

École polytechnique de Louvain

Unveiling the Secrets of IoT Forensics:

Exploring Frameworks, Procedures, and
a Practical GUI Implementation for IoT
Identification

Authors: **Merlin CAMBERLIN, Igor NUNES FERRO**

Supervisor: **Ramin SADRE**

Readers: **Cíntia BORGES MARGI, François DE KEERSMAEKER,
Ramin SADRE**

Academic year 2022–2023

Master [120] in Computer Science and Engineering

UCLouvain
École polytechnique de Louvain
ICTEAM department

Master in Computer Science and Engineering

**Unveiling the Secrets of IoT Forensics: Exploring Frameworks, Procedures,
and a Practical GUI Implementation for IoT Identification**

by Merlin CAMBERLIN and
Igor NUNES FERRO

Abstract

Nowadays, most police departments have computer forensic investigators who examine and analyze computer equipment for evidence recovery. This happens routinely for typical computer equipment, such as laptops, mobile phones, and tablets, as part of criminal investigations. However, procedures and best practices are much less clear for modern IoT devices. The IoT raises several new challenges and prevents investigators from using traditional forensic methods. In this master thesis, we explored the different existing frameworks, procedures, tools and studied their applicability in an investigation context.

We identified a gap in the identification of IoT devices at a crime scene. Most of the research assumes the availability of IoT devices. Yet the identification of IoT devices at a crime scene has become challenging. IoT devices can take on a wide range of forms and be integrated into everyday objects. Because of this, it may be difficult for police officers who are unfamiliar with the Internet of Things to identify IoT devices as such. To address this problem, we proposed a GUI that displays the active IoT devices in the vicinity by sniffing the WiFi, BLE, and ZigBee protocols.

Acknowledgements

We would like to express our deepest gratitude to Pr. Ramin Sadre for his unwavering support, valuable guidance, and dedication throughout the entire duration of the master's thesis. His expertise, quick feedback, and regular meetings were instrumental in shaping our research.

We are also grateful to the police investigators who generously shared their time and knowledge with us. Our discussions and insights into the challenges faced by law enforcement professionals when dealing with crime scenes involving IoT devices were invaluable. Their real-world perspectives greatly enriched our understanding and motivated us to pursue innovative solutions in the field of IoT forensic investigations.

We extend our sincere appreciation to Cíntia Borges Margi and François De Keersmaecker who will join Pr. Ramin Sadre to assess our work.

Lastly, we would like to acknowledge our friends, family and love ones for their support and encouragement. Your unwavering belief in our abilities and continuous support along the way have been the driving force behind our accomplishments.

Contents

Abstract	i
Acknowledgments	ii
List of Abbreviations	vi
List of Figures	vii
1 Introduction	1
1.1 Context	1
1.2 Master Thesis Objectives	2
1.3 Structure	3
1.4 Contribution	3
2 General Background	4
2.1 Introduction	4
2.2 Internet of Things	4
2.2.1 IoT Definition	4
2.2.2 IoT Devices	5
2.2.3 IoT Architecture	6
2.2.4 IoT Communication Protocols	6
2.3 Forensics	8
2.3.1 Forensic Definition	8
2.3.2 Digital Forensics	9
2.3.3 IoT Forensics	9
2.3.4 IoT Forensics vs Digital Forensics	9
2.4 Conclusion	11
I State-of-the-Art in IoT Forensics	12
3 Literature Review	13
3.1 Introduction	13
3.2 IoT Forensics Frameworks	13
3.2.1 Standards in IoT Forensics Frameworks	14
3.2.2 Overview of Existing IoT Forensics Frameworks	16
3.2.3 Smart Home Forensic Frameworks	18
3.3 IoT Forensic Procedures	19
3.4 Conclusion	22

4	The Underlying Problem	23
4.1	Introduction	23
4.2	Legal Challenges	23
4.3	Technical Challenges	25
4.4	Digital Forensics Tools	26
4.4.1	Device Localization	26
4.4.2	Data Extraction	27
4.4.3	Data Analysis	28
4.4.4	Evidence Preservation	29
4.4.5	Tools in IoT Forensics	30
4.5	Advancements Needed	31
4.6	Conclusion	32
II	IoT Device Identification GUI	33
5	GUI Design and Implementation	34
5.1	Introduction	34
5.2	Design Choices	34
5.2.1	Protocols	34
5.2.2	Hardware	36
5.2.3	Programming Language	38
5.2.4	GUI Library	38
5.3	GUI Description	39
5.3.1	GUI Usage Instructions	40
5.3.2	Project Organization	41
5.4	WiFi Device Identification	43
5.4.1	WiFi Networks Discovery	43
5.4.2	WiFi Frames Sniffing	44
5.4.3	IEEE 802.11 Frames	45
5.4.4	Information of Interest to Collect	48
5.4.5	Firmware	49
5.4.6	Integration in the GUI	50
5.5	Bluetooth Low Energy Devices Identification	51
5.5.1	BLE Devices Discovery	51
5.5.2	Information of Interest to Collect	51
5.5.3	Firmware	52
5.5.4	Integration in the GUI	52
5.6	ZigBee Devices Identification	52
5.6.1	IEEE 802.15.4 Network Discovery	53
5.6.2	IEEE 802.15.4 Frames Sniffing	54
5.6.3	IEEE 802.15.4 Frames	56
5.6.4	ZigBee Frames	57
5.6.5	Information of Interest to Collect and Examples	59

5.6.6	Firmware	61
5.6.7	Integration in the GUI	61
5.7	Conclusion	63
6	Experiments and Results	64
6.1	Introduction	64
6.2	Testing Scenario Setup	64
6.2.1	Selection of Devices	64
6.2.2	Placement	66
6.2.3	Configuration	68
6.3	Testing	68
6.4	Results	71
6.5	Limitations	73
6.6	Conclusion	74
7	Conclusion	76
7.1	Key Findings and Contribution	76
7.2	Limitations and Future Work	77
7.3	Final Comments	77
A	GUI setup	79
A.1	Configuration of Serial Ports	79
A.1.1	Finding the Serial Port on Linux	79
A.1.2	Finding the Serial Port on Windows	80
A.2	Timing for ZigBee Sniffer	81
A.2.1	Changing the Timing for Network Discovery	81
A.2.2	Changing the Timing for Sniffing	82
B	Heltec WiFi LoRa V2 WiFi Sniffer Firmware	84
C	Programming the nRF52840 Dongle	88
D	Programming the nRF BLE Sniffer Firmware	89
E	Programming the nRF IEEE 802.15.4 Sniffer Firmware	91
F	Performed Tests	92
F.1	Initializing the Sniffer	92
F.2	Finding WiFi Router	93
F.3	Finding Camera	93
F.4	Seizing the Camera	94
F.5	Finding the Smart Socket	94
F.6	Seizing the Smart Socket	95
F.7	Finding the BLE Headphones	95
F.8	Seizing the BLE Headphones	96

F.9	Selecting the ZigBee Network	96
F.10	Finding Hub and Motion Sensor	97
F.11	Seizing the Motion Sensor	97
F.12	Finding the Multipurpose Sensor	98
F.13	Seizing the Multipurpose Sensor	98
F.14	Finding the Presence Sensor	99
F.15	Seized Presence Sensor and back to Hub	99
F.16	Seizing the Hub	100
F.17	Seizing the Router	100
Bibliography		102

List of Abbreviations

ACK	A C K nowledgment
aDSL	asymmetric D igital S ubscriber L ine
AP	A ccess P oint
BLE	B luetooth L ow E nergy
BSSID	B asic S ervice S et I Dentifier
DF	D igital F orensics
DFI	D igital F orensics I nterface
EM	E lectro m agnetic
EM-SCA	E lectro M agnetic S ide- C hannel A nalysis
ENFSI	E uropean N etwork of F orensic S cience I nstitute
GUI	G raphical U ser I nterface
IBSS	I ndependent B asic S ervice S et
IDE	I ntegrated D evelopment E nvironment
IEEE	I nstitute of E lectrical and E lectronic E ngineers
IP	I nternet P rotocol
IoT	I nternet o f T hings
ISO	I nternational O rganization for S tandardization
ITU	I nternational T elecommunication U ion
MAC	M edium A ccess C ontrol
NIST	N ational I nstitute of S tandards and T echnology
PAN	P ersonal A rea N etwork
RFID	R adio F requency I dentification
RSSI	R eceived S ignal S trength I ndication
SSID	S ervice S et I Dentifier
STA	S T A tion

List of Figures

2.1	Sources of Evidences in Internet of Things Systems [13]	10
3.1	Seven-Phases Smart Home Investigation Framework	19
5.1	<i>Heltec WiFi LoRa V2 - Heltec Automation</i>	37
5.2	<i>nRF52840 Dongle - Nordic Semiconductors</i>	38
5.3	GUI Organization	39
5.4	GUI Displaying Detected Active Devices	41
5.5	Overview of File and Folder Structure in the Project	42
5.6	IEEE 802.11 Management Frame Format [35]	46
5.7	IEEE 802.11 RTS Control Frame Format [35]	47
5.8	IEEE 802.11 Data Frame Format [35]	47
5.9	IEEE 802.11 CTS Control Frame Format	48
5.10	WiFi Sniffing Firmware Data Structures	50
5.11	Terminal with Serial Connection with the Device and Data Acquisition	55
5.12	IEEE 802.15.4 MAC Header Frame Format [39]	56
5.13	ZigBee Frames Stack [41]	57
5.14	ZigBee Network Frame Format [42]	58
5.15	Frame Control Format [42]	58
5.16	Auxiliary Frame Header Format [42]	58
5.17	Security Control format [42]	59
5.18	Parsing of the First ZigBee Frame Type	60
5.19	Parsing of the First ZigBee Frame Type	60
5.20	Parsing of the Third ZigBee Frame Type	61
6.1	Floor Plan of Testing Environment with Position of Devices	67
6.2	Detected WiFi Networks with the Testing One Selected and Highlighted	71
A.1	Serial Ports on Linux.	80
A.2	Changing the code to adapt the serial port in Linux.	80
A.3	Serial Ports on Windows.	81
A.4	Changing the Code to Adapt the Serial port in Windows.	81
A.5	Line that should be changed to alter the Network Discovery time in each channel.	82
A.6	Line that should be changed to alter the Sniffing time in each channel.	83

Chapter 1

Introduction

The chapter 1 of this master thesis is an overview of what to expect from the rest of this work. It is divided into four sections that set the context, present the objectives, describe the structure of the report, and highlight the contribution of this work. The section 1.1 establishes the context of IoT forensics: the rise of the Internet of Things, the interest in investigations, the current usage of IoT for investigation, and the new challenges they pose. The section 1.2 outlines the goals and scope of this master thesis. The structure section describes how this report is organized, highlighting the main topics covered in each chapter. Finally, the last section outlines the contribution of this thesis, describing how the research adds value to the field of IoT computer forensics.

1.1 Context

The Internet of Things (IoT) is a promising combination of technologies that has the potential to revolutionize many sectors by automating processes and enabling new services and applications. Various studies predict the number of IoT devices is expected to increase significantly in the coming years.

The rise of the Internet of Things has led to an explosion in the number of connected devices in our environments. In particular, smart homes are becoming increasingly popular and equipped with IoT devices, ranging from smart locks and security cameras to smart thermostats and home assistants. According to a report by *Statista*, the global number of households in the smart home market is expected to continuously increase between 2023 and 2027 to reach 672.57 million households by 2027 worldwide, up from 360.67 million in 2023.^[1]

This growing adoption of IoT devices has made them a common presence in our environments constituting new sources of evidence for judicial investigations. IoT devices are closely integrated into users' daily lives and constantly interact with and scan the physical world. They act as silent witnesses, quietly recording and storing information at a crime scene. This provides a wealth of information for criminal investigations. They can potentially be used by police officers in a number of ways to assist with investigations

at a crime scene. The data collected can be used to establish timelines, identify suspects or shed light on what happened before, during, and after an event that transpired. This is particularly true in cases involving smart homes. For instance, smart locks and security cameras can record the time and date of entries and exits to a home. Smart home assistants, such as *Amazon Alexa* or *Google Assistant*, can store voice recordings and transcripts of conversations. This information represents a source of information that police investigators cannot overlook anymore as part of their investigation.

Nowadays, most police departments have computer forensic investigators who examine and analyze computer equipment for evidence recovery. In the context of criminal investigations, this happens routinely for standard computer equipment, such as laptops, mobile phones, and tablets. Procedures and practices for evidence recovery from traditional computer equipment have been established and refined over several decades. In contrast, procedures and best practices are much less clear for modern IoT devices. The field of IoT device forensics is still in its early stages, and there is much less clarity about how to conduct examinations of IoT devices for evidence recovery.

Police officers also face new challenges when using IoT devices for evidence recovery. Among them, there is a diversity of IoT devices in terms of hardware, software, and network connectivity. IoT devices may also not be immediately recognizable as such, as they can take a variety of different forms and may be integrated into everyday objects. Chapter 4 will go into more detail about other challenges brought by IoT devices.

In this context in which police officers are working and highlighting the growing importance of IoT devices in crime scenes, there is an increasing need for procedures, techniques, and specialized tools to help them effectively investigate crime scenes that involve IoT devices.

1.2 Master Thesis Objectives

The objectives of this master thesis are twofold. On one hand, this master thesis contributes to the field of IoT computer forensics by conducting a study of the current existing frameworks, procedures, and research for conducting investigations in environments that evolved IoT devices such as smart homes. It also highlights the various challenges encountered by investigators when dealing with IoT devices.

On the other hand, this master thesis aims to provide a Graphical User Interface (GUI) tool that can assist investigators in the process of locating IoT devices at a crime scene. The GUI will be designed to be user-friendly and intuitive and will be developed using the latest software engineering methodologies and tools.

Overall, this master thesis tends to answer the following questions: What are the existing frameworks and procedures used for conducting digital investigations in IoT-based crime scenes? What are the existing tools to support IoT forensics? How to help police officers identify and locate connected objects at crime scenes?

1.3 Structure

This master thesis is divided into two parts. Prior to the first part, chapter 2 sets the general background information required for the understanding of the next chapters. Next, the first part provides the state-of-the-art of existing research on IoT forensics. This part also outlines the underlying problem associated with the use of IoT devices in digital investigations. The second part focuses on the practical implementation of the GUI developed to identify and locate IoT devices. This part includes detailed information and design, implementation details, as well as the results and evaluation of the GUI's performance.

1.4 Contribution

One contribution of this master thesis is to address the current lack of clarity and best practices in the field of IoT device forensics by studying the existing literature on the topic. By reviewing the literature on IoT forensics, this study aims to provide a comprehensive overview of the current state of the field and identify gaps and opportunities for further research.

One second contribution to this master thesis is the implementation of a user-friendly GUI tool intended to help investigators easily visualize detected IoT devices, particularly those found in smart homes. By making the process of IoT device identification more accessible and efficient, this tool has the potential to improve the quality and speed of criminal investigations involving IoT devices.

Chapter 2

General Background

2.1 Introduction

In this second chapter, we establish the necessary background to comprehensively understand the subsequent chapters of this work. The intricate nature of IoT introduces new challenges in the field of forensic investigation. As IoT devices become increasingly prevalent, their potential value as sources of digital evidence in crime scene investigations cannot be overlooked. It is crucial to explore the intersection of IoT and forensics to understand the unique characteristics of IoT forensics.

To achieve this objective, the chapter is structured as follows. The section [2.2](#) provides an overview of the Internet of Things, including its definition, the types of devices involved, the underlying architecture, and the communication protocols commonly employed. This foundation will enable us to comprehend the intricate nature of IoT environments and the potential complexities encountered during forensic investigations.

The section [2.3](#) delves into the domain of forensics. It begins by defining forensic science and then focuses specifically on digital forensics. Subsequently, we explore the emergence of IoT computer forensics as a specialized branch within digital forensics, addressing the unique challenges and considerations associated with IoT devices. A comparative analysis between IoT forensics and traditional digital forensics will be presented to highlight the distinctive characteristics and requirements of each. Furthermore, we investigate the realm of smart home forensics, recognizing the prevalence of IoT devices within residential environments.

2.2 Internet of Things

2.2.1 IoT Definition

The concept of the Internet of Things has been around for over 15 years and refers to the interconnectivity of physical objects through the use of advanced information and communication technologies. The term was originally coined in relation to the work of the

Auto-ID Labs at *MIT* on networked radio-frequency identification (RFID) infrastructures but has since evolved to encompass a wider range of technologies and applications. [2]

The *International Telecommunication Union* (ITU) defines it as:

“ A global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies.” [3]

The official definition of IoT formulated by the *International Organization for Standardization* (ISO) and the *International Electrotechnical Commission* (IEC) takes up a vision centered on value creation through services by generalizing the notion of infrastructure.

“ An infrastructure of interconnected objects, people, systems, and information resources together with intelligent services to allow them to process information of the physical and the virtual world and react.” [4]

Despite the widespread use of the term, there is still no unique globally accepted definition of IoT. There are several definitions that are currently used by different groups. Those definitions do not disagree: they highlight different aspects of what the IoT term covers. Some definitions mention the most important attributes of the IoT phenomenon (connected objects, Internet-related aspects), others describe what the term refers to and others focus on use cases in which the IoT devices are used. [5]

In this paper, the interpretation of the Internet of Things taken is the following: The Internet of Things refers to the interconnected network of physical objects that are embedded with sensors, processing ability, software, and network connectivity, allowing them to collect and exchange data for management, data mining, and access to the data they generate.

2.2.2 IoT Devices

IoT devices are used in a wide variety of contexts, including smart homes, wearables, and industrial settings. Smart homes incorporate IoT devices such as smart thermostats, lighting systems, security cameras, and voice assistants to enhance convenience, energy efficiency, and security. Wearables can monitor health and fitness metrics and enable connectivity with other devices, such as smartphones. Industrial IoT devices are employed in manufacturing plants, transportation systems, and agriculture to improve the efficiency of the operation and collect data to make decisions in real time.

IoT devices exhibit several key characteristics that distinguish them from traditional computing devices. Firstly, IoT devices are often designed with limited processing power, as their primary function is to gather and transmit data rather than perform complex computations. This limitation is imposed to optimize their energy consumption and ensure efficient operation within resource-constrained environments. Additionally, IoT devices typically have limited amounts of memory, restricting their ability to run memory-intensive applications. Furthermore, due to their embedded nature and the need for compact form

factors, IoT devices are equipped with limited battery capacities, emphasizing the importance of energy-efficient algorithms and power management strategies. Finally, IoT devices are characterized by their embedded nature, integrating into various objects and environments, making them ubiquitous and often inconspicuous in their presence.

2.2.3 IoT Architecture

In the rapidly expanding world of the Internet of Things (IoT), various actors play distinct roles in the functioning of IoT systems. These actors, including devices, gateways, cloud platforms, and users are typically organized into multiple layers, each serving a specific purpose and composed of specific types of devices. The common layers in the architecture of IoT systems are described below. [6] [7]

Perception and Action Layer This layer contains the IoT devices. They can be categorized into 2 types. On the one hand, there are sensors that perceive and collect data from the environment. It includes temperature sensors, motion detectors, cameras, and other types of sensors. On the other hand, there are actuators that perform physical actions to change the environment. It includes smart lights, speakers, sockets, and other types of actuators.

Network Layer The network layer is responsible for the communication and connectivity between the devices in the IoT ecosystem. It involves networking technologies such as WiFi, Bluetooth, and ZigBee, which allow the devices to transmit data to each other and to other layers of the architecture.

Gateway Layer The gateway layer acts as a bridge between the devices and the cloud. Gateways perform protocol translation to transmit the data over the internet to the backend services.

Cloud Layer This layer encompasses cloud platforms or backend servers that receive and store the data collected from the devices. It provides computational resources, storage capabilities, and data management infrastructure for processing and analyzing the data.

Application Layer The application layer consists of user interfaces, applications, and software that allow users to interact with the IoT system. Users can monitor and control devices, access data insights, and perform various tasks through applications or web interfaces.

2.2.4 IoT Communication Protocols

Standard communication protocols do not suit IoT devices' communications due to their limited processing power, amount of memory, and limited energy. Specific protocols have been developed for IoT devices. Among them are IEEE 802.11, IEEE 802.15.4, ZigBee, and BLE.

IEEE 802.11 (WiFi)

The IEEE 802.11 standards, commonly known as WiFi, define the specifications for wireless local area networks. These standards outline the protocols and technologies used for wireless communication between devices in a network. WiFi is a widely adopted wireless communication protocol that enables IoT devices to connect to local area networks and the Internet. It provides high-speed data transmission and compatibility with existing internet infrastructure.

The WiFi network protocol is commonly utilized by IoT devices that have a continuous external power source. Due to the higher computing power requirements of WiFi, this technology tends to consume more energy, making it less suitable for resource-constrained IoT-embedded devices. Therefore, WiFi is typically employed by IoT devices that can maintain a constant power supply such as smart lights, smart speakers, thermostats, and security cameras, enabling them to benefit from its higher data transfer rates and broader coverage range. However, for IoT devices with limited processing power and battery life, alternative communication protocols such as IEEE 802.15.4, ZigBee, and BLE are often preferred, as they offer more energy-efficient solutions.

IEEE 802.15.4

IEEE 802.15.4 is another standard for wireless communication specifically designed for low-power, low-data-rate applications. This makes it an ideal choice for battery-powered devices that require long-term operation without frequent battery replacements.[8]

With low energy consumption but low data rates, IEEE 802.15.4 is ideal for applications that require periodic transmission of small amounts of data for battery-powered devices. It also provides robustness in the face of interference, making it reliable in noisy environments. The standard is widely used in industrial automation, home automation, and smart energy management.

IEEE 802.15.4 serves as the foundation for several higher-level protocols, such as ZigBee and Thread. It offers energy-efficient and cost-effective solutions for devices with limited processing power and battery life. Its versatility and support for different network topologies make it applicable in various IoT domains, including home automation, industrial monitoring, healthcare, and smart cities.

ZigBee

ZigBee is a wireless communication technology created by ZigBee Alliance (now called Connectivity Standards Alliance) and built on top of the IEEE 802.15.4 standard. It is specifically designed for low-power, low-data-rate applications, making it well-suited for IoT devices with limited processing power and battery life.[9] ZigBee operates in the 2.4 GHz frequency band and provides a mesh network topology, enabling devices to form self-organizing networks.

Additionally, ZigBee offers reliable and secure communication through its built-in security features, including encryption and authentication mechanisms. This ensures that data transmitted between ZigBee-enabled devices remains protected from unauthorized access, making it suitable for applications that require robust security measures.

IoT devices that employ ZigBee technology benefit from its energy efficiency, which allows for extended battery life and reduced power consumption. This makes ZigBee a popular choice for various applications, including home automation, smart lighting, industrial monitoring, and healthcare devices. The low-power characteristics of ZigBee enable IoT devices to operate for extended periods without the need for frequent battery replacement or recharging.

Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a wireless communication technology designed for data transfer between devices. It is a variant of the classic Bluetooth technology and was introduced with Bluetooth 4.0. Compared to traditional Bluetooth, BLE is optimized for low power consumption making it ideal for battery-operated devices, such as IoT devices where power efficiency is crucial. BLE also uses 2.4 GHz radio frequencies but with a simpler modulation system and optimized for transmitting small amounts of data. It offers data rates up to 2 Mbps, but most BLE applications use lower data rates to save energy.

Because BLE is optimized for low power consumption and designed for low-to-moderate data rate applications, it is widely used in IoT applications. BLE is extensively used in fitness trackers, smartwatches, and other wearables to connect with smartphones and transmit data such as health and fitness information. BLE is also used in smart home applications where it enables the control and monitoring of various IoT devices, including lighting systems, thermostats, door locks, sensors, and appliances. Besides that, BLE is also used for asset tracking. BLE beacons are used for proximity-based services and asset tracking. They can be placed on objects or in physical spaces to transmit signals that can be detected by BLE-enabled devices.

2.3 Forensics

2.3.1 Forensic Definition

Forensic refers to the application of scientific principles and techniques in a criminal investigation process to establish facts. Forensic science encompasses a wide range of disciplines, including chemistry, biology, physics, and engineering, which can be used to analyze physical evidence from crime scenes. The goal of forensic investigation is to gather, preserve, analyze, and interpret evidence in a way that is objective and unbiased so that it can be used to establish the truth about what happened and to help solve crimes. Forensic evidence can be used in criminal court to help establish the guilt or

innocence of a suspect, but it can also be used in other competencies of law, for example, in civil cases to establish liability or to resolve disputes.

2.3.2 Digital Forensics

Digital forensics, also known as computer forensics, is a subfield of forensic science that specifically deals with the investigation of digital devices, data, and systems. Digital forensics involves the recovery and analysis of data from computers, servers, mobile devices, and other electronic storage media to gather evidence for use in criminal or civil proceedings.

The main difference between forensic science and digital forensics is the type of evidence being analyzed. On the one hand, forensic science involves the analysis of physical evidence, such as fingerprints, DNA, fibers, and other physical materials. Digital forensics, on the other hand, involves the analysis of digital data, including files, emails, texts, and other electronic communications. Digital forensics requires specialized knowledge and tools to extract and analyze this data, and it often involves the use of computer software and programming skills.

2.3.3 IoT Forensics

IoT Computer Forensics is a subfield of digital forensic science that specifically deals with IoT devices and their related environment. The IoT device takes part in an ecosystem following the IoT architecture described in section 2.2.3. It evolves IoT devices, communication network devices, cloud platforms, applications, and services.

IoT computer forensics is commonly divided into 3 sub-levels of digital forensics namely: device-level forensics, network forensics, and cloud forensics. Device-level forensics involves the collection of local memory data retrieved from IoT devices, network forensics involves network logs extraction and analysis, and cloud forensics involves getting the data and logs from the cloud provider that is linked to the IoT devices of interest. [10]

Smart Home Forensics

Smart home forensics can be defined as a specialized field inside IoT forensics that focuses on the investigation and analysis of digital evidence found in homes that have smart devices in their environment. Smart homes can be equipped with various interconnected devices and systems that provide automation and control functionalities.

2.3.4 IoT Forensics vs Digital Forensics

Compared to traditional digital forensics, IoT forensics have several differences that are worth noting.[11] In a traditional crime scene, evidence collection involves traditional computer systems such as laptops, tablets, and mobile phones. The widespread adoption of wireless technologies has resulted in a vast number of IoT devices. In an IoT-based crime

scene, evidence collection involves a wide range of devices, including household appliances like thermostats, lighting, locks, dishwashers, and baby monitors. These IoT devices are typically smaller, simpler, and more specialized than general-purpose computers, and they often have limited processing power and storage capacity. This makes them more challenging to analyze from a forensic perspective.

Digital Forensics (DF) investigations typically began with a few devices, such as desktop computers, and gradually expanded to include other devices on the desk, such as USB drives, external hard drives, and mobile computing devices like tablets and smartphones. The focus on these devices is driven by the potential for them to contain information that may be relevant to DF investigations. [12] The IoT devices in the environment, along with their gateways, communication infrastructure, platforms, and interfaces, all produce a large amount of information including sensor readings, logs, and communications which can be used as evidence in forensic investigations. This data is distributed across multiple devices and networks. This means that in an IoT-based crime scene, the focus should be on the system as a whole rather than individual components.

Additionally, there are new sources of evidence that must be analyzed due to the architecture of IoT systems. The FIGURE 2.1 summarizes the different sources of evidence in an IoT ecosystem. There are two main sources of evidence in the IoT: internal and external networks. The internal network refers to the local environment of connected objects and local access points, where raw measured data, information about device communication, and network configuration data can be found. The external network includes networking, service, and interface layers, and may include weblogs, virtual machines, sensor data, and network logs. This external network evidence is particularly important in the IoT due to the diverse range of devices and communication protocols used in this environment. [13]

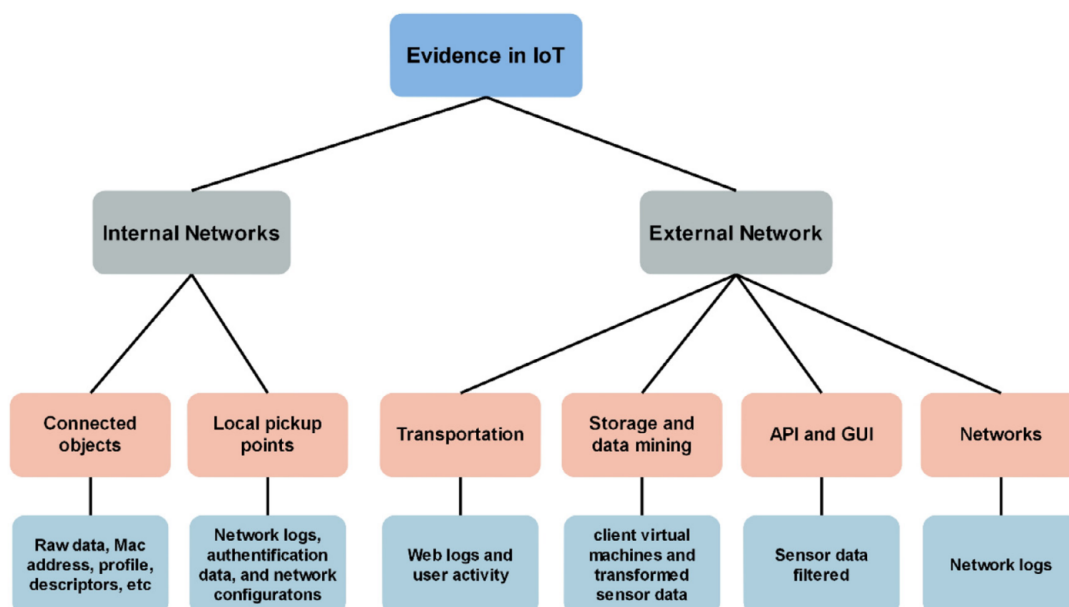


FIGURE 2.1: Sources of Evidences in Internet of Things Systems [13]

2.4 Conclusion

In conclusion, this chapter provided a comprehensive overview of the key concepts and background information necessary for understanding the subsequent chapters. We explored the definition and architecture of the Internet of Things, as well as the communication protocols of IoT devices. The chapter also introduced the field of forensics, including digital forensics and its application in IoT environments. This foundational knowledge sets the stage for the in-depth exploration of IoT forensics in the following chapters.

