.

Following this, the first tool to be highlighted is *Tshark*, whose usage procedure has already been explained above. Nonetheless, it is worth mentioning that its choice was made based on its simplicity of handling.

Regarding the second implementation phase, this tool proves to be insufficient to model the data we need. As a consequence, we resorted to a well-known high-level language: Python. Distinguished by its code readability and numerous libraries, Python is dynamically-typed and garbage-collected, supporting multiple programming paradigms. Within this world of possibilities it offers to the programmer, one particular library was used to perform operations with regular expressions. A regular expression (or RE) specifies a set of strings that matches it. This way it was possible to implement the parser to extract the necessary information contained in the HCI packages.

From there onwards, the whole development of the ML-based algorithm was designed on top of very specific environments for this sort of application.

4.1.1 Model Development

TensorFlow

In this study, the machine learning algorithm was developed using the *Tensorflow* library. *Tensorflow* is an open-source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture allows the deployment of computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. The TensorFlow User Guide provides a detailed overview and looks into using and customizing the *Tensorflow* deep learning framework [56].

Keras

Keras is a high-level neural networks API written in Python and capable of running on top of *Tensorflow*. It supports rapid experimentation, prototyping, and an easy-to-use API, making it very popular in the research and development community. *Keras* focuses on being modular, user-friendly, and extensible. It does not handle low-level computations; instead, it hands them off to another library called *Backend*.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

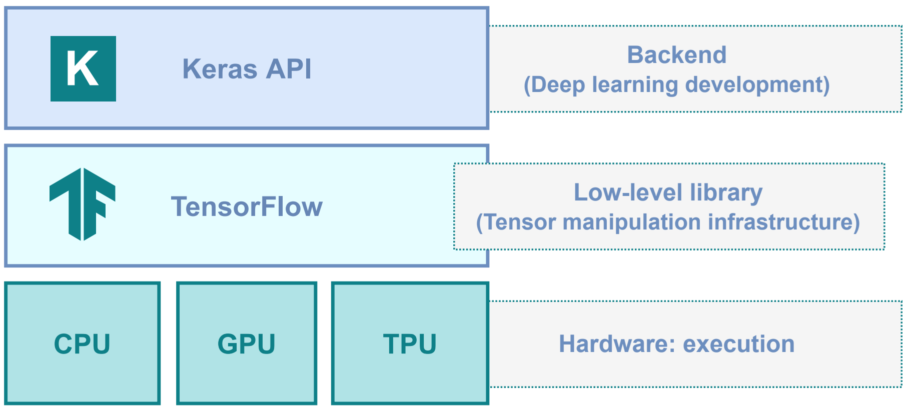


Figure 4.1: *Keras* and *TensorFlow*.

4.1.2 Project Testing

Since this is a proof of concept, it is essential to plan its testing conditions. In order to respond to the main challenges that the tool proposed to solve, we selected a framework that allows us to simulate these circumstances. As part of the test vector, we adopted a framework to perform several attacks and compare the traffic classification with the expected output, making up the essential test cases to prove the value of this proof of concept. Those tests cases are mostly dependent on the features offered by the testing tool as we considered them to be sufficient for our purpose. Moreover, these development stage as it was defined must explore all the test hypotheses and expose possible limitations in order to contribute the most for this research. Resuming hereby is presented *BrakTooth*.

4.1.2.1 BrakTooth

The Bluetooth Classic (BT) protocol is a wireless protocol widely used in laptops, handheld devices, and audio devices. This report discloses *BrakTooth*, a new family of vulnerabilities in commercial BT stacks ranging from DoS to firmware crashes and deadlocks on commodity hardware to arbitrary code execution (ACE) in certain IoTs. Some of the more closely related vulnerabilities are being researched and documented already [57].

4.2 showcases the generic scenario in which BrakTooth attacks are performed. The attacker only requires an ESP32 development kit (ESP-WROVER-KIT [58] in the present case) with a custom (non-compliant) LMP firmware and a computer to run the Proof of Concept (PoC) tool. The tool communicates with the ESP32 board via serial port (*/dev/ttyUSB1*) and launches the attacks according to the specified target BDAddress (*<target bdaddr>*) and exploit name parameter (*<exploit_name>*). Furthermore, the PoC tool logs over-the-air (OTA) packets and checks the health of the target by getting a paging timeout (no response) or alternatively getting status directly from the target via a serial port, ssh connection, etc.

This tool is composed of two main segments: the BT interface and the Host System. The host system is Linux-based and comprises the BT exploiter which will launch the attacks. The exploiter contains a duplication and a mutation module, both developed in C++. When running the toll and specifying the exploit, one of these modules is loaded through the BT Program Controller.

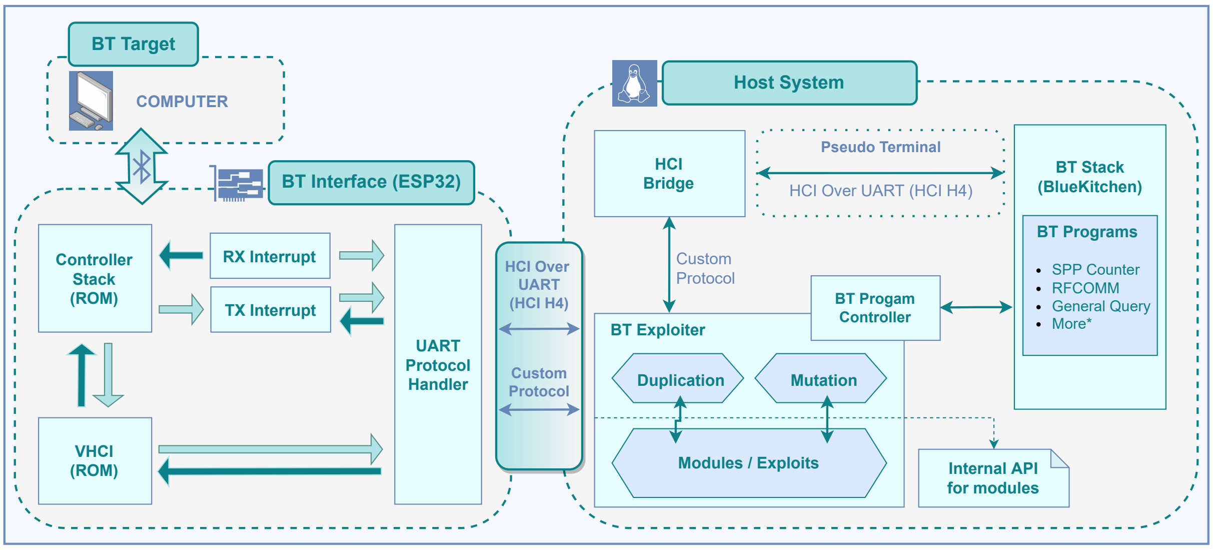


Figure 4.2: *Braktooth* architecture.

The basic operation is quite simple, being that the ESP32 connects directly to the target device. While receiving and sending packets to the destination entity, an interception occurs. Those data are forwarded to the PoC (Host System side), where one of two things can happen: The transmitted packet can be mutated, or code injection may occur, according to the current exploit. Such operations are triggered manually from inside the modules. This deviation of information emerges at each sending or receiving interruption signalized from UART. Since all the steps are intercepted, the toll allows its recording. Furthermore, it is possible to review and analyze them by the end of the communication. This particularity is especially important for our purpose since we aim to use the log files to find irregularities.

After filling in the hardware and software requirements and going through the installation process, one of the following operations can be done: running the BT fuzzer (either with or without a graphical interface), listing exploits, scanning targets, or even launching specific attacks.

4.2 Tool Implementation

Throughout this section, a demonstration of the actual implementation of the PoC will be showcased. The goal is that it can be easily understood and used, or even adapted in future research. The course of this guide will follow, by order, the implementation steps corresponding to the design presented. Thus, it will start with the data collection and preparation, going through the entire algorithm's development stages until it converges into the final classifier module.

4.2.1 Data Preparation

According to the systems' architecture, the primary phase of this implementation consists of preparing the data to enter the model. That process involves capturing the communication among the different devices and shaping that data following certain procedures, then generating the desired sequences of HCI commands.