Part I

State-of-the-Art in IoT Forensics

Chapter 3

Literature Review

3.1 Introduction

Digital forensics experts face unique and complex challenges when attempting to investigate crime scenes involving Internet of Things devices. Those challenges make it impossible to apply traditional computer forensics techniques. Recently, researchers have focused on developing solutions to these challenges, with a particular focus on developing investigation frameworks and procedures specific to IoT. Investigation frameworks are structured approaches for conducting an investigation, while procedures outline the specific steps and techniques used to acquire and analyze digital evidence.

This chapter 3 provides a literature review of the current research advancements in IoT forensics. It provides an in-depth analysis of existing IoT forensic frameworks and procedures. This third chapter is organized into four sections. The section 3.1 briefly introduced the concept of framework and of procedures. It also details the content of this chapter and its structure. The section 3.2 focuses on existing IoT forensic frameworks. The subsequent subsections delve into specific aspects of IoT forensic frameworks, including existing standards, frameworks, and smart home forensics. The following section 3.3 explores some of the procedures that have been developed for some specific devices. In particular, it tries to identify how to acquire data from IoT devices to be used as evidence. Finally, the chapter concludes with a summary and key insights derived from the reviewed literature.

3.2 IoT Forensics Frameworks

IoT forensic investigation frameworks are structured approaches or methodologies used for conducting digital forensic investigations in IoT ecosystems. They provide a general outline or structure for the investigation process by defining the key components of the investigation, such as data collection, analysis, evidence preservation, and presentation. This high-level view of the investigation process is used as a guide to follow for investigators.

By utilizing digital forensic frameworks, several benefits can be achieved. It helps investigators stay organized and focused, ensuring that the investigation process is thorough and

comprehensive, increasing the precision of the investigation and reducing the possibility of human error and so increasing the chance of being admissible in a court of law. These frameworks can streamline repetitive tasks of the case, freeing up investigators' time to focus on the more intricate aspects of the case. Overall, the utilization of high-level frameworks in IoT forensics can result in more efficient, accurate, and consistent investigations even when investigations are carried out by different teams or organizations, leading to stronger cases in the court of law.

3.2.1 Standards in IoT Forensics Frameworks

A standard is a norm or set of criteria established and widely accepted by the expert scientific community to guide and regulate practices and processes in a specific area. Standards provide common guidelines and references for achieving a uniformly high level of authenticity, veracity, integrity, and reliability. Standardized frameworks provide a common ground for consistent practices, methodologies, and techniques.

Adherence to recognized forensic analysis standards and procedures is crucial to the acceptance of evidence in a court of law because it guarantees the evidence collected meets the necessary legal requirements and can withstand scrutiny in court. Evidence obtained and analyzed in a manner that does not comply with established procedures is more likely to be challenged or excluded from the case.

The Complexity of Establishing Standards in IoT Forensics

Establishing standards in the realm of IoT forensics is a complex undertaking. The initial barrier to IoT standardization results from the fact that IoT devices differ in formats, features, functionalities, and manufacturers. There is a lack of uniformity and consistency among them. When trying to establish standardized frameworks for conducting forensic investigations, this diversity poses significant obstacles.

The quick development of IoT technology is another factor that makes standardization difficult. Existing standards are quickly rendered obsolete as new technologies and communication protocols are developed. The IoT environment is dynamic, necessitating continuous efforts to adapt and revise standards to keep up with developments. Forging a thorough and broadly accepted framework will be extremely challenging given the ongoing cycle of technological evolution and standardization.

The standardization process is further complicated by privacy and legal concerns. IoT devices process sensitive personal data, creating questions about privacy rights and legal compliance. Developing standardized frameworks that preserve both investigative criteria and individual rights presents extra issues due to the need to balance the need for efficient forensic investigation methodologies with privacy protection.

Last but not least, the complexity of standardizing is further increased by the multidisciplinary nature of IoT Forensics. Collaboration and cooperation between professionals

from several domains, such as computer science, network engineering, and law enforcement, are essential for the success of standardization initiatives. A fundamental problem in the standardization process is integrating viewpoints from different disciplines and bridging the gap between technical and legal considerations. Also, the standardization process involves a variety of parties with various goals, privacy concerns, and interests. Industry experts, manufacturers, researchers, judicial authorities, and regulatory agencies are some of them. It will be difficult for this heterogeneous group to come to an agreement because each entity may have different viewpoints, objectives, and limits.

Coming Efforts in Standardization

Despite the complexities involved, the importance of standardization in IoT Forensics cannot be overstated. At the time of writing this, the field of IoT device forensics is still relatively new and there are currently no widely established standards for conducting IoT forensic investigation.

However, there are international standards and guidelines for digital forensics investigation in general, which in some cases may be applied to IoT device investigations. These standards include ISO/IEC 27037 [14], which provides guidelines for the identification, collection, acquisition, and preservation of digital evidence, and the ISO/IEC 27042 [15], which provides guidelines for digital forensic evidence analysis.

Similarly, the European Network of Forensic Science Institutes (ENFSI) and the National Institute of Standards and Technology (NIST) have published guidelines and recommendations for digital forensic investigation in general. [16][17] Even if the authors claims: *"The publication is not to be used as an all-inclusive step-by-step guide for executing a digital forensic investigation or construed as legal advice."*[17], these guidelines cover various aspects of the investigation process, including data acquisition, data analysis, and reporting, and may serve as a starting point for developing standards in the future. Those guidelines and recommendations may be adapted for IoT device investigations if special care is taken to deal with the new challenges raised by IoT. Indeed, when it comes to IoT forensics, there are several challenges that avoid resorting to traditional digital forensics. These challenges will be covered in sections 4.2 and 4.3. They necessitate the development of specialized guidelines and recommendations. To give a brief example, the seizure of IoT devices presents a notable difference compared to traditional devices in the context of digital forensics. Unlike conventional devices, IoT devices often have embedded and interconnected components, making the acquisition and preservation of evidence more complex. Traditional methods of seizing devices, such as physically disconnecting them from the network or shutting them down, may not be viable options for IoT devices as they can disrupt ongoing processes or affect the network's integrity. Special considerations need to be taken into account to ensure the proper handling and preservation of IoT devices during the seizure process.[18]

While efforts have been made to develop guidelines and recommendations for IoT forensics, the development of standard IoT forensics frameworks is still needed and requires further

research and active participation in standardization activities.

3.2.2 Overview of Existing IoT Forensics Frameworks

The literature contains reviews and surveys that studied the existing IoT forensic frameworks. Due to the intricate nature of the processes in IoT forensics, a number of models have been proposed. In their paper, Yakubu et al. reviewed the IoT forensic literature and identified gaps and limitations in the sampled paper. Especially, they noticed a lack of standardized methodologies, tools, and techniques that can handle the heterogeneity of devices. On the basis of their research, they concluded that “*none of the forensic models that have been proposed can reliably and timely extract evidence*” and that further research around IoT forensics are required.[19]

In their paper, Lutta, Pantaleon, et al. provides a systematic literature review of the current advancements in IoT forensics. They studied thirteen methodologies, models, and frameworks and their respective variants that were proposed to contribute to IoT forensics. From their paper, we noticed that the majority of research on IoT forensic frameworks is focused on industrial applications for post-incident investigations. These frameworks are often based on prior actions or specific setups before an incident occurs. This limits their applicability and relevance for investigations at a crime scene. In their paper, they also studied the practicality of the available IoT forensics frameworks and methodologies. They concluded that the majority of recent studies are more theoretical than practical and that more pragmatic strategies are required to address the special IoT forensics challenges.[20]

More recently, another systematic literature review was published in April 2023. In their paper, Al-Hussaeni, Khalil, et al. surveyed eighteen IoT forensics frameworks and identify for each proposed framework the IoT challenge addressed. As exposed in TABLE 5 of their paper, there is currently no unique IoT forensic framework that can address all of the IoT challenges. Nevertheless, they also showed that existing IoT forensic frameworks could be combined with one another to address all of the IoT forensic challenges.[21]

In summary, all of the existing literature review studied tends to draw the same conclusions. There is a lack of implementation and testing of proposed frameworks. Most of the existing research is theoretical and lacks pragmatism.

Common IoT Investigation Process

Among all the frameworks and models proposed in the literature, there are similarities that are worth mentioning. Each framework covers, at least partially, a process that can be divided into 4 phases. The four phases identified are the readiness phase, the pre-investigation phase, the investigation phase, and the post-investigation phase.

1. **The readiness phase:** This phase takes place prior to the identification of incidents. The purpose of the readiness phase is to ensure that all necessary preparations and prerequisites are in place. It guarantees that the IoT environment will be ready

for later forensic examination.[22] This readiness phase is typically implemented in industrial use cases and may not be applicable to domestic scenarios.

2. **The pre-investigation phase:** This phase is the preparation step of the investigation phase. It involves securing the scene and maintaining it 'as is' to preserve evidence, obtaining legal authority permission, conducting preliminary assessments, checking necessary tools and resources availability, and ensuring the presence of trained personnel. The goal is to establish a solid foundation for the next crucial phase.
3. **The investigation phase:** This phase is the core phase of the forensic process. It can be divided into the following five steps.
 - (a) Identification: This step is responsible for the identification of the potential source of evidence, their location, and their role in the IoT ecosystem. As illustrated in FIGURE 2.1 from section 2.3.4, investigators may identify several potential storage locations for relevant data, ranging from IoT devices up to cloud servers.
 - (b) Collection: This step focuses on physically gathering the identified potential sources of evidence from the IoT environment while preserving their integrity.
 - (c) Acquisition: This step involves creating a forensic image or copy of the collected evidence for analysis purposes in the next step.
 - (d) Analysis: This step consists of examining and interpreting the acquired evidence to extract meaningful information and draw conclusions.
 - (e) Preservation: This step focuses on preserving the evidence in the states as they were acquired. It involves securely storing them to prevent any unauthorized modifications or tampering.

For each of its sub-phase, three levels of forensics, following the 1-2-3 zone approach proposed by Oriwoh et al. [12], must be considered: cloud forensics, network forensics, and device forensics.

4. **The post-investigation phase:** This phase is the last one. It includes the reporting of findings ensuring compliance with legal requirements, the presentation of evidence in court, and the progress of legal proceedings.

Additionally, some frameworks include another phase that can be concurrent with the previous ones. It involves legal proceedings such as obtaining authorization, documentation, preserving the Chain of Custody, and preserving physical evidence.

The above phases are similar to those of the traditional digital forensics framework. They differ with the greater emphasis on IoT device identification, the various sources of evidence, and the difficulties to preserve volatile data. Between existing frameworks, the main differences lie in their approach, their scope, the specific steps involved, the level of detail provided, and the emphasis on different aspects of IoT forensics. Several tables

from the literature categorize the contribution of the existing research in IoT forensics based on different aspects. The TABLE 3 of [13] categorizes existing work in terms of forensics investigation models, forensics acquisition, forensics challenges, forensics analysis, data synchronization, and privacy-aware forensics. The TABLE 2 of [23] presents a categorization of the existing works, taking into account the limitations and gaps in relation to the practical application of the research in the IoT forensic process. The TABLE 5 of [21] compares the models and frameworks based on the challenges they address and whether they were implemented and tested.

3.2.3 Smart Home Forensic Frameworks

Most of the proposed framework does not suit domestic IoT devices, especially for smart home environments. The issue relies on the readiness phase which requires a set of measures to make the environment ready for further IoT forensic processes. Incorporating a forensic component, such as the collection of logs for centralized storage, into domestic IoT environments would only increase the complexity and cost of devices, which would conflict with the manufacturers' objectives.

The literature contains few research that tries to seek the challenges of domestic IoT devices. Conti et al. conducted a literature review and concluded that smart home forensics is understudied compared to smart home security or other digital forensics topics. [24]

In their paper, Shujun Li et al. proposed yet another investigation framework but this time for smart home automation systems and with pragmatism in mind. This framework is composed of seven phases [24] summarized in the FIGURE 3.1. Each box is numbered and represents a distinct phase.

As the authors claim, their frameworks could be used as “*a quick reference for digital forensic investigators*” [24]. The authors tested their framework in three different case studies. However, their simulations are not enough representative of real-life scenario.[20] Still, the framework proposed in this paper can serve as a useful reference for investigators.

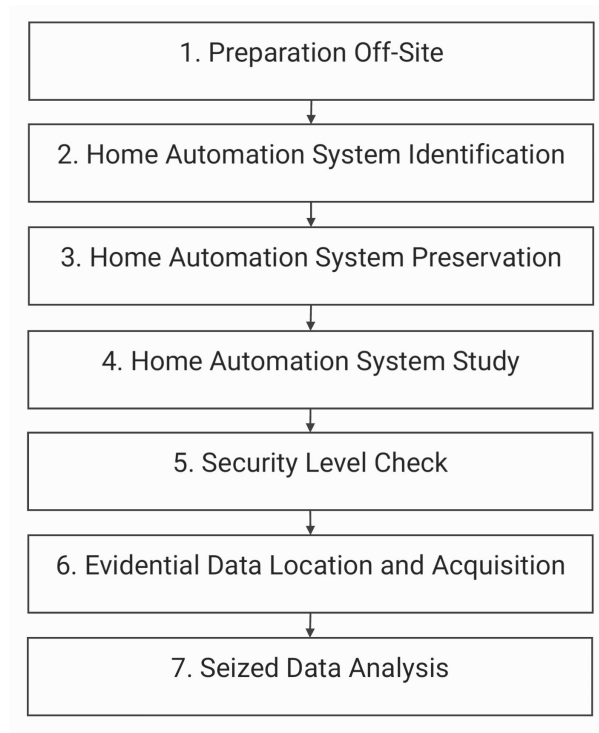


FIGURE 3.1: Seven-Phases Smart Home Investigation Framework

3.3 IoT Forensic Procedures

Besides the myriad of IoT forensic investigation frameworks, the literature also contains IoT forensic procedures. In contrast with frameworks that are structured approaches or methodologies, procedures outline specific steps and techniques to perform forensics.

In order to provide a detailed process for exploring data in IoT devices, procedures need to be more specific. This involves utilizing tools but they cannot be generalized to all devices. Consequently, the existing procedures are usually tied to a device or, at most, a brand or range of devices.

The specific procedures used in an IoT forensic investigation can vary depending on the type of device being investigated and the type of data being collected. For example, procedures for collecting and analyzing data from a smart home device, such as a smart thermostat or a smart lock, will be different from procedures for collecting and analyzing data from a wearable device, such as a fitness tracker. Due to the lack of standardization in IoT devices, the tools and methods employed can vary significantly from one device to another. This variability extends even to devices of the same type but different brands, as each brand may incorporate proprietary files in their software, necessitating the use of distinct analysis methods.

Several research studies have conducted forensic examinations on various IoT devices, providing detailed methods for conducting the investigative process on both the examined devices and others within the same product range. These research papers can be regarded as case studies since they serve as practical demonstrations of the procedures applied to these specific devices. The following subsequent sections explore some of the existing practical forensic works.

Smart TV Forensics: Digital Traces on Televisions

In this paper, Boztas et al. [25] explored Smart TVs to acquire forensic information for an investigation. They chose a television from Samsung, which is a brand that is highly present in this market. They created different scenarios to simulate the real usage and generate realistic data. To acquire the data from the Smart TV, they utilized three methods. The first one studies the signals going to the eMMC memory chip. The second one used a forensic tool named *NFI Memory Toolkit* to read memory data. The last one used the application. They were successful using the two last methods. After acquiring the information, they were able to analyze it and find forensically interesting information, such as system information, app activity, and web browsing activity. However, the used methods are not ideal. The NFI Toolkit method requires desoldering the memory chip from the Smart TV, which has a chance of damaging the chip and/or TV. The application method can be patched by the brands simply through security updates (which happened during their research). Furthermore, the tools that are used here are not a guarantee for other types of devices, because the NFI Toolkit might not be able to extract the memory information from other types of memories, especially from the small IoT devices, that can have more limited types of components, and the used application is specific for that model or brand of device.

Smart Home Forensics—Data Analysis of IoT Devices

Here, Kim et al. [26] analyzed different types of smart home devices. They explored a realistic scenario with multiple devices. They use the *Google Nest Hub* home assistant, the *Kasa* smart camera from *TP-Link*, and IoT sensors from Samsung SmartThings. The researchers explored different types of methods to acquire the data: through the companion apps, through the Google “My Activity” interface, and through a Google Home API. They were able to acquire and analyze different types of data, such as movement, voice information, and calls. This study details methods that can be used to acquire data from those devices. The Google “My Activity” method can also be extended for a wider range of other devices.

Welcome Pwn: Almond Smart Home Hub Forensics

Awasthi et al [27] analyzed the *Almond+* router from the company *Securifi*. This router integrates a smart hub that is used as a central unit for the associated smart devices

such as lamps, dimmers, and sensors. In order to obtain information about smart devices connected to the hub, they explored the router and the companion apps. They were successful to obtain logs from the device, from the cloud, and from the companion apps. The *Almond* ecosystem holds significant potential for forensic investigations as it captures interactions within a specific location, along with the associated time and date. Consequently, the data derived from this device can offer crucial information for an investigation. However, despite the researchers' successful findings in terms of forensic value, they also expressed concerns. They discovered multiple vulnerabilities in the router, raising apprehensions about compromising both user privacy and security. Additionally, the researchers highlighted the potential impact on an ongoing investigation, as a digitally savvy criminal could exploit the device, leading to data erasure and the disappearance of their tracks.

Internet of Things Forensics – Challenges and a Case Study

In this paper, Alabdulsalam et al. [28] proceeded to investigate the device-level forensics for an *Apple Watch Series 2*. The researchers explored two different methods to extract the data.

The first approach involved manual acquisition, where the researchers utilized the device itself to extract data from its screen. Through this method, they were able to retrieve various types of information such as messages, pictures, emails, calendar events, contacts, previous phone calls, and the list of installed apps on the watch. It is important to note that this data extraction process did not require the presence of a phone; however, prior synchronization between the watch and the phone was necessary for successful extraction.

The second method employed in the study was a logical acquisition using digital forensic tools. Specifically, the researchers utilized the *Cellebrite UFED*, a widely-used tool for extracting data from mobile phones on the *iPhone* to extract smartwatch data that was synced with the *iPhone*. Through this method, they were able to recover even more extensive data, including measured health data and GPS data recorded from previous activities performed with the watch.

This study revealed a notable dependence on smartphones when extracting data from IoT devices such as the *Apple Watch*. It demonstrated that without a synced phone or if the synchronization has not occurred recently, the amount of data that can be directly extracted from the Apple Watch itself is limited.

EMvidence: A Framework for Digital Evidence Acquisition from IoT Devices through Electromagnetic Side-Channel Analysis

Sayakkara et al. [29] have developed an application to obtain forensically important data from observing electromagnetic (EM) emissions, by using EM side-channel analysis (EM-SCA) around a studied device. In this work, they introduce an extendable framework named EMvidence. They paired EMvidence with a machine learning classifier to test

the efficiency of their work, and they obtained very good results. When it is difficult or impossible to acquire forensic evidence from an IoT device without having recourse to intruding methods, the proposed solution is a promising approach.

3.4 Conclusion

In conclusion, this chapter presented a comprehensive literature review of existing frameworks and procedures for performing forensics in the IoT ecosystem.

The examination of IoT forensics frameworks highlighted the importance of standardization efforts. Next, the overview of existing IoT forensics frameworks showcased various approaches and methodologies employed in the field. We highlighted the need for future research to concentrate on the implementation and testing of frameworks proposed in the literature. We also identified a common IoT investigation process among all studied frameworks. This process can be divided into 4 phases: the readiness phase, the pre-investigation phase, the investigation phase, and the post-investigation phase. In addition, we investigated smart home forensic frameworks, realizing the lack of studies in the field.

In the second part, we explored some IoT forensic procedures and the particular methods and techniques used to acquire data from IoT devices. We discussed different case studies and showed that methods change depending on each case and device.

Chapter 4

The Underlying Problem

4.1 Introduction

In this chapter, the underlying problems of IoT forensics are addressed. The sections 4.2 and 4.3 highlight the legal and technical challenges that hinder effective investigations in the IoT ecosystem. The legal challenges encompass issues such as the seizing IoT devices, the maintaining of the chain of custody, data localization and jurisdiction, and the applicability and integration of IoT forensics in investigations. Technical challenges include the complexity arising from the heterogeneity of IoT devices, data format, data location, closed software and proprietary hardware, as well as the identification and localization of IoT devices. The section 4.4 explores the role of digital forensics tools and studies their applicability in IoT forensics, focusing on device localization, data extraction, data analysis, and evidence preservation. Despite the existing tools, advancements are needed to enhance IoT forensics practices. The section 4.5 details the need for standardized procedures, specialized tools tailored for IoT investigations, the establishment of best practices, and of training and education for forensic professionals. The conclusion of this chapter emphasizes the significance of addressing challenges such as device localization, bringing into context the purpose of the second part of this work.

4.2 Legal Challenges

Compared to common judicial and digital forensic techniques, IoT forensics raised new legal challenges which are posed by IoT devices and their unique characteristics.

Applicability and Integration in an Investigation

Currently, the literature only contains high-level methodologies named digital forensic frameworks. The methodologies presented in the literature are high-level, broad, and general. They lack specific instructions on how to implement the analysis in each step of the suggested approaches. Also, to date, there is no widely accepted framework for conducting digital forensic investigations in IoT environments. Still, there is an incoming interest in developing such ones. This lack of frameworks exacerbates the difficulties

faced by digital forensics investigators in their efforts to gather and analyze evidence in an IoT-based infrastructure [22].

Furthermore, these frameworks are usually not intended for police officers investigating crime scenes that evolved IoT devices. Instead, they are primarily tailored for companies that experience cyberattacks. Consequently, most IoT devices in corporate environments are interconnected with other devices that collect relevant information, facilitating forensic analysis. However, the situation is different in the context of smart homes. In most cases, if not all, smart home devices are used independently without any additional devices that can aid in an investigation.

Seizing IoT Devices and Maintaining the Chain of Custody

A significant legal concern that extends beyond just IoT devices is the seizure and management of digital evidence. There are multiple aspects of this regarding the legality of the search and also the preservation of privacy, which are all mentioned in the guide by Kerr.[30] It is already crucial to exercise caution when handling digital evidence, but when it comes to IoT, there are even greater demands and requirements.

In IoT forensics, the problem is mostly related to the human part and its compromises, especially the time of seizing. At a crime scene, important information can be obtained from these devices, such as the action of opening or closing a door. However, investigators, unaware of the presence of IoT devices, can contaminate the crime scene digitally by continuing their investigation without taking IoT devices into account. This occurs primarily due to the limited storage capacity of IoT devices, leading to the potential deletion of previously collected data. As these devices typically have restricted storage capabilities, they do not retain a significant amount of data. Consequently, if new information is received, it may overwrite past and potentially crucial data. Additionally, investigators may disable crucial components of the smart home ecosystem, which can result in device malfunctions or even cause them to be reset. Alternatively, they may directly remove the device, which could continue to operate using internal batteries and generate data even when outside of the original environment. Consequently, there is a risk of generating false information, and if investigators fail to exercise caution, it could potentially lead to the dismissal of the entire evidence. The method of seizing each device varies, emphasizing the need for specific guidelines. Bouchaud et al. explore this aspect of evidence gathering in one of his works, and concludes that it is necessary to have these different approaches for IoT devices.[18]

Besides that, the chain of custody is the guarantee of the integrity and reliability of the evidence collected. This requires careful handling of the devices, proper documentation, and secure storage to prevent tampering or unauthorized access. IoT devices are typically smaller, simpler, and more specialized than general-purpose computers, which increases the risk of altering the data during forensic analysis.

Data Location

IoT system involves a local physical structure that extends through the cloud and the Internet, with data that is no longer tied to a specific medium. Data is then spread in multiple locations with different laws and jurisdictions which introduces legal complexities that can make it difficult to determine which laws apply due to the use of the cloud for storing IoT-generated data.[31]

Legal and Ethical Issues

IoT forensics may also raise legal and ethical issues related to privacy and consent, as many IoT devices are used in private or sensitive settings. It is important to consider these issues when collecting and analyzing IoT data for forensic purposes. As mentioned before, in some guides for digital forensics, such as in the one by Kerr [30], privacy is already concerned and the legality of digital evidence is discussed.

4.3 Technical Challenges

Compared to common judicial and digital forensic techniques, IoT forensics raised new technical challenges which are posed by IoT devices and their unique characteristics.

Heterogeneity of IoT devices

One of these challenges is the heterogeneity of IoT devices in terms of their nature, operation, communication protocols, and data management. These devices come in a wide range of types, including smart fridges, smart locks, and smart lights, and they have different hardware and software capabilities. Some devices can connect directly to the Internet, while others require specific hubs, and they use different communication protocols, including some non-standard protocols. This diversity of devices and protocols makes it difficult to apply traditional forensic investigation models to the IoT. [13][32]

Data Format

Additionally, the data generated by IoT devices may be stored in various formats and locations, and it may be fragmented and dispersed across different networks depending on the specific equipment present and their functions.[31]

Closed Software and Proprietary Hardware

Another challenge of IoT forensics is the issue of closed-source software and proprietary hardware, which can make it difficult for investigators to access and analyze the data stored on these devices. This can limit the effectiveness of forensic techniques and make it harder to extract valuable evidence.

Identification and Localization

Another challenge that arises in the realm of IoT is the identification and localization of devices. As mentioned earlier, the proliferation of IoT is becoming increasingly prevalent in society, including the domain of smart homes. Consequently, the level of technical knowledge required to distinguish ordinary devices from smart devices is also escalating. The process of identifying and locating IoT devices can prove to be arduous due to their diverse forms, sizes, and designs. They can range from inconspicuous door sensors to everyday objects that have been augmented with smart capabilities. Unlike traditional devices, IoT devices typically lack explicit labeling to indicate their smart functionality. As a result, the search for these devices must be conducted with great care to avoid overlooking or misplacing them.

4.4 Digital Forensics Tools

At present, there is limited availability of dedicated tools specifically designed for IoT forensics. The majority of tools employed in this domain are those traditionally used in digital forensics. While these tools may not be tailor-made for IoT investigations, they can still offer some degree of assistance. The typical usage scenarios for those tools primarily involve cybercrimes that specifically targeted the seized devices. However, in cases where the devices themselves hold significant relevance to the crime, these tools can also be employed, albeit not to their full potential. In this section, we will delve into various steps of the digital forensic process and explore the tools associated with each step. The purpose of this is to provide an overview of the current tools and parameters, shedding light on their applicability in the context of IoT forensics.

4.4.1 Device Localization

The process of locating devices is a crucial step in forensic investigations as it enables the identification and seizure of relevant evidence. While some devices, such as desktop computers, are easily identifiable in terms of their physical location, others can be concealed or difficult to locate. In such cases, specialized tools are employed to aid in their discovery. Presently, WiFi analyzers serve as the primary tools for this purpose, as they can detect wireless signals in the surrounding environment, alerting the user to the presence of WiFi-enabled devices nearby.

Wireshark

*Wireshark*¹ is a powerful network analyzer that offers real-time traffic capture capabilities. It serves as a valuable tool for examining network communications, enabling investigators to gain a deeper understanding of the transmitted frames. In the context of forensic investigations, Wireshark proves particularly useful in identifying devices that are present

¹<https://www.wireshark.org>

at a crime scene and are actively engaged in wireless communication. By leveraging Wireshark's features, investigators can effectively monitor and analyze the network traffic, revealing the presence of IoT devices.

Nmap

*Nmap*² is an open-source network discovery utility. It is most commonly used for penetration testing and security purposes, but it can be also used in forensics to find devices. Apart from scanning the network and indicating live devices, *Nmap* can also profile the devices, giving information about the operating system and software version that runs on these devices, which can facilitate the work of investigators to discover them.

4.4.2 Data Extraction

During this phase of forensic analysis, it is vital to acquire all available data from the device, including encrypted or deleted data. Equally important is the creation of a forensic image of the device, which serves as a preserved copy for subsequent analysis. Performing analysis directly on the original device can potentially alter or compromise the data. To accomplish these tasks, professionals commonly rely on specialized tools such as *Cellebrite UFED*, *Oxygen*, and *AccessData Forensic Toolkit (FTK)*. These tools offer comprehensive capabilities for data acquisition and preservation, ensuring the integrity of the forensic process and maximizing the information extracted from the device.

Cellebrite UFED

The *Cellebrite UFED*³ is a highly regarded mobile forensics tool widely utilized by law enforcement agencies worldwide for the extraction of data from various devices such as cellphones and tablets. It offers both physical and logical extraction methods, enabling the retrieval of comprehensive data stored in the device's memory. The tool supports a wide range of devices, encompassing diverse architectures and operating systems. To ensure compatibility with the latest device updates, *Cellebrite* regularly updates its toolchain. Notably, the *UFED* tool can employ exploits to bypass protections or unlock devices, providing valuable capabilities for investigative purposes.

Oxygen Forensic Detective

*Oxygen Forensic Detective*⁴, or simply *Oxygen* for short, is another powerful tool when regarding data extraction. It has very similar functionalities as *Cellebrite UFED*, for example. Even though *Oxygen* is an extraction tool, it can provide a preliminary data analysis, which can be useful to decide which direction the investigators should go. It also supports importing images from other forensic tools and is compatible with their

²<https://nmap.org>

³<https://cellebrite.com/en/ufed/>

⁴<https://oxygenforensics.com/en/products/oxygen-forensic-detective/>

proprietary formats, which can be helpful to use some of the features present in *Oxygen*, for example, the preliminary data analysis. *Oxygen* also has a user-friendly GUI, which can be helpful for investigators that are not very used to digital forensics, and it is considerably cheaper than the *UFED*.

AccessData FTK

*AccessData FTK*⁵ is another versatile extraction tool that shares similarities with the previously mentioned tools but with a focus on computers, hard drives, and cloud storage, in addition to mobile devices. While the acquisition methods may differ slightly from other tools, *AccessData FTK* provides various acquisition techniques, including live acquisition and forensic disk imaging. The tool also offers several preliminary data analysis features. Its comprehensive capabilities make it an intriguing tool for the forensic process, facilitating data extraction in a broader context.

4.4.3 Data Analysis

Data analysis is a crucial step in the investigation process as it enables the examination of collected data in a manner that can aid in the investigation. However, maintaining the forensic integrity of the evidence is equally important to ensure its reliability and admissibility in court. Several widely recognized and dependable tools are used worldwide to analyze the data extracted from seized devices.

Volatility

*Volatility*⁶ is an open-source framework primarily designed for analyzing the volatile memory of a computer. This memory, which can contain critical information about running processes, network connections, loaded modules, and more, is of great significance in forensic investigations. By working with samples collected from the RAM of seized devices, *Volatility* enables investigators to uncover the presence of malware or evidence of misconduct. The framework is highly regarded in these scenarios and offers compatibility with various plugins, enhancing its functionality and versatility.

Autopsy

*Autopsy*⁷ places a greater emphasis on general analysis capabilities. It offers support for various types of data sources, including disk images, mobile devices, and network captures. Equipped with multiple features, *Autopsy* proves valuable to investigators by assisting in tasks like file reassembly and generating comprehensive reports. It is also an open-source tool, and benefits from community contributions, which enhance its versatility and enable analysis of data that may not be natively supported by the software.

⁵<https://www.exterro.com/forensic-toolkit>

⁶<https://www.volatilityfoundation.org>

⁷<https://www.autopsy.com>

EnCase Forensic

*EnCase Forensic*⁸ is similar to *Autopsy* but it offers a more comprehensive and powerful set of features. It includes capabilities such as data decryption, integration with AI and Machine Learning algorithms for identifying illicit images, and optical character recognition for analyzing PDF files and extracting data more efficiently. *Encase* also integrates with other *Encase* products, creating a cohesive ecosystem that facilitates investigations.

4.4.4 Evidence Preservation

Preserving evidence is another important aspect of the criminal process, ensuring that the extracted data from seized devices remains uncorrupted and untampered. Maintaining the integrity and credibility of this data is paramount, requiring diligent measures throughout the entire forensic investigation.

One effective method for evidence preservation is creating forensic images, a feature already provided by the aforementioned extraction tools. Additionally, the use of write blockers provides an additional layer of security, preventing any changes to the data during the extraction process. Once obtained, the data must be securely stored using evidence management systems and encryption tools, ensuring its safety and preventing any potential challenges to its admissibility in court.

Tableau Forensic Bridges

*Tableau Forensic Bridges*⁹, developed by *OpenText*, encompass a series of hardware devices designed to facilitate device imaging without compromising their integrity. These bridges provide a secure connection between the seized device and the investigator's computer, ensuring that no alterations or modifications occur during the imaging process. By maintaining the integrity of the collected evidence, these bridges allow for the creation of an accurate image that can be utilized for further analysis and investigation.

AccessData FTK Lab

*AccessData FTK Lab*¹⁰ is a robust system designed for effective evidence management. It plays a critical role in ensuring case organization and preventing any mix-up or loss of evidence. With its advanced functionalities, *AccessData FTK Lab* allows for the efficient handling of large datasets, enabling investigators to analyze and segregate relevant information from irrelevant ones. Additionally, *AccessData FTK Lab* also offers collaboration features, facilitating the collaboration of multiple investigators or forensic teams in a case. But the primary focus of *AccessData FTK Lab* is to maintain the integrity of the

⁸<https://www.opentext.com/products/encase-forensic>

⁹<https://www.opentext.com/products/tableau-forensic-bridges>

¹⁰<https://www.exterro.com/ftk-lab>

evidences. The tool provides secure evidence storage, ensuring the confidentiality and integrity of the data. Only authorized users have access to the stored evidence, guaranteeing its protection and preserving its integrity throughout the investigation process.

VeraCrypt

*VeraCrypt*¹¹ is a powerful open-source tool designed specifically for encrypting disk images, offering enhanced access management capabilities. It allows automatic on-the-fly encryption of data. With support for parallelization and pipelining, *VeraCrypt* ensures efficient data reading, minimizing any performance impact caused by encryption. It also allows for fast access to encrypted data, maintaining the usability of the system. Even if it is not intended to be used as a standalone tool for evidence preservation in the forensic process, it adds an additional layer of protection to sensitive data. When used in conjunction with the aforementioned tools, it significantly enhances data security and confidentiality, providing investigators with an effective means of protecting encrypted disk images during forensic analysis.

4.4.5 Tools in IoT Forensics

As discussed earlier, the existing digital forensic tools, particularly those used for extraction and analysis, are not adequately adapted to address the specific challenges posed by IoT devices. In section 3.3, several cases were examined, and some of these tools were employed. However, it was observed that these tools do not fully meet the requirements of IoT forensics. For instance, tools like *Cellebrite UFED* and *Oxygen* primarily focus on data extraction from mobile phones, but they may not be capable of capturing all the data from other IoT devices such as smartwatches, as demonstrated in the study by Alabdulsalam et al.[28]

This finding aligns with the observations made by Plachkinova et al.[33], who highlighted that digital forensic investigators often lack the necessary tools and expertise to effectively handle smart home investigations. Similarly, Atlam et al. [34] concluded in their review that current digital forensic tools face limitations when it comes to effectively dealing with the complex and decentralized nature of IoT systems.

Related to device localization, tools like *Wireshark* and *Nmap* can be valuable for detecting devices, particularly if they communicate over WiFi. However, when it comes to other protocols commonly used in IoT, these software tools alone may not provide sufficient information. Additional hardware components and expertise are often required to capture and analyze data from these protocols.

Upon examining the other analyzed cases from section 3.3, it becomes apparent that the techniques employed are not ideal. Some methods rely on exploits, which may not be considered forensically sound, while others extract data from the cloud (e.g., Google's "MyActivity") or the device itself, which can be prone to modification or manipulation.

¹¹<https://www.veracrypt.fr/en/Home.html>

As a result, we can deduce that while existing digital forensics tools may provide some assistance in IoT forensics, they are far from being the optimal solution. To effectively investigate IoT devices, it is needed to develop specialized tools that focus specifically on IoT devices and protocols.

4.5 Advancements Needed

Given the challenges present in IoT forensics and the demands that the current tools are not able to sustain, there are some advancements needed regarding this aspect to effectively use IoT devices in criminal investigations for evidence recovery.

Standardized Procedures

The field of IoT device forensics is still relatively new, and there is a lack of standardized procedures for evidence recovery. On the theoretical side, multiple frameworks are present and they were explored, but none of them are officially accepted and there are no regularized guides on how to deal in practice with these kinds of devices. Establishing clear and consistent procedures for evidence recovery will help ensure that investigators can recover evidence reliably and consistently, which can be useful for investigation and also acceptable in court.

Specialized Tools

Given the wide range of hardware, software, and network configurations found in IoT devices, it becomes necessary to develop specialized tools supporting each device type. These tools also should have the capability to retrieve data from various sources, including the device itself, the cloud, or separate networks. Currently, as observed in the section 3.3, the majority of data extracted from IoT devices is obtained from mobile phones or the cloud. While this approach proves effective, it imposes limitations on the range of devices from which data can be obtained. Furthermore, this method does not encompass the entirety of the available data and may result in outdated information due to the need for regular synchronizations. Additionally, standalone devices cannot be analyzed using these methods.

Best Practices

As with any forensic investigation, best practices for handling evidence and preserving its integrity will be critical for successful prosecutions. Investigators will need to understand how to properly collect, analyze, and interpret digital evidence, as well as how to present it in court. This is why there is a need for a well-defined policy guide for collecting and dealing with this kind of evidence.

Training and Education

To effectively use IoT devices in criminal investigations, police officers and forensic investigators will need specialized training and education. This will include understanding the technical aspects of IoT devices, as well as the legal and ethical considerations surrounding the use of digital evidence in criminal proceedings. This communicates entirely with the legal process because if the officers are not trained to deal with these devices, they can make them non-admissible as evidence.

4.6 Conclusion

In conclusion, this chapter has examined the underlying problems of IoT forensics, focusing on the legal and technical challenges that hinder investigations in the IoT ecosystem. The legal challenges encompass issues related to the seizure and chain of custody of IoT devices, the applicability and integration of IoT forensics in investigations, and the complex legal and ethical issues surrounding IoT data. On the technical side, challenges arise from the heterogeneity of IoT devices, data format, and localization, closed software and proprietary hardware, and the identification and localization of IoT devices.

Digital forensics tools play a crucial role in addressing these challenges. However, advancements are still needed to enhance IoT forensics practices. This includes the development of standardized procedures, specialized tools tailored for IoT investigations, the establishment of best practices, and a focus on training and education for forensic professionals to navigate the complexities of IoT ecosystems.

A persistent challenge in IoT forensics revolves around the initial task of device localization. While extraction, data analysis, and preservation are all critical aspects of an investigation, they rely heavily on the ability to swiftly locate IoT devices at a crime scene. This is crucial for extracting vital data without inadvertently overwriting any evidence. Therefore, the development of a new tool should prioritize the accurate identification and location of these devices. Supporting this perspective, the previously mentioned Smart Home forensic framework, illustrated in [FIGURE 3.1](#), highlights the significance of device identification.

To address this challenge, we proposed in the second part of this work an application capable of intercepting and discovering IoT devices. This solution consolidates different protocols and facilitates the identification of IoT devices. By addressing the critical aspect of device localization, this tool would serve as the basis for subsequent phases of the investigation.

Part II

IoT Device Identification GUI

Chapter 5

GUI Design and Implementation

5.1 Introduction

Crime scenes are now even harder to navigate because of the proliferation of Internet of Things devices. IoT devices might not be immediately apparent because they can take on a wide range of forms and be integrated into everyday objects like speakers, light fixtures, and appliances. Because of this, it may be difficult for police officers who are unfamiliar with the Internet of Things to identify IoT devices as such and to recognize the potential value of any data that might be stored on them.

In the previous chapter, we highlighted the lack of forensic-specialized information retrieval techniques. We seek to address those previously mentioned challenges by proposing a Graphical User Interface (GUI) that displays the active IoT devices in the vicinity.

This chapter is divided into 7 sections. The section 5.2 begins with an overview of the design choices made during the development process, including the selection of protocols, hardware components, programming language, and GUI library. Following the design choices, the section 5.3 describes the graphical user interface (GUI) developed. We outline the instructions on how to use the GUI to enable users to interact with the interface. Additionally, we provide insights into the project organization. Subsequently, the sections 5.4, 5.5, and 5.6 respectively delve into the device identification process for WiFi, BLE, and ZigBee devices. For each protocol, we present the necessary steps involved in device discovery and data collection. We explore the specific frame structures of IEEE 802.11, BLE, and IEEE 802.15.4 protocols to highlight the key information of interest that can be extracted. The last section concludes this chapter by reviewing the various points covered.

5.2 Design Choices

5.2.1 Protocols

IoT devices use a variety of communication protocols depending on their specific requirements and use cases. Some of the commonly used protocols for IoT device communication

include WiFi (IEEE 802.11), BLE, ZigBee, Z-Wave, Thread, BLE, LoRaWAN, MQTT, CoAP. The manufacturer's choice of using a specific communication protocol depends on factors such as device compatibility, power consumption requirements, and range. The protocols that are going to be explored in this work have already been detailed in subsection 2.2.4.

We selected BLE, WiFi, and ZigBee for our study. By focusing on these specific communication protocols, the sniffing process targets the most commonly employed technologies in the IoT landscape and increases the chances of successfully identifying IoT devices.

IEEE 802.11 on 2.4 GHz

As explained before, WiFi is a commonly used standard for IoT device communications. WiFi can operate in 2 distinct frequency bands. Only the 2.4 GHz one will be explored in this work. This choice was made for the following reasons.

On the one hand, sniffing the 5 GHz band requires specialized equipment capable of capturing and analyzing packets in that frequency range. In our research, we have not found a chip that is relatively cheap, that supports promiscuous mode on 5 GHz, and that provides a tool to analyze the content of the frames captured. Existing chips usually provide features for sniffing both 2.4 and 5.0 GHz bands simultaneously. This increases the processing power needed and causes the price of chips to increase with them.

On the other hand, while some rare IoT devices utilize the 5 GHz band for communication, the majority of consumer-grade IoT devices still predominantly operate on the 2.4 GHz band. The 5 GHz band has a shorter range compared to the 2.4 GHz band. It is more susceptible to signal degradation caused by obstacles like walls and furniture, making it less practical for scenarios where the devices are dispersed over a larger area such as a house.

The focus of our research is to help investigators in domestic or typical smart home environments. In this scenario, it is likely that a significant portion of IoT devices will be using the 2.4 GHz band. Thus, prioritizing the sniffing of the 2.4 GHz band may yield more relevant results. On the opposite, allocating resources to sniff the 5 GHz band would probably not provide significant additional benefits in terms of the number of IoT devices identified.

Bluetooth Low Energy

As previously discussed in section 2.2.4, Bluetooth Low Energy (BLE) is a widely adopted protocol in domestic IoT environments. Due to the prevalence of devices utilizing this protocol, it is an integral part of the sniffer being employed. Its utilization extends to the tracking of crucial evidence within a crime scene, including wearable devices like fitness trackers and smartwatches. Given its ongoing development, BLE is expected to become increasingly prevalent in daily usage.

IEEE 802.15.4 and ZigBee

The selected protocol to be sniffed in this project is ZigBee, which is based on the IEEE 802.15.4 standard. While there are other protocols that utilize this standard, such as Thread, and competing protocols like Z-Wave and Matter, ZigBee has been chosen due to its widespread adoption in the market, with devices like the Samsung SmartThings ecosystem being commonly available for validation purposes.

As will be explained in the upcoming section, the sniffer is theoretically capable to detect all frames based on the IEEE 802.15.4 standard and can analyze data obtained from the MAC layer, which is common to all protocols built on top of this standard.

5.2.2 Hardware

To successfully sniff WiFi, BLE, or IEEE 802.15.4 traffic, several requirements must be met. Firstly, a device capable of frame sniffing is necessary, such as a dedicated chip or network adapter that supports promiscuous mode. This allows the device to capture all network frames within its range, including those not intended for itself. The device must also be able to operate on the 2.4 GHz band.

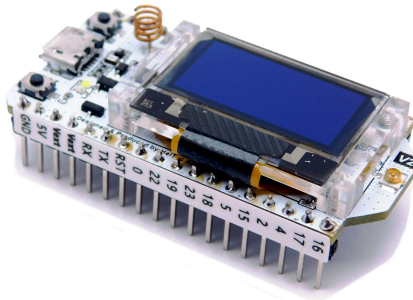
WiFi Sniffing

The use of a dedicated chip for sniffing WiFi traffic offers several advantages over utilizing the WiFi interface of a laptop. Firstly, not all computers have the capability to turn their WiFi interface into promiscuous mode. By using a dedicated chip, specifically designed for sniffing, you can ensure compatibility and avoid limitations associated with certain computer configurations. Additionally, when a laptop's WiFi interface is switched to promiscuous mode, it loses its ability to access the internet. This can be a significant drawback, especially when conducting investigations or monitoring network activity in real time. Using a dedicated chip for sniffing WiFi traffic allows you to separate the monitoring process from the laptop's internet connectivity, ensuring uninterrupted access to online resources while capturing and analyzing network packets.

To perform WiFi 2.4 GHz sniffing, we choose the *Heltec WiFi LoRa V2*¹ chip by *Heltec Automation*. The choice of this chip is justified by the following reasons. First, this chip was in our possession and is equipped with an *ESP32* microcontroller with a network interface that supports promiscuous mode in the 2.4 GHz band. Furthermore, this chip is easily programmable with *Arduino IDE*². The FIGURE 5.1 illustrates the *Heltec WiFi LoRa v2* used for WiFi sniffing.

¹<https://heltec.org/project/wifi-lora-32/>

²<https://www.arduino.cc/en/software>

FIGURE 5.1: *Heltec WiFi LoRa V2 - Heltec Automation*

BLE and IEEE 802.15.4 Sniffing

To perform BLE and IEEE 802.15.4 sniffing, the *nRF52840 Dongle*³ chip by *Nordic Semiconductors* is used. It is a powerful System-on-Chip supporting different communication protocols, which include BLE, IEEE 802.15.4, and protocols derived from it. The FIGURE 5.2 illustrates the *Nordic Semiconductors nRF52840 Dongle* used for BLE and IEEE 802.15.4 sniffing.

Furthermore, *Nordic Semiconductors* freely provides distinct firmware for sniffing BLE⁴ and IEEE 802.15.4⁵ traffic. The firmwares are easily programmable on the chip with *nRF Connect for Desktop* and *nRF Connect Programmer*. The Appendices D and E respectively contain details on how to program the BLE and IEEE 802.15.4 sniffer firmware on the chip.

Additionally, the *nRF52840 Dongle* is a USB dongle that can directly be plugged into the USB port and start sending data over a serial port.

Because the chip cannot sniff traffic from both BLE and IEEE 802.15.4, two distinct chips were required.

³<https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dongle>

⁴<https://www.nordicsemi.com/Products/Development-tools/nrf-sniffer-for-bluetooth-le>

⁵<https://www.nordicsemi.com/Products/Development-tools/nRF-Sniffer-for-802154>

FIGURE 5.2: *nRF52840 Dongle - Nordic Semiconductors*

5.2.3 Programming Language

The GUI proposed is implemented in *Python*. *Python* is a popular and versatile programming language that is well-suited for GUI development. The choice of *Python* as the used programming language is justified by the following reasons:

First of all, *Python* is easy to learn and use. *Python* has a simple and easy-to-learn syntax that makes it accessible to new programmers. This means that police officers who may not have a strong background in programming can quickly learn how to use *Python* to contribute further to the proposed GUI or to modify some parameters. Secondly, *Python* has a large community and ecosystem: *Python* has a large and active community of developers who have created a wide range of libraries and frameworks that make it easy to develop GUIs and get support. Lastly, *Python* is a cross-platform language. It can run on a wide range of operating systems, including Windows, macOS, and Linux. This is useful for police officers who work on different types of devices.

5.2.4 GUI Library

Python has several powerful GUI libraries, including *Tkinter*, *PyQt*, and *wxPython* that make it easy to create complex and interactive GUIs. *Tkinter* library as the others, provides a wide range of widgets and tools that meets the needs for a GUI sniffing IoT devices. The choice of the use of *Tkinter* library in this project is also justified by the following reasons: Firstly, *Tkinter* comes pre-installed. It is included with the standard *Python* distribution, which means that there is no need to install any additional software to use it. This makes it easy to get started with *Tkinter* and eliminates the need to deal with compatibility issues or dependencies. Secondly, *Tkinter* is platform-independent. It works on a wide range of operating systems, including Windows, macOS, and Linux, which makes it a great choice if you want to create a GUI that can be used on different

platforms. Thirdly, *Tkinter* is simple and easy to use. It has a simple and easy-to-learn API that makes it easy to create GUIs quickly. Lastly, *Tkinter* supports a wide range of built-in widgets, including buttons, labels, entry fields, and more, that can be used to create complex and interactive GUIs.

5.3 GUI Description

The Graphical User Interface (GUI) developed is designed to facilitate the identification of IoT devices based on their communication protocols at a crime scene. The GUI is organized into several frames, each serving a specific purpose.

The FIGURE 5.3 illustrates the GUI when launched. In this state, the sniffing is not (yet) started.

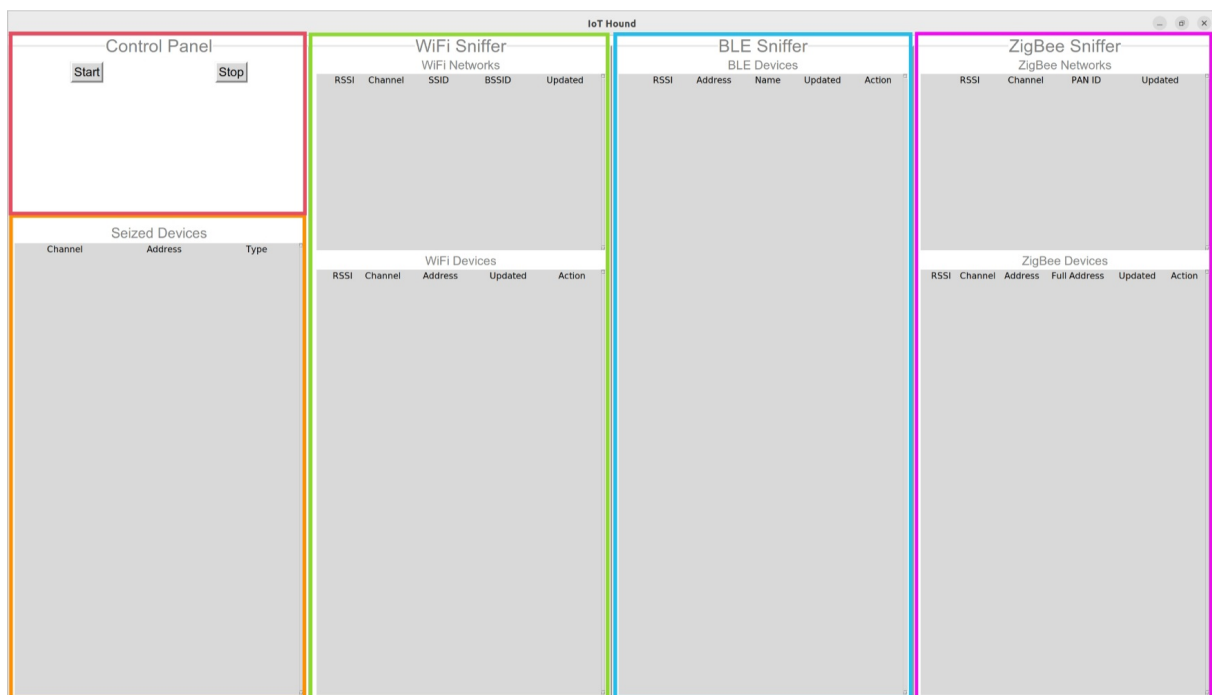


FIGURE 5.3: GUI Organization

- **Control Panel:** This frame is located in the top left corner and is the control panel of the GUI. It contains 2 buttons to **Start** and **Stop** the sniffing.
- **Seized Devices Frame:** This frame is located in the bottom left corner and keeps track of the seized devices. It gives relevant information such as the MAC address, the type of technology used by this device, and the channel on which it operated.
- **WiFi Frame:** This frame is divided into 2 subframes arranged vertically. The top subframe named **WiFi Networks** shows the WiFi networks detected by the WiFi scan. Information related to the network is provided such as the Received Signal

Strength Indication (RSSI), the channel on which the network operates, the Service Set Identifier (SSID), the Basic Service Set Identifier (BSSID), and the last update received time. The bottom subframe named **WiFi Devices** shows the WiFi devices identified matching the channel of the WiFi network selected if any. There is also information provided such as the RSSI, channel, MAC Address, and last sniffing frame received. The last column contains a button to seize the selected device.

- **BLE Frame:** This frame lists the detected BLE devices. It gives relevant information such as the RSSI, MAC Address, name (if any), and the last received frame time. The last column contains a button to seize the selected device.
- **ZigBee Frame:** This frame is organized in the same way as the WiFi frame. It is divided into 2 subframes where the top one provides information on ZigBee networks whereas the bottom one provides information on the identified device matching the channel of the ZigBee network selected if any.

5.3.1 GUI Usage Instructions

0. Configure the serial port number of the BLE, WiFi, and ZigBee sniffers correctly. Appendix A contains the recommended procedure to configure the sniffers correctly.
1. Execute the file `main.py` using *Python3*. A window identical to the **FIGURE 5.3** opens.
2. Click on the **Start** button in the *Control Panel Frame*. The sniffing of the different technologies starts and the different frames will quickly fill up with the active IoT devices in the vicinity.
 - To sniff the traffic associated with a specific network, click on the box at the left of the desired network entry. As a result, the chip will only sniff the channel matching the network entry selected.
 - To sniff all channels, just unclick the current network entry selected if any.
 - For seizing a device, click on the **Seize** button next to the desired device entry.
3. Click on the **Stop** button to stop the sniffing. Once the button is pressed, the sniffing is stopped. Because the sniffing is implemented using threads and that thread can only be launched once, the sniffing cannot be started again on press again on the **Start** button. To restart the sniffing, close the window and restart from step 1.

By following these instructions, users can effectively utilize the GUI to analyze IoT communications protocols and identify IoT devices at a crime scene investigation.

The **FIGURE 5.4** illustrates the GUI once the sniffing started. In this figure, one WiFi, one BLE, and one ZigBee devices are selected and highlighted in red. In addition, two networks are selected: the WiFi network with SSID *Proximus-Home-2B30* and the ZigBee network with PAN ID *DF3D*.

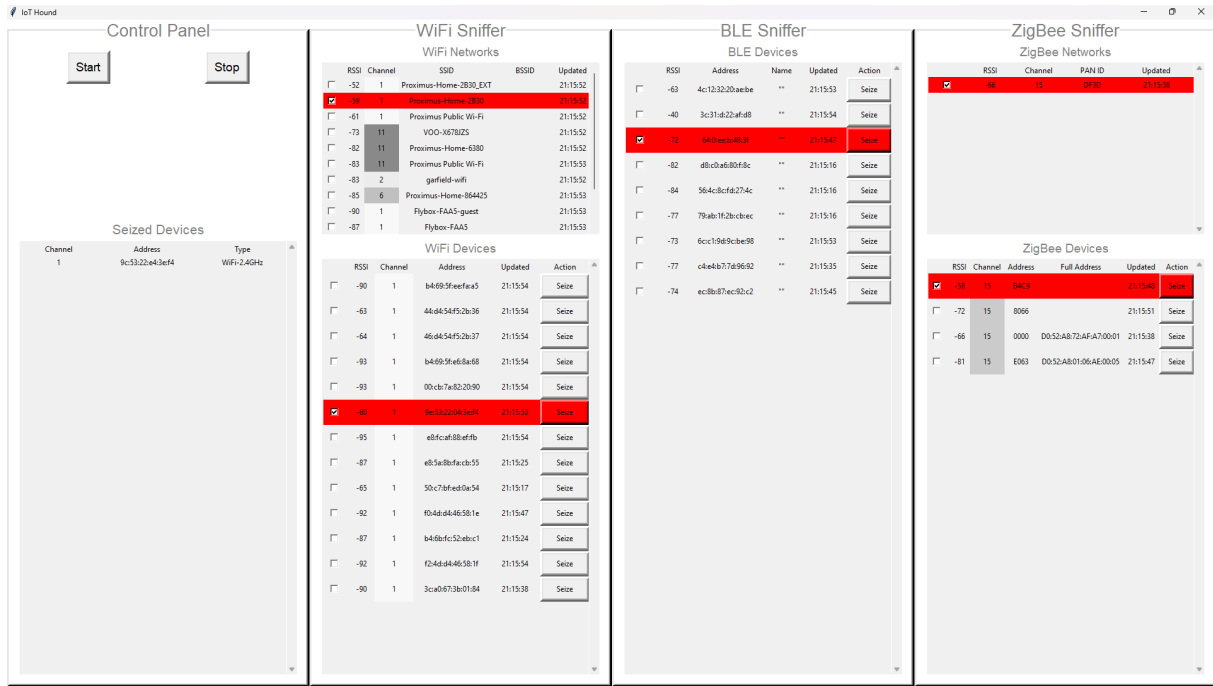


FIGURE 5.4: GUI Displaying Detected Active Devices

Appendix F contains several other figures which illustrate how our IoT identification GUI works.

5.3.2 Project Organization

The project mainly uses Object-Oriented Programming to structure the code, which is divided into different classes that represent different components. For instance, the `BLE_Sniffer` class is used to abstract the BLE sniffer whereas `ScrollableFrame` represents a distinct component of the GUI.

The project follows a modular and organized structure. It is separated into modules based on their functionality and organized into different directories. The files and directory tree are illustrated in FIGURE 5.5.

The `devices` directory contains the `device.py` and `network.py` modules. The `device.py` module defines the `Device` class which is responsible for handling the characteristics and behavior of a device. The `network.py` module defines the `Network` class which is responsible for handling network-related operations.

The `firmwares` directory contains the different firmwares to program on the chip. The filename explicitly tells on which devices it should be programmed and what is the technology targeted.



FIGURE 5.5: Overview of File and Folder Structure in the Project

The `gui` directory contains several modules responsible for building the GUI. The `gui.py` module is responsible for the creation and management of the GUI elements, such as defining the layout, creating buttons and labels, and handling user input. The file `device_gui.py` and `network_gui.py` modules are responsible for handling the display of device and network information respectively. The `scrollable_frame.py` module provides a custom widget for displaying a scrollable frame in the GUI.

The `logs` directory contains the `logger.py` module which defines the formatting and handling of the logger used for logging application events. The `logs.log` file stores those logged events.

The `sniffers` directory contains modules for different types of sniffers: `BLE_Sniffer.py`, `WiFi_Sniffer.py`, and `ZigBee_Sniffer.py`. The `Sniffer.py` module defines the abstract `Sniffer` class that is inherited by each sniffer type. Each sniffer module contains the code specific to the sniffer type and runs as a separate thread. It establishes a serial connection, reads data from the sniffer device, and updates the GUI with the detected devices and networks.

The `utils` directory contains several utility modules. All files in this directory come from the *nRF Sniffer for BLE*⁶ software and are under license. Those files are required to

⁶https://infocenter.nordicsemi.com/topic/ug_sniffer_ble/UG/sniffer_ble/intro.html

acquire BLE data from the *nRF52840 Dongle* from *Nordic Semiconductor*.

Finally, the file `main.py` is the main application file. It serves as the entry point for the GUI application. This file contains the `Tk()` instance and handles the initialization of the GUI and the sniffers.

The project is made publicly available on *GitHub*⁷. It is carefully structured and organized to prioritize maintainability, readability, reusability, extensibility, and collaboration. It follows modular design principles, adheres to consistent naming conventions, utilizes object-oriented programming concepts, and incorporates documentation through relevant comments to explain the purpose and functionality of the code. Design patterns and best practices are employed to ensure easy modification and loose coupling. Logging is implemented for error tracking, and overall, the codebase is aimed at being robust and facilitating efficient development and maintenance.

5.4 WiFi Device Identification

In the context of WiFi device identification, the WiFi sniffer performs two key functions: scanning WiFi networks and capturing WiFi frames.

Firstly, the WiFi sniffer conducts a network scan to identify the available WiFi networks in its vicinity. This scanning process allows it to detect the Service Set Identifiers (SSIDs) of nearby networks. By obtaining the SSID information, the sniffer can then determine which specific channel to monitor for capturing WiFi frames. This targeted approach helps narrow down the focus to the networks of interest and to the relevant IoT devices.