4.2.1.1 Traffic Capture

As figure 4.3 illustrates, Bluetooth traffic capture can be subdivided into three phases. Although they possess a separate representation, each one of these steps is fully interconnected and its implementation is automated and resumed in a single script.

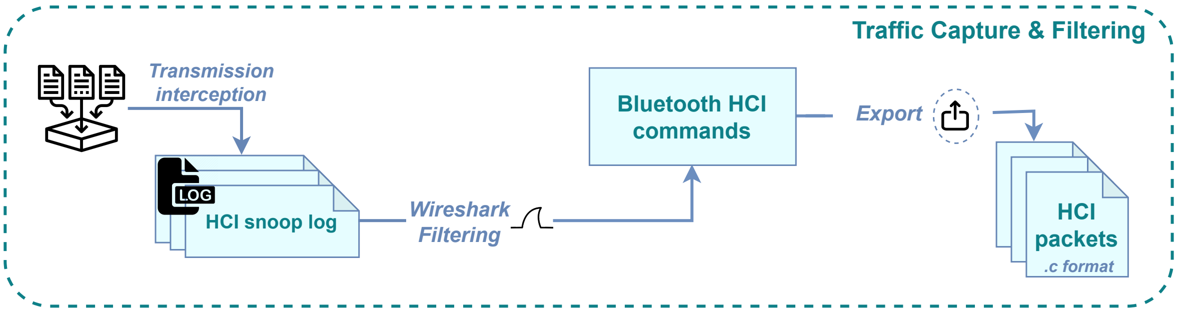


Figure 4.3: Bluetooth traffic capture.

The initial step consists of recording every activity between the host and controller entities. This can be executed either through direct capture using *Tshark* or by activating USB and HCI reporting (in the case of any android device). The generated log file (whose format can also be selected) is then opened necessarily with *TShark*. At that point, some of the numerous filters can be selected. To fit our design the processing commands are resumed to the following listing:

```
tshark [ -i <capture interface>|- ] [ -f <capture filter> ] [ -2 ]
```

Listing 4.1: Capture command.

4.2.1.2 Parsing Technique

The parsing process has the sole purpose of extracting the bytes corresponding to the OCF and OGF that make up the *OpCode* of a HCI command. As figure 4.4 suggests, the bytes in question are in the second and third positions of an array that can take up a variety of sizes. The input of this step is a C-format file with a set of arrays corresponding to the filtered packets, and the output is the list of *OpCodes*.

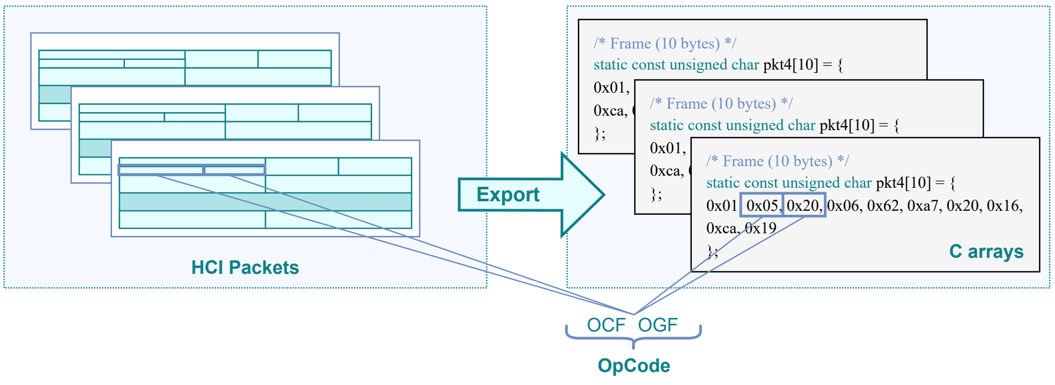

HCI Packets
Export
/* Frame (10 bytes) */
static const unsigned char pkt4[10] = {
0x01, 0x05, 0x20, 0x06, 0x62, 0xa7, 0x20, 0x16,
0xca, 0x19
};
C arrays
OCF OGF
OpCode

Figure 4.4: Data appearance.

The first step for the extraction is to read and split the file by lines. Next, each of the lines is analyzed to find a regular expression. This token is previously defined and corresponds to the hexadecimal representation of a number, whose range must fit the context of the problem. Once this identification is acquired, the algorithm ignores the remaining bytes until the next packet is started. In the end, a vector is returned with the successive results of this operation. This function is referenced whenever it is necessary to convert a new data sequence for the model.

4.2.1.3 Data Encoding/Decoding

One-hot encoding is the representation of categorical variables as binary vectors. This first requires mapping the categorical values. Each integer value is then represented as a binary vector with all zero values except the integer subscript, which is denoted by 1. For a better understanding, one can look at table 4.1.

Table 4.1: One-hot encoding.

Original		**One-hot Encoded**			
Command	**OpCode**	**Inquiry**	**Create_connection**	**...**	**Read_Clock_Offset**
Inquiry	0x0401	1	0	...	0
Create_connection	0x0405	0	1	...	0
...	...	...	...	...	...
Read_Clock_Offset	0x041F	0	0	...	1

Among the data preparation processes, we always have to encode the information into a readable input form. In addition, after training the model, it is necessary to apply the inverse process to compare the values and plot them. This operation consists of the instantiation of the *argmax()* function.

4.2.1.4 N-grams Generator

The last step before incorporating the data into the model is to split it into n-grams. This method (also designated as the sliding window) can be implemented as following expressed, being indispensable to provide the encoded sequence and desired window size to split the data.

```
def gram_generator(encoded_sequence, ngram_size):
    for i in range(len(encoded_sequence)-ngram_size):
        aux = encoded_sequence[i:i+ngram_size]
        X.append(aux)
        y.append(encoded[i+ngram_size])
    return array(X), array(y)
```

Listing 4.2: N-grams generator function.

It is necessary to go through the entire array storing segments of size N in one list and the element consequent to the segment in another list. This way we generate pairs X and Y that correspond to the inputs and the correct model prediction, respectively. Both follow as model inputs to further data training.

4.2.2 LSTM Implementation

Although there is already a lot of information about handling the different APIs that allow us to develop LSTMs, it is a complex process. The following section is intended to simplify it by breaking it down into its essential parts and exposing the different implementation techniques employed. We start by defining the model and its hyperparameters. Following this, it is necessary to specify the compilation specifications until training can proceed. Finally, the model is evaluated according to the predictions achieved, and parameter adjustments are made (if necessary).

4.2.2.1 Define the Model

The first step is to define our network. A neural network is defined with the support of the *Keras* API as a series of layers. The container for these layers is the *Sequential* class. As a stepping stone we must create an instance of the *Sequential* class. Then we can create layers and add them in the desired order. A recursive LSTM layer composed of storage units is called *LSTM()*. A fully connected layer that often follows LSTM layers and is used for outputting a prediction is called *Dense()*.

As designed in the previous section, we defined an LSTM hidden layer with 128 memory cells. Its input must be three-dimensional, comprised of samples, time steps, and features in that order. Following that, we define a *Dense* output layer containing as many neurons as columns of the input data, as listing 4.3 showcases.

```
model = Sequential()
model.add(LSTM(128, input_shape=(window_size, n_features)))
model.add(Dense(n_features))
```

Listing 4.3: Definition of the LSTM model.

4.2.2.2 Compile the Model

Once our network has been defined, we need to compile it. The compilation is an efficiency step. Depending on how *Keras* is configured, it transforms a simple sequence of defined layers into a series of highly efficient matrix transformations in a format designed to run on a GPU or CPU. Compilation can be interpreted as a pre-computation step for the network. It is a fundamental requisite after defining a model.

The configuration requires a set of parameters specific to training the network. In particular, the optimization algorithm used to train the network and the loss function used to evaluate the network is minimized by the optimization algorithm.

```
model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['acc'])
```

Listing 4.4: Compilation of the LSTM model.