Secondly, the WiFi sniffer captures WiFi frames transmitted over the air. To intercept the WiFi frames being transmitted within its range, it either continuously monitors the selected channel or periodically changes the channel to monitor.

5.4.1 WiFi Networks Discovery

The WiFi network discovery relies on beacon frames. Beacon frames are a special type of WiFi management frames that are periodically broadcast by Access Points (APs) in order to provide information about the network. The goal of beacon frames is to allow devices in the surrounding to discover and identify WiFi networks.

Beacon frames consist of various fields that transmit specific information to devices. The most interesting for network discovery are listed below.

- RSSI: The Received Signal Strength Indicator (RSSI) is as the name implies an indication of the strength of the received signal in the antenna. Further in this work, we will use this field as an indication of the distance to the emitter.

⁷<https://github.com/igorferro1/IoT-Hound>

- **SSID:** The Service Set Identifier (SSID) is a case-sensitive alphanumeric identifier assigned to a WiFi network. It enables to identify and differentiate between different WiFi networks in the surrounding.
- **BSSID:** The Basic Service Set Identifier (BSSID) is a unique identifier assigned to a wireless access point in a WiFi network. It is a 48-bit value, typically represented as a series of six pairs of hexadecimal digits. In a wireless network, multiple access points may be present, especially in larger environments to provide adequate coverage. The BSSID enables to differentiate APs with the same SSID.
- **Beacon Interval:** The time between successive beacon frame broadcasts.

The exact timing of beacon transmissions can be configured in the access point's settings, typically ranging from 200 milliseconds to 300 milliseconds. In the writing of the firmware, we decided to listen for at most 500 ms every channel for a beacon frame. Because there are 13 channels, the time required to perform a network scanning is 6.5 s. Some APs are configured such that it changes of transmitting channel if interference on this channel is detected. For this reason, the code is designed to periodically trigger the network scan, between 30 and 35 seconds. This scanning frequency is reasonable to have an updated view of the channel used by the existing WiFi networks but also to let the chip perform frame sniffing.

To collect WiFi network information, a passive scanning approach was used. This approach consists of listening for beacon frames that are already being broadcast by nearby networks. Another approach could be active scanning: it involves actively sending out probe requests to WiFi networks and then waiting for responses from nearby WiFi networks to gather information. Even if passive scanning may not collect as much information as active scanning, this approach was preferred. This choice is justified because this method does not actively send out requests or interact with networks, which makes it less likely to disrupt the networks being scanned. Additionally, using passive network scanning avoids altering the network environment and creating additional traffic. This helps to ensure the integrity of the evidence obtained, increasing its admissibility in legal proceedings.

Because the WiFi network scan can take several seconds to complete, depending on the number of networks in the area and the scanning parameters used, it is running asynchronously. This means that it will not block the execution of any code that comes after it until the scan. This frees up time to perform WiFi frame sniffing or to read the data available on the serial port.

5.4.2 WiFi Frames Sniffing

Besides WiFi networks discovery, the provided code is also able to sniff WiFi frames transmitted on the 2.4 GHz band. The WiFi sniffing functionality can operate in two distinct modes: specific channel sniffing or channel hopping.

By default, the channel hopping mode is the one in which the firmware is running. In this mode, one channel is monitored at a time for a random period of time between 500 and 700 ms. The randomization added helps to capture a broader range of network activity and reduces the likelihood of missing important information that is transmitted periodically. After the period expires, the chip automatically switches to the next channel. It follows a cyclical pattern, where each channel is monitored sequentially, and once the last channel is reached, the monitoring cycle starts again from the first channel, creating a continuous loop of channels sniffed from 1 to 13. By frequently switching between the different channels, the overview of the overall WiFi landscape as well as variations in network activity across different channels can be observed. However, if a frame is transmitted on channel x while the chip is monitoring channel y , then the frame would not be sniffed.

The specific channel sniffing mode focuses on capturing WiFi frames from a single predefined channel. This mode is particularly useful when the investigator already has an idea of which channel IoT devices should communicate. For example, if the domestic WiFi network operates on channel 3, then focusing the WiFi frames capture on this specific channel is recommended (under the assumption that IoT devices cannot communicate to the internet without relying on an AP). The advantage of this mode is to continuously monitors and captures WiFi frames transmitted on a specific channel.

5.4.3 IEEE 802.11 Frames

Understanding the structure of IEEE 802.11 frames is essential for analyzing captured frames and extracting relevant data for further investigation. By examining the specific fields within the frame, such as the sender and destination addresses, investigators can identify the source and destination of network communications, contributing to the overall forensic analysis process. There are three distinct types of frames that can be captured based on the IEEE 802.11 frame exchange protocol.

- **Management Frames:** Management frames are employed to establish the initial communication between a device and the access point (in infrastructure mode) or between stations (in ad hoc mode), and subsequently maintain the connection. Accomplishing these tasks necessitates multiple exchanges of frames. Several types of management frames serve specific purposes in a wireless network. These include:
 - Authentication Frames: which determine access point acceptance or rejection;
 - Association Request Frames: used for resource allocation;
 - Association Response Frames: indicating acceptance or rejection by an access point;
 - Beacon Frames: regularly transmitted by access points to announce their presence and provide wireless information;
 - Deauthentication Frames: terminating communications;

- Disassociation Frames: ending connections;
- Probe Request Frames: seeking information;
- Probe Response Frames: providing details in response to probe requests;
- Reassociation Request Frames: facilitating device movement between cells;
- Reassociation Response Frames: conveying acceptance or rejection notices from access points to requesting devices;

The FIGURE 5.6 illustrates the format of a management frame. Information like the current version number of the standard and whether encryption is being used can be found in the “Frame control” field. The duration of the transmission is specified in the “Duration” field; the value will change depending on the type of wireless transmission being used. The addresses of the receiving and sending devices are listed in the “Address 1 (Destination address)” and “Address 2 (Source address)” fields, respectively. The sequence number for the packet and packet fragment number are contained in the “sequence control” field. [35]

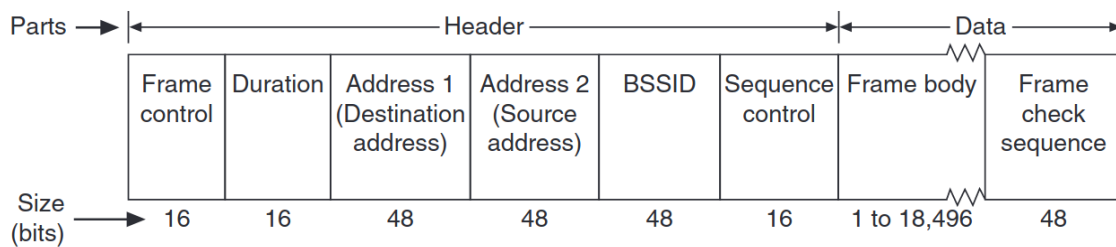


FIGURE 5.6: IEEE 802.11 Management Frame Format [35]

- **Control Frames:** The second type of MAC frames are the control frames. They are responsible for requesting and controlling access to the wireless medium. They help in delivering frames that contain the data after the connection between the stations and AP is established. There are more than thirteen other types of control frames, each with its own format. Among them are:
 - Request-to-Send (RTS) Frames: used by a sender to reserve the wireless medium before transmitting data frames. Its format is illustrated in FIGURE 5.7.
 - Clear-to-Send (CTS) Frames: sent by the receiver in response to an RTS frame;
 - Acknowledgement (ACK) Frames: sent by the receiver to acknowledge successful receipt of a data frame;

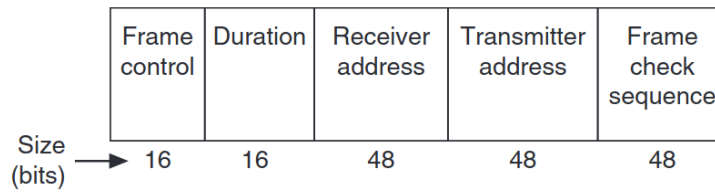


FIGURE 5.7: IEEE 802.11 RTS Control Frame Format [35]

- **Data Frames:** The last type of MAC frames are the data frames. They serve the purpose of encapsulating packets from upper-layer protocols, such as IP. The data frame format is illustrated in FIGURE 5.8. The address fields (Address 1, Address 2, Address 3, and Address 4) contain the address of the BSSID, the destination address, the source address, the transmitter address, or the receiver address. Their contents vary, depending upon the mode of transmission.

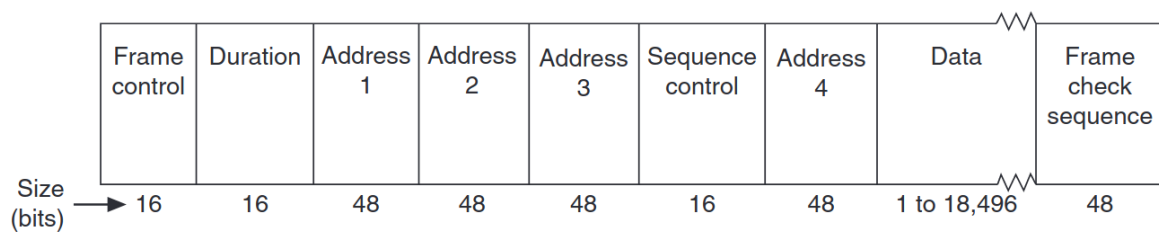


FIGURE 5.8: IEEE 802.11 Data Frame Format [35]

The TABLE 5.1 summarizes the meaning of the address fields for the different values of the field “to SD” and “to DS”. In this table, the sense of the abbreviations is as follows:

- DA: the ultimate Destination Address (DA)
- RA: the Receiver Address (RA)
- SA: the initial Source Address (SA)
- TA: the Transmitter Address (TA)
- BSSID: Basic Service Set Identifier (BSSID)
- IBSS: Independent Basic Service Set (IBSS)

To DS	From DS	Meaning	Address 1	Address 2	Address 3	Address 4
0	0	Station (STA) to Station communications in IBSS mode	RA=DA	SA	BSSID	/
0	1	Downstream traffic from AP to STA	RA=DA	BSSID	SA	/
1	0	Upstream traffic from STA to AP	RA=BSSID	SA	DA	/
1	1	AP to AP communications, when Wireless Distribution System (WDS) in use	RA	TA	DA	SA

TABLE 5.1: IEEE 802.11 address fields meaning function of the subfields “To DS” and “From DS” [36] [37]

5.4.4 Information of Interest to Collect

When sniffing frames to locate IoT devices, the MAC address of the device that generated and sent the frame is valuable information to identify the presence of IoT devices communicating. If this information can be combined with an indication of the distance from the receiver, then police officers would be able to locate the IoT device.

If the sniffed frame is a control frame, the transmitter address might be present inside the “Transmitter Address” (TA) field. However, this field is not contained in every type of control frame. The FIGURE 5.9 below illustrates the format of a CTS control frame. There is no “Transmitter Address” contained in this control frame type.

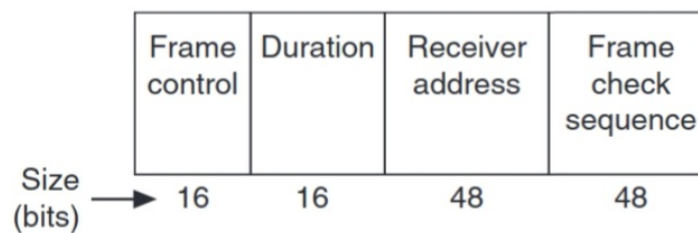


FIGURE 5.9: IEEE 802.11 CTS Control Frame Format

If the sniffed frame is a management frame, the transmitter address is found in the “Source Address” (SA) field.

If the sniffed frame is a data frame, the MAC address of the device that is transmitting the data frame is found in the “Address 2” field. The “Address 2” field meaning is as

described in Table 5.1. In the four scenarios, the “Address 2” field contains information related to the device that generated and sent the control frame.

- In the situation of STA to STA communications, the “Address 2” field contains the Source Address (SA) of the STA transmitting the frame.
- If the frame is a downstream frame from AP to STA, the “Address 2” corresponds to the BSSID of the AP sending the frame. The BSSID is typically the MAC address of the AP emitting the frame.
- If the frame is an upstream frame from STA to AP, the “Address 2” field contains the Source Address (SA) which corresponds to the STA address transmitting the data frame.
- In the situation of AP to AP communications, the “Address 2” field contains the address of the AP transmitting the frame.

Besides the IEEE 802.11 Medium Access Control layer, the IEEE 802.11 physical (PHY) layer also contains information relevant to the identification of IoT devices. Especially, the primary channel on which the data is transmitted. This information tells on which channel the frame sniffed was transmitted. It could help to differentiate the frame sent by the smart ecosystem of the neighbor from others sent by the smart ecosystem of interest.

Additionally, the Received Signal Strength Indication (RSSI) is really valuable. As the name implies, it gives an indication of the signal strength received. This information can be used to help police investigators to determine the location of an IoT device. The stronger the signal, the closer the investigators get to the emitter.

5.4.5 Firmware

Programming the Heltec ESP32 chip was achieved using the [Arduino IDE](https://www.arduino.cc/en/software)⁸ and the [Heltec ESP32 Arduino Development Environment](http://www.heltec.cn/wifi_kit_install/)⁹ provided by the manufacturer. The installation instructions are available online and the links are given in footnotes. The sniffer was implemented by modifying the example codes provided in the libraries in order to both discover WiFi networks¹⁰ and to sniff WiFi frames¹¹ transmitted over the 2.4 GHz frequency band. The whole code is provided in Appendix B.

The FIGURE 5.10 illustrates the different data structures used in the implementation of the firmware. The first line of each box corresponds to the meaning of the structure whereas the second line is the exact name of the structure used in the code. By programming the chip using these structures, the relevant fields are easily accessed, allowing the chip to capture relevant information.

⁸<https://www.arduino.cc/en/software>

⁹http://www.heltec.cn/wifi_kit_install/

¹⁰<https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi/examples/WiFiScan>

¹¹<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/>

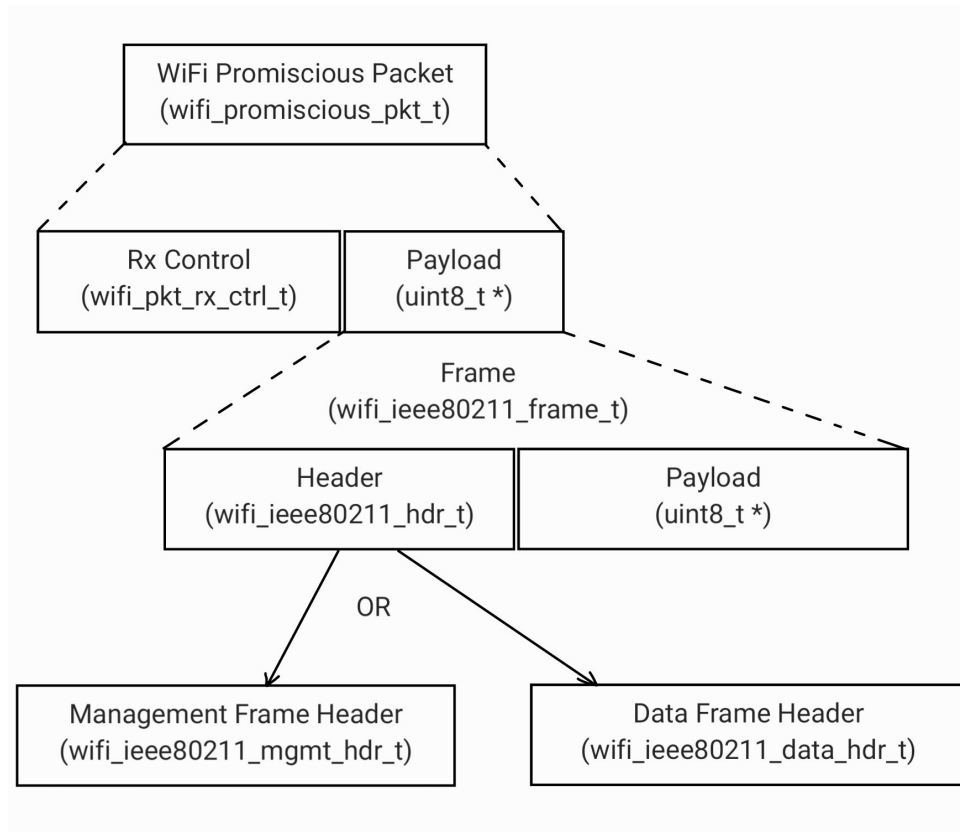


FIGURE 5.10: WiFi Sniffing Firmware Data Structures

5.4.6 Integration in the GUI

To integrate the communication with the ESP32 chip into the GUI, a USB serial connection is utilized to receive the data from the chip, which is then parsed.

The data format chosen for communication consists of two different formats depending on the type of detection: host detection and network detection.

For host detection, the data format follows the pattern: “H, channel, RSSI, address”. Here, “H” represents the host detection identifier, followed by the channel number on which the host was detected. The received signal strength indicator (RSSI) value indicates the signal strength of the detected host, and the address corresponds to the MAC address of the host device.

For network detection, the data format follows the pattern “N, channel, RSSI, SSID, BSSID”. The “N” identifier denotes a network detection, followed by the channel number of the detected network. The RSSI value indicates the signal strength of the network, and the SSID represents the name of the network. Additionally, the BSSID refers to the MAC address of the access point broadcasting network information.

By parsing the received data based on these predefined formats, the GUI can effectively extract the necessary information about the active WiFi devices and networks in the vicinity. This allows it to display relevant details such as channel, signal strength, MAC address, SSID, and BSSID, providing investigators with valuable insights into the WiFi environment and the devices present within it.

5.5 Bluetooth Low Energy Devices Identification

5.5.1 BLE Devices Discovery

The Bluetooth Low Energy (BLE) device identification relies on beacon frames. BLE beacon frames are a specific type of advertising frame used in BLE technology. They serve as a means of broadcasting information about a device or a particular location to nearby BLE devices.

When a device sends advertising frames, other BLE devices that are scanning can detect these beacon frames and identify the presence and proximity of the advertising device. This allows for efficient device discovery of BLE devices.

In practice, advertising messages are sent on three advertising channels: channel 37, channel 38, and channel 39. These channels are in the 2.4 GHz ISM (Industrial, Scientific, and Medical) band and are spaced 2 MHz apart. The advertising packets are transmitted to at least one of the three advertising channels, with a repetition period called the advertising interval. For reducing the chance of multiple consecutive collisions, a random delay of up to 10 milliseconds is added to each advertising interval. The use of multiple advertising channels reduces the likelihood of message collision and lost data.

It also enables the receiver to detect advertising packets from a BLE device even if one or more channels are affected by interference or signal attenuation. To detect BLE devices, a scanner passively listens to advertising packets on all three channels for a duration called the scan window, which is periodically repeated every scan interval.

5.5.2 Information of Interest to Collect

BLE advertisement frames contain interesting information about the device such as the MAC address of the device that generated and sent the frame. This is valuable information that can be combined with the received signal strength, also contained in the beacon frame, to help police investigators to locate the IoT device advertising. The name might also be present in the beacon frame. Typically, the name is the model and or the brand of the device emitting the beacon frame. This information can guide police officers in their research of the device that is advertising.

5.5.3 Firmware

Nordic Semiconductors provides firmware for sniffing BLE frames. This firmware is included in the project under the `firmwares` directory. Its name is `sniffer_nrf52840dongle_ble.hex` and it has to be programmed onto the dongle. The instructions on how to do so are given in Appendix D and are available online¹². Programming the *nRF52840 Dongle* requires special actions to be performed that are described in the Appendices C or online¹³. The *nRF Connect for Desktop*¹⁴ and the *nRF Connect Programmer*¹⁵ are also required.

5.5.4 Integration in the GUI

The provided firmware is designed to be used in combination with a capture plugin for *Wireshark* that will record and analyze the detected data. To interact with their firmware, *Nordic Semiconductor* developed a **Sniffer** API and added it to the *Wireshark* capture plugin. To communicate with the chip from our own GUI, we analyzed the behavior of this **Sniffer** API. Based on this analysis, we retrieve the information to send on the serial to the chip to start the sniffing and we also deduced how to interpret the data responded to by the sniffer.

The class `BLE_Sniffer` contains the implementation of the BLE Sniffer. The function `setup_scan` sends the appropriate signal to set the chip into a scanning state whereas the function `run` continuously listens to the serial port for data from the chip. The files needed for the interaction with the chip are put inside the `utils` folder and come from the provided **Sniffer** API.

5.6 ZigBee Devices Identification

In the context of ZigBee device identification, the ZigBee sniffer performs two key functions: scanning existing ZigBee networks and capturing ZigBee frames.

Firstly, the ZigBee sniffer conducts a network scan to identify the available networks in its vicinity. This scanning process allows it to detect the devices of nearby networks. This way, the sniffer can determine which specific channel to monitor for capturing ZigBee frames, only on the ones in which there are devices communicating. This targeted approach helps narrow down the focus to the networks of interest and to the relevant IoT devices.

¹²https://infocenter.nordicsemi.com/topic/ug_sniffer_ble/UG/sniffer_ble/programming_firmware.html

¹³https://infocenter.nordicsemi.com/topic/ug_nc_programmer/UG/nrf_connect_programmer/ncp_programming_dongle.html

¹⁴https://infocenter.nordicsemi.com/topic/struct_nrftools/struct/nrftools_nrfconnect.html

¹⁵https://infocenter.nordicsemi.com/topic/ug_nc_programmer/UG/nrf_connect_programmer/ncp_introduction.html

Secondly, the ZigBee sniffer captures IEEE 802.15.4 frames transmitted over the air on the respective channels that have been detected in the previous step. To intercept the frames being transmitted within its range, it either continuously monitors the selected channel, or if no channel is selected, it periodically changes between the aforementioned channels.

The sniffer was specifically designed and validated for ZigBee devices, however, in theory, it should also be compatible with other protocols that are built on top of the IEEE 802.15.4 standard, such as Thread. But for the sake of consistency, in this section, it will be referred to as ZigBee sniffing only.

5.6.1 IEEE 802.15.4 Network Discovery

First, as mentioned in the section 2.2.4, IEEE 802.15.4 operates on 16 different channels. This poses a small problem for the sniffer because the nRF device cannot be on all the channels at the same time. The solution for this can be simple: the sniffer can keep hopping between the channels, stay in each channel for a while detecting for devices, and then go to the next channel. This is the simplest way to resolve this but it is highly inefficient because, for each hub present, ZigBee operates in a single frequency in the MAC layer [38], therefore it is very rare that there are multiple devices inside a crime scene, each one communicating in different channels, and that all of the channels are used at the same time. Consequently, it would be better to have a way to detect which channels are being used and only sniff those channels.

But there is a challenge in doing this: what is the most efficient way to check if a channel is being used? In WiFi, we have beacon frames, but looking for those is not a very efficient approach in IEEE 802.15.4 technologies because they are not always sent, to save bandwidth and power. So in this scenario, it is not an optimal way to look for announcement messages. To understand how this could be done, we analyzed the channels with the nRF, going manually through them one by one, and from this, we noticed that there is a constant data flow in the channels that have devices on them. It is not uncommon to have data requests, data messages, or acknowledgments present in these cases. Therefore, it would be possible to check if a channel has data just by listening to it and seeing if there is a data flow, no matter what is the type of message, and with this, there is no need to check for announcement messages from the hub. But this also poses another question: how should the scanning be done? Which channel should be checked at which moment?

Inherently, channel hopping is necessary because there is no way at checking multiple channels in parallel, but for how long the device should stay in each channel is an interesting question, but with an expected answer: this depends on the intensity of the data flow, so that the device doesn't change channels too soon, before obtaining the data traffic, but it shouldn't stay for so long, wasting more time in empty channels. From empirical measurements with devices, we noticed that there is a considerable data flow in 10 seconds, so it would be hard to miss a channel while staying 10 seconds in it. This is

a variable that can be easily changeable by the user of the program in case they think it is necessary. This combination proved itself good to detect the channels that have data flow, as will be shown in the tests.

It has been explained how the scanning is done, but what happens during and after the scanning and the information obtained from it is equally important. For each channel, during the stay on it and listening to the messages, two possible paths can be taken by the software:

1. **There is data on that channel:** In this case, the device will capture data, the software will check if it's actually a ZigBee frame, and then add that channel to a list of channels to sniff and hop to the next channel.
2. **There is no data on that channel:** Within this scenario, the device will keep waiting some time in one channel, 10 seconds, before just going to the next channel, concluding that there is no data flowing on this one.

This procedure takes around 3 minutes for scanning all channels, which is considered an acceptable amount of time before starting the sniffing. It can be repeated if considered to be necessary, and this can be set on the program as well. If there is no ZigBee traffic detected, the program will stay in this phase, hopping through the channels and looking for data.

After the execution of this part of the process, the output will be a list of channels that have traffic on them. This list will be passed to the second part, the sniffing of the frames, so that only those channels are checked for the presence of data, making the sniffer more efficient.

5.6.2 IEEE 802.15.4 Frames Sniffing

As mentioned, the output of the previous step is a list of channels that contain data flow through them. This data is useful in this phase because those channels are the ones that will be listened to. Having that list in hand, then the software will start sniffing. The sniffing process is relatively simple: the software sends a serial command to start receiving the data on the first channel on the list. Since the sniffer is already configured with the firmware, it will just start sniffing on that channel and sending the data serially to the program. After a while, it changes the channel by clearing the buffer to avoid delayed data being attributed to the next channel, and it starts listening to the next channel in the list.

The most important part is what comes after the sniffing: the processing of the data. The entire serial message is composed of 4 parts, as shown in [FIGURE 5.11](#):

```

> Type and press enter to send
Send

> channel 15
channel 15
> receive
receive
received: 4188FD3DDFFFFF00000912FCFF000001CB0100A7AF72A852D028332BB2000100A7
AF72A852D000C9F9BDA145BD0E00 power: -79 lqi: 56 time: 77042470
received: 6388243DDF00006680042100 power: -59 lqi: 132 time: 80634234
received: 0200240CDF power: -80 lqi: 48 time: 80634777
received: 6388993DDF000063E0042A00 power: -51 lqi: 168 time: 80802366
received: 0200990DDF power: -80 lqi: 52 time: 80802909
received: 61889A3DDF000063E00802000063E01EEF2829B205000500AE0601A852D0000BEF
33F9FC5277A70656EC9294A700D2902A00 power: -51 lqi: 168 time: 80807042
received: 02009A0CDF power: -81 lqi: 48 time: 80807585
CLEAR CONSOLE

```

FIGURE 5.11: Terminal with Serial Connection with the Device and Data Acquisition

- **The Received Message:** This is a hexadecimal representation of the entire raw ZigBee frame.
- **The Power:** The RSSI value, which indicates the strength of the signal with the sniffer in that position. It is measured in dBm.
- **The LQI:** This is the Link Quality Indicator. In ZigBee, it indicates the quality of the link in which this frame was sent.
- **The Time:** This is the representation in microseconds of the moment in which that frame was captured, beginning from when the nRF device was connected.

The most important data for the sniffer in this are the received message and the RSSI. The message is important because, clearly, it means that there is at least one device exchanging messages in that location, and we might be able to extract the source address(es) from that message. The RSSI is important because it can help determine the distance from the device to the sniffer, and it is a piece of information that will be displayed to the user.

After receiving the data from the serial port, the software proceeds to save locally only the frame and the RSSI. It will then convert the message to binary in order to find control bits which will indicate the presence of one or more address(es) in the message. If the message doesn't contain any of the addresses, it will be discarded.

It is worth noting that the sniffer does this capture for every channel in the list of detected channels, and it is not simultaneous because, as mentioned multiple times, the sniffer can only be configured for one channel at a time. This means that it also does channel hopping, but instead of being only 10 seconds in each channel as in the channel discovery phase, it stays between 20 and 30 seconds, with this value being random for each time it runs. This chosen period is because the duration of each channel stay should be different, in a way that recurrent messages should be captured. If it stays, for example, 30 seconds in each channel every time, a message that is sent recurrently on one channel might be lost because it's always sent when the sniffer is listening to the other channel. With

this randomness, it won't be "looped" to go to a certain channel at a certain time only, breaking the regularity and being able to capture those "out-of-time" frames.

With this sniffing and data processing composition, the data is collected, analyzed, and passed to the GUI in order to show it in a simple way to the investigators, being updated regularly.

5.6.3 IEEE 802.15.4 Frames

For this work, we'll try to obtain the source address from the device that is sending the frame. This will be further explained in section 5.6.5. One of the most important aspects is that IEEE 802.15.4 is defined to be a lightweight protocol, and one of its properties is that it tries to optimize communication with a low overhead, this means that it contains a considerably small header. The FIGURE 5.12 shows the frame for IEEE 802.15.4, on the MAC layer. It omits a footer at the end with the checksum of the frame.

802.15.4 MAC header						MAC payload
Octets:2	1	0/2	0/2/8	0/2	0/2/8	variable
Frame Control	Sequence number	Destination PAN ID	Destination address	Source PAN ID	Source address	Frame payload
Bits: 3	1	1	1	1	3	2
Frame Type	Security enabled	Frame pending	ACK required	Pan ID	Reserved	Dest addr mode
						Frame Version
						Src addr mode

FIGURE 5.12: IEEE 802.15.4 MAC Header Frame Format [39]

The most important fields to analyze in this header are "Frame Control", "Source address" and "Frame Payload". The frame control field contains flags of the frame, which functionality includes defining the type of the frame and also which fields it has.

Inside the "Frame Control" field, the important subfields will be "Frame Type", "Security Enabled", "Source addressing mode", "PAN ID", and "Destination addressing mode". The "Frame Type" can indicate if it's a beacon, an acknowledgment (ACK), a data frame, or a command frame. The "Security Enabled" bit will indicate if the frame is cryptographically protected by the MAC layer. It is necessary that this bit is set to 0 in order to be able to explore the data fields in a further way, without encryption [40]. The "Source addressing mode" indicates if it will use the short address for the source (if this field is 10), the extended address (if this field is 11), or no address at all (if this field is 00, and for example, it's not necessary to include an address for an ACK frame). The other mentioned fields won't have the data used, but it is necessary to check them because they influence the positioning of the "Source address" field in the number of bytes. In our case, "PAN ID" must be set to 1 and "Destination addressing mode" set to 10.

The "Source address" field is important to acquire the address of the sending device. If present, this address can be in one of two forms: an intra-network unique 16-bit address

generated by the coordinator when the node joined the network (called the short address), or a 64-bit address entirely unique to that device, which was defined in the manufacturing of the device (called the extended or IEEE address). The source address is the one that will be obtained by the sniffer to show the presence of a device, which will be explained later, but at this moment, it is only interesting to obtain the short address.

The “Frame Payload” is where the upper layers and the data will be. This is important because the further layers of ZigBee and their headers will be in this field.

Now knowing about the structure of an IEEE 802.15.4 frame and its important fields of the headers, it is necessary to understand the ZigBee frame and headers.

5.6.4 ZigBee Frames

As mentioned before, ZigBee is built on top of IEEE 802.15.4, and the header for the MAC layer is the same as described in the previous section. But ZigBee also has layers on top of the MAC one, and that will include headers with information. The FIGURE 5.13 shows the stack of a ZigBee frame.

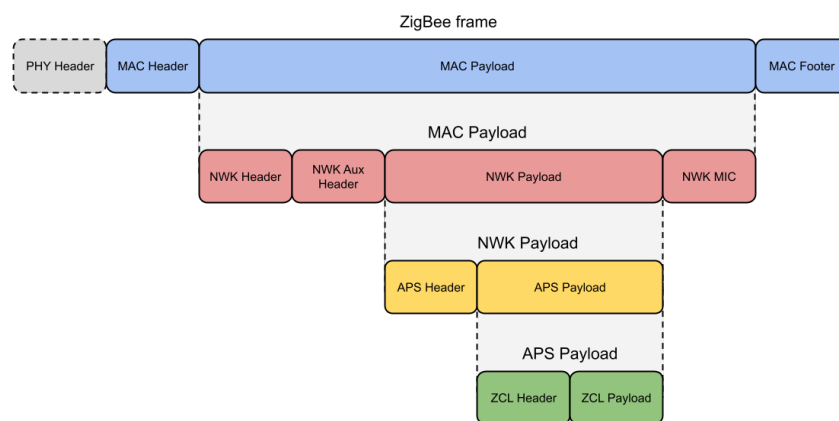


FIGURE 5.13: ZigBee Frames Stack [41]

This shows that apart from the MAC header, which contains retrievable information, we still have a network and auxiliary headers (NWK layer), and an application header (APS layer). The network layer will be the most important to analyze in this step. The fields of the network frame are shown in the following figure:

Octets: 2	2	2	1	1	0/8	0/8	0/1	Variable	Variable
Frame control	Destination address	Source address	Radius	Sequence number	Destination IEEE Address	Source IEEE Address	Multicast control	Source route subframe	Frame payload
NWK Header									Payload

FIGURE 5.14: ZigBee Network Frame Format [42]

It is important to note that this enhanced analysis is only done if the short source address was already present before, but if it's possible to obtain the IEEE address as well it's better because those addresses are unique to each device, so this is why it's notable to continue going deeper into the layers to analyze them. The necessary fields here are the "Frame Control" and the "Source IEEE Address". The "Frame Control" is similar to the homonymous field on the previous layer, and it contains control bits. It is represented in FIGURE 5.15.

Bits: 0-1	2-5	6-7	8	9	10	11	12	13	14-15
Frame type	Protocol version	Discover route	Multicast flag	Security	Source Route	Destination IEEE Address	Source IEEE Address	End Device Initiator	Reserved

FIGURE 5.15: Frame Control Format [42]

The bits of interest here are "Source IEEE address" and "Security". The "Source IEEE address" indicates if the 64-bit extended address will be present in the header (if its value is 1), and "Security" indicates that the auxiliary frame header will be present (also if its value is 1), which gives another opportunity to obtain the IEEE extended address. If the extended address is present, it can be obtained by the sniffer in this step by checking the bits of the "Source IEEE address" field on the network header.

In case the 64-bit address is not present, the sniffer will detect if the auxiliary frame header is present and analyze it. The FIGURE 5.16 represents the auxiliary frame header format.

Octets: 1	4	0/8	0/1
Security control	Frame counter	Source address	Key sequence number

FIGURE 5.16: Auxiliary Frame Header Format [42]

This is the last header analyzed by the program. Inside it, the “Security control” field will be analyzed, which is represented in FIGURE 5.17. The “Source address” field can contain the 64-bit desired IEEE address.

Bit: 0-2	3-4	5	6-7
Security level	Key identifier	Extended nonce	Reserved

FIGURE 5.17: Security Control format [42]

Inside this field, the only bit that is checked is the “Extended nonce” sub-field. If this bit is 1, then the “Source address” field of the header will be the extended address, making it possible for the program to obtain it in this step.

The application layer won’t be analyzed by the program, one reason is that when the auxiliary header is present, it usually means that the application layer is encrypted. In the tested scenarios, this was true for the whole communication.

An example showing and highlighting the analyzed bits is present in the following step, in which the obtained data is now parsed and passed to the program to show it to the user.

5.6.5 Information of Interest to Collect and Examples

The general idea for this sniffer is to obtain the source address(es) of the sniffed frames, which is shown together with the received signal strength, alongside the channel in which the sniffing is happening and the time of the last sniffed frame. This demonstrates that there is a unique present device in the scene. Obtaining these values posed a challenge because it is necessary to extract the data from the raw frames, but it was possible by parsing the frames based on their structure, seen previously, and will be demonstrated in the following part.

ZigBee Frame Parsing and Examples

With the knowledge of the structure of ZigBee frames explored in the previous subsection, now it is possible to obtain the data from the frames by looking at the aforementioned fields in the structure. In this part, we will demonstrate a few examples of ZigBee frames that are parsed to obtain important data. This will be shown through a manual analysis on Wireshark, but the sniffer does this automatically. For the sake of simplicity, only 3 types of frames will be shown in this step, all of them containing at least one type of source address, that is collected by the software.

The first type of frame that is analyzed is the simpler one, with only the short address. It is represented in FIGURE 5.17. This first highlighted line shows the “Source addressing mode” bits, which are set to the “Short” option, and the second highlighted line demonstrates the bits of this address.

```

Frame 1: 10 bytes on wire (80 bits), 10 bytes captured (80 bits) on interface COM7, id 0
IEEE 802.15.4 Command, Dst: 0x0000, Src: 0xb4c9
  ▾ Frame Control Field: 0x8863, Frame Type: Command, Acknowledge Request, PAN ID Compression:
    .... .011 = Frame Type: Command (0x3)
    .... .0... = Security Enabled: False
    .... .0... = Frame Pending: False
    .... .1. .... = Acknowledge Request: True
    .... .1.. .... = PAN ID Compression: True
    .... .0... .... = Reserved: False
    .... .0... .... = Sequence Number Suppression: False
    .... .0. .... = Information Elements Present: False
    .... 10.. .... = Destination Addressing Mode: Short/16-bit (0x2)
    .... 00... .... = Frame Version: IEEE Std 802.15.4-2003 (0)
    10..... = Source Addressing Mode: Short/16-bit (0x2)
Sequence Number: 96
Destination PAN: 0xdf3d
Destination: 0x0000
Source: 0xb4c9
Command Identifier: Data Request (0x04)

```

FIGURE 5.18: Parsing of the First ZigBee Frame Type

On this second frame, represented in the FIGURE 5.19, we have a frame with the extended address present in the Network Layer. These bits that indicate the presence of the longer address are highlighted in the figure, along with the short and extended addresses themselves.

```

> Frame 105: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface COM7, id 0
  ▾ IEEE 802.15.4 Data, Dst: Broadcast, Src: 0x0000
    > Frame Control Field: 0x8841, Frame Type: Data, PAN ID Compression, Destination Addressing Mode: Short/16-bit,
      Sequence Number: 218
      Destination PAN: 0xdf3d
      Destination: 0xffff
      Source: 0x0000
      [Extended Source: Physical_72:af:a7:00:01 (d0:52:a8:72:af:a7:00:01)]
      [Origin: 9]
    ▾ ZigBee Network Layer Command, Dst: Broadcast, Src: 0x0000
      ▾ Frame Control Field: 0x1209, Frame Type: Command, Discover Route: Suppress, Security, Extended Source Command
        .... .01 = Frame Type: Command (0x1)
        .... .00 10.. = Protocol Version: 2
        .... 00.. .... = Discover Route: Suppress (0x0)
        .... .0... .... = Multicast: False
        .... .1. .... = Security: True
        .... .0.. .... = Source Route: False
        .... 0... .... = Destination: False
        .... 1..... = Extended Source: True
        ..0. .... = End Device Initiator: False
      Destination: 0xffff
      Source: 0x0000
      Radius: 1
      Sequence Number: 75
      Extended Source: Physical_72:af:a7:00:01 (d0:52:a8:72:af:a7:00:01)
    > ZigBee Security Header
    > Data (2 bytes)

```

FIGURE 5.19: Parsing of the First ZigBee Frame Type

The third frame, in the FIGURE 5.20 represents the last type of analyzed frame, with the extended address present in the Security Header. Again, the highlighted values indicate the addresses and the bits that indicate the presence of the extended one. It is worth

noting that while Wireshark shows the “Extended Source” field under the Network Layer header, it is actually not there, and it shows the value from the Security Header.

```

> Frame 103: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface COM7, id 0
▼ IEEE 802.15.4 Data, Dst: 0x0000, Src: 0xe063
  > Frame Control Field: 0x8861, Frame Type: Data, Acknowledge Request, PAN ID Compression, De
    Sequence Number: 5
    Destination PAN: 0xdf3d
    Destination: 0x0000
    Source: 0xe063
    [Extended Source: Physical_01:06:ae:00:05 (d0:52:a8:01:06:ae:00:05)]
    [Origin: 7]
  ▼ ZigBee Network Layer Data, Dst: 0x0000, Src: 0xe063
    ▼ Frame Control Field: 0x0200, Frame Type: Data, Discover Route: Suppress, Security Data
      .... ..00 = Frame Type: Data (0x0)
      .... ..00 10.. = Protocol Version: 2
      .... ..00 00.. = Discover Route: Suppress (0x0)
      .... ..0 .... = Multicast: False
      .... ..1. .... = Security: True
      .... ..0.. .... = Source Route: False
      .... ..0... .... = Destination: False
      .... ..0 ..... = Extended Source: False
      .... ..0. .... = End Device Initiator: False
    Destination: 0x0000
    Source: 0xe063
    Radius: 30
    Sequence Number: 200
    [Extended Source: Physical_01:06:ae:00:05 (d0:52:a8:01:06:ae:00:05)]
    [Origin: 7]
  ▼ ZigBee Security Header
    ▼ Security Control Field: 0x28, Key Id: Network Key, Extended Nonce
      ...0 1... = Key Id: Network Key (0x1)
      ...1..... = Extended Nonce: True
    Frame Counter: 251870
    Extended Source: Physical_01:06:ae:00:05 (d0:52:a8:01:06:ae:00:05)
    Key Sequence Number: 0
    Message Integrity Code: 653ef384
    > [Expert Info (Warning/Undecoded): Encrypted Payload]
  > Data (13 bytes)

```

FIGURE 5.20: Parsing of the Third ZigBee Frame Type

From this analysis, we can see that it is possible to obtain the addresses from the sniffed frames, even for different frame formats. Therefore, it was necessary to build a program to sniff those frames and parse them.

5.6.6 Firmware

Just like described in the BLE section, the device can be programmed to be used as an IEEE 802.15.4 sniffer with a code provided by *Nordic Semiconductors*, and this application allows communication with the device through the serial port, and with these tools in hand, it is possible to obtain the data for this standard.

5.6.7 Integration in the GUI

The main issue is that the provided firmware is projected only for two scenarios: obtaining the data in real-time with Wireshark or through an API that is provided in a *Python* file by Nordic, but this is mostly useful to create .pcap files and not to do live observation. Therefore, for obtaining the data in real-time outside Wireshark, it was necessary to adapt the utilization of the device.

Since the data is not directly sent by the sniffer without the use of Wireshark or following Nordic usage for Python, it was necessary to understand how to obtain that data using other tools. Since it establishes a serial connection, the first step was to open a serial terminal in order to explore this connection with the nRF. Initially, it is clear and had nothing inside, but we found it possible to send a few control messages to configure the sniffer for the desired usage. The first important message is `channel X`, where X is a number between 11 and 26, included, and this controls the channel to which the nRF is going to listen. After this, the message `receive` can be inputted, because it will then start the collection of data. It is worth noting that it is possible to change the channel after starting the collection of data just with the `channel` command again. This configuration and reception of data were shown previously in the FIGURE 5.11.

When a frame is received by the software, if at least the short address is present, then it will save this value and instantiate an object called “Device”, which will be created with the short source address, the RSSI, the type, the timestamp, the channel which the device is in, the name, and the extended address, if also present. All of these fields have already been explained, except for the name, type, and timestamp. The name of the device is just a representation if the device is a coordinator: in ZigBee, the short address `0x0000` is reserved for the coordinator, so if the message contains a source address with this value, the name of the device will be set as “Hub”, otherwise, no value will be put there, the type is just mentioning that this is a ZigBee device, and the timestamp keeps track of when this device was last updated.

After instantiating this device, the program will call a function that will either append or update that device to the GUI. To keep track of what devices have been already sniffed, the program keeps a local list of devices. The new device will then be compared to each device on this list. If the device is not in this list, it means that it is not yet shown in the GUI, and it will be simply added to it and to the list. If the device already exists on this local list, then it’s only necessary to update its information in it and also in the GUI. The fields that have to be updated are the RSSI, the timestamp, and the extended address if the latter has been obtained in the sniffed message.

As mentioned before, it was necessary to create a program to adapt to the sniffer, and it was done through Python, with the device communication being done with pySerial, a library to control and send/receive data through serial ports in Python. It became especially useful in this scenario because, with it, we are able to retrieve the data and give the right commands to configure the channels and the sniffing.

The developed program, therefore, sends the control signals to the serial port and receives the data. It formats the received message and then processes it to show the address(es) in the GUI, along with the RSSI, channel, and the time of the last received frame that it was currently sniffing.

5.7 Conclusion

In this chapter, we delved into the design and implementation aspects of our IoT device identification GUI, focusing on WiFi, Bluetooth Low Energy, and ZigBee devices. The goal was to develop a solution to help police investigators identify IoT devices at a crime scene. The proposed solution listens to information transmitted by these communications protocols to detect and display the active IoT devices onto the GUI.

We began by discussing the design choices made during the development process, including the selection of protocols, hardware, programming language, and GUI library. We then provided an overview of the graphical user interface. Instructions on how to use the GUI were presented, enabling police investigators to interact with the interface.

The subsequent sections focused on the identification of WiFi, BLE, and ZigBee devices. For each protocol, we outlined the necessary steps involved in device discovery. We explored the specific frame structures highlighting the key information of interest that could be gathered from these frames.

To support the identification of WiFi devices, a firmware was developed. This firmware enabled the extraction and processing of relevant data for display in the GUI. To support the identification of BLE and IEEE 802.15.4/ZigBee devices, firmware from the manufacturer was used.

In conclusion, the developed IoT device identification GUI provides a user-friendly solution for identifying WiFi, BLE, and ZigBee devices. By leveraging the capabilities of each protocol and integrating them into a cohesive GUI, users can effectively monitor and analyze the IoT landscape in their vicinity. This system has the potential to assist in forensic investigations and enhance the process of evidence collection. In the next chapter, we will present the experiments conducted to evaluate the performance and effectiveness of the system.

Chapter 6

Experiments and Results

6.1 Introduction

This chapter focuses on the evaluation and results of the IoT device identification GUI. It aims to provide a comprehensive overview of the testing process, the obtained results, as well as any limitations encountered during the evaluation. The section 6.2 describes the setup of the testing scenario, encompassing the selection of devices, their placement, and the necessary configuration. Careful consideration was given to selecting a diverse range of IoT devices that are commonly encountered in real-world scenarios. The placement of these devices was strategically planned to simulate realistic conditions. The testing phase, discussed in section 6.3, involved executing the IoT device identification GUI in the established testing scenario. The GUI was used to identify and locate the IoT devices present. The obtained results are presented in section 6.4, highlighting the efficiency of the IoT device identification GUI. The limitations encountered during the evaluation will be discussed in section 6.5. These limitations include factors such as the RSSI value, channel hopping issue, or ZigBee network discovery.

6.2 Testing Scenario Setup

In this section, we will describe the testing scenario that was set up to evaluate the effectiveness and performance of the developed IoT device identification GUI. The objective is to simulate a real-world environment where multiple IoT devices are present and run as in a typical smart home.

6.2.1 Selection of Devices

To create a diverse and representative testing scenario, a range of IoT devices was selected. These devices were chosen because we were in possession of them and because they are prevalent in domestic settings. They also use WiFi, BLE, or ZigBee protocols to communicate. The selected devices include a smart socket, a security IP camera, a motion sensor, and a presence sensor. The devices are going to be used simultaneously to

simulate real-world scenarios and to evaluate the capabilities of the sniffer to detect and differentiate between the devices within these protocols.

WiFi Devices

- Smart Socket: The *TP-Link* smart socket is a WiFi-enabled socket that allows remote connections to turn it on and off and offers energy consumption monitoring.
- IP Camera: The *Xiaomi* security camera is a WiFi-based camera that offers high-quality photos and videos to be stored locally or through remote streaming and access.

BLE Devices

- Headphones: The *Bose QuietComfort 35* is a pair of BLE-enabled headphones offering wireless audio playback. This device cannot be considered an IoT device, but it was used because it utilizes Bluetooth Low Energy, and no other BLE devices were available for the testing.

ZigBee Devices

- Motion Sensor: The *Samsung SmartThings* motion sensor is, as the name implies, a sensor for detecting motion.
- Presence Sensor: The *Samsung SmartThings* is a sensor detecting the arrival and presence of individuals.
- Multipurpose Sensor: The *Samsung SmartThings* multipurpose sensor is a sensor that can monitor doors, windows, cabinets, or drawers depending on usage. In our case, the sensor will be used to monitor a door.

Other Devices

In addition to the IoT devices, other devices are evolved in the setup.

- WiFi Router: The *Proximus BBox 3V+* router provides internet access to the environment via aDSL.
- ZigBee Hub: The *Samsung SmartThings* hub is the central control unit for the other *Samsung SmartThings* devices. It brings the ZigBee communication in contact with the internet through Ethernet.
- Laptop: The laptop runs the developed IoT devices identification GUI. It will not be detected by the GUI itself.

The TABLE 6.1 details the different devices involved in the test setup. For each of them, the device type, the model, the MAC or IEEE extended address, the communication protocol, and an identifier are provided. The identifier will be used in the next section.

Device	Model	MAC/Extended Address	Technology	ID
Router	Proximus BBox 3V+	44:D4:54:F5:2B:36 & 44:D4:54:F5:2B:37	WiFi	(1)
Hub	Samsung STH-ETH-200	D0:52:A8:7:AA:27 & D0:52:A8:72:AF:A7:00:01	ZigBee	(2)
Motion Sensor	Samsung F-IRM-UK-V2	00:0D:6F:00:0B:BC:41:15	ZigBee	(2)
IP Camera	Xiaomi MJSXJ02CM	78:8B:2A:B2:20:EA	WiFi	(3)
Laptop	ASUS Vivobook	Variable	BLE, WiFi & ZigBee	(4)
Wireless Headphones	Bose QC35	5B:6B:DB:97:2:FE	BLE	(5)
Multipurpose Sensor	Samsung F-MLT-UK-2	00:0D:6F:00:0B:BB:05:92	ZigBee	(6)
Smart Socket	TP-Link HS110	50:C7:BF:ED:0A:54	WiFi	(7)
Presence Sensor	Samsung STSS-PRES- 001	D0:52:A8:01:06:AE:00:05	ZigBee	(8)

TABLE 6.1: Devices Involved in the Setup

6.2.2 Placement

The tests were conducted in a 4-story house, serving as the testing environment for evaluating the performance of the developed sniffer. The location of the devices is represented in FIGURE 6.1. The red numbers in the figure are the identifiers associated with the IoT devices in TABLE 6.1.

- **First Floor** The FIGURE 6.1A illustrates the first floor of the testing house.
 - (1) The WiFi router is attached to the wall for the aDSL connection.
 - (2) The *SmartThings* Hub and the motion sensor are placed in the same location on the first floor of the house. The motion sensor is on top of the hub.
 - (3) The IP camera points through the window, towards the garden.
- **Second Floor** The FIGURE 6.1B illustrates the second floor of the testing house.

- (4) The laptop runs the IoT devices identification GUI. Initially, it is located on the counter of the second floor. To detect the IoT devices, the laptop will be later moved to different places to study the evolution of the received signal strength. This will be further discussed in section 6.3.
- (5) The headphones are on the second floor and stay on a table.
- **Fourth Floor** The FIGURE 6.1C illustrates the bedroom on the fourth floor. This room contains the remaining IoT devices.
 - (6) The multipurpose sensor is placed on the door of the bedroom to detect opening and closing.
 - (7) The smart socket is plugged into an outlet in the bedroom.
 - (8) The presence sensor is placed on the bedroom table.



FIGURE 6.1: Floor Plan of Testing Environment with Position of Devices

The testing scenario was designed to resemble a typical residential environment, mirroring real-life situations. This approach ensures that the sniffer's performance and results can be effectively replicated in actual crime scenes without significant deviations.

The positioning variation inside the house also introduces a realistic test scenario because the devices are in different areas, which adds obstacles such as walls between the IoT

devices, the ZigBee hub, the router, and the laptop. Another aspect that can be noted in this testing scenario is the influence of the antennas. The Hub and the Motion Sensor are placed in proximity, and that allows us to observe the differences between the RSSI values of both devices. This can be an interesting analysis because the investigators can understand how much they should take into account if a device is a smaller, battery-powered one, or a bigger one that is connected to the wall, and its influence on the signal strength. This enables the assessment of the GUI's ability to detect and keep track of IoT devices under challenging signal propagation conditions.

6.2.3 Configuration

The devices' configuration would not matter in a real scenario, because the investigators would not know this information in most of the situations. But for the sake of transparency, the configuration of these devices will be explained in this section.

The WiFi devices are connected to the home router through the network with SSID "Proximus-Home-2B30" and operating on channel 1. Both WiFi devices were connected through their respective companion app, "Kasa Smart" for the smart socket and "Mi Home" for the smart camera.

The Bluetooth Headphones are not connected to any device and are simply switched on.

The ZigBee devices and the smart socket are configured and connected to the hub through the "Samsung SmartThings" app. The hub operates on channel 5 and is connected to the internet through an Ethernet connection to the router.

6.3 Testing

Prior to initiating the detection process, all personal equipment utilizing WiFi, Bluetooth, or ZigBee technologies was turned off to avoid corrupting the crime scene and eliminate interference. The GUI was launched from the laptop, and the scanning process was initiated to capture wireless communication signals emitted by IoT devices. Initially, the laptop running the GUI was positioned on the second floor of the house.

The procedures to identify WiFi, BLE or ZigBee devices have similarities and are described in the following subsections. Even if the WiFi, BLE, and ZigBee devices are concurrently monitored, we recommend first identifying all WiFi devices, then all BLE devices, and finally all the ZigBee devices. This facilitates the process for the investigator. We also recommend ending with the identification of ZigBee devices because the ZigBee network discovery can take up to 3 minutes whereas the WiFi network discovery only takes around 30 seconds.

During the testing procedure, the laptop was initially positioned on the second floor and then moved around the house. This approach of starting from a central location and gradually exploring the surroundings allows for the identification of all devices in the

house. By moving closer to each device, the received signal strength indicator (RSSI) increases, indicating the presence of nearby active devices.

WiFi Devices Identification

1. While walking around the house, the WiFi Access Point (AP) was identified by checking the values of the RSSI increasing, and as a result, it was possible to visually locate it. When near the router, the RSSI value can get close to -30, suggesting that it was the desired WiFi network and then it was selected. The selection causes the WiFi sniffer chip to only sniff the channel associated with the selected WiFi network.
2. For each displayed device on the **WiFi Devices** frame:
 - (a) If the RSSI was around -50, the leftmost case of the device entry was clicked to highlight the device, turning the device entry red for easy identification. If the RSSI value never approached -50 in any location within the house, it indicated that the device did not belong to the testing environment and therefore was not selected.
 - (b) We proceeded to move around the house while observing the RSSI value on the GUI. If the RSSI value increased, it indicated an increase in received signal strength, suggesting proximity to the transmitting device.
 - (c) If the RSSI value reaches approximately -30, it indicates that the IoT device was in close proximity to the current position of the laptop running the GUI. We focused our search in the immediate vicinity to locate the IoT device.
 - (d) Once the device was found, the investigators performed the necessary actions, such as documenting relevant information or gathering evidence, before proceeding to seize the device.
 - (e) Upon completion of the necessary actions, the **Seize** button corresponding to the identified device entry (highlighted in red) was clicked, moving the seized device into the **Seized Devices** frame.
3. As there is only one WiFi AP in the house, the procedure has not to be followed again from step 2.

As far as we know, we are not aware of IoT devices using WiFi in a peer-to-peer fashion, or without relying on AP. In such scenario, the selection of the channel used by the WiFi router could hide the peer-to-peer communications if the peer-to-peer network uses a distinct channel than the one of the WiFi router. To overcome this problem, we then recommend unchecking the WiFi network after the relevant WiFi devices were seized.

BLE Devices Identification

The procedure to identify the BLE devices shares a common step with the WiFi devices identification one.

1. For each displayed device on the **BLE Devices** frame:
 - (a) If the RSSI was around -50, the leftmost case of the device entry was clicked to highlight the device, turning the device entry red for easy identification. If the RSSI value never approached -50 in any location within the house, it indicated that the device did not belong to the testing environment and therefore was not selected.
 - (b) We proceeded to move around the house while observing the RSSI value on the GUI. If the RSSI value increased, it indicated an increase in received signal strength, suggesting proximity to the transmitting device.
 - (c) If the RSSI value reaches approximately -30, it indicates that the IoT device was in close proximity to the current position of the laptop running the GUI. We focused our search in the immediate vicinity to locate the IoT device.
 - (d) Once the device was found, we performed the necessary actions, such as documenting relevant information or gathering evidence, before proceeding to seize the device.
 - (e) Upon completion of the necessary actions, the **Seize** button corresponding to the identified device entry (highlighted in red) was clicked, moving the seized device into the **Seized Devices** frame.

ZigBee Devices Identification

1. While walking around the house, the ZigBee hub was identified by checking the values of the RSSI increasing, and as a result, it was possible to visually locate it. When close to it, an RSSI value close to -50 indicated the strongest signal, suggesting that it was the desired ZigBee network. The selection causes the ZigBee sniffer chip to only sniff the channel associated with the selected ZigBee network.
2. For each displayed device on the **ZigBee Devices** frame:
 - (a) If the RSSI was around -50, the leftmost case of the device entry was clicked to highlight the device, turning the device entry red for easy identification. If the RSSI value never approached -50 in any location within the house, it indicated that the device did not belong to the testing environment and therefore was not selected.
 - (b) We proceeded to move around the house while observing the RSSI value on the GUI. If the RSSI value increased, it indicated an increase in received signal strength, suggesting proximity to the transmitting device.
 - (c) If the RSSI value reaches approximately -30, it indicates that the IoT device was in close proximity to the current position of the laptop running the GUI. We focused our search in the immediate vicinity to locate the IoT device.

- (d) Once the device was found, we performed the necessary actions, such as documenting relevant information or gathering evidence, before proceeding to seize the device.
 - (e) Upon completion of the necessary actions, the **Seize** button corresponding to the identified device entry (highlighted in red) was clicked, moving the seized device into the **Seized Devices** frame.
3. As there is only one ZigBee hub in the house, the procedure has not to be followed again from step 2.

6.4 Results

By following the procedures described in the previous section, we were able to systematically detect and identify all IoT devices within the testing environment. The Appendix F contains several figures that illustrate the testing process and the obtained results.

WiFi Device Identification Results

After evaluating the GUI in the testing environment, the results revealed two challenges in the identification of WiFi devices.

Firstly, the GUI displayed a higher number of devices than expected, even after selecting the specific channel on which the WiFi router of the testing environment is running. This discrepancy can be attributed to the presence of neighboring networks that operate on the same channel as the testing environment. As a result, devices from these neighboring networks were also detected, leading to a larger pool of displayed devices that were not part of the intended testing environment. The FIGURE 6.2 illustrates the different WiFi networks discovered. As shown, there are multiple APs that use channel one to communicate.

	RSSI	Channel	SSID	BSSID	Updated
☐	-66	1	Proximus-Home-2B30_EXT		15:48:23
☒	-71	1	Proximus-Home-2B30		15:48:23
☐	-85	1	Proximus Public Wi-Fi		15:48:23
☐	-76	6	VOO-X678JZS		15:48:23
☐	-79	6	Proximus-Home-6380		15:48:23
☐	-79	6	Proximus Public Wi-Fi		15:48:23
☐	-94	1	Flybox-FAA5		15:48:23
☐	-92	1	Flybox-FAA5-guest		15:48:23
☐	-88	2	Philips TAPB400-800f8d.n090		15:48:23
☐	-90	8	Proximus-Home-5818		15:47:47

FIGURE 6.2: Detected WiFi Networks with the Testing One Selected and Highlighted

Secondly, the other challenge encountered during the identification of WiFi devices is the difficulty in determining the specific network to which the devices are connected. Because we set up the testing environment we knew it in advance but in a real scenario, the investigators wouldn't. To avoid missing a single transmitted frame, it is necessary to identify the channel on which the target network operates. This is achieved by scanning the surrounding networks and retrieving their RSSI, SSID, BSSID, and channel values. However, when multiple access points (APs) are present in the vicinity, it becomes challenging to accurately determine the network to which the devices of interest are connected. The similar RSSI values and similar SSIDs from different APs can cause confusion and make it harder to isolate the desired network for further analysis. To address this issue, a recommended solution is to physically move closer to the WiFi router within the testing environment if it is possible to see it. Moving closer to the router helps in reducing signal interference from neighboring networks and improves the accuracy of identifying the target network. By doing so, the GUI will update the RSSI values of the different networks identified, and the network with the highest RSSI is more likely to be the one to which the devices of interest are connected. This approach assumes that the WiFi router can be easily located within the environment.

If the WiFi router is hidden or not easily identifiable, it poses a challenge for the police investigator to select the correct channel for sniffing.