For example, here is when we compile the predefined model and specify the *Adam* optimization algorithm and the mean squared error loss function for regression-type problems. Ultimately, there were also specified metrics to collect while fitting the model in addition to the loss function.

4.2.2.3 Fit the Model

Once the network is compiled, it can be adapted. This means fitting the weights according to the training dataset. To fit a network, we need to provide training data, both a matrix of input patterns X and an array of matching output patterns Y.

When compiling the model, backpropagation algorithm requires training the network for a specified number of epochs or exposures on every sequence in the training dataset. Each epoch can be divided into a set of input/output pattern pairs called stacks. This defines how many patterns the network will be exposed to before the weights are updated within an epoch. It is also an efficiency optimization to avoid loading too many input patterns into memory at once. Please, observe listing 4.5:

```
cp = ModelCheckpoint('model/', save_best_only=True)
history = model.fit(X_train, y_train,
          validation_data=(X_val, y_val),
          epochs=250,
          batch_size=64)
```

Listing 4.5: Fit of the LSTM model.

After fitting, a history object is returned that provides an overview of how the model performed during training. This includes both the loss and additional metrics specified when the model was compiled and recorded at each epoch. We can record, graph, and analyze these metrics to gain insight into whether the network is overfitting or underfitting the training data. Also, as we train the model, it will save the best configuration. After each episode, the result obtained is compared with the previous ones and, if better, a replacement is made.

4.2.2.4 Evaluate and Make Predictions on the Model

The network can be evaluated using the training data, but all that data has been seen before and is not a useful indicator of the network's performance as a predictive model. Therefore we use a separate dataset. The model evaluates the loss of all test patterns and other metrics specified when compiling the model, returning a list of metrics. For instance, a model's performance can be evaluated with an accuracy metric against a new dataset.

As soon as the performance of the fit model is satisfactory, we can load the best attempt and use it to make predictions on new data. All it requires is to call the predict function on the model with an array of new input patterns. Observe the following listing:

```
loss, accuracy = model.evaluate(X_val, y_val)
X_test, y_test = get_sequence("real_capture.c", window_size)
val_predictions = model.predict(X_test)
```

Listing 4.6: Fit of the LSTM model.

The predictions are returned in the format provided by the network output layer. In our case, it is necessary to perform further decoding in order to compare with the predicted results.

4.2.3 Classification Module

More than functional this proof of concept should be simple to use and test. To cope with that, we start the testing phase with a single module, isolated and abstracted from everything that precedes it. In preparation for the analysis, we start by loading the best model obtained. Next, using the same techniques, we read and parse the HCI packets file. After entering these data into the model, all the predictions are made. Finally, according to the accuracy achieved, a classification of the observed behavior is launched. Listing 4.7 shows how simple can be handling this tool from the command line.

```
Usage: classifier.py [OPTIONS] FILE
Arguments:
  FILE  [required]
Options:
  --w INTEGER             [default: 25]       # specify window size
  --t / --no-t            [default: no-t]     # print table w/ predictions
  --f / --no-f            [default: no-f]     # plot results & save figure
  --help                  Show this message and exit.
```

Listing 4.7: Python command line tool.

4.3 Model Evaluation

As previously mentioned, the process of developing a ML-based algorithm does not have a linear evolution, being quite often necessary to take the training to the end, interpret the results and adapt certain parameters. To this succeeds a training repetition, and so on. Following the theoretical references of the tool's design, after a few iterations, some promising results were obtained.

The present section starts by showing this process of understanding and criticizing the model's outputs. A good understanding of the returned data is critical for manipulating the inputs and diversifying the test situations. Having surpassed that barrier, the testing phase moves to a different stage. Hence, the model is submitted to a tool capable of simulating attack environments in the sense of validating the overall purpose of the thesis. Specific test cases were stipulated for both moments in order to achieve the highest guarantees.

Table 4.2: Test cases for model evaluation.

Test case ID	**Test case description**	**Expected result**
1	Check predictions accuracy from training data.	Near maximum accuracy.
2	Evaluate the influence of the sliding window.	Higher accuracy for larger contexts.
3	Measure models' acknowledgment of normal traffic.	Pattern recognition and high rate of successful predictions for all devices.
4	Confront the model with random data.	No pattern recognition and failed predictions.

Regarding the first testing phase, it is necessary to predict what to expect and antecipate the classification results by confronting the algorithm with data from different natures. Besides, some predefined parameters may require validation to ensure they represent the optimal solution, reason why there were defined four different test cases with the respective expected output. The testing sequence showcased in table 4.2 is an iterative process that gradually adds confidence to the model as it performs.

4.3.1 Training Results

The most rudimentary way to test the model's performance obtained during training is by testing its ability to predict the outputs according to the input sequence. Such test is performed using the data selected to train the model. If it has, indeed, learned, it will present values identical to the predicted ones.

Table 4.3: Predictions for training data: 98,61% accuracy.

	Train Predictions	**Actual Values**	...		**Train Predictions**	**Actual Values**
0	169	169	...	18 994	112	112
1	174	174	...	18 995	173	173
2	98	98	...	18 996	167	167
3	100	100	...	18 997	188	188
4	96	96	...	18 998	173	173
5	71	71	...	18 999	134	134

Table 4.4 shows the comparison between the values debited by the model and those that were part of the actual sequence. Since the training sequence contained about 19 000 points (corresponding to different commands), only its resume is presented. In this sense, it can be stated that the model was able to correctly predict the correct output in 98.61% of the cases.

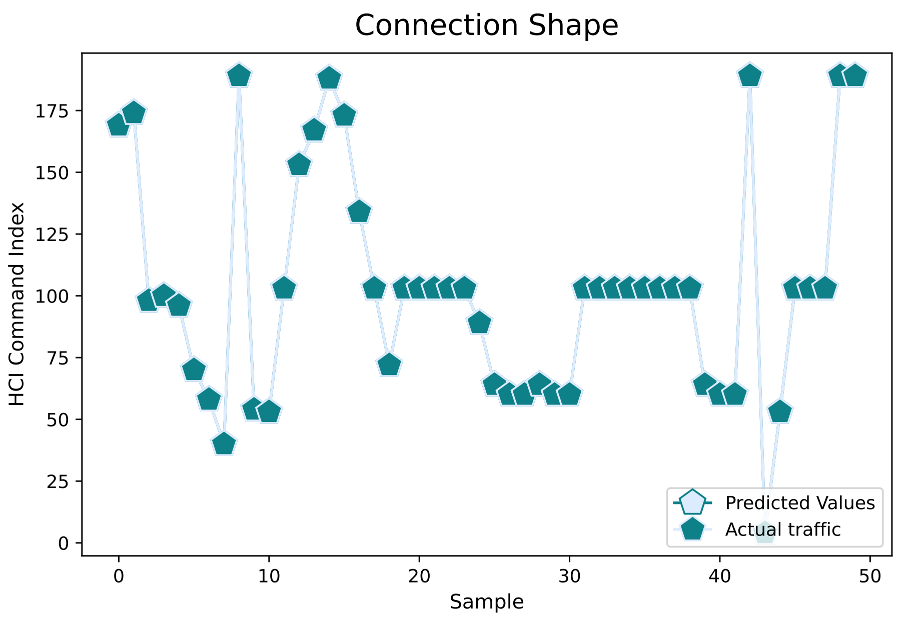
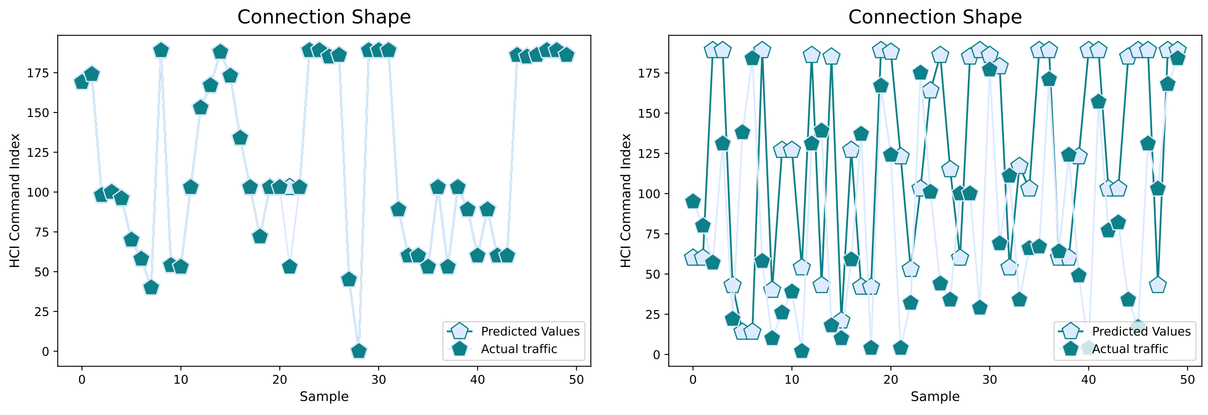


Figure 4.5: Model evaluation using training data.