Despite these challenges, the WiFi part of the sniffing process successfully detected a variety of devices, including the Smart Socket and the IP Camera. Even with a lot of devices around, it was possible to find the ones that were inside the house. Most of the other devices usually had their RSSIs very weak, going around -90 dBm, while inside the house, the minimum was -70 dBm, and getting closer to the devices was at most -30 dBm.

The results from the WiFi sniffing highlight the need for careful consideration of the testing environment, device density, and access point selection when conducting WiFi sniffing. These factors play a significant role in the identification and monitoring of WiFi devices, and they should be taken into account to ensure accurate and reliable results.

BLE Device Identification Results

During the identification of BLE devices in the testing environment, we observed that the RSSI value of BLE devices experiences a higher level of degradation with distance and obstacles compared to WiFi technology. This could indicate that the signal strength of BLE devices weakens more rapidly as the distance between the BLE device and the sniffer increases, or when obstacles such as walls are present in the signal path.

The BLE sniffer autonomously scanned the environment for BLE signals and displayed the results in the GUI. The GUI successfully displayed the Bose QC 35 headphones along with their relevant information, allowing for their easy identification and localization.

The identification of BLE devices in the testing environment underscores the need for careful consideration of signal propagation characteristics when working with BLE devices. It also demonstrates the effectiveness of the GUI in identifying BLE devices.

ZigBee Devices Identification Results

During the identification of ZigBee devices, the GUI successfully detected all the ZigBee devices within the testing environment.

At first, the ZigBee sniffer employed a comprehensive scan across all available channels to detect those that contain communicating ZigBee devices. This process took approximately three minutes. Afterward, the sniffer was able to identify the home network and detected all ZigBee devices along with the extended address, including those located in the distant bedroom with multiple obstacles in between. This highlights the sniffer's capability to identify ZigBee devices even in challenging conditions. ZigBee didn't suffer from the same problems as WiFi because there were no other ZigBee networks around, which made device identification easier.

6.5 Limitations

RSSI Value

No particular physical parameter has a standardized relationship to the RSSI reading. For instance, the IEEE 802.11 standard does not define a direct correlation between RSSI value and the power level, whether in milliwatts or decibels referenced to one milliwatt (dBm). Each vendor and chipset manufacturer establishes their own accuracy, granularity, and range for the actual power measurement and the corresponding range of RSSI values (from 0 to the highest RSSI value) [43].

Despite the lack of precise accuracy, RSSI remains a viable indicator for localization purposes due to its widespread availability across wireless devices without requiring additional hardware. While RSSI measurements may not always provide sufficient accuracy to determine the precise location, it still serves as a practical and accessible means for localization within wireless networks.

Channel Hopping

In the presence of multiple WiFi (resp. ZigBee) networks and when no specific network is selected, the period of switching between detected channels is approximately 600 milliseconds (resp. approximately 25 seconds).

For WiFi sniffing, it is not a problem because WiFi devices frequently send information and the switching a channel period is relatively short. In other words, if a device is transmitting on a channel that is not currently sniffed, that is not a big problem because

it is very likely that this device will also transmit data later when the channel will be sniffed.

For ZigBee sniffing, this results in infrequent updates of information from other devices, which hinders the accuracy of the locating process. Therefore, we recommend selecting a specific channel and tracking the devices operating within it when using the ZigBee sniffer and then selecting the other channel.

ZigBee has this longer period for sniffing between changing channels because the idea is to leave some time for the investigators to update the signal strength of these devices before hopping to the next channel since ZigBee messages are not as frequent as WiFi. This longer window in which a channel will be sniffed helps with this, although it is still recommended to select a specific channel to sniff on. And since it will only sniff the channels in which there is the presence of a data flow, it is still efficient in detecting the devices. However, there is a tradeoff involved: only one channel is updated during the approximately 25-second interval. Consequently, if an investigator approaches a device not on the current channel, it will not be updated, potentially causing the device to be missed. The duration of this time period can be adjusted within the code, as will be shown in [Appendix A](#).

ZigBee Network Discovery

During the ZigBee network discovery phase, if ZigBee devices fail to communicate within the time frame during which a specific channel is being listened to, that channel will be ignored afterward. In such cases, the program would need to be restarted to ensure proper sniffing. To avoid this situation, the period for listening to a specific channel is set to 10 seconds, which has been tested thoroughly and it was concluded that it is enough time for data exchange in a ZigBee network, but it can also be changed in the code, as shown in [Appendix A](#).

6.6 Conclusion

The experiments conducted in this chapter provided valuable insights into the effectiveness of our IoT device identification GUI in a real-world testing environment. The testing scenario setup involved carefully selecting devices, strategically placing them within the environment, and configuring the necessary parameters. The testing process consisted of launching the GUI and going around the house and checking the devices' RSSI. The obtained results highlighted the challenges and strengths associated with identifying WiFi, BLE, and ZigBee devices.

Regarding WiFi device identification, one of the challenges observed was the presence of neighboring networks using the same channel as the testing environment. This resulted in the display of additional devices that were not part of the testing scenario.

For BLE devices, it was found that the RSSI values experienced greater degradation with distance and obstacles compared to WiFi technology. Despite this, the GUI proved effective in identifying the BLE devices present in the environment.

When it comes to ZigBee devices, everything worked as expected, but we can see that channel hopping presented a challenge both during the discovery and sniffing phases. If devices did not communicate within the listening timeframe, the corresponding channel would not be sniffed unless the program was restarted. Additionally, sniffing multiple channels without specifying a particular one led to infrequent updates and impacted the accuracy of the locating process. Therefore, selecting a specific channel was recommended for optimal ZigBee device identification.

While the experiments provided valuable insights, it is important to acknowledge the limitations of the testing environment and methodology. Factors such as the physical layout of the environment, signal interference, and variations in device performance may have influenced the results.

In conclusion, the experiments and results presented in this chapter demonstrate the functionality and effectiveness of our IoT device identification GUI. The GUI proved capable of identifying WiFi, BLE, and ZigBee devices, albeit with some challenges specific to each technology.

Chapter 7

Conclusion

In this final chapter, we will revisit the key aspects discussed throughout this master thesis and provide a summary of the main findings. We will also address the limitations of our solution and propose avenues for future work in the field of IoT forensics.

7.1 Key Findings and Contribution

The aim of this master thesis was to explore the field of IoT forensics and develop a solution to help police investigators deal with IoT devices at a crime scene. To achieve this, we first began by introducing the context in which this master thesis takes place and the objectives. We then provided a general background on the Internet of Things and on IoT forensics. We then divided our work into two distinct parts. Part I provided the state-of-the-art in IoT computer forensics. Part II described the solution we developed to identify IoT devices.

In chapter 3, we studied the existing IoT forensics frameworks and procedures. We identified a lack of study in the field of IoT forensic investigations, all the more so in smart home forensics. Additionally, the existing frameworks and procedures reviewed revealed a lack in their implementation and testing.

In chapter 4, we examined the underlying problem associated with IoT forensics. We explored existing digital forensics tools and highlighted their relevance and limitations in the context of IoT forensics. We emphasized the need for specialized tools, standardized procedures, best practices, and enhanced training and education. We also noticed that most research advancements make the assumption of the direct availability of IoT devices. Yet the identification of IoT devices at a crime scene has become challenging.

In chapter 5, we described the design and implementation of our IoT Device Identification GUI that enables the identification of IoT devices that use WiFi, BLE, and ZigBee protocols. We provided detailed descriptions and integration steps for each device type.

Finally, in chapter 6, we conducted experiments to test the effectiveness of our solution. The results demonstrated the successful identification of IoT devices using our GUI.

7.2 Limitations and Future Work

While our solution is promising in addressing the challenges of locating IoT devices, it also has certain limitations that should be mentioned and that can be covered by future works.

Active Devices

One limitation of our GUI is its focus on detecting active devices (devices engaged in communication activities). Consequently, devices that have infrequent data transmission patterns, such as a sensor that rarely sends data, may not be detected by our GUI. This represents a gap in our current approach, as it overlooks a subset of IoT devices that operate on sporadic data transmission schedules. To address this limitation, future work should explore strategies to enhance our GUI's capabilities and incorporate mechanisms to identify such devices with minimal data activity. By doing so, we can provide a more comprehensive solution that covers a wider range of IoT devices, regardless of their communication frequency.

Limited Device Support

Our solution currently focuses on WiFi, BLE, and ZigBee protocols. Although they are widely used in the IoT field, some IoT devices use different protocols to communicate. Future works can expand our GUI by adding new hardware capable of sniffing other protocols and incorporating them into the GUI.

Integration with Forensic Tools

Our solution operates as a standalone GUI. Future work should explore the integration of our solution with existing digital forensics tools to enhance the overall investigative capabilities and streamline forensic investigations ensuring forensically sound evidence management.

7.3 Final Comments

In the end, the findings from this work contribute to the overall understanding and improvement of IoT forensics. We want to highlight the importance of continuous research in the field of IoT forensics. As the IoT ecosystem continues to evolve, the capabilities of forensic tools have to evolve as well to keep pace with the challenges that are emerging in this aspect.

Lastly, the IoT device identification GUI we built for locating IoT devices at crime scenes can be considered a significant advancement in the IoT forensic field. Further refinements and enhancements can be made to the GUI to improve its performance, expand its device compatibility, and integrate it with other existing forensic tools. However, the successful

tests conducted to locate the devices demonstrate its practical applicability and potential for IoT forensic investigations. It can also serve as a foundation for further research in this field.

The journey does not end here but rather opens up new avenues for research and innovation in the field of IoT forensics.

Appendix A

GUI setup

This Appendix will verse about how to configure the GUI for the use of the investigators. It is necessary to adapt the serial ports in which the devices are connected, and if the investigators want, they can change the timing of both phases of ZigBee sniffing.

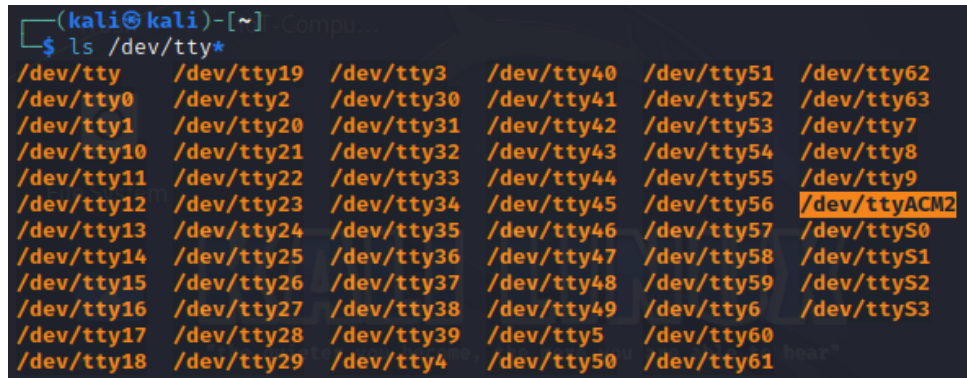
A.1 Configuration of Serial Ports

For both devices, it is first necessary to install the drivers of them. For the nRF devices, it is only necessary to install the *nRF Connect For Desktop* software that the drivers are installed. For the ESP32, the drivers are available at their website.

After correctly installing the drivers and programming the devices, it is necessary to find out to what serial port the devices are connected. For this step, it is better to connect one device at a time and execute the procedure that will be described, because then it is possible to keep track of which device corresponds to the serial port. It is different for Linux and for Windows.

A.1.1 Finding the Serial Port on Linux

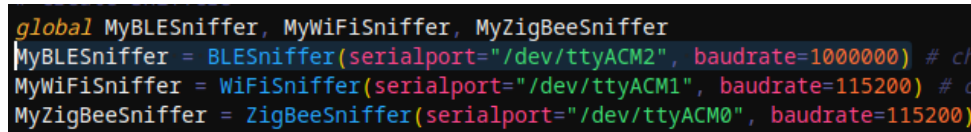
1. Connect the device via USB. In this example, it will be demonstrated for the BLE Sniffer.
2. Open a terminal window and insert the following command: `ls /dev/tty*`.
3. It is possible that lots of results appear. The port for the sniffer starts by `/dev/ttyACM` or `/dev/ttyUSB`. This is represented in the figure below.



```
(kali㉿kali)-[~] -Compu
$ ls /dev/tty*
/dev/tty      /dev/tty19  /dev/tty3   /dev/tty40  /dev/tty51  /dev/tty62
/dev/tty0     /dev/tty2   /dev/tty30  /dev/tty41  /dev/tty52  /dev/tty63
/dev/tty1     /dev/tty20  /dev/tty31  /dev/tty42  /dev/tty53  /dev/tty7
/dev/tty10    /dev/tty21  /dev/tty32  /dev/tty43  /dev/tty54  /dev/tty8
/dev/tty11    /dev/tty22  /dev/tty33  /dev/tty44  /dev/tty55  /dev/tty9
/dev/tty12    /dev/tty23  /dev/tty34  /dev/tty45  /dev/tty56  /dev/ttyACM2
/dev/tty13    /dev/tty24  /dev/tty35  /dev/tty46  /dev/tty57  /dev/ttyS0
/dev/tty14    /dev/tty25  /dev/tty36  /dev/tty47  /dev/tty58  /dev/ttyS1
/dev/tty15    /dev/tty26  /dev/tty37  /dev/tty48  /dev/tty59  /dev/ttyS2
/dev/tty16    /dev/tty27  /dev/tty38  /dev/tty49  /dev/tty6   /dev/ttyS3
/dev/tty17    /dev/tty28  /dev/tty39  /dev/tty5   /dev/tty60
/dev/tty18    /dev/tty29  /dev/tty4   /dev/tty50  /dev/tty61
```

FIGURE A.1: Serial Ports on Linux.

4. If there is more than one port with these values, unplug the device, and go back to step 2. From this, eliminate the ports that are still open, and the value is the one that is not present this second time.
5. Input the port on the respective field for this sniffer in the code on the file `main.py`, as represented and highlighted in the next figure.



```
global MyBLESniffer, MyWiFiSniffer, MyZigBeeSniffer
MyBLESniffer = BLESniffer(serialport="/dev/ttyACM2", baudrate=1000000) # ch
MyWiFiSniffer = WiFiSniffer(serialport="/dev/ttyACM1", baudrate=115200) # c
MyZigBeeSniffer = ZigBeeSniffer(serialport="/dev/ttyACM0", baudrate=115200)
```

FIGURE A.2: Changing the code to adapt the serial port in Linux.

6. Repeat the entire process for the other pieces of hardware until the three sniffers are correctly put into the code.

A.1.2 Finding the Serial Port on Windows

1. Connect the device via USB. In this example, it will be demonstrated for the BLE Sniffer.
2. Search for the “Device Manager” in the search bar, and open it.
3. On this new window, look for the “Ports (COM & LPT)” tab and expand it. From there, the nRF device will appear as USB Serial Device (COM*), and the ESP32 as Silicon Labs CP210x USB to UART Bridge (COM*), with * being a number. The result is shown in the following figure:

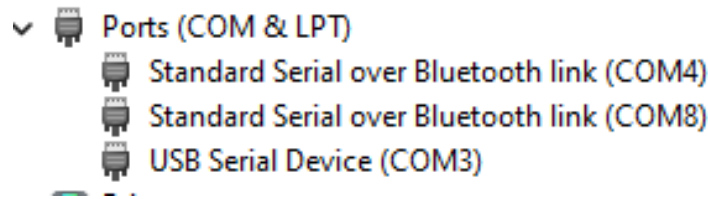


FIGURE A.3: Serial Ports on Windows.

4. If there is more than one port with these values, unplug the device, and go back to step 2. From this, eliminate the ports that are still open, and the value is the one that is not present this second time.
5. Input the port on the respective field for this sniffer in the code on the file `main.py`, as represented and highlighted in the next figure.

```
global MyBLESniffer, MyWiFiSniffer, MyZigBeeSniffer
MyBLESniffer = BLESniffer(serialport="COM3", baudrate=1000000) # ch
MyWiFiSniffer = WiFiSniffer(serialport="COM5", baudrate=115200) # c
MyZigBeeSniffer = ZigBeeSniffer(serialport="COM7", baudrate=115200)
```

FIGURE A.4: Changing the Code to Adapt the Serial port in Windows.

6. Repeat the entire process for the other pieces of hardware until the three sniffers are correctly put into the code.

A.2 Timing for ZigBee Sniffer

For the ZigBee sniffer, it has been mentioned that the time periods for the Network Discovery and for the Sniffing processes are able to be changeable according to the needs of the investigators. To do this, the procedure is similar for both cases.

A.2.1 Changing the Timing for Network Discovery

1. Go to the `ZigBee_Sniffer.py` file.
2. Find the `findChannels` function.
3. Inside it, there is a `while` command with a value of 10 being added to the rightmost operand. This value is the time that will be waited. This line is highlighted and represented in the following figure:

```
def findChannels(self):

    for _ in range(1):
        for channel in range(11, 27):

            self.serial.reset_input_buffer()
            timeout_start = time.time()
            self.serial.write("receive\r\n".encode())
            rec = self.serial.readline()
            self.serial.write("channel {}\r\n".format(channel).encode())

            rec = self.serial.readline()
            rec = None

            while time.time() < timeout_start + 10:
                rec = self.serial.readline()

            if not rec:
                break

            if (rec.decode().split())[0] == "channel":
                continue
            elif channel not in self.channels:
                self.channels.append(channel)
                self.serial.reset_input_buffer()
                break
```

FIGURE A.5: Line that should be changed to alter the Network Discovery time in each channel.

A.2.2 Changing the Timing for Sniffing

1. Go to the ZigBee_Sniffer.py file.
2. Find the `sniff` function.
3. Inside it, there is a `while` command with a call to a function that has two values, 20 and 30, being added to the rightmost operand. This function is to generate a random time between those two values. To change the value, it is possible to change the lower or the higher one, and this will change the window of random time that time will be for each channel in the sniffing phase. This line is highlighted and represented in the following figure:

```
def sniff(self):
    if not self.channels:
        return

    for channel in self.channels:
        self.dataVal = 0
        timeout_start = time.time()
        self.serial.write("channel {}".format(channel).encode())

        rec = self.serial.readline()
        rec = None

        while time.time() < timeout_start + random.randint(20, 30):

            if not self.running or self.restart:
                self.restart = False
                return

            rec = self.serial.readline()
```

FIGURE A.6: Line that should be changed to alter the Sniffing time in each channel.

Appendix B

Heltec WiFi LoRa V2 WiFi Sniffer Firmware

```
// WiFi Network Discovery inspired by the available example in the library Arduino ESP32:
// https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi/examples/WiFiScan
// WiFi Packet Sniffing inspired by the provided code: https://github.com/ESP-E0S/ESP32-WiFi-Sniffer
// ESP32 documentation available at: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/
#include "driver/gpio.h"
#include "esp_wifi.h"
#include "esp_wifi_types.h"
#include "esp_system.h"
#include "esp_event.h"
#include "esp_event_loop.h"
#include "freertos/FreeRTOS.h"
#include "nvs_flash.h"
#include "WiFi.h"

#define WIFI_SCAN_MIN_PERIOD (30000) // [ms]
#define WIFI_CHANNEL_MIN_SWITCH_INTERVAL (500) // [ms]
#define WIFI_CHANNEL_MAX_SWITCH_INTERVAL (700) // [ms]
#define WIFI_CHANNEL_MAX (13)

/* = GLOBAL VARIABLES ===== */
bool channel_hopping_mode = true;
long last_scan_millis = 0;
long scan_period = 0;
uint8_t channel = 1;
static wifi_country_t wifi_country = { .cc = "CN", .schan = 1, .nchan = 13 };

/* = STRUCTURES ===== */
// Data Frame Header Structure
typedef struct {
    unsigned frame_ctrl : 16;
    unsigned duration_id : 16;
    uint8_t addr1[6];
    uint8_t addr2[6];
    uint8_t addr3[6];
    unsigned sequence_ctrl : 16;
    uint8_t addr4[6];
} wifi_ieee80211_mac_data_hdr_t;

// Management Frame Header Structure
typedef struct {
    unsigned frame_ctrl : 16;
    unsigned duration_id : 16;
    uint8_t addr1[6]; // < destination address
    uint8_t addr2[6]; // < source address
    uint8_t BSSID[6];
    unsigned sequence_ctrl : 16;
} wifi_ieee80211_mac_mgmt_hdr_t;
```

```

// Frame Header Structure
typedef struct{
    union{
        wifi_ieee80211_mac_mgmt_hdr_t mgmt;
        wifi_ieee80211_mac_data_hdr_t data;
    }header;
}wifi_ieee80211_mac_hdr_t;

// Frame Structure
typedef struct {
    wifi_ieee80211_mac_hdr_t hdr; // header
    uint8_t payload[0]; // payload
} wifi_ieee80211_frame_t;

/* Promiscuous Info Structure
typedef struct {
    wifi_pkt_rx_ctrl_t rx_ctrl; // < metadata from PHY layer
    uint8_t payload[0]; // < frame
} wifi_promiscuous_pkt_t;
*/

/* = WIFI SCAN FUNCTIONS ===== */
void wifi_network_formatter(int networkItem) {
    // channel, RSSI, SSID, BSSID
    printf("N,%d,%d,%s,%s\n",
        WiFi.channel(networkItem),
        WiFi.RSSI(networkItem),
        WiFi.SSID(networkItem).c_str(),
        WiFi.BSSIDstr(*(WiFi.BSSID(networkItem))).c_str());
}

/* = WIFI SNIFFER FUNCTIONS ===== */
esp_err_t event_handler(void *ctx, system_event_t *event) {
    return ESP_OK;
}

void setup_wifi_sniffer(void) {
    nvs_flash_init();
    tcpip_adapter_init();
    ESP_ERROR_CHECK(esp_event_loop_init(event_handler, NULL));
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    esp_wifi_init(&cfg);
    esp_wifi_set_country(&wifi_country);
    esp_wifi_set_storage(WIFI_STORAGE_RAM);
    esp_wifi_set_mode(WIFI_MODE_NULL);
    esp_wifi_start();
    esp_wifi_set_promiscuous(true);
    esp_wifi_set_promiscuous_rx_cb(&wifi_frame_handler);
}

```

```

void wifi_frame_handler(void *buff, wifi_promiscuous_pkt_type_t type) {
    if(type == WIFI_PKT_CTRL || type == WIFI_PKT_MISC){
        return;
    }
    const wifi_promiscuous_pkt_t *raw_data = (wifi_promiscuous_pkt_t *)buff;
    const wifi_pkt_rx_ctrl_t *metadata = &raw_data->rx_ctrl;
    const wifi_ieee80211_frame_t *frame = (wifi_ieee80211_frame_t*) raw_data->payload;
    if(type == WIFI_PKT_MGMT){
        const wifi_ieee80211_mac_mgmt_hdr_t* header = (const wifi_ieee80211_mac_mgmt_hdr_t*) &frame->hdr;
        printf("H,%u,%d,%02x:%02x:%02x:%02x:%02x,MGMT\n", metadata->channel, (int8_t)metadata->rssi, header->addr2[0], header->addr2[1], header->addr2[2],header->addr2[3], header->addr2[4], header->addr2[5]);
    }
    else if (type == WIFI_PKT_DATA) {
        const wifi_ieee80211_mac_data_hdr_t *header = (const wifi_ieee80211_mac_data_hdr_t*)&frame->hdr;
        printf("H,%u,%d,%02x:%02x:%02x:%02x:%02x,DATA\n", metadata->channel, (int8_t)metadata->rssi, header->addr2[0], header->addr2[1], header->addr2[2],header->addr2[3], header->addr2[4], header->addr2[5]);
    }
}

/* = SETUP ===== */
// the setup function runs once when you press reset or power the board
void setup() {
    Serial.begin(115200);
    setup_wifi_sniffer();
    delay(1000);
}

/* = LOOP ===== */
// the loop function runs over and over again forever
void loop() {

    // -----
    // SERIAL READ
    // -----
    // check if there is data coming in the serial
    if (Serial.available()) {
        int i = Serial.parseInt();
        if (i > 0 && i <= WIFI_CHANNEL_MAX) { // sniff a specific channel in the range [1, 13]
            channel = (uint8_t)i;
            esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);
            channel_hopping_mode = false;
        } else if (i == -1) { // sniff all channels
            channel_hopping_mode = true;
        }
        delay(10);
    }
}

```

```

// -----
// CHANNEL HOPPING
// -----
// check if channel hopping mode enabled
if (channel_hopping_mode) {
    long random_switch_interval = random(WIFI_CHANNEL_MIN_SWITCH_INTERVAL, WIFI_CHANNEL_MAX_SWITCH_INTERVAL);
    vTaskDelay(random_switch_interval / portTICK_PERIOD_MS);
    esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);
    channel = (channel % WIFI_CHANNEL_MAX) + 1;
}

// -----
// WIFI NETWORKS SCAN
// -----
// regularly trigger Wi-Fi network scan (between 30 and 35 [s])
long current_millis = millis();
if (current_millis - last_scan_millis > scan_period) {
    WiFi.scanNetworks(true /*async*/, true /*show_hidden*/, true /*passive_scan*/, 500 /*max_ms_per_chan*/, 0
/*channel*/, nullptr /*ssid*/, nullptr /*bssid*/);
    last_scan_millis = current_millis;
    scan_period = random(WIFI_SCAN_MIN_PERIOD, 5000); // Add some randomness in the scan period
}
// check if Wi-Fi network scan completed
int n = WiFi.scanComplete(); // return -1 if scan not finished, -2 if scan not triggered
if (n >= 0) {
    for (int i = 0; i < n; i++) {
        wifi_network_formatter(i);
    }
    WiFi.scanDelete(); // delete last scan result from RAM
    esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE); // set back the channel before network scan started
}
}

```

Appendix C

Programming the nRF52840 Dongle

Programming the nRF52840 Dongle in nRF Connect Programmer requires a different approach than programming the nRF51 Dongle.

To program the nRF52840 Dongle:

1. Open nRF Connect for Desktop and launch [nRF Connect Programmer](#).
2. Insert the nRF52840 Dongle into a USB port on the computer.
3. Put the dongle into bootloader mode by pressing the **RESET** button.

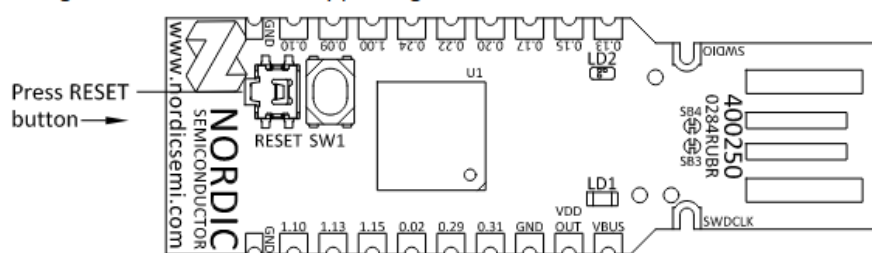


Figure 5: nRF52840 Dongle overview

Note:

- This step is not needed if the currently running application uses the DFU trigger library.
- If this is the first time the dongle is connected, a driver needed for the nRF52840 USB DFU feature is also installed as part of this step.

The status light (**LD2**) will start pulsing red, which indicates that the dongle is powered up and in bootloader mode. After a few seconds, the computer recognizes the dongle as a USB composite device.

4. In the navigation bar in the Programmer app, click **Select device** and select the serial number of the dongle from the drop-down list.
5. Drag and drop the HEX file into the **File Memory Layout** section. Alternatively, click **Add file** to add the files you want to program, by using one of the following options:
 - Select the files you used recently.
 - If there are no recently used files, click **Browse** from the drop-down list.
6. Select the firmware image file (with the extension `.hex`) from the file browser that opens up.
7. Click **Write** to program the firmware onto the dongle.

When the writing process completes, the device resets and – unless the application uses the DFU Trigger Library – the dongle will no longer show up in the Programmer app, as it is no longer in the bootloader mode.

Appendix D

Programming the nRF BLE Sniffer Firmware

You must connect a *DK* or dongle running the nRF Sniffer firmware to your computer to be able to use the nRF Sniffer for Bluetooth LE.

See [Supported development kits and dongles](#) for a list of development kits and dongles that can run the nRF Sniffer firmware.

There are various ways to program the nRF Sniffer firmware. The following instructions use [nRF Connect Programmer](#), but you can also use the command-line tool **nrfjprog** (which is part of the [nRF Command Line Tools](#)).

To program your *DK* or dongle, complete the following steps:

1. Install nRF Connect Programmer.
See [Installing the Programmer](#) for instructions.
2. On macOS and Linux, install the SEGGER J-Link software.
It is available from [SEGGER J-Link Software](#).

Note: On Windows, the J-Link software is included in nRF Connect for Desktop, so you can skip this step.

3. Locate the firmware HEX file for your *DK* or dongle.

All firmware HEX files are located in *Sniffer_Software/hex/*. Use the suitable file for your *DK* or dongle:

Development kit/dongle	Firmware file name
nRF52840 DK (PCA10056)	sniffer_nrf52840dk_nrf52840_*.hex
nRF52840 Dongle (PCA10059)	sniffer_nrf52840dongle_nrf52840_*.hex
nRF52 DK (PCA10040)	sniffer_nrf52dk_nrf52832_*.hex
nRF51 DK (PCA10028)	sniffer_nrf51dk_nrf51422_*.hex
nRF51 DK (PCA10031)	sniffer_nrf51dongle_nrf51422_*.hex

Table 2: Firmware file names

4. Follow the instructions in [Programming a Development Kit or the nRF51 Dongle](#) or [Programming the nRF52840 Dongle](#) to program the HEX file.

[45]

The third step is slightly different than the one depicted on the figure because we added the firmware in the project directory. The firmware HEX file for the chip is named

`sniffer_nrf52840dongle_ble.hex` and is located inside the folder `firmwares` as explained earlier in section [5.3.2](#).

Appendix E

Programming the nRF IEEE 802.15.4 Sniffer Firmware

You must connect a development kit or dongle running the nRF Sniffer firmware to your computer to use the nRF Sniffer for 802.15.4.

See [Supported development kits and dongles](#) for a list of development kits and dongles that can run the nRF Sniffer firmware.

There are various ways to program the nRF Sniffer firmware. The following instructions use [nRF Connect Programmer](#), but you can also use the command line tool `nrfjprog` (which is part of the [nRF Command Line Tools](#)).

To program your development kit or dongle, complete the following steps:

1. Install nRF Connect Programmer.
See [Installing the Programmer](#) for instructions.
2. On macOS and Linux, install the [SEGGER J-Link Software](#).

Note: On Windows, the J-Link software is included in nRF Connect for Desktop, so you can skip this step.

3. Locate the firmware hexadecimal file for your development kit or dongle.
All firmware hexadecimal files are located in `Sniffer_Software/nrf802154_sniffer/`. Use the suitable file for your development kit or dongle:

Development kit/dongle	Firmware file name
nRF52840 DK (PCA10056)	nrf802154_sniffer.hex
nRF52840 Dongle (PCA10059)	nrf802154_sniffer_dongle.hex

Table 1. Firmware file names

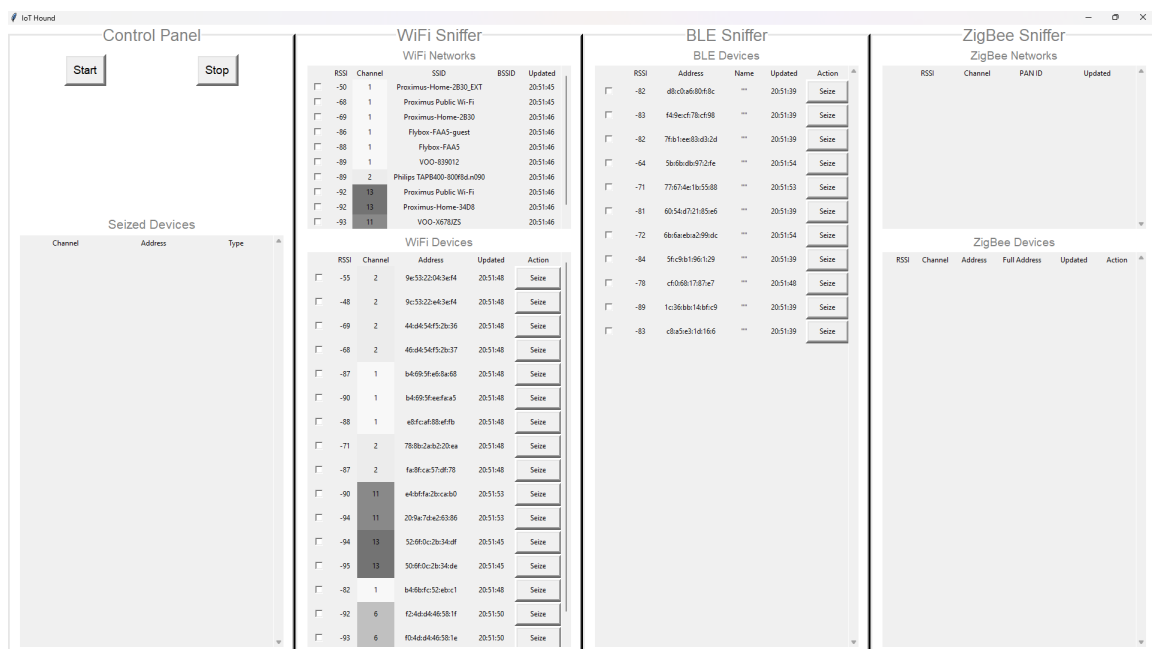
4. Follow the instructions in [Programming a Development Kit or the nRF51 Dongle](#) or [Programming the nRF52840 Dongle](#) to program the HEX file.

The third step is slightly different than the one depicted on the figure because we added the firmware in the project directory. The firmware HEX file for the chip is named `sniffer_nrf52840dongle_802154.hex` and is located inside the folder `firmwares` as explained earlier in section [5.3.2](#).

Appendix F

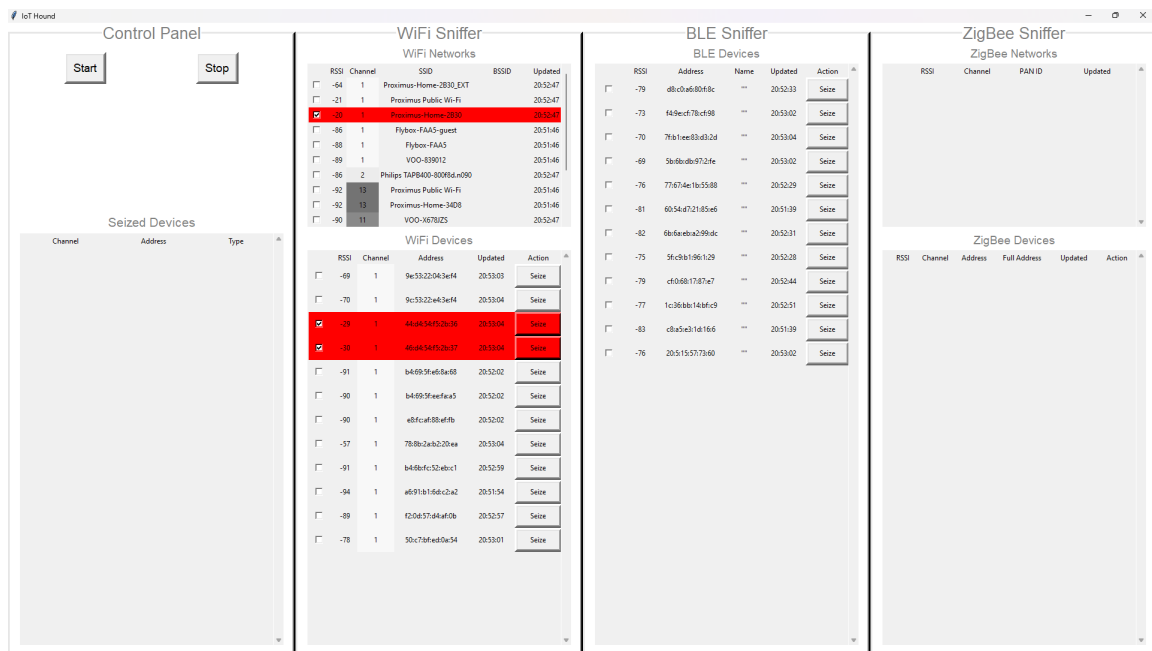
Performed Tests

F.1 Initializing the Sniffer



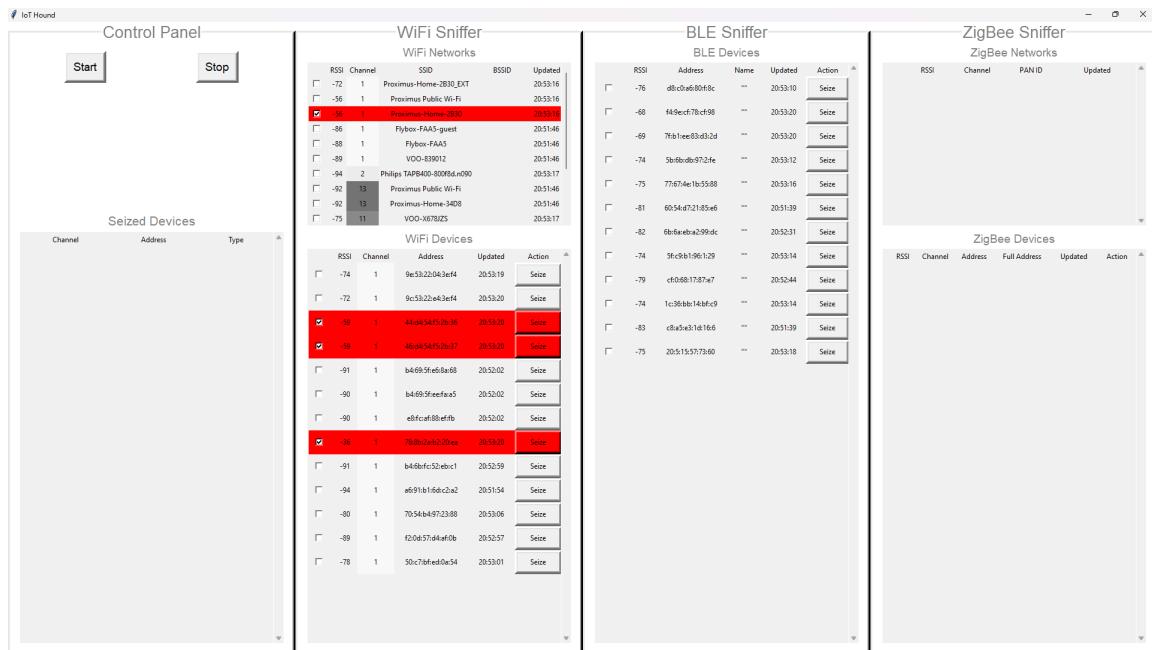
The first screenshot of the testing process. It simply shows the GUI after around 10 seconds of initialization.

F.2 Finding WiFi Router



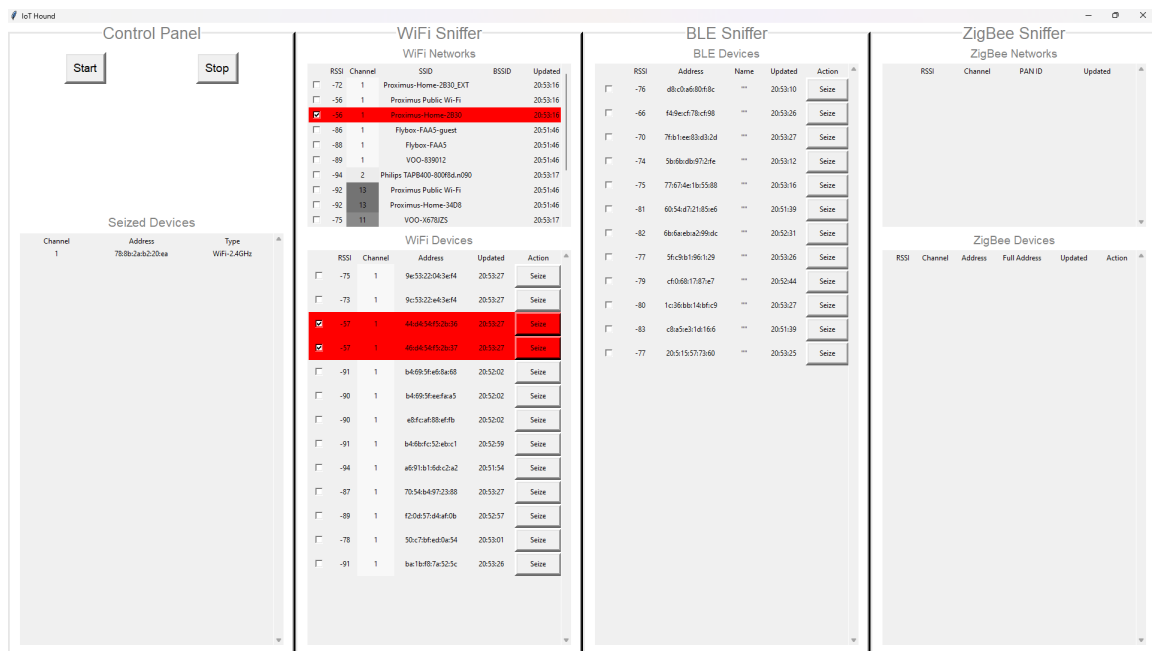
The second screenshot shows that the router was found and the network was selected, alongside with the router antennas.

F.3 Finding Camera



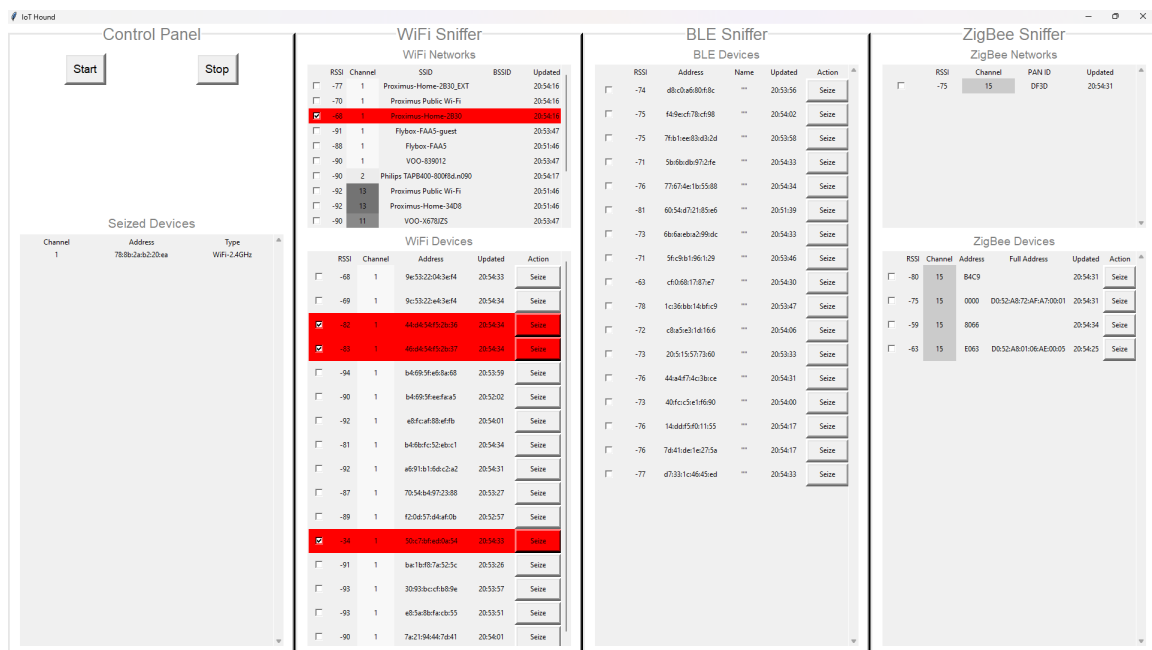
This screenshot shows when the camera was found in the GUI and selected.

F.4 Seizing the Camera



This screenshot shows the GUI after seizing the Smart Camera.

F.5 Finding the Smart Socket



Here, the Smart Socket has an increase in its RSSI, being selected and looked for. The ZigBee scan has already finished as well and it starts showing

F.6 Seizing the Smart Socket

The screenshot shows the Iot Hound GUI with four main sections: Control Panel, WiFi Sniffer, BLE Sniffer, and ZigBee Sniffer. The Control Panel has 'Start' and 'Stop' buttons. The WiFi Sniffer section displays a table of detected WiFi networks. The 'Proximus-Home-3408' network is highlighted in red, indicating it has been seized. The BLE Sniffer section displays a table of detected BLE devices. The device with address '77b7-4e1b-55-88' is highlighted in red, indicating it has been seized. The ZigBee Sniffer section is currently empty.

RSSI	Channel	SSID	BSSID	Updated
-77	1	Proximus-Home-3830_EXT		20:54:16
-70	1	Proximus Public Wi-Fi		20:54:16
-49	1	Proximus-Home-3408		20:54:16
-81	1	Flybox-FAA5-guest		20:53:47
-88	1	Flybox-FAA5		20:51:46
-90	1	VOO-839012		20:53:47
-90	2	Philips TAP8400-800B6-n090		20:54:17
-92	13	Proximus Public Wi-Fi		20:51:46
-92	13	Proximus-Home-3408		20:51:46
-90	11	VOO-8678025		20:53:47

RSSI	Channel	Address	Updated	Action
-68	1	9c:53:22:e4:3e:f4	20:54:39	Seize
-68	1	9c:53:22:e4:3e:f4	20:54:39	Seize
-79	1	44-a4:54:f5:2b:35	20:54:39	Seize
-81	1	46-a4:54:f5:2b:37	20:54:39	Seize
-94	1	b4-69:3f:ee:f6:a5	20:53:59	Seize
-90	1	b4-69:3f:ee:f6:a5	20:53:02	Seize
-92	1	e5:fca:f8:af:b9	20:54:01	Seize
-84	1	b4-6b:fc:52:eb:c1	20:54:38	Seize
-92	1	a6:91:b1:d6:c2:a2	20:54:39	Seize
-87	1	70:54:b4:97:23:88	20:53:27	Seize
-89	1	f2:0d:57:d4:af:0b	20:52:57	Seize
-91	1	ba:1b:f9:7a:52:5c	20:53:28	Seize
-93	1	30:93:bc:cf:b8:9e	20:53:57	Seize
-93	1	e5:5a:8b:fa:cb:55	20:53:51	Seize
-90	1	7a:21:94:44:7d:41	20:54:01	Seize

RSSI	Address	Name	Updated	Action
-74	d8:c7:a8:80:f4:8c		20:53:56	Seize
-75	f4:9c:cf:78:cf:98		20:54:02	Seize
-75	77b7-4e1b-55-88		20:53:58	Seize
-75	5b:8b:db:97:2d:fe		20:54:37	Seize
-83	77b7-4e1b-55-88		20:54:39	Seize
-81	60:54:d7:21:85:e5		20:51:39	Seize
-88	6b:8e:eb:a2:99:dc		20:54:39	Seize
-71	5f:c9:b1:96:1:29		20:53:46	Seize
-65	cf:0b:81:78:f7:e7		20:54:39	Seize
-78	1c:36:b0:14:bfc:9		20:53:47	Seize
-72	cd:a5:e3:1d:16:6		20:54:06	Seize
-73	20:51:55:73:60		20:53:33	Seize
-76	44:a4:7f:4c:3b:ce		20:54:31	Seize
-73	40:fc:c5:e1:f6:90		20:54:00	Seize
-76	14:d8:f5:8b:11:55		20:54:17	Seize
-76	7d:41:de:1e:27:5a		20:54:17	Seize
-76	d7:33:1c:46:45:ed		20:54:35	Seize

RSSI	Channel	PAN ID	Updated
-75	15	DF3D	20:54:37

RSSI	Channel	Address	Full Address	Updated	Action
-81	15	B4C9		20:54:37	Seize
-75	15	0000	D0:52:AB:72:AF:A7:00:01	20:54:37	Seize
-59	15	8066		20:54:34	Seize
-63	15	E063	D0:52:AB:01:06:AE:00:05	20:54:25	Seize

Now, the Smart Socket has been found and seized on the GUI.

F.7 Finding the BLE Headphones

The screenshot shows the Iot Hound GUI with four main sections: Control Panel, WiFi Sniffer, BLE Sniffer, and ZigBee Sniffer. The Control Panel has 'Start' and 'Stop' buttons. The WiFi Sniffer section displays a table of detected WiFi networks. The 'Proximus-Home-3408' network is highlighted in red, indicating it has been seized. The BLE Sniffer section displays a table of detected BLE devices. The device with address '77b7-4e1b-55-88' is highlighted in red, indicating it has been seized. The ZigBee Sniffer section is currently empty.

RSSI	Channel	SSID	BSSID	Updated
-73	1	Proximus-Home-3830_EXT		20:55:16
-53	1	Proximus Public Wi-Fi		20:55:16
-54	1	Proximus-Home-3408		20:55:16
-85	1	Flybox-FAA5-guest		20:55:17
-88	1	Flybox-FAA5		20:51:46
-91	1	VOO-839012		20:55:17
-85	2	Philips TAP8400-800B6-n090		20:55:17
-92	13	Proximus Public Wi-Fi		20:51:46
-92	13	Proximus-Home-3408		20:51:46
-89	11	VOO-8678025		20:55:17

RSSI	Channel	Address	Updated	Action
-59	1	9c:53:22:e4:3e:f4	20:55:23	Seize
-62	1	9c:53:22:e4:3e:f4	20:55:23	Seize
-65	1	44-a4:54:f5:2b:35	20:55:23	Seize
-81	1	46-a4:54:f5:2b:37	20:55:23	Seize
-91	1	b4-69:3f:ee:f6:a5	20:55:23	Seize
-90	1	b4-69:3f:ee:f6:a5	20:55:22	Seize
-90	1	e5:fca:f8:af:b9	20:55:23	Seize
-91	1	fa:8f:ca:57:d8:78	20:55:18	Seize
-80	1	b4-6b:fc:52:eb:c1	20:55:23	Seize
-91	1	a6:91:b1:d6:c2:a2	20:55:08	Seize
-87	1	70:54:b4:97:23:88	20:53:27	Seize
-89	1	f2:0d:57:d4:af:0b	20:52:57	Seize
-91	1	ba:1b:f9:7a:52:5c	20:53:28	Seize
-91	1	30:93:bc:cf:b8:9e	20:55:05	Seize
-93	1	e5:5a:8b:fa:cb:55	20:53:51	Seize
-91	1	7a:21:94:44:7d:41	20:55:18	Seize

RSSI	Address	Name	Updated	Action
-73	d8:c7:a8:80:f4:8c		20:55:23	Seize
-68	f4:9c:cf:78:cf:98		20:55:16	Seize
-73	77b7-4e1b-55-88		20:55:22	Seize
-47	77b7-4e1b-55-88		20:55:23	Seize
-88	77b7-4e1b-55-88		20:55:23	Seize
-81	60:54:d7:21:85:e5		20:51:39	Seize
-88	6b:8e:eb:a2:99:dc		20:55:21	Seize
-72	5f:c9:b1:96:1:29		20:55:01	Seize
-70	cf:0b:81:78:f7:e7		20:55:23	Seize
-73	1c:36:b0:14:bfc:9		20:55:02	Seize
-72	cd:a5:e3:1d:16:6		20:54:06	Seize
-73	20:51:55:73:60		20:53:33	Seize
-70	44:a4:7f:4c:3b:ce		20:55:14	Seize
-74	40:fc:c5:e1:f6:90		20:54:49	Seize
-76	14:d8:f5:8b:11:55		20:54:17	Seize
-76	7d:41:de:1e:27:5a		20:54:17	Seize
-66	d7:33:1c:46:45:ed		20:55:03	Seize
-75	26:37:7a:9d:34:d		20:54:49	Seize

RSSI	Channel	PAN ID	Updated
-45	15	DF3D	20:55:16

RSSI	Channel	Address	Full Address	Updated	Action
-70	15	B4C9	00:0D:4F:00:0B:BC:41:A5	20:55:23	Seize
-65	15	0000	D0:52:AB:72:AF:A7:00:01	20:55:16	Seize
-64	15	8066	00:0D:4F:00:0B:8B:05:92	20:55:20	Seize
-79	15	E063	D0:52:AB:01:06:AE:00:05	20:55:05	Seize

Now that all WiFi devices that appeared on the GUI with a considerably strong signal have been seized (except for the router, to keep the system alive), we went to the BLE process. The headphone has been selected because of the signal strength that it assumed.

F.8 Seizing the BLE Headphones

The screenshot displays the Iot Hound interface with four main panels:

- Control Panel:** Contains 'Start' and 'Stop' buttons.
- Seized Devices:** A table showing two seized devices:

Channel	Address	Type
1	783b2a3b20ea	WiFi-2.4GHz
1	50c78aed0a54	WiFi-2.4GHz
- WiFi Sniffer:** A table showing WiFi networks. The network 'Proximus-Home-3408' is selected.

RSSI	Channel	SSID	BSSID	Updated
-73	1	Proximus-Home-3830_EXT		20:55:16
-53	1	Proximus Public Wi-Fi		20:55:16
-54	1	Proximus-Home-3408		20:55:16
-85	1	Flybox-FAA5-guest		20:55:17
-88	1	Flybox-FAA5		20:51:46
-91	1	VOO-839012		20:55:17
-85	2	Philips TAP8400-800B64-n090		20:55:17
-92	13	Proximus Public Wi-Fi		20:51:46
-92	13	Proximus-Home-3408		20:51:46
-89	11	VOO-1678025		20:55:17
- BLE Sniffer:** A table showing BLE devices. The device '6b6e6ba2b9dc' is selected.

RSSI	Address	Name	Updated	Action
-73	db6c0ab8048c		20:55:27	Seize
-69	649e0cf78cf98		20:55:37	Seize
-70	78b1eed83d32d		20:55:38	Seize
-77	77d74e1b5588		20:55:36	Seize
-81	6054a712185e6		20:51:39	Seize
-70	6b6e6ba2b9dc		20:55:37	Seize
-72	5fcb61961129		20:55:01	Seize
-77	c068b1787e7		20:55:37	Seize
-73	1c36b6b14bfc9		20:55:02	Seize
-72	c8a5a1d1d166		20:54:06	Seize
-73	20515577360		20:53:33	Seize
-73	44a4f74c3b3ce		20:55:27	Seize
-74	40fccc5e1f690		20:54:49	Seize
-76	14dd45401155		20:54:17	Seize
-76	78d41de1e275a		20:54:17	Seize
-66	d7331c4645ed		20:55:03	Seize
-75	26377a7e934d		20:54:49	Seize
- ZigBee Sniffer:** A table showing ZigBee networks. The network 'D032AB72AFA70001' is selected.

RSSI	Channel	Address	Full Address	Updated	Action
-85	15	B4C9	00:0D:4F:00:0B:BC:41:A5	20:55:37	Seize
-85	15	0000	D0:32:AB:72:AFA7:00:01	20:55:37	Seize
-89	15	8066	00:0D:4F:00:0B:8B:05:92	20:55:32	Seize
-89	15	E063	D0:32:AB:01:06:AE:00:05	20:55:25	Seize

After locating the BLE Headphones, they have been seized.

F.9 Selecting the ZigBee Network

The screenshot displays the Iot Hound interface with four main panels:

- Control Panel:** Contains 'Start' and 'Stop' buttons.
- Seized Devices:** A table showing two seized devices:

Channel	Address	Type
1	783b2a3b20ea	WiFi-2.4GHz
1	50c78aed0a54	WiFi-2.4GHz
- WiFi Sniffer:** A table showing WiFi networks. The network 'Proximus-Home-3408' is selected.

RSSI	Channel	SSID	BSSID	Updated
-70	1	Proximus-Home-3830_EXT		20:56:16
-58	1	Proximus Public Wi-Fi		20:56:16
-52	1	Proximus-Home-3408		20:56:16
-82	1	Flybox-FAA5-guest		20:55:47
-89	1	Flybox-FAA5		20:55:47
-90	1	VOO-839012		20:55:47
-93	2	Philips TAP8400-800B64-n090		20:56:17
-92	13	Proximus Public Wi-Fi		20:51:46
-92	13	Proximus-Home-3408		20:51:46
-83	11	VOO-1678025		20:56:17
- BLE Sniffer:** A table showing BLE devices. The device '6b6e6ba2b9dc' is selected.

RSSI	Address	Name	Updated	Action
-74	db6c0ab8048c		20:56:18	Seize
-69	649e0cf78cf98		20:56:11	Seize
-88	78b1eed83d32d		20:56:18	Seize
-65	77d74e1b5588		20:56:09	Seize
-81	6054a712185e6		20:51:39	Seize
-69	6b6e6ba2b9dc		20:55:46	Seize
-71	5fcb61961129		20:56:18	Seize
-73	c068b1787e7		20:55:45	Seize
-72	1c36b6b14bfc9		20:55:02	Seize
-72	c8a5a1d1d166		20:54:06	Seize
-75	20515577360		20:56:18	Seize
-73	44a4f74c3b3ce		20:55:27	Seize
-74	40fccc5e1f690		20:54:49	Seize
-76	14dd45401155		20:54:17	Seize
-76	78d41de1e275a		20:54:17	Seize
-66	d7331c4645ed		20:55:03	Seize
-71	ab2c2e7736443		20:56:09	Seize
- ZigBee Sniffer:** A table showing ZigBee networks. The network 'D032AB72AFA70001' is selected.

RSSI	Channel	Address	Full Address	Updated	Action
-38	15	B4C9	00:0D:4F:00:0B:BC:41:A5	20:56:14	Seize
-48	15	0000	D0:32:AB:72:AFA7:00:01	20:56:17	Seize
-91	15	8066	00:0D:4F:00:0B:8B:05:92	20:56:16	Seize
-83	15	E063	D0:32:AB:01:06:AE:00:05	20:56:05	Seize

Then, after attesting that no more BLE devices were in the scene, the ZigBee network has been selected to analyze.

F.10 Finding Hub and Motion Sensor

The screenshot shows the Iot Hound interface with four main panels: Control Panel, WiFi Sniffer, BLE Sniffer, and ZigBee Sniffer. The Control Panel has 'Start' and 'Stop' buttons. The WiFi Sniffer panel displays a list of WiFi Networks and a list of Seized Devices. The BLE Sniffer panel displays a list of BLE Devices. The ZigBee Sniffer panel displays a list of ZigBee Networks and a list of Seized Devices.

WiFi Networks

RSSI	Channel	SSID	BSSID	Updated
-70	1	Proximus-Home-2830_EXT		20:56:16
-58	1	Proximus Public Wi-Fi		20:56:16
-82	1	Flybox-FAA5-guest		20:55:47
-89	1	Flybox-FAA5		20:55:47
-90	1	VOO-839012		20:55:47
-93	2	Philips TAP8400-800B4-n00		20:56:17
-92	13	Proximus Public Wi-Fi		20:51:46
-92	13	Proximus-Home-3408		20:51:46
-83	11	VOO-867025		20:56:17

WiFi Devices

RSSI	Channel	Address	Updated	Action
-71	1	9c:53:22:e4:3e:f4	20:56:33	Seize
-71	1	9c:53:22:e4:3e:f4	20:56:33	Seize
-40	1	44:a4:54:f5:2b:35	20:56:33	Seize
-41	1	46:a4:54:f5:2b:37	20:56:33	Seize
-91	1	b4:69:3f:ee:f5:a5	20:56:05	Seize
-93	1	b4:69:3f:ee:f5:a5	20:56:05	Seize
-93	1	e5:fca:f8:af:b5	20:56:06	Seize
-89	1	b4:6b:f5:52:ab:c1	20:56:31	Seize
-93	1	a6:91:b1:6d:c2:a2	20:56:03	Seize
-88	1	70:54:b4:97:23:88	20:56:30	Seize
-89	1	f2:0a:57:4a:af:0b	20:52:57	Seize
-91	1	ba:1b:f9:7a:52:5c	20:55:28	Seize
-91	1	30:93:bccf:b8:9e	20:55:05	Seize
-93	1	e5:5a:8f:fa:cb:55	20:55:51	Seize
-91	1	7a:21:94:44:7d:41	20:55:18	Seize
-92	1	90:5c:44:b0:c5:38	20:54:58	Seize

Seized Devices

Channel	Address	Type
1	78:3b:2a:b2:20:ea	WiFi-2.4GHz
1	50:c7:8f:ee:f5:a5	WiFi-2.4GHz
	9a:0b:ab:97:32:fe	BLE

BLE Devices

RSSI	Address	Name	Updated	Action
-78	d8:c7:ad:80:f8:c		20:56:21	Seize
-45	f4:9c:d7:7b:c7:98		20:56:23	Seize
-70	7b:b1:ee:f3:d3:2d		20:56:28	Seize
-45	77:67:4e:1b:55:88		20:56:09	Seize
-81	60:54:e7:21:85:e6		20:51:39	Seize
-49	6b:fa:eb:a2:99:dc		20:55:46	Seize
-70	5f:cb:b1:96:1:29		20:56:29	Seize
-73	c0:68:17:87:e7		20:55:45	Seize
-73	1c:36:b0:14:b4:c9		20:55:02	Seize
-72	c8:a5:ic:h1:d:16:6		20:54:06	Seize
-70	20:51:55:73:60		20:56:29	Seize
-73	44:a4:f7:4c:3b:ce		20:55:27	Seize
-74	40:fc:c5:f1:66:90		20:54:49	Seize
-76	14:dd:f5:40:11:55		20:54:17	Seize
-76	7d:41:de:1e:27:5a		20:54:17	Seize
-66	d7:33:1c:46:45:ed		20:55:03	Seize
-71	a2:c2:c7:73:64:43		20:56:09	Seize

ZigBee Networks

RSSI	Channel	PAN ID	Updated
-37	15	00:00:00:00:00:00:00:00	20:56:02

ZigBee Devices

RSSI	Channel	Address	Full Address	Updated	Action
-30	15	0000	D0:52:AB:72:AF:A7:00:01	20:56:05	Seize
-81	15	8066	00:00:00:00:00:00:00:00	20:56:34	Seize
-87	15	E063	D0:52:AB:01:06:AE:00:05	20:56:25	Seize

The first devices that were located were the Hub and the Motion Sensor.