These statistics were taken from the best-produced model during the training phase. It is objective to conclude from the table that if the model finds data sequences similar to the one used for its training, it will be able to recognize its patterns and predict its variation. To complete that initial analysis graphic 4.5 is also shown, in which it is possible to observe the variation of the index of the commands. Notice that the predicted index indicator is always overlaid by the true output.

4.3.2 Real Capture vs Random Data Confrontation

The first real challenge to the built model is a confrontation with data from different natures. Hence, this test case is composed of two data entries: one from an actual capture (very similar to the ones used for training) and a completely random vector.



((a)) Real capture (accuracy=99,38%). ((b)) Random data (accuracy=0,21%).

Figure 4.6: Model prediction for different situations.

As it can be concluded from the observation of figure 4.6, the model behavior went as expected. When taking Bluetooth traffic as an analysis object, it was perfectly capable of identifying it as such. It is possible to perceive that from the overlapping lines. In contrast, while reading a sequence of random values within the same range, it did not detect any usual pattern. Hence, the blue dots (corresponding to the index of the expected command) do not match the green ones along the sequence. Because of so, it is clear that predictions of future behavior have failed almost on every occasion.

4.3.3 N-gram Analysis

One decisive element that has been left for analysis during the testing phase is the size of the input window. This parameter is considered, from the beginning, as a determinant success factor of this experiment. The goal is to arrive at a value that delivers guarantees of a good prediction. The algorithm needs to know the context of the data under analysis to understand and predict the next step. The longer and more complete this context is, the more confidence the algorithm will have in the prediction it makes.

On the other hand, being tempted to extend this length to get better results can hinder the learning capacity of the system. Furthermore, accepting smaller tokens allows for shorter situations to be analyzed and decisions to be made in environments that contain little information. If we train the system to make classifications based on too much data, it can become difficult to do so when communications are short. That stated, look at the following table to notice how varying this size affects the model results in terms of its accuracy:

Table 4.4: Accuracy variation according to the given window.

N-gram	n = 3	n = 5	n = 10	n = 25	n = 50
Model Accuracy (%)	88,51	91,47	95,96	98,53	99,12

From the analysis of the values presented on table 4.4, it was clear that, for this investigation course, a window of size 25 would be the best appropriate. That judgment relies on the initial arguments, being that obtaining a prediction accuracy of 98,53% is a value that gives us the necessary guarantees to proceed. Besides, it is expected to fit any test condition, even if the captures have to be briefly extended.

4.3.4 Normal Traffic

Table 4.5: Normal traffic recognition rate.

Capture No.	**Device A**	**Device B**	**Length (packets)**	**Predictions Accuracy (%)**
1	MSI Modern 14	Xiaomi Mi 9 SE	75	100
2	Samgung Galaxy A22	Pioneer Car DEH-09BT	157	99,16
3	Xiaomi Redmi Note 8	Mi Smart Band 5	134	95,89
4	MSI Modern 14	Sony SRS-XB13	47	97,63
5	Xiaomi Mi 9 SE	Sony WH-CH500	175	99,28
6	MSI Modern 14	Xiaomi Redmi Note 10s	314	91,61
7	Xiaomi Redmi Note 10s	Redmi Watch 2	568	97,01
8	Xiaomi Mi 9 SE	Multiple devices	1 253	97,76
9	Xiaomi Redmi Note 10s	TV Smart Tech 32HA10V3	201	99,38
10	Xiaomi Redmi Note 10s	Novoteck BT-001	286	98,15
11	Samgung Galaxy A22	Huawei Y5P DRA-LX9	417	97,45
12	MSI Modern 14	Samsung A20e	121	96,02

After an initial test in which the model has been confronted with extreme realities, it is time to understand its consistency regarding ”normal traffic”. Addressing this, several captures were selected using a strategy involving devices very similar to global training. Table 4.5 makes a summary of the different communications presenting the results obtained for the predictions of each of them. Regardless of the capture size, it can be observed that the model gets more than 95% of the predictions right almost every time. In addition, those statistics are never below 90%. Such values mean that this traffic is very familiar, and its behavior corresponds to absolutely normal situations. Thus, the pattern recognition task can be considered as successful.

Although it is admitted for different operations and devices that the sequence spectrum may vary, when the predictions deviate too much from these values, the classification module must identify the traffic as abnormal. It is expected that in all failed predictions more than a third of the occasions may be associated with abnormal traffic. In other words, an accuracy below 67% may indicate an alert situation.

4.4 Proof of Concept Evaluation

Having proved an effective Bluetooth traffic recognition, from an artificial intelligence point of view, in already a valuable achievement. However, in the context of an intrusion detection system, its value is dependent on more specific tests. To validate the implemented approach we hereby simulate some attack environments to understand how our algorithm reacts and classifies those behaviors.

As described earlier in this section, the BrakTooth tool allowed us to simulate the different alert situations for testing purposes. As a result of its complexity and development time, this framework allows the launching of attacks of different natures. Most of them aim to cause DoS via firmware crashes or deadlocks.

In common, these attacks have the fact that they exploit several vulnerabilities in the LMP firmware. Since this protocol includes processes such as authentication, pairing, encryption, synchronization, or link supervision, it can be considered an absolutely complete range of tests. On the other hand, this choice meets what has been identified as the most critical moment in Bluetooth communication: the establishment of the connection. Through the execution of the different threats, the connection moment happens to be violated in the ways that the model has been trained to identify.

Although the tool allows tests from temporally incorrect responses to address manipulation or overflows, a selection of the most relevant threats was conducted. According to the results obtained, and in agreement with the theory already presented, three harmful attacks are fully documented in this test section. This evaluation is done in a comprehensive way, including traffic analysis and deduction of the respective anomalies.

The selection presented is based on very specific criteria established from the beginning. It is our intention to group similar test situations that can be summarized and represented by a particular attack situation. If this is not possible, many demonstrations will be made as necessary. In addition, it is crucial to ensure the inclusion of different test results. Only by comparing these results and searching for their explanations can the best conclusions be drawn about the model's efficiency. Ultimately, the test situations involved should together be able to represent as closely as possible the problem defined in the motivation of this dissertation.

4.4.1 BrakTooth Exploits Summary

As indicated in the introduction to the tool, *BrakTooth* has a full range of attacks. The deepening of the last three arises from the need to expose the different perspectives regarding the results obtained. However, it is relevant to refer to what was the complete outcome of the stipulated test cases. To cope with that, we next present a table with the interpretation of the different attack environments.

Table 4.6: Test results for the complete set of *BrakTooth* exploits.

Exploit designation	**HCI sensibility to the exploit**	**Traffic classification**
Invalid timing accuracy	Low	Abnormal
Repeated host connection	High	Abnormal
SDP unknown element type	Low	Abnormal
Knob	Moderated-high	Abnormal
Random authentication number flooding	High	Abnormal
LMP maximum slot overflow	Low	Abnormal
Duplicated encapsulated payload	No data	No classification
Feature response flooding	No data	No classification
LMP packets DM1 overflow	Low	Abnormal
Invalid feature page execution	Moderated	Abnormal
Feature response flooding	Low	Abnormal
Truncated SCO link request	Low	Abnormal
Wrong encapsulated payload	Moderated	Abnormal
Paging scan disable	No data	No classification
Invalid maximum slot	No data	No classification
Truncated LMP accepted	Low	Abnormal
SDP oversized element size	Moderated	Abnormal
Duplicated *IO* capability requested packets	High	Abnormal
LMP auto rate overflow	Low	Abnormal
Noncompliance invalid stop encryption	Moderated	Abnormal
Noncompliance duplicated encryption request	Moderated	Abnormal

Going further, we tried to infer which situations the model would be able to give us the certainty of a dangerous situation. In contrast, we also identified those in which it would be more reasonable to generate a simple alert. According to the sensitivity of the capture to identify these moments, patterns were established in order to group these situations and draw general conclusions about the developed tool.

In short, a ranking was established to characterize the content of the extracted packet sequence. In case these segments do not give us a good sense of what is going on at the progression level, their sensitivity is rated as elevated. Conversely, if we cannot deduce that progression, sensitivity is designated low. In extreme situations, there is no data for analysis at all. This has not to do with the capture method capability but with the nature of the attack itself. It should be understood that attacks that disable, for example, the paging procedure lead to no commands appearing. This absence is directly related to the disabling of such a feature. In between, we have cases where it is possible to observe the chaining of connection moments up to the point where they fail. Although this occurrence can derive from numerous indistinguishable factors, it is visible that an error has been obtained. The procedure is repeated, the traffic is considered anomalous, and the sensitivity is moderated.

When trying to analyze and cluster these results, we can derive the following aspects: attacks designed to attack very specific protocols, or involving the overflow of a certain element, usually do not give us much information about what is happening behind them. On the other hand, noncompliance exploits or well-referenced attacks like a knob, in addition to being detectable, can be partially dissected. On top of that, attacks that involve manipulating/adulterating a Bluetooth address are perfectly identifiable. Similarly, it is also possible to distinguish all code injection that leads to the duplication of a given action. Duplicated procedures, and consequent errors, are detectable since they are literally described by the associated command. In general, assuming the provided data is enough to constitute a capture sample, the model ends up identifying all these behaviors as anomalous.

4.4.2 LMP Random Authentication Number Flooding

In order to construct a more informed analysis of this exploit, it is convenient to start by analyzing its fingerprints. Table 4.7 shows an extract of the capture made at the time of its performance. Although shortened, one can see that the rest of the spectrum resembles the exposed command set. The first column corresponds to the ordered position of the extracted line in the global sequence before filtering. The second pane depicts the OpCode attributed by the HCI interface and, afterward, the matching description. The dominant statement is undeniable "Link Key Request Negative Reply". This command is issued in response to a link key request event from the controller to the host if there is no link key associated with the connection.

This occurs only for authentication after the pairing process has been completed successfully. Thus, when two devices are paired, mutual authentication takes place (assuming security mode 4). The procedure involves the controller asking the host for the link key previously stored. The HCI command with code 0x040C is used to inform about the absence of such a link key and resulting in a connection failure.

Table 4.7: HCI commands from *au_rand_flooding* capture.

No.	OpCode	Extended info	No.	OpCode	Extended info
1	0x200C	LE Set Extended Scan Enable	26	0x0401	Inquiry
5	0x040C	Link Key Request Negative Reply	28	0x200C	LE Set Extended Scan Enable
8	0x040C	Link Key Request Negative Reply	30	0x2005	LE Set Random Address
11	0x040C	Link Key Request Negative Reply	33	0x200B	LE Set Extended scan Parameters
14	0x040C	Link Key Request Negative Reply	36	0x040C	Link Key Request Negative Reply
17	0x040C	Link Key Request Negative Reply	37	0x040C	Link Key Request Negative Reply
20	0x2005	LE Set Random Address	39	0x040C	Link Key Request Negative Reply
22	0x200B	LE Set Extended scan Parameters	41	0x040C	Link Key Request Negative Reply
24	0x200C	LE Set Extended Scan Enable	44	0x040C	Link Key Request Negative Reply

After analyzing the data, one realizes that there are several failed connections in the communication between the host and the controller. Since that is outside the expected range, the model starts failing its predictions. This situation is clearly distinct from the one it was trained to learn. Hence, the accuracy obtained is around 29%. The way the system has been successively failing can be seen in figure 4.7.

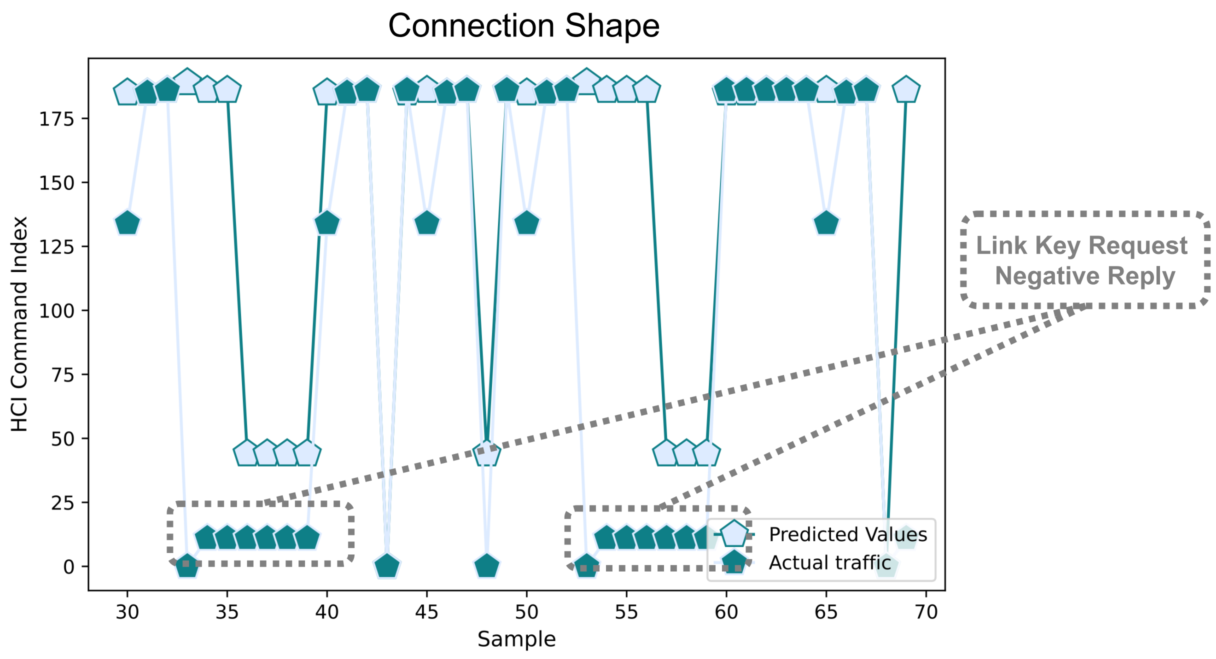


Figure 4.7: Connection shape while performing *au_rand_flooding* attack.

Furthermore, it is possible to visualize the repetition of this command by association with its index over the exposed samples. As the connection shape differs from the blue line, one can at least conclude that we are towards abnormal behavior. Therefore, this test has successfully corresponded to the expectations.

In an attempt to understand what has led to so, let's review the stages. The first thing to emphasize is the fact that our capture is at a level with enough sensitivity to discern the communication incoherences. Moreover, the consecutive translation of that thread has kept its features. Settle on that, the final classification is a trivial assignment.

4.4.3 Repeated Host Connection

As in the previous example, it is good practice to start by looking at the command sequence to understand how this attack causes a DoS. Table 4.8 shows that, for every *N* commands, there is an accepted connection request. While this repetition is suspicious by itself, it is made worse by being in duplicate. Actually, it is the fact that the Bluetooth address present in the connection request is repeated that leads to the disconnection and consequent repetition of the procedure. This is why we see a repeated cycle throughout the capture.

Table 4.8: HCI commands from *repeated_host_connnection* capture.

No.	OpCode	Extended info	No.	OpCode	Extended info
803	0x0034	IO Capability Req Negative Reply	878	0x041C	Read Remote Extended Features
812	0x0409	Accept Connection Request	881	0x0419	Remote Name Request
815	0x0409	Accept Connection Request	892	0x0034	IO Capability Req Negative Reply
818	0x041B	Read Remote Supported Features	898	0x0409	Accept Connection Request
823	0x041C	Read Remote Extended Features	901	0x0409	Accept Connection Request
826	0x0419	Remote Name Request	904	0x041B	Read Remote Supported Features
866	0x0409	Accept Connection Request	910	0x041C	Read Remote Extended Features
869	0x0409	Accept Connection Request	913	0x0419	Remote Name Request
872	0x041B	Read Remote Supported Features	945	0x0409	Accept Connection Request

Even though no error commands are posted, one can sense the failure in the connection establishment by the expressed pattern. The data presented above is translated graphically in figure 4.8. Again, from the command index it is possible to identify the "Accept Connection Request" and how the model fails to predict this double event. In this case, the percentage of correct predictions is set at 39.25%.

Considering the extract presented, one might think that this number is too high for a system that is clearly under DoS attack. However, this can be explained by the fact that between the connection attempts, commands such as "Read Remote Supported Features" or "Remote Name Request" make sense to the model and such predictions are correct in the context of this sequence. Ultimately, the traffic is still classified as anomalous by a wide margin, leaving no space for uncertainty.

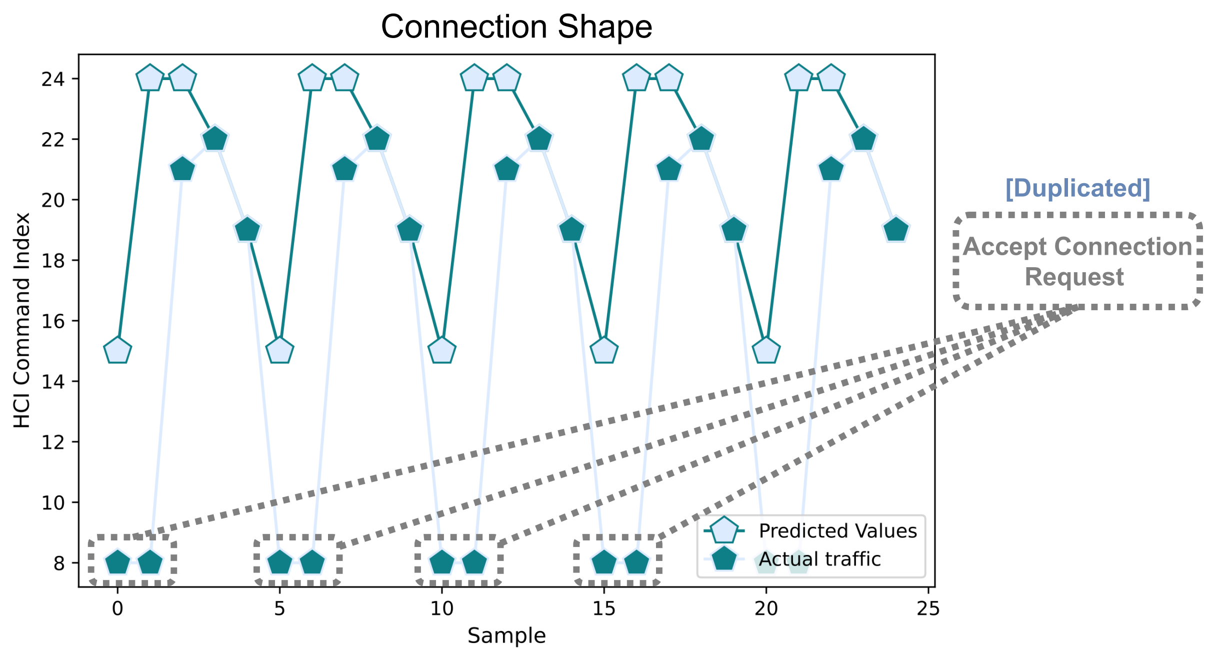


Figure 4.8: Connection shape while performing repeated host address attack.

4.4.4 Invalid Timing Accuracy

To extend our range of testing, let's see how the system reacts to malformed LMP accuracy responses followed by multiple reconnections. This is a very specific exploit designed to break the authentication process, whose appearance does not raise much suspicion. Its capture can be analyzed from the observation of the following table:

Table 4.9: HCI commands from *invalid_timing_accuracy* capture.

No.	**OpCode**	**Extended info**	**No.**	**OpCode**	**Extended info**
64	0x0401	Inquiry	90	0x0401	Inquiry
68	0x0402	Inquiry Cancel	92	0x200C	LE Set Extended Scan Enable
70	0x200C	LE Set Extended Scan Enable	96	0x200C	LE Set Extended Scan Enable
72	0x2005	LE Set Random Address	98	0x200C	LE Set Extended Scan Enable
74	0x200B	LE Set Extended scan Parameters	100	0x0402	Inquiry Cancel
76	0x200C	LE Set Extended Scan Enable	102	0x200C	LE Set Extended Scan Enable
78	0x0401	Inquiry	104	0x2005	LE Set Random Address
84	0x200C	LE Set Extended Scan Enable	106	0x200B	LE Set Extended scan Parameters
86	0x200C	LE Set Extended Scan Enable	109	0x0401	Inquiry

Thus, what can be concluded from the sequence of HCI packets is that the inquiry process to find a stable connection keeps on going. The cyclic repetition of these attempts can be camouflaged in the middle of the connection shape. However, by looking at a longer segment, we realize that the system is in a deadlock. This means that a device is not able to overcome these steps and complete the connection. Nevertheless, as it passes through our algorithm, it is a spectrum classified as abnormal. But is this conclusion really that reliable?

Even from the user's point of view, how many connections do we not see failing on the first attempt in our daily lives? The answer is quite a few. And what is visible to us as a failed attempt, can be translated into countless cycles that look similar to this attack. This means that, although the model's classification is correct, we cannot guarantee that the same will not happen in a normal situation where something went wrong at the user's fault.

A simple example of such is the pairing process between two devices that do not know each other. When trying to connect them for the first time, it is necessary to confirm at the opposite device. This interactivity with the user is subject to some failures, such as response time violations. While this would be easily rectified in a second approach, it is expected that our algorithm would generate an alert right from the beginning.

Therefore, model ratings should be interpreted as deviations from normality rather than absolute attacks. Circumstances of this nature were anticipated during the model capture and training process. We tried to include captures of unsuccessful first-time connections in normal traffic. However, in the multitude of data submitted, LSTM was able to filter out these circumstances and realize that they were not ideal. By injecting more traffic of this type into the model, it would be possible to reduce the number of false positives generated. On the other hand, as we do so, we take the risk that the model starts to ignore this type of situation, which is not intended. Accordingly, we should accept the fact that the model points out abnormal traffic as a possible attack environment, even if it is not sensitive enough to distinguish what is causing it.

In this sense, it is difficult to establish a threshold to classify the data. In the present test case, the data sequence looks considerably similar to normal behavior. Still, the model failed in about 60% of the predictions. Where should we place the threshold, then? It is known that anything that leads to forecast failures of more than 20% consists of a warning situation. Such a mark is supported by the results previously shown. Regardless of the situation, we witnessed a correctness rate always above 90%, which gives us great confidence in the judgments now made. However, assuming the 20% barrier is raised above, we can only state that we are facing anomalous communication. The concept of attack is, somehow, difficult to infer, especially in attacks like invalid timing accuracy.

Conclusion

This dissertation aim to study an intrusion detection system for automotive environment. One of its greatest motivations was to cope with the lack of security tools for Bluetooth networks when compared to homologous protocols such as Wi-fi. On the other hand, it is widely used and there are not too many papers on the subject, making it even more challenging. Conducting an investigation into the level of intrusion systems, we followed an approach that we consider to best suit the Bluetooth technology. Solving this problem required understanding the connection mechanisms employed by the communication protocol and the attack fronts to which it is exposed.

Combining the embedded system's world and the way the Bluetooth stack is designed to work in a given operating system with the dimension of artificial intelligence, forms quite a challenge. Nevertheless, that bridge has been established by forming an intrusion detection system or, as the results shout, a communication anomaly detection system. This section starts by enhancing the goals achieved throughout the work development, also seeking to take conclusions from them and opening suggestions for future implementations.

5.1 Discussion

For that purpose, we suggest an approach in reference to anomaly-based intrusion detection capable of adapting a machine-learning model to the problem statement. The solution consists of a neural network that detects non-compliant patterns based on the communication between the host and the controller. To cope with that, there were selected packets representing HCI commands from a connection, defining a regular working pattern. Even though we cannot claim that all the package content has been explored, the HCI commands represent the best characterization option, and its sequence has been trained by using an LSTM. Hence, while surveying several threats during the state-of-the-art, this work's evaluation metrics can only consider attacks detectable by its sequence of operations, which is translated into the collected data.

Concerning the proof of concept results itself, the outcomes are entirely satisfactory. From the artificial intelligence perspective, we achieved a working solution, where all the theory of an anomaly-based system was converted into a practical model with a tremendous self-learning capability. According to the dataset limitation, the model has succeeded in perceiving which patterns form normal behavior and which ones don't. For traffic similar to the sequences comprised in the training data, the results allow us to declare an outstanding identification of any anomalous behavior in a network data exchange. Despite dependencies such as the testing framework, this implementation has been proven to be a robust instance of how to approach problems of this nature, forming an accurate intrusion detection.

5.2 Future Work

Following up on the overall conclusions of this research it is an added value to point out the fields of expansion that will lead to a more significant advance for the scientific community. Based on the final classifier achieved, it would be interesting to adapt it at different levels.

First, review the data used to train the model. Further exploration of the information present in the transmitted packets may lead to the discovery of new attack indicators. As a primary suggestion, it is considered relevant to go deeper into the LMP protocol and all the data exchange associated with this level. As a matter of fact, most of the exploits that can be replicated are based on the adulteration of this behavior.

Secondly, one could follow an approach regenerated in the literature. It is certain that there are several signature-based IDS. Combining the guarantees that these systems give us by comparison with known attack signatures with all types of traffic analysis would strengthen this approach. Although it is not assured that building a hybrid system would extend the coverage of the danger area, it would in a fact increase the confidence level on particular outputs.

On the other hand, it would be interesting, from an uppermost perspective, to associate the traffic classifier with other awareness mechanisms. In this sense, we believe that interactivity with the user would be something indispensable to the validation of certain suspicions. As observed in the results chapter, there are moments in which the developed IDS lacks some sensitivity to guarantee the occurrence of a specific fraud. Since signalization is still done, the generation of a warning and confirmation by the user would help clarify the threat.

This is not a one size fits all solution. However, it tries to identify any behavior that is in potential violation of the system's normal operation. Taking this proof of concept results as evidence, it is possible to adapt them to more specific problems. This work proposal translates, for example, into the exploration of the Bluetooth stack and monitoring the communication at the level of a given protocol or an underlying protection mechanism.

Bibliography

[1] Karl Koscher et al. “Experimental Security Analysis of a Modern Automobile”. In: Jan. 2010, pp. 447–462. DOI: `10.1109/SP.2010.34`.

[2] Pratik Satam, Shalaka Satam, and Salim Hariri. “Bluetooth Intrusion Detection System (BIDS)”. In: *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*. 2018, pp. 1–7. DOI: `10.1109/AICCSA.2018.8612809`.

[3] Stephen Checkoway et al. “Comprehensive Experimental Analyses of Automotive Attack Surfaces”. In: *Proceedings of the 20th USENIX Conference on Security*. SEC’11. San Francisco, CA: USENIX Association, 2011, p. 6.

[4] Sandro Rodriguez Garzon. “Intelligent In-Car-Infotainment Systems: A Contextual Personalized Approach”. In: *Proceedings of the 2012 Eighth International Conference on Intelligent Environments*. IE ’12. USA: IEEE Computer Society, 2012, pp. 315–318. ISBN: 9780769547411. DOI: `10.1109/IE.2012.70`. URL: `https://doi.org/10.1109/IE.2012.70`.

[5] Computest. “The Connected Car. Ways to get unauthorized access and potential implications”. In: (2018).

[6] Edwin Franco Myloth Josephlal and Sridhar Adepu. “Vulnerability Analysis of an Automotive Infotainment System’s WIFI Capability”. In: *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*. 2019, pp. 241–246. DOI: `10.1109/HASE.2019.00044`.

[7] Samuel Woo, Hyo Jin Jo, and Dong Hoon Lee. “A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN”. In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2015), pp. 993–1006. DOI: `10.1109/TITS.2014.2351612`.

[8] Saurabh Kale Kunal Karnik Manandeep and Ajinkya Medhekar. “On Vehicular Security for RKE and Cryptographic Algorithms: A Survey”. In: vol. 09. 2020. DOI: `10.17577/IJERTV9IS050693`.

[9] *Bluetooth Core Specification*. Version 5.1. Jan. 2019. URL: `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`.

[10] “How can Bluetooth services and devices be effectively secured?” In: *Computer Fraud Security* 2006.1 (2006), pp. 4–7. ISSN: 1361-3723. DOI: `https://doi.org/10.1016/S1361-3723(06)70294-9`. URL: `https://www.sciencedirect.com/science/article/pii/S1361372306702949`.

[11] Dai Davis. “Bluetooth”. In: *Network Security* 2002.4 (2002), pp. 11–12. ISSN: 1353-4858. DOI: `https://doi.org/10.1016/S1353-4858(02)00413-0`. URL: `https://www.sciencedirect.com/science/article/pii/S1353485802004130`.

[12] Edward Au. “Bluetooth 5.0 and Beyond [Standards]”. In: *IEEE Vehicular Technology Magazine* 14.2 (2019), pp. 119–120. DOI: `10.1109/MVT.2019.2905520`.

[13] K. Matheus and S. Magnusson. “Bluetooth radio network performance: measurement results and simulation models”. In: *International Workshop on Wireless Ad-Hoc Networks, 2004.* 2004, pp. 228–232. DOI: 10.1109/IWWAN.2004.1525576.

[14] Yang Hua and Yuexian Zou. “Analysis of the packet transferring in L2CAP layer of Bluetooth v2.x+EDR”. In: *2008 International Conference on Information and Automation*. 2008, pp. 753–758. DOI: 10.1109/ICINFA.2008.4608099.

[15] Xue Zhangcheng and Wu Shunxiang. “File Transferring via BlueTooth Based on the Obex Protocol”. In: *2009 International Conference on Wireless Networks and Information Systems*. 2009, pp. 77–80. DOI: 10.1109/WNIS.2009.56.

[16] “Chapter 4 - Security Management”. In: *Bluetooth Application Developer’s Guide*. Ed. by David Kammer et al. Burlington: Syngress, 2002, pp. 125–166. ISBN: 978-1-928994-42-8. DOI: https://doi.org/10.1016/B978-192899442-8/50007-0. URL: https://www.sciencedirect.com/science/article/pii/B9781928994428500070.

[17] Pushpa R. Suri and Sona Rani. “Bluetooth security - Need to increase the efficiency in pairing”. In: *IEEE SoutheastCon 2008*. 2008, pp. 607–609. DOI: 10.1109/SECON.2008.4494365.

[18] Asif Ikbal Mondal and Bijoy Kumar Mandal. “Architecture of Bluetooth Security for Pairing Key and Better Authentication”. In: *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*. 2021, pp. 1–6. DOI: 10.1109/ISCON52037.2021.9702335.

[19] Jian Wang and Nan Jiang. “Secure authentication and authorization scheme for mobile devices”. In: *2009 IEEE International Conference on Communications Technology and Applications*. 2009, pp. 207–211. DOI: 10.1109/ICCOMTA.2009.5349208.

[20] Robert Morrow. McGraw-Hill, 2002. ISBN: 9780071409636.

[21] John Padgette et al. *Guide to Bluetooth Security*. en. 2022-01-19 05:01:00 2022. DOI: https://doi.org/10.6028/NIST.SP.800-121r2-upd1. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=934038.

[22] Ashley Podhradsky, Cindy Casey, and Peter Ceretti. “The Bluetooth honeypot project”. In: vol. 4. Apr. 2012, pp. 1–10. ISBN: 978-1-4577-0579-3. DOI: 10.1109/WTS.2012.6266078.

[23] Richard Kissel. *Glossary of Key Information Security Terms*. en. 2006-04-25 2006. DOI: https://doi.org/10.6028/NIST.IR.7298.

[24] Turuk Panigrahy Jena. “Security in Bluetooth, RFID and Wireless Sensor Networks”. In: *International Conference on Communication, Computing and Security* (2011), pp. 628–633.

[25] Sunny Shrestha et al. "SoK: A Systematic Literature Review of Bluetooth Security Threats and Mitigation Measures". In: ACM. DOI: 10.2139/ssrn.3959316. URL: https://ssrn.com/abstract=3959316.

[26] Redjem Bouhenguel, Imad Mahgoub, and Mohammad Ilyas. "Bluetooth Security in Wearable Computing Applications". In: *2008 International Symposium on High Capacity Optical Networks and Enabling Technologies*. 2008, pp. 182–186. DOI: 10.1109/HONET.2008.4810232.

[27] Nishitkumar Patel, Hayden Wimmer, and Carl M. Rebman. "Investigating Bluetooth Vulnerabilities to Defend from Attacks". In: *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. 2021, pp. 549–554. DOI: 10.1109/ISMSIT52890.2021.9604655.

[28] Marwan Albahar. "BLUETOOTH PAIRING SECURITY THREATS AND COUNTERMEASURES". PhD thesis. Nov. 2017.

[29] Keijo Haataja. *Security Threats and Countermeasures in Bluetooth-Enabled Systems: Practical Experiments, Vulnerability Evaluation, and Solutions*. LAP Lambert Academic Publishing, 2009. ISBN: 978-3838313443.

[30] Yaniv Shaked and Avishai Wool. "Cracking the Bluetooth PIN". In: *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*. MobiSys '05. Seattle, Washington: Association for Computing Machinery, 2005, pp. 39–50. ISBN: 1931971315. DOI: 10.1145/1067170.1067176. URL: https://doi.org/10.1145/1067170.1067176.

[31] Nateq Be-Nazir, Nateq Ibn Minar, and Mohammed Tarique. "Bluetooth Security Threats And Solutions: A Survey". In: *International Journal of Distributed and Parallel Systems* 3 (Feb. 2012). DOI: 10.5121/ijdps.2012.3110.

[32] Goutam Chakraborty et al. "Analysis of the Bluetooth device discovery protocol". In: *Wireless Networks* 16 (Feb. 2010), pp. 421–436. DOI: 10.1007/s11276-008-0142-1.

[33] Dennis Kügler. ""Man in the Middle" Attacks on Bluetooth". In: *Financial Cryptography*. Ed. by Rebecca N. Wright. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 149–161. ISBN: 978-3-540-45126-6.

[34] Shaker Alanazi et al. "On Resilience of Wireless Mesh Routing Protocol against DoS Attacks in IoT-based Ambient Assisted Living Applications". In: Oct. 2015. DOI: 10.1109/HealthCom.2015.7454499.

[35] Yicai Huang, Pengcheng Hong, and Bin Yu. "Design of Bluetooth DOS Attacks Detection and Defense Mechanism". In: *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*. 2018, pp. 1382–1387. DOI: 10.1109/CompComm.2018.8780851.

[36] Bluetooth SIG. *Bluetooth Core Specification v5.2*. Sept. 2022. URL: https://www.bluetooth.com/specifications/specs/core-specification-5-3/.

[37] URL: http://www.bluez.org/.

[38] Peng Zhou and Xiang Ling. "HCI-based bluetooth master-slave monitoring system design". In: *International Conference on Computational Problem-Solving*. 2010, pp. 406–409.

[39] Keijo Haataja. "New Efficient Intrusion Detection and Prevention System for Bluetooth Networks". In: Jan. 2008, p. 16. DOI: 10.1145/1361492.1361512.

[40] Rashmi Ravindra Chaudhari and Sonal Patil. "Intrusion Detection System. Classification, Techniques and Datasets to Implement". In: vol. 04. 2017.

[41] Usman Shuaibu Musa et al. "Intrusion Detection System using Machine Learning Techniques: A Review". In: *2020 International Conference on Smart Electronics and Communication (ICOSEC)*. 2020, pp. 149–155. DOI: 10.1109/ICOSEC49089.2020.9215333.

[42] Hung-Jen Liao et al. "Intrusion detection system: A comprehensive review". In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24. ISSN: 1084-8045. DOI: https : / / doi .org / 10 .1016 / j .jnca .2012 .09 .004. URL: https://www.sciencedirect.com/science/article/pii/S1084804512001944.

[43] Peter Stavroulakis and Mark Stamp. *Handbook of Information and Communication Security*. Jan. 2010. ISBN: 978-3-642-04116-7. DOI: 10.1007/978-3-642-04117-4.

[44] Terrence OConnor and Douglas Reeves. "Bluetooth Network-Based Misuse Detection". In: *2008 Annual Computer Security Applications Conference (ACSAC)*. 2008, pp. 377–391. DOI: 10.1109/ACSAC.2008.39.

[45] Tahir Mehmood and Helmi Rais. "Machine learning algorithms in context of intrusion detection". In: Aug. 2016, pp. 369–373. DOI: 10.1109/ICCOINS.2016.7783243.

[46] T. Saranya et al. "Performance Analysis of Machine Learning Algorithms in Intrusion Detection System: A Review". In: *Procedia Computer Science* 171 (Jan. 2020), pp. 1251–1260. DOI: 10 .1016/j .procs .2020.04.133.

[47] Hind Tribak et al. "Statistical analysis of different artificial intelligent techniques applied to Intrusion Detection System". In: *2012 International Conference on Multimedia Computing and Systems*. 2012, pp. 434–440. DOI: 10.1109/ICMCS.2012.6320275.

[48] Anish Halimaa A. and K. Sundarakantham. "Machine Learning Based Intrusion Detection System". In: *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. 2019, pp. 916–920. DOI: 10.1109/ICOEI.2019.8862784.

[49] Sara A. Althubiti, Eric Marcell Jones, and Kaushik Roy. "LSTM for Anomaly-Based Network Intrusion Detection". In: *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*. 2018, pp. 1–3. DOI: 10.1109/ATNAC.2018.8615300.

[50] Kazuki Hara and Kohei Shiomoto. “Intrusion Detection System using Semi-Supervised Learning with Adversarial Auto-encoder”. In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. 2020, pp. 1–8. DOI: 10.1109/NOMS47738.2020.9110343.

[51] Shalaka Satam, Pratik Satam, and Salim Hariri. “Multi-level Bluetooth Intrusion Detection System”. In: *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*. 2020, pp. 1–8. DOI: 10.1109/AICCSA50499.2020.9316514.

[52] Shalaka Satam, Pratik Satam, and Salim Hariri. “Multi-level Bluetooth Intrusion Detection System”. In: *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*. 2020, pp. 1–8. DOI: 10.1109/AICCSA50499.2020.9316514.

[53] Sepp Hochreiter and Jurgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* (May 2016).

[54] Pavlo Radiuk. “Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets”. In: *Information Technology and Management Science* 20 (Dec. 2017), pp. 20–24. DOI: 10.1515/itms-2017-0003.

[55] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence. Springer Berlin, Heidelberg, Jan. 2012. ISBN: 978-3-642-24797-2. DOI: 10.1007/978-3-642-24797-2.

[56] Google Researchers. *TensorFlow Guide*. Aug. 2022. URL: https://www.tensorflow.org/guide.

[57] Matheus Garbelini et al. *BRAKTOOTH: Causing Havoc on Bluetooth Link Manager*. Aug. 2022. URL: https://asset-group.github.io/disclosures/braktooth/.

[58] Espressif Systems. *Official ESP32 Development Kit ESP32-WROVER-KIT*. Aug. 2022. URL: https://www.espressif.com/en/products/hardware/esp-wrover-kit/overview.