F.11 Seizing the Motion Sensor

The screenshot shows the Iot Hound interface with four main panels: Control Panel, WiFi Sniffer, BLE Sniffer, and ZigBee Sniffer. The Control Panel has 'Start' and 'Stop' buttons. The WiFi Sniffer panel displays a list of WiFi Networks and a list of Seized Devices. The BLE Sniffer panel displays a list of BLE Devices. The ZigBee Sniffer panel displays a list of ZigBee Networks and a list of Seized Devices.

WiFi Networks

RSSI	Channel	SSID	BSSID	Updated
-49	1	Proximus-Home-2830_EXT		20:57:16
-81	1	Proximus Public Wi-Fi		20:57:16
-82	1	Flybox-FAA5-guest		20:55:47
-89	1	Flybox-FAA5		20:55:47
-90	1	VOO-839012		20:55:47
-93	2	Philips TAP8400-800B4-n00		20:57:17
-95	13	Proximus Public Wi-Fi		20:57:17
-92	13	Proximus-Home-3408		20:51:46
-95	11	VOO-867025		20:57:17

WiFi Devices

RSSI	Channel	Address	Updated	Action
-94	1	9c:53:22:e4:3e:f4	20:57:29	Seize
-89	1	9c:53:22:e4:3e:f4	20:57:30	Seize
-25	1	44:a4:54:f5:2b:35	20:57:30	Seize
-74	1	46:a4:54:f5:2b:37	20:57:30	Seize
-92	1	b4:69:3f:ee:f5:a5	20:57:30	Seize
-93	1	b4:69:3f:ee:f5:a5	20:57:30	Seize
-88	1	e5:fca:f8:af:b5	20:57:30	Seize
-81	1	b4:6b:f5:52:ab:c1	20:57:23	Seize
-92	1	a6:91:b1:6d:c2:a2	20:57:28	Seize
-91	1	70:54:b4:97:23:88	20:56:47	Seize
-89	1	f2:0a:57:4a:af:0b	20:52:57	Seize
-91	1	ba:1b:f9:7a:52:5c	20:55:28	Seize
-91	1	30:93:bccf:b8:9e	20:55:05	Seize
-93	1	e5:5a:8f:fa:cb:55	20:55:51	Seize
-91	1	7a:21:94:44:7d:41	20:55:18	Seize
-92	1	90:5c:44:b0:c5:38	20:54:58	Seize

Seized Devices

Channel	Address	Type
1	78:3b:2a:b2:20:ea	WiFi-2.4GHz
1	50:c7:8f:ee:f5:a5	WiFi-2.4GHz
	9a:0b:ab:97:32:fe	BLE
15	BaC9	ZigBee

BLE Devices

RSSI	Address	Name	Updated	Action
-78	d8:c7:ad:80:f8:c		20:56:21	Seize
-45	f4:9c:d7:7b:c7:98		20:56:23	Seize
-70	7b:b1:ee:f3:d3:2d		20:56:28	Seize
-45	77:67:4e:1b:55:88		20:56:09	Seize
-81	60:54:e7:21:85:e6		20:51:39	Seize
-49	6b:fa:eb:a2:99:dc		20:55:46	Seize
-70	5f:cb:b1:96:1:29		20:56:29	Seize
-73	c0:68:17:87:e7		20:55:45	Seize
-73	1c:36:b0:14:b4:c9		20:55:02	Seize
-72	c8:a5:ic:h1:d:16:6		20:54:06	Seize
-70	20:51:55:73:60		20:56:29	Seize
-73	44:a4:f7:4c:3b:ce		20:55:27	Seize
-74	40:fc:c5:f1:66:90		20:54:49	Seize
-76	14:dd:f5:40:11:55		20:54:17	Seize
-76	7d:41:de:1e:27:5a		20:54:17	Seize
-66	d7:33:1c:46:45:ed		20:55:03	Seize
-71	a2:c2:c7:73:64:43		20:56:09	Seize

ZigBee Networks

RSSI	Channel	PAN ID	Updated
-34	15	0000	20:57:29

ZigBee Devices

RSSI	Channel	Address	Full Address	Updated	Action
-24	15	0000	D0:52:AB:72:AF:A7:00:01	20:57:29	Seize
-46	15	8066	00:00:00:00:00:00:00:00	20:57:29	Seize
-40	15	E063	D0:52:AB:01:06:AE:00:05	20:57:28	Seize

Since the Hub is necessary for the system, only the motion sensor was seized at this moment.

F.12 Finding the Multipurpose Sensor

The screenshot shows the IoT Hound Control Panel with four main sections: Control Panel, WiFi Sniffer, BLE Sniffer, and ZigBee Sniffer. The Control Panel has 'Start' and 'Stop' buttons. The WiFi Sniffer section displays a list of WiFi Networks and a list of WiFi Devices. The ZigBee Sniffer section displays a list of ZigBee Networks and a list of ZigBee Devices. In the ZigBee Networks list, the entry with RSSI -74, Channel 15, PAN ID 0000, and Full Address D052A072AFA70001 is highlighted in red. In the ZigBee Devices list, the entry with RSSI -49, Channel 15, Address 8066, and Full Address D052A072AFA70001 is also highlighted in red.

Then, locating the rest of the devices began, starting with the multipurpose sensor. It was selected and tracked.

F.13 Seizing the Multipurpose Sensor

This screenshot is similar to the previous one, showing the IoT Hound Control Panel. In this state, the 'ZigBee Networks' list shows the entry with RSSI -49, Channel 15, PAN ID 0000, and Full Address D052A072AFA70001 highlighted in red. The 'ZigBee Devices' list shows the entry with RSSI -49, Channel 15, Address 8066, and Full Address D052A072AFA70001 highlighted in red. The 'WiFi Sniffer' and 'BLE Sniffer' sections remain unchanged from the previous screenshot.

After finding it, it was seized.

F.14 Finding the Presence Sensor

The screenshot displays the IoT Hound interface with four main panels: Control Panel, WiFi Sniffer, BLE Sniffer, and ZigBee Sniffer. The Control Panel on the left includes 'Start' and 'Stop' buttons and a 'Seized Devices' table. The WiFi Sniffer panel shows a list of WiFi networks and devices. The BLE Sniffer panel shows a list of BLE devices. The ZigBee Sniffer panel shows a list of ZigBee networks and devices.

Control Panel

Start Stop

Seized Devices

Channel	Address	Type
1	78:3b:2a:b2:20:ea	WiFi-2.4GHz
1	50:c7:b8:ed:0a:54	WiFi-2.4GHz
15	9a:0b:db:97:24:fe	BLE
15	BAC9	ZigBee
15	8066	ZigBee
15	E063	ZigBee

WiFi Sniffer

WiFi Networks

RSSI	Channel	SSID	BSSID	Updated
-69	1	Proximus-Home-2830_EXT		20:58:46
-78	1	Proximus Public Wi-Fi		20:58:46
-79	1	Proximus-Home-2830		20:58:46
-80	1	Flybox-FAA5-guest		20:58:17
-89	1	Flybox-FAA5		20:55:47
-91	1	VOO-839012		20:58:47
-93	2	Philips TA98400-800864-n090		20:58:47
-95	13	Proximus Public Wi-Fi		20:57:17
-92	13	Proximus-Home-3408		20:51:46
-88	11	VOO-8678025		20:58:47

WiFi Devices

RSSI	Channel	Address	Updated	Action
-77	1	9e:53:22:04:3e:f4	20:58:58	Seize
-76	1	9e:53:22:04:3e:f4	20:58:59	Seize
-81	1	44:a4:54:f5:2b:39	20:58:55	Seize
-81	1	44:a4:54:f5:2b:39	20:58:55	Seize
-92	1	b4:69:3f:ed:8a:68	20:58:30	Seize
-93	1	b4:69:3f:ed:8a:68	20:58:30	Seize
-89	1	e8:fc:af:88:af:b9	20:58:58	Seize
-88	1	b4:6b:fc:52:ab:c1	20:58:56	Seize
-93	1	a6:91:b1:6d:c2:a2	20:58:40	Seize
-91	1	70:54:b4:97:23:88	20:56:47	Seize
-89	1	f2:0a:57:4a:af:0b	20:52:57	Seize
-91	1	ba:1b:fb:7a:52:5c	20:55:28	Seize
-91	1	30:93:bccf:b8:b9	20:55:05	Seize
-93	1	e8:5a:8f:ac:cb:55	20:55:51	Seize
-91	1	7a:21:94:44:7d:41	20:55:18	Seize
-92	1	90:5c:a4:b0:c5:38	20:54:58	Seize

BLE Sniffer

BLE Devices

RSSI	Address	Name	Updated	Action
-78	d8:c3:af:80:f8:c		20:56:21	Seize
-65	f4:9e:cf:78:c9:8		20:56:23	Seize
-70	78:b1:ee:d3:d3:2d		20:56:28	Seize
-65	77:b7:4e:1b:55:88		20:56:09	Seize
-81	60:54:d7:21:85:e6		20:51:39	Seize
-69	6b:6e:ba:a2:99:dc		20:55:46	Seize
-70	5f:c9:b1:96:1:29		20:56:29	Seize
-73	c0:68:17:b7:e7		20:55:45	Seize
-73	1c:36:b0:14:bfc:9		20:55:02	Seize
-72	cb:a5:e3:1d:16:6		20:54:06	Seize
-70	20:51:55:73:60		20:56:29	Seize
-73	44:a4:f7:4c:3b:ce		20:55:27	Seize
-74	4b:fc:c5:e1:f6:90		20:54:49	Seize
-76	14:dd:f5:8b:11:55		20:54:17	Seize
-76	7d:41:de:1e:27:5a		20:54:17	Seize
-66	d7:33:1c:46:45:ed		20:55:03	Seize
-71	a2:c2:c7:73:64:43		20:56:09	Seize

ZigBee Sniffer

ZigBee Networks

RSSI	Channel	PAN ID	Updated	Action
-78	15	0000	20:58:38	Seize

ZigBee Devices

RSSI	Channel	Address	Full Address	Updated	Action
-78	15	0000	D0:52:AB:72:AF:A5:00:01	20:58:38	Seize

Then, the presence sensor was selected and tracked.

F.15 Seized Presence Sensor and back to Hub

The screenshot displays the IoT Hound interface with four main panels: Control Panel, WiFi Sniffer, BLE Sniffer, and ZigBee Sniffer. The Control Panel on the left includes 'Start' and 'Stop' buttons and a 'Seized Devices' table. The WiFi Sniffer panel shows a list of WiFi networks and devices. The BLE Sniffer panel shows a list of BLE devices. The ZigBee Sniffer panel shows a list of ZigBee networks and devices.

Control Panel

Start Stop

Seized Devices

Channel	Address	Type
1	78:3b:2a:b2:20:ea	WiFi-2.4GHz
1	50:c7:b8:ed:0a:54	WiFi-2.4GHz
15	9a:0b:db:97:24:fe	BLE
15	BAC9	ZigBee
15	8066	ZigBee
15	E063	ZigBee

WiFi Sniffer

WiFi Networks

RSSI	Channel	SSID	BSSID	Updated
-80	1	Proximus-Home-2830_EXT		20:59:47
-42	1	Proximus Public Wi-Fi		20:59:46
-82	1	Proximus-Home-2830		20:58:46
-90	1	Flybox-FAA5-guest		20:58:17
-89	1	Flybox-FAA5		20:55:47
-91	1	VOO-839012		20:58:47
-87	2	Philips TA98400-800864-n090		20:59:47
-95	13	Proximus Public Wi-Fi		20:57:17
-92	13	Proximus-Home-3408		20:51:46
-91	11	VOO-8678025		20:59:47

WiFi Devices

RSSI	Channel	Address	Updated	Action
-78	1	9e:53:22:04:3e:f4	20:59:47	Seize
-77	1	9e:53:22:04:3e:f4	20:59:47	Seize
-80	1	44:a4:54:f5:2b:39	20:59:40	Seize
-81	1	44:a4:54:f5:2b:39	20:59:47	Seize
-93	1	b4:69:3f:ed:8a:68	20:59:17	Seize
-92	1	b4:69:3f:ed:8a:68	20:59:17	Seize
-91	1	e8:fc:af:88:af:b9	20:59:20	Seize
-86	1	b4:6b:fc:52:ab:c1	20:59:34	Seize
-92	1	a6:91:b1:6d:c2:a2	20:59:17	Seize
-94	1	70:54:b4:97:23:88	20:59:48	Seize
-94	1	3c:a0:67:3b:01:84	20:59:37	Seize
-89	1	f2:0a:57:4a:af:0b	20:52:57	Seize
-91	1	ba:1b:fb:7a:52:5c	20:55:28	Seize
-91	1	30:93:bccf:b8:b9	20:55:05	Seize
-86	1	e8:5a:8f:ac:cb:55	20:59:20	Seize
-88	1	7a:21:94:44:7d:41	20:59:59	Seize

BLE Sniffer

BLE Devices

RSSI	Address	Name	Updated	Action
-78	d8:c3:af:80:f8:c		20:56:21	Seize
-65	f4:9e:cf:78:c9:8		20:56:23	Seize
-70	78:b1:ee:d3:d3:2d		20:56:28	Seize
-65	77:b7:4e:1b:55:88		20:56:09	Seize
-81	60:54:d7:21:85:e6		20:51:39	Seize
-69	6b:6e:ba:a2:99:dc		20:55:46	Seize
-70	5f:c9:b1:96:1:29		20:56:29	Seize
-73	c0:68:17:b7:e7		20:55:45	Seize
-73	1c:36:b0:14:bfc:9		20:55:02	Seize
-72	cb:a5:e3:1d:16:6		20:54:06	Seize
-70	20:51:55:73:60		20:56:29	Seize
-73	44:a4:f7:4c:3b:ce		20:55:27	Seize
-74	4b:fc:c5:e1:f6:90		20:54:49	Seize
-76	14:dd:f5:8b:11:55		20:54:17	Seize
-76	7d:41:de:1e:27:5a		20:54:17	Seize
-66	d7:33:1c:46:45:ed		20:55:03	Seize
-71	a2:c2:c7:73:64:43		20:56:09	Seize

ZigBee Sniffer

ZigBee Networks

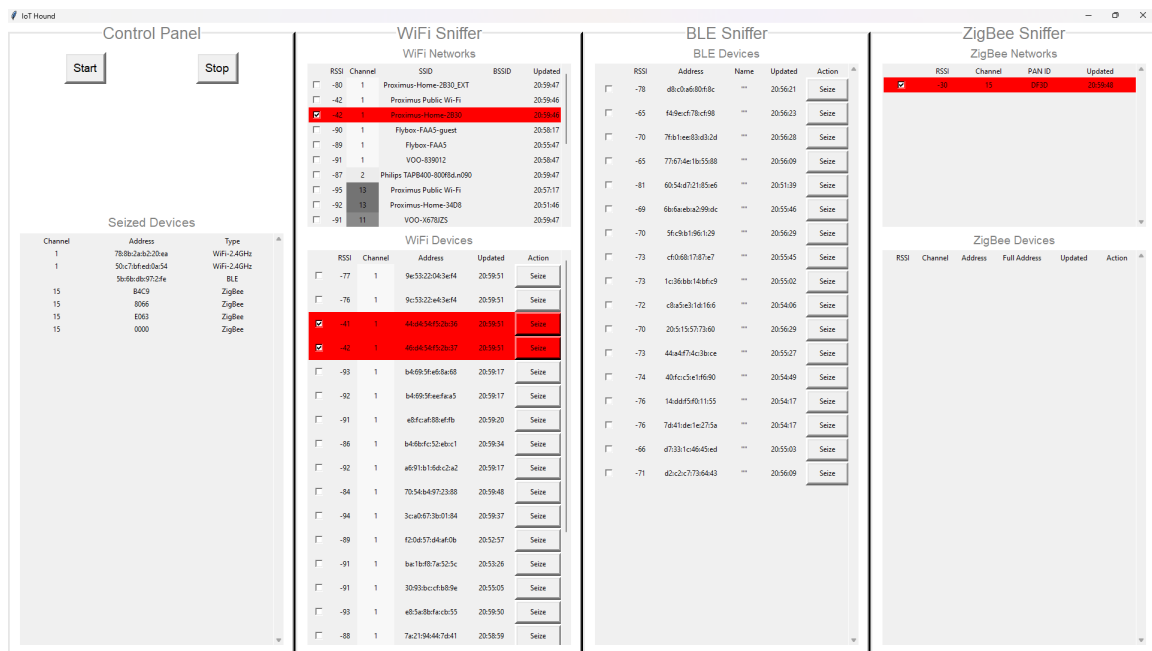
RSSI	Channel	PAN ID	Updated	Action
-78	15	0000	20:58:37	Seize

ZigBee Devices

RSSI	Channel	Address	Full Address	Updated	Action
-78	15	0000	D0:52:AB:72:AF:A5:00:01	20:58:37	Seize

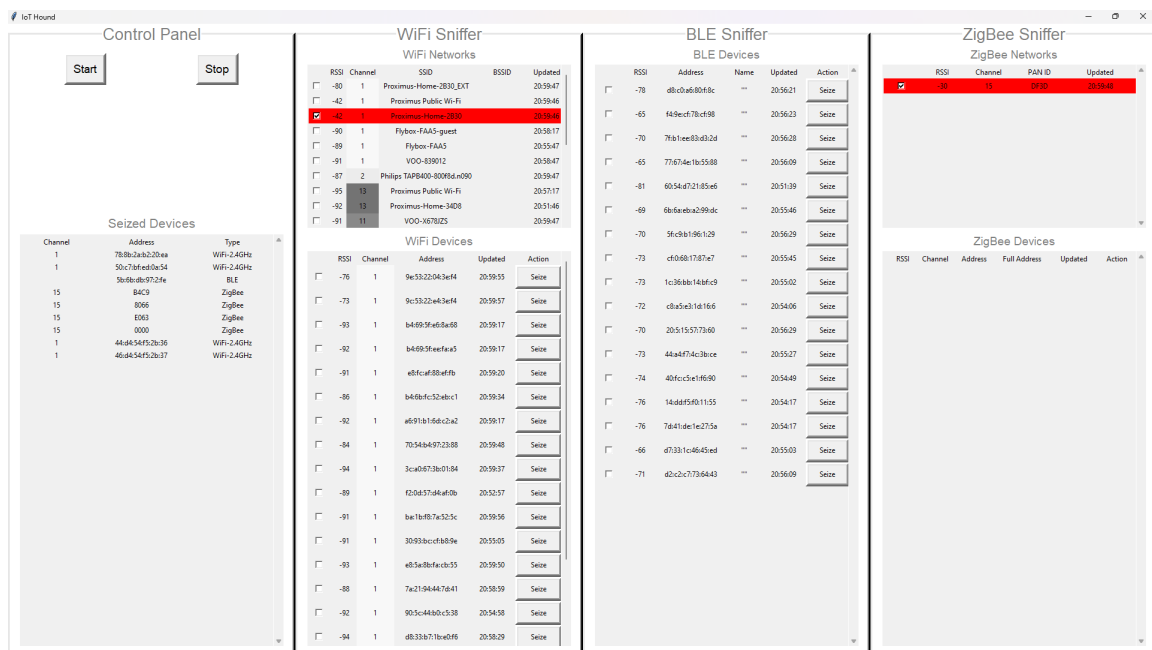
It was seized and then we came back to the Hub to seize it as well.

F.16 Seizing the Hub



The Hub was then seized.

F.17 Seizing the Router



And lastly, since there are no more devices connected, the router was seized. It was left for last because it makes the connectivity between most of the devices, and turning it

off could cause the devices to stop emitting signals. This should be a step taken by the investigators in a crime scene.

Bibliography

- [1] Jeremiah Lasquety-Reyes. *Number of users of smart homes worldwide 2018-2027*. <https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-in-the-world>. Accessed: 27/03/2023. 2023.
- [2] Felix Wortmann and Kristina Flüchter. “Internet of things”. In: *Business & Information Systems Engineering* 57.3 (2015), pp. 221–224.
- [3] Philippa Biggs et al. “Harnessing the Internet of Things for global development”. In: (2016).
- [4] International Organization for Standardization. *ISO/IEC JTC1 Internet of Things primary report 2014*. 2017. URL: https://www.iso.org/files/live/sites/isoorg/files/developing_standards/docs/en/internet_of_things_report-jtc1.pdf.
- [5] Karen Rose, Scott Eldridge, and Lyman Chapin. “The internet of things: An overview”. In: *The internet society (ISOC)* 80 (2015), pp. 1–50.
- [6] Muhammad Burhan et al. “IoT elements, layered architectures and security issues: A comprehensive survey”. In: *sensors* 18.9 (2018), p. 2796.
- [7] Mohammad Ali Jabraeil Jamali et al. “IoT Architecture”. In: *Towards the Internet of Things: Architectures, Security, and Applications*. Cham: Springer International Publishing, 2020, pp. 9–31. ISBN: 978-3-030-18468-1. DOI: [10.1007/978-3-030-18468-1_2](https://doi.org/10.1007/978-3-030-18468-1_2). URL: https://doi.org/10.1007/978-3-030-18468-1_2.
- [8] “IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPAN)”. In: *IEEE Std 802.15.4-2003* (2003), pp. 1–680. DOI: [10.1109/IEEESTD.2003.94389](https://doi.org/10.1109/IEEESTD.2003.94389).
- [9] Connectivity Standards Alliance. *Zigbee Frequently Asked Questions*. URL: <https://csa-iot.org/all-solutions/zigbee/zigbee-faq/>.
- [10] Shams Zawoad and Ragib Hasan. “FAIoT: Towards Building a Forensics Aware Eco System for the Internet of Things”. In: *2015 IEEE International Conference on Services Computing*. 2015, pp. 279–284. DOI: [10.1109/SCC.2015.46](https://doi.org/10.1109/SCC.2015.46).
- [11] Maria Stoyanova et al. “A survey on the internet of things (IoT) forensics: challenges, approaches, and open issues”. In: *IEEE Communications Surveys & Tutorials* 22.2 (2020), pp. 1191–1221.
- [12] Edewede Oriwoh et al. “Internet of things forensics: Challenges and approaches”. In: *9th IEEE International Conference on Collaborative computing: networking, Applications and Worksharing*. IEEE. 2013, pp. 608–615.

- [13] Hany F Atlam et al. “Internet of things forensics: A review”. In: *Internet of Things* 11 (2020), p. 100220.
- [14] I Techniques. “Iso/iec 27037: 2012 information technology security techniques guidelines for identification collection acquisition and preservation of digital evidence”. In: *ISO/IEC-The standard was published in October* (2012).
- [15] ISO/IEC. “Information Technology—Security Techniques—Guidelines for the Analysis and Interpretation of Digital Evidence”. In: *ISO/IEC 27042* (2015).
- [16] European Network of Forensics Sciences Institutes. *Best Practice Manual for the Forensic Examination of Digital Technology*. en. 2015. URL: https://enfsi.eu/wp-content/uploads/2016/09/1._forensic_examination_of_digital_technology_0.pdf.
- [17] Timothy Grance et al. *Guide to Integrating Forensic Techniques into Incident Response*. en. 2006. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=50875.
- [18] François Bouchaud, Thomas Vantroys, and Gilles Grimaud. “Evidence Gathering in IoT Criminal Investigation”. In: *Digital Forensics and Cyber Crime*. Ed. by Sanjay Goel et al. Cham: Springer International Publishing, 2021, pp. 44–61. ISBN: 978-3-030-68734-2.
- [19] Osman Yakubu, N.C. Babu, and O. Adjei. “A review of digital forensic challenges in the internet of things (IOT)”. In: *International Journal of Mechanical Engineering and Technology* 9 (Jan. 2018), pp. 915–923.
- [20] Pantaleon Lutta et al. “The complexity of internet of things forensics: A state-of-the-art review”. In: *Forensic Science International: Digital Investigation* 38 (2021), p. 301210.
- [21] Khalil Al-Hussaeni et al. “A Review of Internet of Things (IoT) Forensics Frameworks and Models”. In: *Information Systems*. Ed. by Maria Papadaki et al. Cham: Springer Nature Switzerland, 2023, pp. 515–533. ISBN: 978-3-031-30694-5.
- [22] Victor R Kebande and Indrakshi Ray. “A generic digital forensic investigation framework for internet of things (iot)”. In: *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE. 2016, pp. 356–362.
- [23] Pantaleon Lutta et al. “The complexity of internet of things forensics: A state-of-the-art review”. In: *Forensic Science International: Digital Investigation* 38 (2021), p. 301210.
- [24] Arnoud Goudbeek, Kim-Kwang Raymond Choo, and Nhien-An Le-Khac. “A forensic investigation framework for smart home environment”. In: *2018 17th IEEE international conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE. 2018, pp. 1446–1451.
- [25] Abdul Boztas, ARJ Riethoven, and Mark Roeloffs. “Smart TV forensics: Digital traces on televisions”. In: *Digital Investigation* 12 (2015), S72–S80.
- [26] Soram Kim et al. “Smart home forensics—data analysis of IoT devices”. In: *Electronics* 9.8 (2020), p. 1215.

- [27] Akshay Awasthi et al. “Welcome pwn: Almond smart home hub forensics”. In: *Digital Investigation* 26 (2018), S38–S46.
- [28] Saad Alabdulsalam et al. “Internet of things forensics—challenges and a case study”. In: *Advances in Digital Forensics XIV: 14th IFIP WG 11.9 International Conference, New Delhi, India, January 3-5, 2018, Revised Selected Papers 14*. Springer. 2018, pp. 35–48.
- [29] Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. “EMvidence: a framework for digital evidence acquisition from IoT devices through electromagnetic side-channel analysis”. In: *Forensic Science International: Digital Investigation* (2020).
- [30] Orin S Kerr. *Searching and seizing computers and obtaining electronic evidence in criminal investigations*. Office of Legal Education, Executive Office for United States Attorneys, 2001.
- [31] François Bouchaud. “Analyse forensique des écosystèmes intelligents communicants de l’internet des objets”. PhD thesis. Université de Lille, 2021.
- [32] Sundresan Perumal, Norita Md Norwawi, and Valliappan Raman. “Internet of Things (IoT) digital forensic investigation model: Top-down forensic approach methodology”. In: *2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC)*. IEEE. 2015, pp. 19–23.
- [33] Miloslava Plachkinova, Au Vo, and Ala Alluhaidan. “Emerging trends in smart home security, privacy, and digital forensics”. In: (2016).
- [34] Hany F Atlam et al. “Internet of things forensics: A review”. In: *Internet of Things* 11 (2020), p. 100220.
- [35] Mark Ciampa. *CWNA guide to wireless LANs*. Cengage Learning, 2012.
- [36] Sheila Frankel et al. *NIST Special Publication 800-97, Establishing Wireless Robust Security Networks*. Feb. 2007.
- [37] “IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”. In: *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)* (2021), pp. 1–4379. DOI: [10.1109/IEEESTD.2021.9363693](https://doi.org/10.1109/IEEESTD.2021.9363693).
- [38] Chia-Hsin Cheng and Chung-Ching Ho. “Implementation of multi-channel technology in ZigBee wireless sensor networks”. In: *Computers & Electrical Engineering* 56 (2016), pp. 498–508.
- [39] Dong-Kyu Kang, Hyung-Sin Kim, and Saewoong Bahk. “ORPL-DT: opportunistic routing for diverse traffic in multihop IoT networks”. In: *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE. 2017, pp. 1–6.
- [40] “IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)”. In: *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)* (2011), pp. 1–314. DOI: [10.1109/IEEESTD.2011.6012487](https://doi.org/10.1109/IEEESTD.2011.6012487).
- [41] *Autopsy of a ZigBee frame*. URL: <https://lucidar.me/en/zigbee/autopsy-of-a-zigbee-frame/>.

- [42] *ZigBee Specification*. URL: <https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf>.
- [43] Gough Lui et al. “Differences in RSSI readings made by different Wi-Fi chipsets: A limitation of WLAN localization”. In: *2011 International conference on localization and GNSS (ICL-GNSS)*. IEEE. 2011, pp. 53–57.
- [44] Nordic Semiconductors. *Programming the nRF52840 Dongle*. [Online; accessed 25-May-2023]. 2022-09-06. URL: https://infocenter.nordicsemi.com/topic/ug_nc_programmer/UG/nrf_connect_programmer/ncp_programming_dongle.html.
- [45] Nordic Semiconductors. *Programming the nRF BLE Firmware*. [Online; accessed 25-May-2023]. 2022-09-06. URL: https://infocenter.nordicsemi.com/topic/ug_sniffer_ble/UG/sniffer_ble/programming_firmware.html.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl