Offensive and defensive approaches for wireless communication protocols security in IoT

Romain Cayre

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *30/06/2022* **par :**

Romain CAYRE

Offensive and defensive approaches for wireless communication protocols security in IoT

JURY

Aurélien FRANCILLON	Professeur des universités	Rapporteur
Mathieu CUNCHE	Maître de conférences	Rapporteur
Mohamed KAANICHE	Directeur de recherche	Directeur de thèse
Guillaume AURIOL	Maître de conférences	Co-directeur de thèse
Vincent NICOMETTE	Professeur des universités	Examinateur
Maryline LAURENT	Professeure des universités	Examinatrice
Valérie VIET TRIEM TONG	Professeure des universités	Examinatrice
Damien CAUQUIL	Ingénieur	Examinateur
Géraldine MARCONATO	Ingénieure	Membre invitée du jury
Romain VIGNER	Ingénieur	Membre invité du jury

École doctorale et spécialité :

EDSYS : Informatique 4200018

Unité de Recherche :

Laboratoire d'analyse et d'architecture des systèmes

Directeur(s) de Thèse :

Guillaume AURIOL et *Mohamed KAÂNICHE*

Rapporteurs :

Aurélien FRANCILLON et *Mathieu CUNCHE*

Acknowledgments

During these last three years, this research work has occupied most of my thoughts and learned me a lot about myself. For me, these three years have been one of the most exciting and beautiful chapter of my life, and I realize how lucky I have been to be surrounded and supported by amazing people that helped me to become the person I am today, and forged my passion for science and computer security. These few words are dedicated to them: my colleagues, my friends and my family.

First, I want to thank Liviu Nicu and Mohamed Kaâniche, CNRS Research Directors at the Laboratory of Analysis and Architecture of Systems, for welcoming me in the laboratory, and Hélène Waeselynck, Team Leader, for allowing me to perform this research work in the Tolérance aux Fautes et Sûreté de fonctionnement (TSF) team. This environment was a perfect match for me, and it allowed me to work in a stimulating place, surrounded by brilliant and inspiring people. I also would like to sincerely thank Aurélien Francillon, Matthieu Cunche, Marilyne Laurent, Damien Cauquil and Romain Vigner for accepting to share their expertise with me and review this PhD thesis. Thanks to them, I enjoyed every second of my PhD defense, and it was a real pleasure and honor for me to present them my work and answer their questions. I'm very proud that they consider this work as a relevant contribution to IoT security, and I've been particularly touched by their words and encouragements. I would like to express special thanks to my rapporteurs, Aurélien Francillon and Matthieu Cunche, for their valuable feedback on this manuscript, and to Damien Cauquil for his inspiring contributions to wireless security and the time he regularly took during this thesis to provide me some feedback and sharing his technical expertise with me.

I want to warmly thank my supervisors for their advices, their constant support and their friendship. There are no words strong enough to say to Vincent Nicomette how precious his humanity, his advices and his kindness were for me. I want him to know how grateful and respectful I am for the incredible work he did and the person he is. I want to express to Guillaume Auriol my friendship and my appreciation of the constant efforts he made to support me during this research work. Without him, his scientific advices, his empathy and his sense of humor, these three years would have been way more difficult, and I'm already missing our passionated talks about computer science, teaching, electronics or physics. I also would like to thank Mohamed Kaâniche for his kind and caring support and the great value of his scientific expertise. His feedback and advices have been incredibly useful for me, and learned me a lot about what a research work is and how to conduct it properly. Finally, I want to express to Geraldine Marconato how pleasant it was to work with her, and how valuable her feedback, her advices and her enthusiasm was for me. I wish her the best for her new career at Airbus, and hope she will continue to build collaboration between the academic and the industrial world, allowing other students to discover how beautiful and stimulating computer science can be.

Much credit of this work is to be given to my colleagues and friends, especially Jonathan Roux, Florent Galtier and Clement Chaine. I want to thank sincerely Jonathan Roux for his warm welcome during my internship, his constant support during this thesis and the inspiring ideas he shared with me. I want him to know that his empathy, his optimism and his kindness helped me a lot during the unavoidable periods of doubts that happened during this research work. I want to thank Florent Galtier for his help on many topics covered by this thesis. Collaborating with him during these three years was a pleasure, and I hope he enjoyed these stimulating conversations as much as I do. I also want to highlight the impressive work performed by Clement Chaine during his internship and thank him for his support and his enthusiasm. Last but not least, many PhD students and supervisors from the LAAS-CNRS TSF team contributed to make these three years a beautiful experience, and I want them to know how proud I am to have been a member of such a great research team.

Moreover, I would like to thank my friends, who were always here for me during this PhD thesis. A lot of them took care of me during these three years, probably more than myself. Thanks to Alicia Vigny for being the wonderful friend I needed during these years, and welcoming me with a warm tea and a lot of love when I needed it. Thanks to Pierre Lefebvre for his constant support and his ability to turn every moment, even the worst one, into a good joke. It was incredibly precious for me. Thanks to Cédric Guerri and Rose-Hélène Michon for every second we spent together and for the exciting discussions about science, flowers, theater, litterature, politics, movies and life. I want to thank Pierre-Hugo Coste for his delicious toasted sandwiches and for his friendship. Thanks to Damien Gales for sharing with me way too much stimulating documentaries and for contributing to make humanity better. Thanks to Pauline Cayzac for all the beautiful moments we spent together.

Finally, I want to warmly thank my brother and my parents for their constant support and love. Their presence during my PhD defense contributed a lot to this wonderful experience, and they have always been here when I needed them. I want them to know how proud I am of our family and how much I love them.

Contents

List of Figures

List of Tables

Introduction

In the recent years, there has been a fundamental shift in the world linked to the rapid and massive expansion of a new kind of devices, so-called connected objects, that are spreading everywhere in our daily life. These new devices aim to provide connectivity to a wide variety of physical objects used in our daily life, allowing them to provide new features and an improved user-experience. The interconnection of these cyber-physical systems is commonly named *Internet of Things*. The fast and early adoption of these new devices by end-users shows that there is a real interest for these emerging technologies. A wide variety of devices, from smartwatches to medical devices, are already deployed in the wild and massively used by the general public and the professional circles.

This situation is a real game changer for computer science: indeed, many characteristics of these new devices challenge numerous assumptions and models commonly used in computer science, especially from a security perspective. The most obvious impact of this technology is linked to the fact that a connected device is intended to interact with the physical world using sensors or actuators by design. As a consequence, an attacker able to compromise such a system could cause considerable damages, not only to the system itself or the data it processes, but also to the physical world. Depending on the considered objects, the consequences of such an attack may be critical: for example, a compromised medical device could cause severe injuries or even death in extreme cases. These new risks dramatically highlight the necessity of securing these devices.

Another key characteristic of these devices is that they are generally limited in terms of resources: most of them being dedicated to a mobile use, they are mostly based on embedded systems with limited resources and optimized for low energy consumption. While this choice is logical according to the functional constraints of these devices, it has severe consequences upon the security of these systems. Indeed, multiple security approaches that are commonly implemented on traditional systems cannot be easily embedded in such devices because of their computational cost. For example, many mitigation measures aiming at hardening traditional systems, such as ASLR or cryptography, are too costly to apply to these types of devices (because of the balance between resources and functional requirements).

Another characteristic of these technologies is related to the connectivity they provide. Indeed, each connected object needs to interact with other systems, using a given communication medium. While any type of medium could theoretically be used, the use of wireless communication protocols to provide this connectivity is a common practice in the Internet of Things industry. From a functional perspective, this choice is coherent and relevant: it allows the development of a wide variety of devices, including mobile devices such as smartwatches, with very few constraints regarding their interconnection. This choice has led to the development of multiple

wireless communication protocols in recent years, engaged in a fierce competition to conquer this new market and meet the technical requirements of these new devices.

However, the emergence of these new wireless communication protocols introduces many challenges from a security perspective. Indeed, no standard solution has yet emerged, leading to the co-existence of dozens of heterogeneous protocols in the same environments. Some of these protocols, such as Zigbee, Thread or Bluetooth Low Energy are well known, their specifications being publicly available, while others, such as ANT or Enhanced ShockBurst, are proprietary and poorly documented, resulting in black box technologies that are extremely difficult to evaluate and secure. The interconnection of connected objects using these wireless protocols also generates complex and dynamic topologies, mainly due to the fact that many of them are dedicated to mobile use. Moreover, because the multiple wireless communication protocols commonly used in IoT are based on peer-to-peer communications, there are generally no gateways nor central nodes to comprehensively monitor and analyze traffic to identify and mitigate threats.

The new paradigm emerging from the complex interconnection of connected objects, exchanging potentially sensitive data through heterogeneous wireless communication protocols sharing similar frequency bands or physical layers, raises multiple issues from a security perspective. First, evaluating the attack surface associated with the deployment of wireless technologies is a challenging task. While a considerable number of threats inherent to wireless media have already been studied and discussed in the literature, the increasing number of vulnerabilities recently discovered in wireless protocol stacks shows that further investigations are needed to accurately identify these threats. This involves developing a deeper understanding of the internals of these stacks, as well as innovative methods to analyze the lower layers, which are generally difficult to instrument or monitor. Second, the specific threats linked to the co-existence of these protocols in the same environment due to the IoT context have not been actively studied for now. New attack vectors may take advantage of this situation, exploiting the wild deployment of mobile connected devices to reach new targets. Finally, the relevance of the deployed security models in this new context can also be questioned : for example, classic mitigations such as perimetric security could be potentially defeated by an attacker exploiting these types of attack vectors. Similarly, deploying traditional security solutions such as intrusion detection systems (IDS) or intrusion prevention systems (IPS) is extremely difficult because of multiple factors, such as the absence of central node, the dynamic nature of IoT environments and the heterogeneity of wireless protocols. Adapting these solutions to this new context is another challenge that remains unsolved for now.

This PhD thesis is a contribution to the identification and mitigation of the potential security issues related to the design of these wireless communication protocols and their co-existence in IoT environments. It mainly focuses on the security of peer-to-peer wireless communication protocols commonly used in IoT, such as Zigbee or Bluetooth Low Energy, especially from a low level perspective. More precisely, the background research question that motivated this work is the following

one: *how to identify, assess and mitigate the new threats linked to the deployment of peer-to-peer wireless communication protocols in the IoT context ?*

This PhD thesis explores this problem both from an offensive and a defensive perspective. The offensive contributions are mainly focused on vulnerability analysis, highlighting critical low level attack vectors in wireless communication protocols commonly used in IoT, such as Bluetooth Low Energy or Zigbee. We also contribute to the field by providing offensive tools as open-source software, facilitating future research works on IoT security. The defensive contributions are dedicated to Intrusion Detection and Prevention in IoT wireless networks, exploring two innovative approaches adapted to the IoT context in order to both detect and prevent attacks targeting the wireless protocols. More precisely, the contributions of this PhD thesis are the following ones:

- We discovered and explored a new attack vector which could be critical in the context of IoT, allowing to take advantage of a transceiver dedicated to a given wireless protocol to perform pivoting attacks targeting another wireless protocol that is not natively supported, by exploiting similarities in their physical layers. We have demonstrated multiple critical attack scenarios based on this vector and implemented them on several off-the-shelf devices, including a smartphone or a connected object.

- We discovered a critical injection vulnerability in the Bluetooth Low Energy protocol design, which is currently deployed on a wide number of devices, allowing to perform multiple critical offensive scenarios. We practically demonstrated its feasibility and highlighted that every Bluetooth Low Energy-enabled device implementing the specification is vulnerable.

- We designed and implemented an offensive auditing framework for wireless communication protocols, aiming at standardizing the development of offensive tools and allowing to interact with up to six wireless protocols. This framework currently supports a wide variety of offensive hardware tools and implements a large number of attacks that can be easily customized and combined to build complex attack workflows, considerably facilitating both the vulnerability analysis of these protocols and the evaluation of defensive solutions.

- We designed a decentralized Intrusion Detection System embedded in Bluetooth Low Energy devices, and showed that it can efficiently detect low level vulnerabilities targeting the protocol. We implemented a defensive instrumentation framework aiming at facilitating the development of detection modules and the analysis of the internals of the wireless stacks.

- We built an experimental wireless firewall based on a reactive jamming approach, allowing to monitor, intercept or drop the traffic generated by two widely used wireless protocols. We performed a set of preliminary experiments demonstrating that it may efficiently prevent low level attacks.

Let us note that the contributions of these research works covered the security of Internet of Things in various contexts, from smart home to smart factory. Moreover, these research works have been co-directed by the LAAS-CNRS laboratory and Apsys.Lab, which is a company specialized in Aircraft Security and Industry 4.0, allowing us to take into account an insightful industrial perspective.

This document is divided into three main parts. Part I describes the context of Internet of Things from a security perspective and introduce the state of the art of wireless security, Part II describes the offensive contributions and discuss their impact upon the security of connected objects while Part III introduces our defensive contributions aiming to explore new Intrusion Detection and Prevention approaches.

Part I

Internet of Things security overview

Chapter 1

Internet of things: a security perspective

Contents

This first chapter introduces the context of this PhD thesis, highlights the challenges that have motivated this work and outlines the contributions.

We first introduce the context of Internet of Things expansion, underlining its main characteristics and guidelines. Then, we discuss the impact of this situation from a security perspective, underlining the security issues linked to this new paradigm. Finally, we discuss the scientific challenges implied by this situation that have motivated this work and introduce our contributions.

1.1 General context

In the recent years, a new kind of devices has been actively developed and deployed both in public and professional environments, named "connected objects". These devices aim to implement new features into physical objects by providing them with connectivity, their interconnection with existing networks and especially the Internet network forming the so-called "Internet of Things" (or IoT).

Providing relevant definitions of terms like connected objects or Internet of Things is already a challenge, considering the increasing amount of heterogeneous technologies used in the wild that are making use of these terms. Indeed, depending on the use case involved, from health care to domotics, it seems that multiple visions of Internet of Things coexist, and we lack a global picture that could harmonize these specific visions. This situation has been discussed multiple times in the literature these past few years, without leading to a consensus as far as we know.

In this work, we will use the definitions proposed by B. Dorsemaine et al. in [Dorsemaine 2015], which attempt to embrace the variety of use cases and technologies in general definitions.

(1) Connected object. *Sensor(s) and/or actuator(s) carrying out a specific function and that are able to communicate with other equipment. It is part of an infrastructure allowing the transport, storage, processing and access to the generated data by users or other systems.*

(2) Internet of Things. *Group of infrastructures interconnecting connected objects and allowing their management, data mining and the access to the data they generate.*

From these definitions, we can underline the key aspects that define a connected object and its environment:

- Interaction with the physical world through sensor(s) and/or actuator(s),
- Capacity to communicate with other equipment,
- Data generation and processing,
- Part of an infrastructure dedicated to the transport, storage, processing and access to the data.

While these key aspects are obviously very general, they already provide us a good overview of the implications resulting from this deployment. First, we can note that these systems are able to interact with the physical world using sensor(s) or/and actuator(s). From a functional perspective, this capability allows a lot of new exciting use cases and applications by expanding the limits of the computing world. These systems being already deployed and massively used, we can already observe in our daily life how useful and innovative they can be. For example, a smartwatch can monitor your heart rate during a sport session or record indicators allowing to improve your sleep quality. A connected insulin pump may considerably improve the life quality of diabetics people. Your connected alarm may alert you during a burglary attempt. Some of these devices, such as a connected pacemaker, can even sometimes save lives. However, from a security perspective, we can also note that this capability opens up a new critical attack surface, allowing a potential attacker to impact the physical world and cause severe damages. Your smartwatch can now be compromised, allowing an attacker to collect sensitive information about your health state or your position. A security flaw in the connected alarm system may allow to disable it remotely, facilitating the physical intrusion. Finally, health systems such as insulin pumps or pacemakers could be used to hurt or even kill their user if they are compromised. The ambivalence of this situation dramatically highlights the necessity to build secure and reliable connected objects, especially when the attack impact is high. Indeed, the compromise of such systems being relatively simple in most cases, the risk level is directly related to the impact.

The second aspect mentioned in these definitions is the capacity to communicate with other equipment, which is indeed a key characteristic of these devices. This communication may allow to control the device and its actuators or collect data from the sensors. It can theoretically be based on any kind of mediums, from wired to wireless mediums such as Radio Frequencies (RF) or Infrared (IR). The wide variety of communication mediums available gives a great flexibility to IoT designers, allowing them to adopt the solution that is the most suited to fit their requirements. For example, a static sensor in a building may be connected to the company network using Ethernet, while a mobile smartwatch could use Bluetooth Low Energy to allow a direct communication with the user's smartphone while saving power. However, this multiplicity of communications protocols complicates the security analysis of IoT environments, and leads to the co-existence of heterogeneous technologies increasing the complexity of such environments. In this work, we mainly focus on wireless communication protocols such as Zigbee or Bluetooth Low Energy, that have been widely deployed in IoT environments in the recent years: we introduce these protocols and discuss their impact upon IoT security in section 1.2.

The third key aspect emphasized by these definitions is the fact that connected objects are intended to generate and process data. In the recent years, we have observed the development of data-oriented technologies and approaches, aiming at analyzing and exploiting huge amount of data to achieve different objectives. Internet of Things is especially interesting according to these new technologies, as it may allow to provide valuable data sets that could lead to exciting new applications and use cases. On the other hand, the undeniable value of data generated and processed by connected devices could become a privileged target for attackers, leading to significant security risks. Indeed, some connected objects such as medical or sport oriented devices, process sensitive or personal data, sometimes without a full understanding of the associated risks by the end-user. In this context, preserving the privacy of users becomes a priority, and protecting the confidentiality, integrity and availability of the data processed and generated by connected objects remains a significant challenge.

Finally, the fourth key aspect underlined by these definitions is the existence of an infrastructure allowing the interconnection of connected objects, the so-called Internet of Things. In the current state of connected objects deployment, we can note that this infrastructure cannot be assimilated to a traditional network based on an homogeneous and standardized technology, such as the IP network. The Internet of Things is mainly composed of an interconnection of multiple networks, based on very heterogeneous technologies. This situation generates a lot of complexity and requires the deployment of equipment aiming at allowing communications between heterogeneous communication protocols, such as gateways. Some specific use cases linked to IoT generate even more complexity in the resulting environments: for example, some devices being dedicated to a mobile use, they can lead to complex and dynamic network topologies, in which it is difficult to identify whether the presence of a given node is legitimate or not. From the previous observations,

we already know how critical this infrastructure is from a security perspective: the complexity resulting from this interconnection makes it especially difficult to analyze and secure, and we obviously need security-oriented research to adapt to this situation and secure the Internet of Things.

This high level overview of the Internet of Things underlines both the incredible potential of connected objects from a functional perspective and the significant risks it raises from a security perspective: securing these networks and devices is obviously a major concern. However, the fast expansion of the Internet of Things is a game changer for computer science, as it deeply impacts various aspects of computing. The fast emergence of new heterogeneous technologies, their massive interconnection, the wide range of new use cases and applications, the generation of complex and dynamic topologies, create a new computing paradigm which significantly differs from the previous one, and question some of the assumptions that support our existing security models.

1.2 Wireless communication protocols

One of the most significant evolution linked to the Internet of Things is the rapid and massive development of wireless communication protocols. Indeed, the flexibility provided by these wireless protocols is especially interesting according to the constraints of connected objects, and resulted in a widespread adoption of these technologies. It has also lead to the design and deployment of a large amount of new wireless communication protocols, that are competing to win this new IoT market by fitting the requirements of connected objects and continuously releasing new features.

The chaotic deployment of these protocols in IoT environments resulted in a sensitive situation from a security perspective, and highlights significant scientific challenges that must be solved to protect these environments. In the current section, we highlight some of the characteristics of these protocols and how they impact IoT security.

The first thing that must be noted is linked to the proliferation of heterogeneous wireless technologies, that prevent a global understanding of the associated risks. Some of these technologies, such as Zigbee [Zig 2015] or Bluetooth Low Energy [Blu 2019], are well documented and their specification is publicly available, while others are based on proprietary technologies and only provide a partial, or even no documentation at all. For example, some protocols only provide the specification of the upper layers of the corresponding stacks, and do not disclose any information about the lower layers: this situation is quite common in the wild, and applies to various protocols such as LoRA [LoR 2017] or ANT [Dyn 2014]. In the worst case, the protocols are not documented at all, and only a costly and time-consuming reverse engineering process may allow to acquire enough knowledge about their internals to estimate the risks associated to their deployment while raising potential legal issues. This situation is especially bad from a security

perspective, as identifying the corresponding security risks obviously requires a deep technical understanding of the protocols internals. More precisely, the risks linked to the lowest layers are especially difficult to estimate, as they are hard to analyze using common security approaches.

This proliferation of heterogeneous communication protocols also raises some significant issues regarding the interconnection of the corresponding networks. This situation leads to the deployment of multi-protocols gateways aiming at allowing communications between heterogeneous technologies. As an example, the Phillips Hue connected lightbulbs [Philips 2022] being based on Zigbee protocol, a WiFi to Zigbee gateway is needed to control the lightbulb from a computer or a smartphone. This situation increases the complexity of IoT environments and dramatically expands the attack surface, because they imply the deployment of complex devices interacting with multiple protocols at the same time, that are only necessary because of the lack of standardization of IoT protocols. These gateways being generally connected to private or professional networks also induce an attack surface expansion for such networks, allowing a potential attacker to use this IoT gateway as an intermediary to perform pivoting attacks targeting them.

Some of the wireless communication protocols that are commonly used in IoT provide a set of security features, that could theoretically mitigate a wide number of threats. For example, 802.15.4-based communications can be encrypted using AES at multiple levels, from Network to Application layers. Similarly, Bluetooth Low Energy pairing and bonding mechanisms allow the nodes to securely negociate cryptographic parameters and settings to establish a secure communication based on state-of-the-art encryption. Unfortunately, configuring and enabling such security mechanisms is generally not trivial and may have an impact on user experience. This situation, combined to the lack of expertise in security of IoT manufacturers, may result to the deployment of insecure wireless networks because of technical errors or bad practices. A typical example is the use of default credentials or example keys, which was found to be a recurrent problem in IoT networks, as illustrated by the Mirai botnet [Antonakakis 2017]. It is also quite common to observe the use of custom security implementations, especially cryptography implemented upon applicative layers, that may be flawed and cannot protect efficiently the lower layers of the protocols. For example, the number of Bluetooth Low Energy-enabled devices that are using poorly or not encrypted at all communications is surprisingly high, as noted by C. Zuo et al. in a qualitative study of the percentage of BLE devices activating encryption mechanisms [Zuo 2019].

The fast evolution of protocols also induces a latency between the protocol specification and its practical implementations in the wild. This is especially visible with the Bluetooth Low Energy protocol, that introduced significant security fixes in the latest versions of its specification [Blu 2019], while a vast majority of BLE-enabled devices are implementing older versions of the stack and can't be easily patched, leaving them vulnerable to known attacks.

It must also be noted that a significant part of connected objects are based on tiny embedded systems with limited resources, that could considerably complicate

the use of traditional mitigation techniques. This is a main limitation to IoT security as a significant part of our existing security mechanisms are costly in terms of resources, and cannot be easily applied to such embedded systems. While this situation may evolve in the future, it's obviously mandatory to take into account the vast amount of connected objects that are already deployed in the wild and cannot be easily patched because of hardware or software limitations. We can also underline the sensitivity of some security mechanisms such as the software updates mechanisms from a security perspective, as it is absolutely mandatory to fix potential security issues after the device deployment while being a critical feature that can potentially be abused by an attacker, especially if the update is performed over the air through a wireless communication protocol.

Another problem is linked to the dynamicity of networks and topologies in IoT environments. Indeed, we already mentioned the fact that some connected objects being dedicated to a mobile use, they tend to generate complex and dynamic environments, which can't be easily monitored. The increasing number of wireless protocols allowing peer-to-peer communications makes this situation even worse, as no central point may allow an exhaustive monitoring or filtering of traffic. This situation disqualifies many traditional network security approaches, and requires the development of new defensive strategies that are more suited to this kind of systems.

A more subtle issue is linked to the co-existence of these wireless protocols in the same environments. Indeed, no standard being adopted yet, multiple wireless protocols may coexist in the same environments. However, a wireless medium is open by design: the direct consequence is that some protocols may share the same frequency bands, or use similar modulation schemes. This is especially visible in the 2.4GHz ISM band, which is overcrowded because of the co-existence of multiple heterogeneous wireless protocols. While this situation is well known from a functional perspective, and has lead to the development of a set of techniques aiming at limiting the interference between these different protocols [Chiasserini 2003] (e.g., the frequency hopping algorithms used by Bluetooth), it may lead to new security threats that are especially difficult to anticipate. Estimating the attack surface exposed by a given protocol implies to take into account the potential interactions between this protocol and the others that may co-exist in the environments, and that could be abused by an attacker.

The main concern that emerges from these observations is the obsolescence of several existing security models in this new context. Indeed, the highlighted characteristics, especially the combination of **heterogeneous**, **dynamical** and **decentralised** networks deeply question some classical security models or techniques while introducing significant challenges to design new ones. We can also note that the deployment of connected objects introduces a significant new attack surface, targeting both the IoT-related technologies and the traditional systems they may be interconnected with.

1.3 Security threats

In this section, we highlight the most significant security threats induced by the deployment of connected objects.

While some of these devices are based on architectures and technologies that are quite similar to traditional computers and are not very constrained in terms of resources (e.g., smart fridge, connected thermostat, domestic alarm), we focus in this work on the specific threats raised by connected objects based on small embedded systems that are limited in term of resources (e.g., smartwatches, lightbulb, keyfob) as we consider them as significantly different from more traditional technologies to justify a dedicated analysis.

The main characteristics of these devices are the following ones:

- they rely on embedded hardware with very limited resources,
- they are optimized for low energy consumption,
- they are using bare metal firmware or lightweight real time operating systems,
- they are using wireless communication protocols to interact with other systems.

We can classify the security threats linked to these systems in three main categories: the **hardware-related threats**, the **software-related threats** and the **network-related threats**.

1.3.1 Hardware-related threats

Connected objects being physically deployed in the wild, they are especially exposed to hardware-level attacks according to the high probability that an attacker could gain a physical access to them. As a consequence, we must obviously take into account the security threats that could result from hardware-level manipulations aiming at breaking the device security.

A typical design of such embedded system is composed of one (or several) micro-controllers, connected to sensors and/or actuators and communicating using a radio transceiver dedicated to a specific wireless protocol. The most interesting target for a hardware attack is obviously the main micro-controller, that could be attacked using several strategies.

A typical issue that has been observed in practice is the abuse of debugging features [Vishwakarma 2018]. Indeed, programming and debugging the software embedded in such devices can generally be performed using debugging physical interfaces, such as JTAG or SWD. If these features have not been properly disabled by the manufacturer, or if the chip security can be bypassed to gain access to these debugging features, it allows an attacker to gain a privileged access to the micro-controller itself. This privileged access could result in a code execution through a malicious firmware installation [Bettayeb 2019] or an unauthorized access to secrets or sensitive data such as cryptographic secret information or personal data.

Some specific hardware attacks can also be used to inject faults to the micro-controller, allowing to generate unexpected behaviour that may compromise the security of the software running. For example, glitching the micro-controller power delivery during a short amount of time could result to a misinterpretation of some instructions or to bypass some instructions [Timmers 2017]. Depending on the software architecture, such a behaviour could lead to bypass some software security checks or access privileged section of code [Timmers 2016]. Similar offensive techniques, such as laser injection, could lead to similar results.

Some electronic components can also be abused by exploiting an unexpected source of information, such as electromagnetic leaks, power consumption or vibrations, that could reveal sensitive information about the running algorithms or the data processed. These attacks, known as side-channel attacks [Le 2008], may allow to infer information by analyzing the resulting traces, and could lead to the acquisition of secrets. This situation is especially critical when it involves cryptographic algorithms, as previous works [Lo 2017, Gandolfi 2001, Genkin 2016] have already demonstrated that such attack could allow to recover an encryption key by monitoring the power consumption or electromagnetic leaks generated during an encryption process. In [Camurati 2018], G. Camurati et al. also demonstrated that in some conditions, electromagnetic leakage from digital logic can be mixed with the radio carrier, allowing to remotely collect sensitive information by analyzing radio emissions, and constituting what the authors call *screaming channels*.

Attacking surrounding components could also lead to security issues. For example, the presence of flash or memory chips may allow an attacker to read arbitrary content, such as firmware or cryptographic material, and under some specific conditions to write arbitrary data to these memories. Similarly, the communications between components is generally based on standard protocols such as I2C or SPI, that could be monitored or instrumented to leak some information or generate unexpected behaviour [Khelif 2021].

Some devices could also be targeted by hardware implants, that could allow an attacker to install a backdoor or collect sensitive data when the device is in use [Bojovic 2019, FitzPatrick 2016]. Depending on the system targeted, such implants may seriously compromise the security of connected devices while being difficult to identify and mitigate for a non expert.

Damaging or destroying some specific components, including mechanical ones, could also help an attacker to achieve a specific objective. It could be used to perform a denial of service of a specific security system, or bypass some defensive measures. For example, this kind of attack has already been used in the past to perform lockpicking targeting connected smartlocks, allowing to bypass the multiple software-level checks that were implemented.

While all these threats may require a physical access, it must also be noted that hardware level vulnerabilities can potentially be attacked from software components [Kocher 2019, Lipp 2018]. For example, performing some specific operations from software may generate an unexpected behaviour at the hardware level, such as the generation of a glitched signal in an adjacent line, that could potentially

be exploited by an attacker to perform a privilege escalation or attack a sensitive component.

1.3.2 Software-related threats

Connected objects are making use of software components, exposing them to a set of software-related threats. However, giving a general overview of these threats is a difficult task, because the software architecture greatly depends on the considered objects: some of them may embed an operating system, exposing a software stack similar to traditional computing, whereas others may use bare metal firmware or lightweight real-time operating system.

The devices embedding operating systems are mainly exposed to the same threats as a traditional system exposed over the Internet network: the services and applications that are remotely accessible may be exploited to gain remote code execution or leak data. These vulnerabilities are mainly implementation dependent, and may be linked to memory corruption issues [Tsoutsos 2018] (e.g., buffer overflow, use after free ...) or insufficiently sanitized inputs (e.g., code injection, format string) [Ray 2012]. We can also note that this kind of systems generally exposes a web interface, allowing to configure them or to control their behaviour easily. In this situation, such web application generally becomes a privileged target as it may be vulnerable to various web vulnerabilities (e.g., local or remote file inclusion, XSS, CSRF) [Singh 2019, Lai 2008] and potentially exposed over the Internet network. This is especially critical if the targeted web application has a privileged access over the system in order to trigger specific behaviours, allowing the attacker to divert or abuse these features. Similarly, some services that are considered as insecure, such as telnet or FTP, are sometimes exposed by such devices [Pa 2015], increasing the risks of compromise.

Unfortunately, multiple examples over the past few years have highlighted the presence of severe vulnerabilities in this kind of devices: as an example, the IoT search engine shodan.io [SHO 2009], that continuously scans Internet to identify IoT devices, reveals the presence of a significant number of surveillance cameras that can be accessed remotely without authentication all around the globe. It also highlights a typical issue in IoT devices: a vast amount of devices makes use of default credentials or configuration settings, allowing attackers to gain control over them easily using automated tools. For example, the Mirai botnet [Antonakakis 2017] takes advantage from this situation by exploiting a wordlist of default credentials to compromise IoT cameras.

On the other hand, a significant amount of devices are based on tiny embedded systems with not enough resources to run a full operating system, and are based on bare-metal firmwares or lightweight real time operating systems. In this situation, the main software threats are related to insecure firmware implementations [Cui 2013]. These firmwares being generally developed using low level languages such as C or C++, they are especially prone to introduce memory-related issues, such as buffer overflow, format string or use-after-free. Depending on the

context, this kind of vulnerabilities may result to leak sensitive data from memory, alter the execution flow or even code execution in some cases. We can note that exploiting such software issues in this kind of device is generally easier than in traditional systems, as they generally don't include typical security mechanisms aiming at complicating the exploitation such as ASLR, W⊕X or stack canaries.

We must also underline that one of the biggest issue linked to IoT devices is the lack of secure and efficient update mechanisms. Indeed, it may be difficult, and sometimes impossible, to update the software components running on this kind of devices. This situation is common in the wild, resulting in devices which are difficult to maintain and may embed outdated or vulnerable software components. This is very problematic as the software industry mainly relies on automatic software updates to fix software-related vulnerabilities. Deploying such an automatic process in the IoT ecosystem is especially hard, because these devices don't necessarily include an IP connectivity and may be deployed in environments without network access. When the security mechanisms allowing to update the software components are embedded, they should also be efficiently hardened and secured to avoid being abused by an attacker to install malicious updates [Bettayeb 2019]: this kind of attack is especially critical as it may lead to a remote code execution and a full control over the device.

Let us note that some software vulnerabilities can also be related to software components that are less visible, such as the ones which are involved in the network management: for example, a Bluetooth Low Energy protocol stack could be vulnerable to a buffer overflow that can be triggered over the air [Garbelini 2020]. We still consider this kind of implementation-dependent issues as software vulnerabilities even if the network is involved in their exploitation, as they are not linked to the communication technology design itself and can theoretically be patched by software.

1.3.3 Network-related threats

Because of the connectivity they provide, connected objects are also exposed to network-related threats. These threats are mainly linked to the communication protocol design: we do not consider here the potential vulnerabilities linked to a specific implementation of the protocol stack [Garbelini 2020, Armis 2017, Armis 2018], as they can be considered as software-related. We also mainly focus on wireless communication protocols because of their omnipresence in IoT networks. However, some of the threats we mentioned could also be relevant for a wired protocol.

The first threat that must be taken into account is jamming, that allows an attacker to impact the availability. Jamming attacks allows to corrupt a transmitted message by transmitting a malicious signal simultaneously: generally, it abuses features such as CRC, that aims to detect errors during the transmission. Such features are common in wireless protocols because of the high probability of interferences, especially in overcrowded frequency bands such as the 2.4 GHz ISM band. The adversary jamming signal corrupts the legitimate message by introducing some bitflips

in the demodulated bitstream, forcing the surrounding receivers to drop the frame because of an invalid CRC. Multiple jamming strategies can be used [Xu 2005], from the simplest (such as continuous jamming [Shintani 2020], that transmits a strong signal continuously on the target channel) to the more complex ones (e.g., reactive jamming [Bräuer 2016, Schulz 2017], that selectively jams specific messages by identifying a specific pattern on-the-fly to take the jamming decision).

Sniffing is another major network-related threat, especially in a wireless context which is open by definition. Indeed, an attacker may perform an eavesdropping attack allowing him to impact the confidentiality of the communications. Even if the transmitted packets are encrypted, such an attack may still allow an attacker to acquire some knowledge about the network, by exploiting some specific fields that are transmitted in plaintext [Newlin 2016a] or some metadata such as the RSSI. We can note that, depending on the targeted protocol, performing this kind of attack can be more or less complex from a technical perspective: for example, sniffing Bluetooth communications is a non trivial task [Ryan 2013a, Cauquil 2017b, Cauquil 2019] because of the use of a channel hopping algorithm. We can also underline that sniffing is commonly involved in more complex attack workflows, for example to collect some information needed to perform active operations later (e.g., spoofing a specific node may require to acquire some knowledge about the device to mimic, such as address or payloads).

Another threat that can have a severe impact is traffic injection, when an attacker injects some malicious messages into the network to trigger a specific behaviour or achieve a malicious objective. Such injections can be more or less difficult depending on the targeted protocol and the mitigation they provide, and may involve some complementary actions (e.g., spoofing a device, disrupting a legitimate node, etc). Moreover, they can be performed from various setups: as an example, packet in packet attack [Goodspeed 2011b] can be used to perform a packet injection with low privileges from upper layers. An injection doesn't necessarily require the ability to modulate a signal according to the protocol specification: a physical layer replay attack, that can be assimilated to an injection attack, can be performed by transmitting a signal previously collected, even if the attacker is not able to decode it. We can also note the existence of a specific class of attacks, named overshadowing attacks [Wilhelm 2012, Yang 2019], that could be assimilated to a complex packet injection: the attacker transmits a signal simultaneously to a legitimate one, which is carefully synchronized and crafted to alter the transmitted symbols in real time and generate a specific malicious message during demodulation.

Another main threat targeting communication protocols is Man-in-the-Middle attacks, where the attacker is able to present himself as an intermediary between two (or more) nodes, allowing him to transparently manipulate the traffic. Such an attack may impact both the confidentiality, the availability and the integrity as the attacker can modify the frames, drop them or inject fake ones. Multiple configurations may allow to achieve such an attack: for example, it can be performed by abusing routing mechanisms (e.g., Wormhole attack [Hu 2006]), or link layer procedures (e.g., GATTacker [Jasek 2016] or BTLEJuice [Cauquil 2016]). A Man-

in-the-Middle setup can also be exploited to attack only one side of the targeted communication, resulting in an hijacking attack, that could impersonate a specific device during a communication with another one.

The coexistence of several wireless protocols in the same frequency bands, which is a common situation in IoT environments, can also lead to specific threats. For example, a malicious code in a compromised device may exploit this kind of similarities to perform pivoting attacks, allowing the compromised node to perform eavesdropping or injection attacks targeting a different protocol [Jiang 2018, Jiang 2017]. This kind of threats is especially difficult to anticipate, as it is linked to unexpected interactions between wireless communication protocols.

1.4 Mitigations

The high number and variety of high impact threats previously mentioned dramatically highlights the necessity to secure connected objects. One of the first key requirement to mitigate security threats is to have methodologies and tools allowing to efficiently identify and analyze them. For example, vulnerability scanning, fuzzing and auditing are commonly used from a defensive perspective in order to identify potential security weaknesses. However, the particular situation of IoT significantly complicates the development of this kind of methodologies and tools. The wide number of heterogeneous technologies, including proprietary ones, prevents the design of a generic or global approach and leads to the multiplicity of methods and tools covering only a small fraction of threats.

The identification and analysis of hardware-related threats are especially difficult to cover, as they may involve costly analysis equipment and knowledge in electronics and physics that might be outside of the core skills of a security analyst. Similarly, auditing network-related threats, especially the ones linked to the link and physical layer of wireless protocols, implies the manipulation of costly hardware to collect and analyze RF signals, while involving very specific skills in physics and signal processing. The IoT context also raises significant issues complicating software-related threats identification, because of the lack of instrumentation tools targeting embedded systems, the use of heterogeneous hardware architectures and the significant differences between a bare-metal firmware and a regular process running on a standard operating system. Globally, we lack a systematic approach allowing to efficiently detect security vulnerabilities, both from a software, hardware and network perspective, and we can't easily deploy previous approaches in this context because of specific constraints.

Once identified, fixing the vulnerabilities remains a significant challenge, especially if the concerned system is already deployed in the wild. We previously mentioned issues about software updates, that are the usual way to fix vulnerabilities once the system has been deployed but can be tricky to implement in practice depending on the concerned system and the technologies in use (e.g., some connected objects are not connected to Internet) and may themselves introduce security vul-

nerabilities. Fixing hardware-related vulnerabilities is even more complicated, as it generally implies physical actions targeting the device such as component replacements or PCB modifications: as a consequence, only a small subset of hardware vulnerabilities are corrected in deployed devices, as it implies product recalls and massive costs from the manufacturers (generally, manufacturers correct them in the next hardware revisions). The security threats linked to communication protocols, especially when they are linked to the protocol design itself, lead to similar issues, as the only way to fix them is to write a new version of the specification. The consequence is the massive presence of insecure protocol stacks in the wild, that are outdated and can't be fixed without replacing the hardware.

Another approach that could be used to secure these devices is the deployment of Host-based mitigations, aiming to detect malicious code (e.g., antivirus) or prevent the exploitation of security flaws (e.g., anti-exploit techniques such as ASLR or stack cookies). However, embedding such programs in IoT devices is a real challenge because of the use of tiny embedded systems with very limited resources and simplified hardware architectures. Moreover, programs like antivirus may be based on signature-based approaches and could require regular update of their database, which has already been pointed out as difficult in this context. The technical limitations in terms of speed and memory on some systems also complicate the deployment of mitigations based on cryptography, and could have a significant impact upon the functional performance in such constrained systems.

Complementary approaches could be the deployment of Network-based mitigations, such as IDS, IPS or firewalls. While these technologies are common in traditional networks, they are also difficult to apply to IoT communication protocols. Indeed, most of these technologies are supposed to be placed at a strategic location in the monitored network, such as a router or a central point, facilitating the monitoring and filtering of the traffic. However, the massive use of peer-to-peer communication protocols in IoT considerably complicates the deployment of such surveillance nodes, as no central point can be instrumented to perform traffic analysis. Deploying a system which is able to exhaustively monitor the peer-to-peer communications in an IoT environment implies to install sniffers or probes, which are limited by the radio range they cover, their localization and the hardware they use. Moreover, let us note that some protocols use complex physical layers (e.g., Bluetooth, based on a channel hopping algorithm) that could considerably increase the complexity and the cost of a sniffer. Similarly, the presence of heterogeneous protocols makes the situation even worst as it implies the use of several different sniffers or the deployment of generic RF hardware such as Software Defined Radio, which is costly and implies a significant effort of engineering. Moreover, it's especially difficult for such a monitoring system to selectively filter or remove malicious packets as the probes are only passive nodes and don't play an active role in the packet routing. Last but not least, the presence of mobile nodes generates dynamic environments, where identifying whether the presence of a specific node is legitimate or not remains a non-trivial task: it increases the complexity of behavioural approaches.

We can also note that security models and assumptions aiming at protecting traditional networks are also challenged by this new paradigm. For example, perimeter security, which is a very common security model applied in a wide amount of networks, must be questioned considering the deployment of personal mobile devices that could easily have been compromised outside of the perimeter and used as intermediary by an attacker.

Unfortunately, we must also underline that some key obstacles that significantly complicate the development and deployment of suited mitigations can't be solved by a technical approach. There is an economic constraint linked to the *time-to-market*: various competitors try to conquer this new IoT market, leading the manufacturers to continuously release new systems and features, sometimes without adequate attention to security requirements. Indeed, building a secure IoT system would require a significant amount of work, from the design to the deployment, that is costly and more difficult to promote compared to new functional features. We can also note that most of manufacturers involved in this new market don't have a lot of experience about security: contrarily to the software industry, they are not familiar with the security requirements and don't benefited from the learning effects linked to previous attacks.

1.5 Challenges and contributions

The various observations we developed in the previous sections highlights that the deployment of connected objects, especially when they support peer-to-peer wireless communication protocols, leads to the development of new threats while considerably complicating both their identification and their mitigation. Multiple indicators seems to indicate that a significant attack surface emerges from this particular context. For example, in the past few years, a wide number of Bluetooth vulnerabilities have been reported, some of them being present in the wild for several years: it obviously shows that their impact dramatically changes in the context of IoT, as these protocols are becoming a central part of connected objects connectivity. Moreover, a protocol stack is a very complex system involving multiple intricated components and layers interacting together. Some threats may be linked to very low level components of these stacks, which are very difficult to analyze from a security perspective as they are especially difficult to instrument and involve skills and technologies that are trans-disciplinary: it results in a new attack surface which remains unknown and particularly difficult to explore. In this context, the background research question that motivated this work is the following one: how to identify, assess and mitigate the new threats linked to the deployment of peer-to-peer wireless communication protocols in the IoT context ?

More accurately, we focused on the following objectives inferred from this research question:

- Understanding the internals of heterogeneous wireless communication protocols used in IoT, including proprietary or poorly documented ones, as well

as the use cases they cover, the guidelines motivating their design and their evolution over time.

- Designing tools and methodologies allowing to identify and assess security threats linked to the design of these heterogeneous protocols and their coexistence in the same environments.

- Identifying and analyzing the new unexplored attack surface involved by the complex intrication of hardware and software components in a protocol stack.

- Understanding the consequences of IoT context upon our existing defensive models and adapting them or designing new ones allowing to fit these new constraints.

Matching these objectives involves to overcome multiple challenges. First, we need to solve the technical constraints linked to the heterogeneity and complexity of wireless protocols: we must especially build tools and methodologies allowing to analyze these protocols and challenge their security assumptions. Testing the security of the lowest layers, especially the physical layer, is very challenging because most of the existing hardwares, softwares and libraries are not designed to perform security research and don't allow to access the low level components nor performing non-standard actions, which is necessary to efficiently identify threats.

We must also identify and evaluate both the differences and similarities between these protocols, highlighting the potential interactions between them and the risks associated. The environments being dynamic and decentralized, it also makes traditional defensive approaches unpractical, forcing us to rethink them and build suited defensive systems that fit the connected objects constraints.

From our perspective, solving these challenges implies to adopt a transdisciplinary approach allowing to take into account both the signal processing, hardware and software components, as well as their interactions. The intersection between these components, especially when they are designed and developed according to very different engineering fields and tend to be considered as black boxes, are very critical and could lead to a significant attack surface that is especially difficult to evaluate. We must also keep in mind the particular context that leads to the deployment of these protocols, and the impact it has upon the environments where these technologies are deployed: critical attack vectors could be linked to the chaotic deployment of heterogeneous technologies and especially the unexpected interactions between them. We also tried to combine and explore ideas and approaches from different technical areas, both from an offensive and defensive perspective. For example, we actively explored the use of techniques that are generally used in an offensive context, such as code injection or jamming, to achieve defensive objectives and circumvent some technical issues linked to protocol stacks. More generally, we consider that the research works we initiated from an offensive perspective highlighted some specific weaknesses and issues, that allowed us to feed our thoughts about the development of defensive solutions for IoT and

underlined some specific issues and challenges that must be solved to build relevant mitigations. Similarly, we explored different perspectives linked to this central thematic, from signal analysis to software reverse engineering, allowing us to gain a better understanding of the big picture of IoT security.

In this PhD thesis, we mainly focused on the lowest layers of these protocols, especially the link layer and the physical layer, and explored various challenges related to these technologies, both from an offensive and a defensive perspective. The offensive-oriented research works we produced were mainly focused on vulnerability analysis, and highlighted several critical attack vectors linked to the way IoT wireless communication protocols are designed and to their co-existence, while providing a significant set of open-source tools aiming at facilitating the reproducibility of our work and future research work in this field. Our defensive work was focused on the specific issues linked to the monitoring and filtering of this kind of protocols and explored two innovative approaches aiming at facilitating the design of IDS and IPS suited for this context.

Our contributions are the following ones:

- We discovered a new attack vector, allowing to divert a transceiver dedicated to a given protocol to interact with non natively supported ones, that could be easily abused to perform pivoting attacks or covert-channel attacks. This attack vector mainly abuses the physical similarities between wireless protocols coexisting in the same environments. We explored the threats linked to this vector by demonstrating multiple critical attack scenarios based on it and implemented it on various off-the-shelf devices, including a smartphone and various connected objects, sometimes with very low privileges. We first demonstrated the feasibility of this kind of attack from a Bluetooth Low Energy transceiver targeting Zigbee networks, resulting in an offensive strategy named WazaBee. Then, we enlarged the scope to cover various other protocols, and showed that building such pivot attacks is practically feasible and could target a lot of IoT protocols deployed in the 2.4 ISM band.

- We discovered and practically implemented a critical injection vulnerability in the Bluetooth Low Energy protocol design, linked to a protocol feature dedicated to compensate clocks desynchronization and named InjectaBLE. Every device which is compliant with the BLE specification is vulnerable to this injection by design, as it is linked to the protocol design itself. We showed that multiple critical scenarios can be implemented using this vulnerability, leading to a full compromission of the communication between two BLE devices.

- We designed and implemented an offensive auditing framework aiming at facilitating the security analysis of wireless communication protocols and the development of offensive tools. We designed the framework to be generic, modular and easily customizable, and successfully implemented support for dozens of heterogeneous hardware and up to six wireless protocols. It allows to easily perform and implement wireless attacks targeting these protocols,

while allowing to build complex attack workflows combining multiple attacks. We actively used and improved it all along this thesis, because it considerably facilitates both the vulnerability analysis of protocols and the evaluation of defensive solutions.

- We designed a decentralized Intrusion Detection System embedded in Bluetooth Low Energy controllers, allowing to detect up to six critical design-related vulnerabilities while providing a very low level access to the instrumented stacks. We implemented a defensive framework aiming at facilitating both the development of detection modules and the analysis of the internals of wireless stacks.

- We built an experimental wireless firewall based on a reactive jamming approach, aiming at circumventing the issues linked to the absence of central node in peer-to-peer wireless communication protocols. The firewall was successfully implemented and tested on two common IoT protocols, demonstrating that this approach is generic and can successfully intercept and drop malicious traffic. We performed a set of preliminary experiments showing promising results to prevent wireless attacks.

In chapter 2, we introduce the state of the art of wireless security while chapters 3 to 6 present our contributions.

Chapter 2

State of the art of wireless security

Contents

This chapter presents the state of the art of wireless communication protocols security. It first focuses on the offensive tools, both from a software and hardware perspective, highlighting the main issues related to these tools. Then, it introduces the most relevant protocol-related attacks targeting several widely used protocols such as Zigbee or Bluetooth, as well as some proprietary protocols. We also introduce the existing strategies allowing to perform pivoting attacks from a transceiver supporting natively a given protocol to interact with another wireless technology. Finally, we discuss the defensive researches focused on Intrusion Detection Systems and Intrusion Prevention Systems in IoT wireless networks.

2.1 Auditing tools

Analyzing the security of wireless communication protocols requires the use of dedicated tools, allowing to monitor or interact with the targeted protocol. In the recent years, multiple research works in this area have contributed to the development of such tools, allowing to facilitate the analysis of wireless communications. In this section, we first introduce the hardware tools that can be used in an security context, then we present the corresponding software tools. In both cases, we

discuss the pros and cons of these tools, then we highlight some general issues and limitations that must be solved to improve the reliability and efficiency of wireless security analysis.

2.1.1 Hardware tools

Testing the security of wireless communication protocols may imply the use of dedicated hardware devices, aiming to interact with the protocol. More precisely, a security researcher may analyze a given protocol from various perspectives, that can be complementary and allow to perform different kind of attacks or analyze different aspects of the protocol: various hardware may provide different capabilities, that can be more or less suited to the situation. Two main category of hardware are commonly used to analyze the security of a wireless protocol: **Software Defined Radios** and **dedicated transceiver**.

2.1.1.1 Software Defined Radios

The most powerful and flexible hardware components that can be used to interact with wireless protocols are Software Defined Radios. These devices aim to provide a generic radio interface by allowing the user to access the RF signal directly, both in transmission and reception. The Digital Signal Processing (DSP) components (e.g., mixers, filters, modulators) that are usually implemented in hardware can be instead implemented in software.

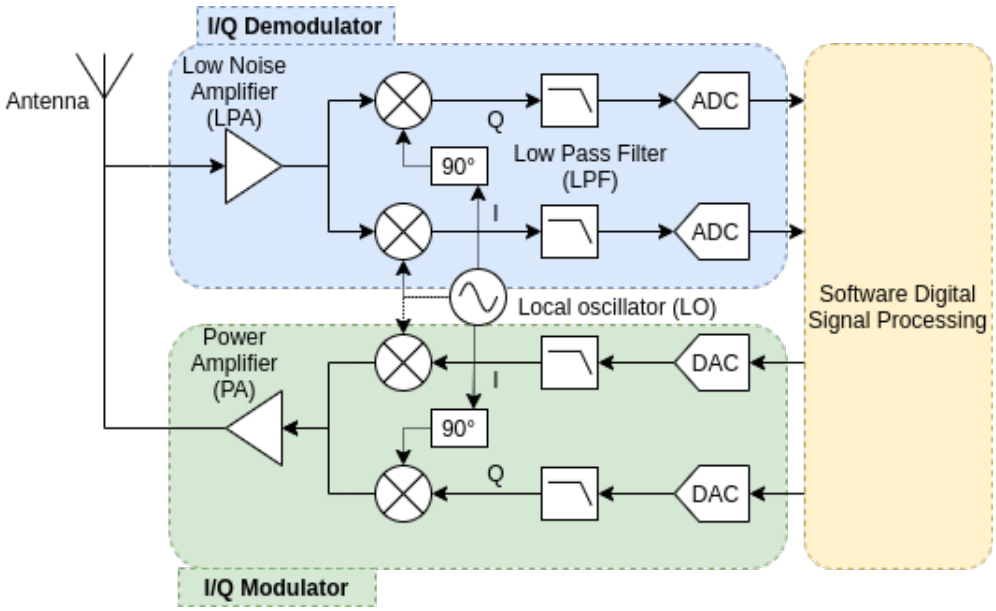

Figure 2.1: Typical Software Defined Radio architecture

These devices are generally based on an IQ modulator/demodulator, illustrated in figure 2.1 allowing to provide a very generic way to interact with the RF signals.

It offers a great flexibility to the user as it allows to manipulate the signal at a very low level, and can be used to modulate or demodulate a wide variety of wireless protocols. Such a property is especially interesting from a security perspective as it may theoretically be used to implement a given protocol stack from scratch entirely in software, giving the security analyst a significant control over the whole stack architecture, from the lower to the upper layers.

While most of Software Defined Radios are based on the same kind of architecture, they greatly vary in terms of price, depending on the RF characteristics provided by the underlying hardware. The cheaper SDR device that can be found is RTL-SDR [RTL-SDR 2022], a low-cost DVB-T USB dongle based on a Realtek RTL2832U chip that can be diverted to act as an SDR. This device is widely used because of its low cost but presents several significant limitations: it can only be used as a receiver (no transmission capabilities are provided), it can only process sub-1GHz signals and present a limited bandwidth of 2.4 MHz.

Another SDR device which is commonly used during security analysis is HackRF one from Great Scott Gadgets [greatscottgadgets 2022]. This device provides a good compromise between its price and the provided capabilities, as it can be used in reception or transmission mode, on a frequency range of 1 MHz to 6 GHz with a bandwidth up to 20 MHz. The project being based on open-hardware, it has built a consequent community of hackers and radio-amateurs in the recent years, and has been widely used both in academic researches [Roux 2018,Yuniati 2019,Galtier 2020] and technical contexts. It is also supported by a lot of DSP tools and provides its own open-source software environment, allowing to easily record, transmit or replay an arbitrary signal. It has also been extended to support wide band monitoring [Ossmann 2017] using a sweep-based strategy. However, the device still suffers from several limitations, mainly linked to its half duplex nature (reception and transmission can't occur at the same time) and its low sampling rate that don't allow to interact with complex modulations or high throughput protocols. The communication with the host being based on USB, it also implies a speed limit, complicating the monitoring of some protocols (for example, Bluetooth uses a frequency hopping algorithm that can't be followed by such a device because of its limited speed to switch frequency). Finally, it is based on a combination of Complex Programmable Logic Device (CPLD) and micro-controller, which is a less flexible architecture than Field-Programmable Gate Array (FPGA), which are generally used in SDR.

BladeRF [Nuand 2022] presents similar capabilities, while extending the bandwidth to 56 MHz and allowing a full duplex mode using Multiple-Input Multiple-Output (MIMO) technology. It also includes an Altera Cyclone 4 FPGA. MyriadRF [MyriadRF 2022] also released two SDR devices, the Lime SDR and Lime SDR mini, which are also based on an Altera Cyclone 4 FPGA and cover a frequency band from 100kHz to 3.8GHz with a bandwidth up to 61.44 MHz. These devices are especially interesting from a security perspective as they are designed as open-hardware and rely on an open-source tooling, facilitating the prototyping of security tools while offering a good compromise between price and quality.

Finally, USRP devices [Research 2022] from Ettus Research have to be men-

tioned, as they are probably the most commonly used SDR devices in research works. Multiple series have been released over the years, providing various features such as High Speed Ethernet connectivity, multiple channels or wide bandwidth, allowing to reach very good performances thanks to their FPGA-based design. However, these devices are generally very expensive compared to other SDR and are generally bulky and fragile, complicating their use in mobile-oriented applications or outdoor use.

We can also note the existence of experimental research works aiming at diverting commodity hardware dedicated to a specific protocol (e.g., WiFi) to turn them into a SDR, allowing the transmission or reception of arbitrary signals. In 2018, Schulz et al. [Schulz 2018a] presented during MobiSys conference an experimental approach based on Nexmon framework [Schulz 2018b], allowing to patch the Wi-Fi firmware of BCM4339 devices installed in Nexus 5 smartphones and BCM43455c0 devices installed in Raspberry Pi B3+ computers in order to transmit arbitrary IQ signals in the 2.4 GHz and 5 GHz frequency bands, abusing calibration features. However, it seems limited to specific WiFi Broadcom chips and can only be used as a transmitter.

While these devices are interesting because of their genericity and flexibility, they also suffer from significant limitations that must be taken into account. First, they imply a significant amount of work as they require to implement the protocol stacks from scratch, including the Digital Signal Processing components which implies specific knowledge in physics and signal processing. Moreover, they also force to perform most of the signal analysis in the Host, which may require a big amount of resources to process some complex protocols. Similarly, the implementation on the Host also implies the introduction of delays linked to the Device to Host transmission, especially when using USB connection: these delays may complicate the analysis of specific protocols that require a fast processing (e.g., exhaustively monitoring a Bluetooth connection is impossible on most of SDR because of the use of a channel hopping algorithm which requires fast frequency changes). These constraints can sometimes be solved using a specific kind of device (e.g., some USRP provide fast Ethernet connection to limitate the transmission delays) or by implementing time-sensitive components of the stack in the device directly, but it obviously impacts both the genericity and flexibility of these devices.

2.1.1.2 Dedicated transceivers

An alternative solution to interact with a protocol is to use a dedicated transceiver, generally embedded in a *System on Chip* (SoC) or a dongle. Such devices are generally dedicated to one or two specific protocols and may provide various features depending on the underlying hardware and the embedded software: we can note that multiple devices used for security testing have required the development of custom firmware, in order to add specific capabilities (e.g., KillerBee [Wright 2009] firmware providing injection feature for RZUSBStick) or provide a low level control over the device (e.g., RFStorm firmware for nRF24 chips [Newlin 2016b]).

The most straightforward way to interact with a given protocol is to use commodity hardware specially designed for this protocol. For example, standard WiFi dongles can be used to perform some attacks targeting the WiFi technology. Similarly, some Bluetooth Low Energy attacks (e.g., BTLEJuice [Cauquil 2016] or GATTacker [Jasek 2016]) are making use of standard HCI dongle, used to perform Bluetooth Low Energy communications. However, these devices generally present several limitations that must be highlighted. First, some features which are necessary for performing security analysis are not standard and linked to proprietary implementations. For example, performing some specific WiFi attacks implies the use of a non-standard mode named "monitoring", allowing to forward every received frames to the Host even if the MAC address included in the frame doesn't match the receiver configuration: depending on the used hardware, this feature may be available or not, making some specific models and vendors more relevant from a security perspective. Similarly, some Bluetooth Low Energy attacks [Cauquil 2016, Jasek 2016] rely on spoofing a targeted device and require the capability to set an arbitrary BD address. The Bluetooth specification doesn't include a standard way to perform such a modification, resulting in the absence of this feature in some devices while others implement it using vendor-specific commands. Second, these devices being generally dedicated to a regular use of the protocol, they generally provide a high level API which can significantly complicate their use for security testing. For example, a Bluetooth dongle exposes a HCI interface that only allows to manipulate the highest layers of the protocol, so it is not relevant to analyze the security of the link layer or the physical layer. Similarly, some proprietary protocols rely on security by obscurity and hide the implementation of the lowest layers: for example, the ANT specification [Dyn 2014] only provides the specification of upper layers and the ANT transceivers only provide a high level API, complicating the analysis of the low level layers.

Another strategy which is commonly used in security testing of wireless protocols is the use of a commodity hardware which has been diverted to unlock new capabilities, generally to gain access over the lowest layers of the protocol. These strategy generally rely on the use of custom firmware [Newlin 2016a, Newlin 2016b] specially dedicated to security testing, or the patching of existing proprietary firmwares [Schulz 2018b, Mantz 2019]. In 2009, J. Wright released at Toorcon 11 the KillerBee framework [Wright 2009], dedicated to security analysis of 802.15.4 communications, especially Zigbee. For this purpose, he developed a custom KillerBee firmware for the Atmel RZUSBStick including injection and jamming features which were not natively provided by the manufacturer. T. Goodspeed has also discovered a vulnerability in the *nRF24L01+* chip, that facilitates sniffing and frame injection on a set of protocols (such as *Bluetooth Low Energy* or *Enhanced ShockBurst*) using *Gaussian Frequency Shift Keying* modulation. He was able to divert the use of a register dedicated to the address selection to select an arbitrary preamble [Goodspeed 2011a]. Exploiting this vulnerability allowed him to add a promiscuous mode for the *Enhanced ShockBurst*, which is not natively supported by the chip. However, it is also possible to divert the use of this register to detect specific preambles used

by different wireless technologies, as long as similar modulations and bit rates are used. This vulnerability has been used by M. Newlin to develop a firmware named RFStorm [Newlin 2016b] aiming to add advanced sniffing capabilities for the *Enhanced ShockBurst* and *Mosart* protocols to the *nRF24* chip [Newlin 2016a]. This custom firmware can be flashed to nRF24-based dongles, such as Logitech Unifying dongles or Crazy Radio PA.

A similar vulnerability has also been identified by D. Cauquil in another chip from *Nordic SemiConductors*, named nRF51. This chip supports Bluetooth Low Energy communication, and abusing this vulnerability allowed D. Cauquil to implement a custom firmware for the BBC:MicroBit [Cauquil 2017c], an educational device embedding nRF51 chip, allowing him to add Bluetooth Low Energy sniffing and jamming capabilities [Cauquil 2017b]. Similarly, a powerful Bluetooth Low Energy sniffer named *Sniffle* has also been implemented by S. Qasim Khan in [Qasim Khan 2019], based on Texas Instruments Bluetooth Low Energy transceivers (TI CC1352/CC26x2). It increases the probability of successfully synchronizing the sniffer with a connection by tracking the targeted device during its advertising phase, while supporting the new physical layers (LE 2M and LE coded) introduced in the latest versions of the specification.

Some research works have also focused on reverse engineering and patching the native firmware used by commodity hardware, in order to gain access to low level features. In [Schulz 2018b], M. Schulz provided a framework called *Nexmon*, aiming to facilitate the patching of BCM4339 WiFi chip from Broadcom embedded in the Nexus 5 smartphone. He demonstrated that patching the firmware of this chip may allow the implementation of advanced low level attacks, such as reactive jamming [Schulz 2017] or covert channel attacks. A similar work has been realized for Cypress and Broadcom Bluetooth chips in [Mantz 2019], where D. Mantz et al. demonstrated that vendor-specific HCI commands could be abused by an attacker to patch the native firmware, allowing the instrumentation and implementation of low level attacks targeting the Bluetooth protocol [Antonioli 2019]. In [Vanhoef 2014], M. Vanhoef et al. demonstrated a similar strategy allowing to add low level capabilities to Atheros AR7010 and AR9271 WiFi chips. He showed that implementing a custom firmware into those chips may allow to perform low level WiFi attacks, such as Jamming, unfair channel usage or channel-based Man-in-the-Middle.

The latest strategy that can be used is the design and development of a dedicated hardware, optimized for performing security testing. This approach has been used for the *Ubertooth One* device from Great Scott Gadgets, an open-hardware dongle based on the TI CC2400 transceiver. It implements the sniffing strategy proposed by M. Ryan in [Ryan 2013a] for Bluetooth Low Energy and the research work from D. Spill et al. [Spill 2007] allowing to sniff Bluetooth BR/EDR packets. Multiple proprietary Bluetooth and Bluetooth Low Energy sniffers have also been developed, such as the Bluefruit LE Sniffer [Adafruit 2014] from AdaFruit, allowing to follow a Bluetooth Low Energy connection, or the Ellisys Bluetooth Explorer [Ell 2021], a device allowing wide-band monitoring of Bluetooth BR/EDR and Bluetooth Low Energy communications. During their research work targeting proprietary pro-

tocols used by wireless keyboards, T. Schroeder and M. Moser [Schroeder 2010] developped a specific hardware allowing to sniff packets transmitted by these input devices. Another security-oriented RF dongle named Yard Stick One [Atlas 2012] has also been released, based on a TI CC1111 transceiver. It allows the analysis of wireless protocols using the sub-1GHz frequency band, and can interact with several modulation schemes such as 2-FSK, 4-FSK or ASK. A dedicated open-source firmware, named RFCat, has been released and allow to easily manipulate such dongle from a python library.

While these dedicated transceivers obviously allow a wide range of use cases, we can highlight the heterogeneity of these devices, their separated development and the various capabilities they provide. Some of them allow high level operations but can't be used to investigate the lower layers, while others are focused on low level capabilities but may imply a significant work of tooling and can be limited in term of usability. Moreover, a significant part of transceivers diverted using vulnerabilities are dependent of specific hardware and can't be easily maintained (for example, Broadcom has fixed some of the insecure vendor-specific commands that allowed InternalBlue patching, forcing the maintainer to bypass the new firmware design [Classen 2021]). This chaotic development also leads to a wide variety of custom tools presenting various limitations, implementing different features and exposing custom API. It also significantly impacts the development of associated software tools.

2.1.2 Software tools

A wide variety of software tools have been developed in the recent years, aiming at interacting with previously mentioned hardware to perform wireless security testing.

2.1.2.1 Physical layer analysis

Multiple softwares have been developed to interact with Software Defined Radios. One of the main software used in this field is GNU Radio [GNU 2021], an open-source toolkit aiming at providing easy to use Digital Signal Processing blocks in order to process and analyze RF signals from a wide range of SDR devices. Multiple projects have been built on top of this software, aiming at interacting with specific wireless protocols by implementing the corresponding demodulators and modulators (M. Newlin first implemented a Enhanced ShockBurst sniffer using GNU Radio [GRN 2016] for analyzing wireless keyboards security). A similar tool, named PothosFlow [POT 2021], has also been released, and has been used by some security researchers to analyze specific protocols (for example, during his research work on ANT protocol, T. Szakaly [Szakaly 2016] built an ANT sniffer [ANT 2016] based on this software suite to analyze ANT-FS communications).

Some specific tools are also commonly used to analyze RF signals. Spectrum analyzer, such as GQRX [GQR 2013], allows to visualize the RF Spectrum in a convenient way. A set of tools are also dedicated to the analysis of RF signals,

and can be used to analyze the physical layer of a wireless protocol. For example, Inspectrum [INS 2021] can be used to visualize a captured signal with different representations to infer the symbol rate or the modulation scheme. Universal Radio Hacker [Pohl 2018] is another reverse-engineering oriented toolkit, that can be used to capture signals and perform an advanced analysis to identify physical parameters such as the modulation scheme, the encoding in use or the symbol rate. While these softwares are generally developed to support multiple devices and make use of generic backends such as SoapySDR [SOA 2022], we can also note the existence of specific tools dedicated to a given hardware. For example, the HackRF maintainers provide a set of command-line tools [greatscottgadgets 2022] that can be used to easily manipulate HackRF one to receive and transmit signals or perform a wide-band monitoring using a sweeping strategy [Ossmann 2017]. Similarly, Myriad RF provides several tools allowing to perform simple signal analysis using LimeSDR [MyriadRF 2022] devices. We can also note the existence of complementary tools that are not directly related to SDR but can be relevant during a reverse engineering process targeting a wireless protocols, such as Reveng [REV 2022], that allows to automatically identify the CRC algorithm in use by providing a set of sample packets, or Scapy [SCA 2022], a python library facilitating the dissection and construction of arbitrary packets from various protocols.

2.1.2.2 Link Layer analysis and attacks

Similarly, most of the dedicated transceivers that have been developed or diverted in a security perspective comes with their own library, allowing to manipulate them from a Host device. For example, Ubertooth [Spill 2012] one provides a set of command-line tools allowing to sniff Bluetooth BR/EDR and Bluetooth Low Energy packets. Similarly, D. Cauquil provided a python tool named BTLEJack [Cauquil 2018] that can be used to sniff or jam Bluetooth Low Energy connections using a Micro:Bit embedding his custom firmware. S. Qasim Khan released a set of python scripts that can be used to interact with Sniffle firmware [Qasim Khan 2019], providing features such as sniffing, connection establishment or advertising for the Bluetooth Low Energy protocol. J. Wright implemented a python framework named KillerBee [Wright 2009], allowing to weaponize multiple 802.15.4 transceivers (including the RZ USB Stick embedding KillerBee firmware) to perform packets injection, eavesdropping or wardriving. M. Newlin released a python library [Newlin 2016b] facilitating the interaction with nRF24 devices embedding the RFStorm firmware, while the Yard Stick One can also be manipulated using a python library named RFCat [RFC 2021]. We can note that maintaining, documenting and improving these tools and libraries is a very difficult task, as most of them have been developed by a single security researcher or small security teams to fit the constraints of a specific project. As an example, some of the projects previously mentioned are broken at the time of writing because of outdated dependencies or the use of Python 2, which is no longer maintained. The variety of hardware and software tools leads to an uncoordinated effort and complicates the

use of these libraries and tools.

Finally, some software tools have been developed to perform a specific attack or implement high level features that may be relevant during a security analysis. For example, a tool named JackIt [JAC 2020] has been developed, implementing a subset of MouseJack [Newlin 2016a] vulnerabilities targeting Logitech Unifying mouse and keyboards. It relies on the python library provided by M. Newlin to interact with a nRF24 dongle embedding the RFStorm firmware [Newlin 2016b] and perform a keystroke injection attack. Similarly, multiple tools have been built over the HCI interface for Bluetooth Low Energy, allowing to interact with standard Bluetooth dongles. While some tooling is generally provided by the Bluetooth stack implemented on the Host (for example, the BlueZ [BLU 2000] linux stack provides a set of command line tools such as gatttool, hcitool and bluetoothctl), they are generally difficult to use and not very flexible, as they can't be easily customized nor integrated into a custom workflow without significant modification. As a consequence, several alternative tools have been developed to fill this gap. For example, evilsocket provided BLEAH [BLE 2019], an information gathering tool allowing to connect to a Bluetooth Low Energy device to collect the services and characteristics exposed by the GATT server, and displaying them in a human-readable format. Another tool, named nRF Connect [NRF 2022] and released by Nordic SemiConductors, is available on Android and iOS, and can be used to perform information gathering targeting Bluetooth Low Energy devices. It allows to scan the advertisements transmitted by BLE devices, establish a connection with them and interact with the GATT server easily. It also provides basic features to mimick a simple Bluetooth Low Energy device. While this device implement a significant number of features useful from a security perspective, it is proprietary and can't be easily customized nor automated.

Two major tools have also been released, allowing to perform Man-in-the-Middle attacks targeting Bluetooth Low Energy devices. BTLEJuice [Cauquil 2016], released by D. Cauquil, takes advantage of the fact that a device stops advertising when it enters connected mode to perform a connection with the target device, expose a cloned version of the device and wait for an incoming connection from another device to setup a Man-in-the-Middle scenario. Similarly, GATTacker [Jasek 2016], developed by S. Jasek, exploits a race condition by advertising cloned advertising packets faster than the legitimate spoofed device to capture the connection before it and establish a Man-in-the-Middle scenario. While these tools implement a very relevant attack from a security perspective, they are significantly limited by the high level libraries they rely on. Indeed, both are based on two nodeJS libraries called bleno [BLE 2018] and noble [NOB 2018]. These libraries allow to implement Bluetooth Low Energy Peripherals and Centrals, but they are not designed to be used together and imply the use of two different operating systems. To address this issue, GATTacker's developer rewrote the code of these libraries, resulting in a non standard forked version which is especially difficult to maintain, while the architecture of BTLEJuice was composed of two software components running on different OS and communicating thanks to a complex WebSockets architecture,

significantly increasing the complexity of the tool. The use of high level software components and libraries, which are not suited for security analysis, is a recurrent issue of these tools. Some projects, such as PyBT [PYB 2015] from Mike Ryan, have tried to interact directly with the HCI interface, but are still incomplete and require a significant effort of engineering to be efficiently used in a security context.

2.2 Protocol attacks

In this section, we present the state of the art of offensive security for various wireless communication protocols used in IoT. We first present the major attacks that have been identified for Bluetooth Low Energy, Zigbee and various proprietary protocols such as Enhanced ShockBurst or ANT. We also discuss the existing strategies that could be leveraged during a cross protocol pivoting attack, aiming at exploiting a given protocol to interact with another one which is not natively supported.

2.2.1 Bluetooth Low Energy

In the past few years, multiple attack strategies and tools targeting the BLE protocol have been released.

Sniffing a Bluetooth Low Energy connection is a non-trivial task, because of the channel hopping algorithm used by the devices when they are in *connected mode*. In [Ryan 2013a], M. Ryan demonstrated that a specific connection can be easily sniffed if the sniffer successfully receives the packet initiating the connection which includes the initial channel hopping parameters. He also showed that an attacker may be able to retrieve the parameters of an already established connection by monitoring specific events. This approach was then improved by D. Cauquil in [Cauquil 2017b] to infer the channels to listen to.

In [Cauquil 2019], D. Cauquil also adapted the sniffing strategy to deal with a new algorithm based on a pseudo-random generator that has been introduced in the BLE 5.0 specification [Blu 2019], called *channel selection algorithm #2*. Finally, a new tool named *Sniffle* has also been released [Qasim Khan 2019] by S. Qasim Khan. It provides interesting features such as support for the new physical layers introduced in the BLE 5.0 specification or a mode allowing to follow the target device hopping along the advertising channels. Since these channels are used to broadcast data and indicate the presence of a specific device, the probability of a successful sniffing is increased. Sniffing Bluetooth Low Energy advertisements is easier because it only relies on three advertising channels with predictable parameters. Several works have focused on the privacy of Bluetooth Low Energy protocol, especially by leveraging the passive observation of these advertisements to acquire privacy-sensitive information [Celosia 2020b, Celosia 2020a, Celosia 2019].

Multiple active attacks have also been presented in recent years. First, jamming-based attacks have been explored by Brauer et al. in [Bräuer 2016], they demonstrated an attack allowing to selectively jam advertisements. D. Cauquil also presented a new offensive tool named *BTLEJack* [Cauquil 2018] allowing to disrupt an

existing connection by jamming packets transmitted by one of the devices involved in a connection, called *Slave*. The direct consequence of this jamming strategy is a disconnection of the other device, named *Master*, allowing the attacker to synchronise with the *Slave* instead of the legitimate device, resulting in hijacking the *Master* role during an established connection. However, this strategy cannot be used to hijack the *Slave* role, which could also be relevant from an offensive perspective, and, being based on a jamming technique, is highly invasive and visible.

Second, two major tools, *GATTacker* [Jasek 2016] by S. Jasek and *BTLEJuice* [Cauquil 2016] by D. Cauquil, can be used to perform a *Man-in-the-Middle* attack. *GATTacker* clones the advertisements transmitted by the target device (called *Peripheral*) to indicate its presence and tries to advertise them faster, forcing the device initiating the connection (also known as *Central*) to connect on a cloned *Peripheral* controlled by the attacker. The approach adopted by *BTLEJuice* directly establishes a connection with the target *Peripheral*, forcing it to stop advertising, then it exposes a cloned *Peripheral* to the *Central*. Both of these strategies are based on advertisements spoofing: as a consequence, they can only perform a Man-in-the-Middle attack if the connection is not already established.

Multiple studies have also addressed the security of authentication and encryption mechanisms in BLE connections. In 2013, M. Ryan presented *CRACKLE* [Ryan 2013b], a tool exploiting a weakness in the first version of the BLE pairing process to quickly bruteforce the keys involved in the BLE *connected mode* security. In [Antonioli 2019], Antonioli et al. introduced an attack named *KNOB* (Key Negotiation of Bluetooth), to downgrade the key entropy from 16 to 7 bytes, which drastically reduces the attacker's effort to bruteforce the key. In [Antonioli 2020], they also analysed the Cross-Transport Key Derivation, a mechanism allowing to share keys between Bluetooth Classic and BLE, and demonstrated four attacks named *BLUR attacks* abusing this feature, allowing to impersonate a device, manipulate traffic or establish a malicious session. Similarly, Wu et al. demonstrated *BLESA* [Wu 2020a], an active attack abusing the reconnection process of an already paired *Central* to impersonate the corresponding *Peripheral* and transmit some unencrypted spoofed data. Von Tschirschnitz et al. presented a method confusion attack [von Tschirschnitz 2021] aiming at forcing the pairing of two devices using different methods. While some of these attacks can be used to impersonate a device, none of them can hijack such a device during an established BLE connection.

Previous research have also focused on discovering vulnerabilities that are linked to the stack implementation rather than the protocol specification, such as *Blueborne* [Armis 2017] in 2017 or *BleedingBit* [Armis 2018] in 2018. Also, in [Garbelini 2020], Garbelini et al. presented a fuzzing framework named *SweynTooth* targeting various BLE stacks, discovering a dozen of vulnerabilities. While their consequences are generally severe, they are related to specific implementations and cannot be generalised.

2.2.2 Zigbee

Multiple attacks targeting the Zigbee protocol have also been discovered in the past few years. Zigbee is a very commonly used protocol in Internet of Things, its lower layers being based on 802.15.4 specification.

One of the major contribution to Zigbee offensive security is the presentation at ToorCon 11 by J. Wright, who presented both its KillerBee [Wright 2009] framework and a set of attacks allowing to impact a Zigbee network. He presented several solutions allowing to recover the encryption key of a Zigbee network, so-called Network Key. Indeed, Zigbee keys can be pre-installed in devices by the manufacturer, but can also be provisioned using Over the Air (UTA) procedure. He suggested to dump the RAM of a node to recover the key if the pre-installed method is used, and noted that extracting the key during an OTA provisioning process is trivial as the key is transmitted in plaintext, allowing a passive attacker to recover it by sniffing packets. He also noted that 802.15.4 doesn't provide any anti-replay mechanism, allowing an attacker to replay previously observed packets even without knowing the network key. He also described a Denial of Service attack based on flooding fake Association Request packets to the network's center node, named Coordinator. These packets being transmitted in plain text, they are easy to inject from an attacker's perspective and don't require the acquisition of the network key. The coordinator provides a new address for each fake Association Request, leading to a fast consumption of the address space: as a result, a legitimate End Device trying to associate with the network is rejected because no address is available. The provided framework allows to easily manipulate the traffic by providing injection, sniffing and jamming primitives.

In [Zillner 2015], T. Zillner et al. present at BlackHat 2015 a new offensive framework named SecBee [SEC 2015] and multiple attacks targeting Zigbee networks. They mainly studied security mechanisms involved by Zigbee and noted that even if encryption can be performed at multiple layers (especially the Network and Link Layer), only the Network Key is used in practice. They also noted the existence of default keys provided as examples in the specifications, that could be used in the wild for badly configured networks. They also propose an offensive strategy based on the transmission of a Reset To Factory Settings command packet, forcing the targeted node to perform a new association with the network. They were then able to impersonate the Coordinator and enroll the corresponding node into a malicious network. They also underline that a jamming attack could trick a user into initiating a new pairing.

In [Cao 2016], X. Cao et al. discuss the impact of denial of services attacks exploiting energy depletion to consume the legitimates nodes energy and disrupt a Zigbee network. They propose several strategies allowing to perform such an attack. They noted that some mechanisms, especially those related to processing security headers, are costly in term of energy resources and that injecting malicious packets including malformed security headers could lead to such a denial of service. They also proposed the exploitation of channel sensing and contention

based access nature of the IEEE 802.15.4 CSMA/CA protocol: by continuously flooding the link, the attacker may disallow any legitimate use of the link by legitimate nodes, while forcing them to monitor it continuously. They noted that such a consumption attack may result in resetting some security-related values in the targeted nodes, situation that could be abused to perform more complex attacks. In [Vidgren 2013b], Vidgren et al. proposed an attack, allowing to force an End Device into polling mode by spoofing a Router or Coordinator node and continuously answering to poll requests from the targeted End Device, forcing it to stay awake and consume its battery life. In [Olawumi 2014], Olawumi et al. described another Denial of Service attack based on artificially incrementing the sequence number using a malformed encrypted packet to desynchronize the legitimate node's sequence numbers and force the rejection of legitimate frames. In [Krivtsova 2016], I. Krivtsova et al. presented a broadcast storm attack targeting a Zigbee network by flooding Broadcast frames. It forces every legitimate node to retransmit the malicious broadcast packets, impacting both their power consumption and the link usage. This attack is especially effective as it amplifies the malicious traffic by abusing broadcast mechanism.

Finally, some attacks are also linked to specific implementations of the Zigbee stack. XBee is one of the most popular implementation of Zigbee, and includes its own proprietary mechanisms over Zigbee. In [Vaccari 2017], I. Vaccari et al. noted the existence of proprietary XBee packets allowing to remotely transmit AT commands to XBee node, allowing an attacker to remotely alter the configuration of the node. Such a situation is especially critical as it may allow to perform Denial of Service, Hijacking or Man-in-the-Middle attacks by forcing the node to join a malicious network. Similarly, in [Ronen 2017], Ronen et al. show that a vulnerability affecting Philips Hue light bulbs allows an attacker to control the connected objects and corrupt their *firmware* in order to create an IoT worm.

2.2.3 Proprietary protocols

In this subsection, we describe the known attacks that have been presented in the recent years, targeting several proprietary protocols commonly used in wireless devices and connected objects. Indeed, a significant amount of devices relies on proprietary communication protocols to communicate, complicating their security analysis. For example, the ANT protocol is wildly used by fitness and sport-oriented devices, such as heart rate sensors or sport-oriented smartwatches. Only the higher layers of the protocol are documented, requiring an extensive reverse engineering process to analyze the corresponding communications. Similarly, wireless keyboards have massively made use of proprietary protocols over the years, and several severe security flaws linked to the design of these protocols have been found recently.

2.2.3.1 ANT protocol

ANT protocol is a proprietary wireless communication protocol commonly used in fitness and sport-oriented devices. The upper layers are documented but no specification is available for the lowest layers, especially the physical and the link layer. Two main protocols are built over ANT: ANT+ and ANT-FS. ANT+ is a protocol aiming at facilitating the interconnection between devices, by providing a default ANT configuration that is publicly available and can be used by any ANT+ compliant device. ANT-FS aims to facilitate file sharing between ANT-enabled devices, and is commonly used to provide over the air firmware update.

As far as we know, the most advanced work about ANT protocol security is the presentation at DEFCON 24 by T. Szakaly [Szakaly 2016]. He presented the ANT protocol internals and highlighted several serious security issues. He noted that some security mechanisms provided by the protocol seems to rely on security by obscurity and could be easily bypassed, such as the pairing bit mechanism or the network key. Similarly, he noted that fundamental security measures such as encryption can't be used in ANT+ protocol, which is the most commonly used implementation of ANT. He also reverse engineered the ANT-FS protocol and showed that the packets are transmitted in plain text over the air and can be easily sniffed using an SDR device. He highlighted that authentication methods provided by ANT-FS can be bypassed and managed to setup a Man-in-the-Middle attack between two ANT-FS devices. Similarly, B. Dixon presented at DEFCON 27 an attack aiming to cheat in e-cycling by injecting ANT+ data to an ANT+ dongle [Dixon 2019].

2.2.3.2 Enhanced ShockBurst, Logitech Unifying and Mosart

Wireless input devices, such as wireless keyboards and mices, are generally based on proprietary wireless protocols. Most of them seem to be derived from the ShockBurst and Enhanced ShockBurst protocols, two variants of a proprietary protocol provided by Nordic SemiConductors based on a Gaussian Frequency Shift Keying (GFSK) modulation in the 2.4GHz ISM band. While the Enhanced ShockBurst is well documented, several manufacturers, such as Logitech, Microsoft or HP, seem to implement their own undocumented proprietary applicative layers over ShockBurst or Enhanced ShockBurst protocol.

These kind of devices have been actively studied in the recent years, as they are especially critical from a security perspective. The first major contribution to this field is Keykeriki, presented at the CanSecWest 2010 conference by T. Schroeder and M. Moser [Schroeder 2010]. They both studied old protocols using the 24MHz frequency band and new protocols based on the 2.4 GHz ISM band, and built a custom hardware allowing them to sniff these proprietary protocols. They highlighted several security issues linked to these protocols, such as weak encryption used by Microsoft keyboards (the address can be easily eavesdropped and is used as input key by a custom XOR-based encryption, allowing the attacker to decrypt the traffic) or Logitech mouse packets being transmitted without encryption.

M. Newlin has presented his own research about wireless mices security at DEFCON 24 [Newlin 2016a]. He presented MouseJack, a set of critical vulnerabilities targeting wireless mices from multiple manufacturers (e.g., Microsoft, Logitech, Amazon, HP, EagleTek). He demonstrated several security attacks allowing to inject unencrypted packets (allowing to inject arbitrary keystrokes), bypass some encryption mechanisms allowing to force valid encrypted packets without knowing the encryption key, force a malicious pairing, perform a denial of service and in some specific cases, eavesdrop the keystrokes-related packets, allowing to build a wireless keylogger. He also significantly contributed to tooling by releasing RFStorm firmware for nRF24 dongle, as we previously mentioned in the tool section.

In 2019, M. Mengs released Logitacker [LOG 2019], a set of critical vulnerabilities targeting the Logitech Unifying protocol. He mainly focused his work on the pairing process used to share the encryption key between the dongle and the wireless keyboard, and highlighted that this process is insecure and could be passively eavesdrop by an attacker, allowing him to recover the key by applying simple transformations to the collected data. He also published several bypasses for some of the MouseJack security patches, allowing to inject malicious packets to a Logitech Unifying dongle. He also released a custom firmware implementing these attacks on a nRF52840 dongle from Nordic SemiConductors.

2.2.3.3 Other protocols

Various other proprietary protocols have also been analyzed by security researchers. For example, S. Kamkar presented OpenSesame [OPE 2015], an attack based on a custom hardware he built by diverting a toy computer to open garage doors. The attack exploits the fact that the corresponding wireless protocol relies on fixed codes and doesn't use preamble to quickly bruteforce it using a De Bruijn sequence transmitted continuously, allowing to dramatically reduce the bruteforce duration. He also presented at DEFCON 23 [Kamkar 2015] a complex replay attack aiming to break rolling codes based protocols (generally used by car keys) by jamming two consecutive valid signals, then replaying the first one to allow access to the user, while being able to replay the second one which is still valid.

D. Cauquil also presented RadioBit [Cauquil 2017a], a custom firmware for the BBC:MicroBit allowing to interact with several RF protocols using a simple python CLI, including ShockBurst, Enhanced ShockBurst and a proprietary protocol used by a mini-drone, the Cheerson CX-10. He managed to hijack the legitimate remote to control the drone by flooding control commands faster than the legitimate device.

2.2.4 Cross-technologies attacks

This subsection briefly presents different attack strategies to carry out a pivoting attack. Firstly, the case of IoT devices supporting multiple radio protocols is discussed, then an overview of the few existing research works that considered such an attack on a device supporting a single specific radio protocol is presented.

2.2.4.1 Multi-protocol devices

A pivoting attack aims at taking advantage of the coexistence of multiple protocols in the same environment in order to compromise new objects. The most natural approach for this attack is to compromise an object supporting multiple radio communication protocols, allowing to perform the attack using the provided API. As an example, in [Bachy 2015], Bachy et al. compromise a smart-TV using *HbbTV* communication protocol, then use it to reconfigure the firewall embedded in the ADSL box using LAN protocols (Ethernet or WiFi).

Several hardware devices allow such attacks to be carried out. For instance, *Software Defined Radio* devices are designed for a generic purpose, allowing communications through multiple protocols, regardless of their modulation and frequency bands. However, so far, these devices are only used for prototyping and experimentation purposes.

There are also chips that integrate different wireless devices. For example, *B-L475E-IOT01A* [IOT 2018], based on the *STM32L4* micro controller intended for IoT devices, supports multiple wireless protocols (such as *Bluetooth*, *WiFi* or *NFC*). Similarly, the *CC2652R* [CC2 2019] from *Texas Instruments* is compliant to multiple radio technologies in the ISM band. The compromise of such a chip greatly facilitates the implementation of a pivoting attack targeting one of the wireless protocols supported by the chip. However, such chips are expensive and their use is quite specific, which limits their deployment in IoT networks.

2.2.4.2 Single-protocol devices

Since most connected objects only embed one wireless device, the practical implementation of a pivoting attack is much more complex. We are not aware of existing research specifically addressing this issue from an offensive perspective. However, some contributions explored related topics.

The most relevant contributions are related to *Cross-Technology Communications* (*CTC*) solutions, that are aimed providing a communication system between two single-protocol devices supporting heterogeneous wireless communication protocols. However, to our knowledge these contributions did not investigate the use of this technology in security or in an offensive perspective. There are two main categories of *CTC*, named *Packet-level CTC* and *Physical layer CTC*.

The *Packet-level CTC* approach relies on some information linked to the packets. As an example, K. Chebrolu et al. purposely adapt the packet duration in order to encode data [Chebrolu 2009], while the *FreeBee* [Kim 2015] approach by S. Min Kim is based on the time interval between beacon frames. From an offensive perspective, these approaches could be interesting to exfiltrate some data, but they are not relevant for pivoting attacks. Other limitations, such as a low data throughput, are inherent to these approaches and hamper their practical use.

Physical layer CTC approaches consist in emulating a technology using the signal generated by another one. As an example, Z. Li et al. simulate a *Zigbee* frame using a WiFi transceiver [Li 2017]. Similarly, W. Jiang et al. have presented

an approach named *BlueBee* [Jiang 2017], allowing to simulate *Zigbee* frames using a *BLE* transceiver, and another approach called *XBee* [Jiang 2018], enabling to receive *Zigbee* frames from a *BLE* receiver. However, these solutions have major limitations that prevent their use in an offensive perspective. As an example, the selection of a *Zigbee* channel by *BlueBee* is based on the channel hopping algorithm of *BLE* connected mode, so it requires to establish a *BLE* connection with another *BLE* device. Similarly, adding a specific identifier before the data included in the frame is needed in order to receive a *Zigbee* frame using *XBee*, so it requires the cooperation of the *Zigbee* transmitter. These constraints can be easily addressed if the use of *CTC* is legitimate and deliberate, however they prevent the use of these solutions in a context of attack and especially for pivoting attacks.

The *Packet-in-Packet* strategy [Goodspeed 2011b], proposed by T. Goodspeed et al. consists in encapsulating a complete radio frame into an application-level *payload*: a misidentification of the beginning of the encapsulating frame by the receiver (e.g., due to interferences causing bitflips during the demodulation) can lead to the interpretation of the encapsulated frame. This strategy is particularly interesting for bypassing software checks performed at the protocol layer, and may thus allow attackers to access and control the lower layers of the radio device. The authors highlight a possible use of this attack to perform a pivoting attack, e.g., to inject radio traffic corresponding to a wireless protocol different from the protocol natively supported by the radio device, under certain specific conditions. However, this strategy can only be applied to a limited number of protocols, and can only be achieved if the modulations used have similar characteristics (frequency bands, bandwidth, etc). For instance, M. Millian and V. Yadav discuss the possibility of encapsulating *802.15.4* traffic into 802.11 frames [Millian 2015]. However, they stress the difficulty of such a strategy due to the differences between the two technologies.

T. Goodspeed [Goodspeed 2011a] has also discovered the vulnerability in the *nRF24L01+* chip we mentioned earlier, that allows to sniff and inject packets on various protocols based on *Gaussian Frequency Shift Keying* modulation. The attack abuses a chip register initially designed to hold an address, that can be diverted to select an arbitrary preamble and sniff Enhanced ShockBurst frames without prior knowledge of the address. However, it is also possible to divert the use of this register to detect specific preambles used by different wireless technologies, as long as similar modulations and bit rates are used. This vulnerability has been used by M. Newlin to develop a firmware aiming to add advanced sniffing capabilities for the *Enhanced ShockBurst* and *Mosart* protocols to the *nRF24* chip [Newlin 2016a].

D. Cauquil has also disclosed a similar vulnerability in other *Nordic Semiconductors* chips [Cauquil 2017b], and has developed a similar tool for the *nRF51*. He was then able to implement communication primitives for a proprietary protocol not initially supported by the chip, allowing it to control a mini-drone [Cauquil 2017c]. An implementation of these primitives has been integrated into the *radiobit* [Cauquil 2017a] project.

Finally, in [Camurati 2022], G. Camurati et al. demonstrated that shaping arbitrary signals out of electromagnetic noise is possible from unprivileged software,

and can be used to transmit arbitrary signals from a smartphone. They demonstrate the practicability of such an attack to interact with several protocols. While this result is one of the most promising work related to this topic, the attack is constrained in term of power, frequency and bandwidth by the properties of the leakage and can only transmit arbitrary signals. Similarly, in [Bratus 2016], Bratus et al. discuss this kind of cross-technology attacks by illustrating it on simple modulations and proposes a classification of physical layers to facilitate the identification of such strategies. We can also note the existence of research works leveraging a similar strategy in other areas, aiming at diverting specific file formats to imitate other ones [Koch 2022, Albertini 2013] (so-called polyglot files).

These research works present some first techniques and experimental results that illustrate the practical feasibility of pivoting attacks targeting wireless protocols. However, these techniques have several limitations which strongly restrict their use: they require an active cooperation of other devices, or the modulation of the native protocol and the pivoting protocol must be similar and sometimes depend on the use of specific chips (such as *Nordic SemiConductors nRF24* and *nRF51* chips).

2.3 Defensive approaches

In this section, we present some of the research works that have implemented defensive approaches adapted to this new wireless communication protocols. Several studies discuss the relevance of classical security approaches and investigate new mitigation measures, especially intrusion detection systems. We mainly focus on intrusion detection oriented works, as we explored this specific topic in our defensive contributions.

2.3.1 Signal-based approaches

Some works have focused on exploiting physical level indicators to identify security threats: they analyze the RF signals linked to the wireless communication protocols to protect in order to detect intrusions. For example, Roux et al. [Roux 2018] provide a protocol-independent approach based on the analysis of the physical layer using *Software Defined Radios*, to monitor wireless communications and detect intrusion attempts. They rely on the sweep features provided by HackRF one tooling to monitor wide frequency bands and identify malicious behaviours using a Machine Learning-based approach. While the approach is especially interesting because of its genericity and because it doesn't rely on prior knowledge of the protocols, the results seem less convincing in the overcrowded 2.4 GHz band, which is widely used by IoT devices. In [Fragkiadakis 2012], A. Fragkiadakis et al. exploited a metric based on Signal-to-Interference-plus-Noise-Ratio (SINR) to detect denial of service attacks based on jamming techniques. They evaluate their approach on a standard WiFi network. Another approach by P. Umashankar et al. [Ghugar 2018] relies on analyzing some physical-layer based metrics to calculate a specific trust value per node, allowing to detect periodic jamming attacks.

Some physical approaches also try to fingerprint nodes based on their physical transmission characteristics, allowing to identify spoofing attacks. Some of these fingerprinting approaches [Hall 2005,Ur Rehman 2012,Köse 2019] exploit the transient of transmitted signals in order to identify artifacts linked to imperfections of the transmitter. The main limit of such an approach is the need of expensive devices, as the transient signal must be captured at a high sampling rate. In [Brik 2008], fingerprinting is performed by comparing the received signal with the corresponding ideal waveform and using the error as an indicator. Such an approach seems effective but is obviously dependent of prior knowledge of the modulation scheme to calculate the ideal waveform. Similarly, in [Helluy-Lafont 2020], E. Helluy-Lafont et al. use timing indicators linked to frequency hopping and frequency deviation to identify Bluetooth nodes. In [Galtier 2020], F. Galtier et al. provide a fingerprinting approach aiming at detecting spoofers by analyzing the DSP of the received signals. They demonstrate the relevance of such an approach on Bluetooth Low Energy and Zigbee nodes.

2.3.2 Packet-based approaches

Some works also exploit higher level indicators to perform detection in multiple protocols. In [Siby 2017], S. Siby et al. monitor traffic of several protocols using dedicated transceivers (e.g., Ubertooth, RZUSBStick) in order to automatically identify and classify the nodes. The solution is however especially difficult to maintain as it relies on specific transceiver for each monitored protocol. In [Tournier 2020], J. Tournier et al. use an abstract representation of packets to monitor and analyze heterogeneous wireless protocols, providing a tool named IoTMAP which is able to analyze BLE, Zigbee and 6LowPAN networks.

However, most of Intrusion Detection approaches focus on a specific protocol. For example, in [Miettinen 2016], M. Miettinen et al fingerprint devices using WiFi link layer indicators and identify vulnerabilities based on a CVE database. In [Raza 2013], S. Raza et al. focus on 6LowPAN and provid a Network Intrusion Detection System named SVELTE allowing to detect routing attacks, which relies on monitoring software embedded both in the nodes and in a 6LowPAN router. In [Jokar 2016], F. Sadikin et al. propose a Zigbee Intrusion Detection System combining rule-based intrusion detection and machine learning-based anomaly detection to cover efficiently both known and unknown attacks.

In the specific case of Bluetooth Low Energy, we already highlighted several major research works [Ryan 2013a, Cauquil 2017b, Cauquil 2019] focused on allowing eavesdropping of Bluetooth Low Energy communication, which is not trivial because of the use of channel hopping algorithms. Although these works are not necessarily focused on a defensive perspective, they are still relevant for BLE Intrusion Detection, as most of existing defensive works rely on them to monitor BLE traffic. Unfortunately, these passive approaches suffer from several serious limitations that have a consequent impact on the completeness and representativity of monitored communications. First, most of these sniffers are only capable of monitoring one

connection at a time. G. del Arroyo et al. have attempted in [Gutierrez del Arroyo 2017] to solve this issue by implementing an opportunistic algorithm based on a scheduler in the Ubertooth one that allows monitoring multiple connections simultaneously. While this work looks promising, it is still limited by the underlying hardware and may miss some packets depending on the environment. Second, most existing implementations are unstable, partially because of the use of various heuristics, that are not suitable for some devices (for example, some smartphones change the channel map frequently, making it difficult for the sniffer to infer this parameter). As far as we know, comprehensive monitoring of the Link Layer traffic of BLE communications from an external probe, especially in connected mode, remains an open challenge. This has a consequent impact on the research works aiming at developing intrusion detection systems for this protocol: since most of them are based on the aforementioned sniffers, they generally rely on advertisements monitoring only, limiting their scope to spoofing or denial of services attacks targeting the advertising mode. In [Wu 2020b], J. Wu et al. presented BlueShield, an approach to detect spoofing attacks by profiling monitored devices using multiple features inferred from the advertisements packets. In [Sung 2016], Y. Sung et al. explored the use of Received Signal Strength Indicators (RSSI) to detect intruders. In [Yaseen 2019], M. Yaseen et al. presented *MARC*, a framework to detect Man-in-the-Middle attacks, by exploiting four features inferred from advertisement packets such as the advertising interval or RSSI levels. Other research works also explore the analysis of traffic in connected mode. In [Newaz 2020], A. Newaz et al. combine an n-gram-based approach with various machine learning techniques to detect various attacks by analyzing irregular traffic-flow patterns on Personal Medical Devices. Similarly, in [Lahmadi 2020], A. Lahmadi et al. explore the use of Machine Learning techniques to identify Man-in-the-Middle attacks by building a model of legitimate behaviors based on features such as RSSI, channels numbers or distance. While these works provide interesting results regarding traffic analysis, they performed offline detection on datasets, and are difficult to deploy in practice.

Some works also focused on building IDS for Bluetooth Low Energy Mesh networks. In [Lacava 2021], A. Lacava et al. provide a distributed IDS based on the deployment of watchdogs nodes within the network, capable of collecting local traffic and detecting attacks using cooperative decision making. Furthermore, in [Krzysztoń 2020], M. Krzyszton et al. perform simulations to choose optimal placements for watchdogs in this type of cooperative IDS. The distributed approach adopted by these research works are especially relevant for Mesh networks, but seems difficult to apply to existing Bluetooth Low Energy devices, that generally do not use routing mechanisms.

2.4 Outline

In the previous sections, we highlighted several research topics, issues and challenges that have motivated the contributions of this work. In this section, we present

the structure of the following chapters, the corresponding contributions and our scientific positioning for every covered topic. The following chapters are splitted into two main parts: the first part describes our offensive contributions, while the second presents our defensive contributions.

The first offensive contribution we introduce in chapter 3 is an exploration of cross-technology pivoting attacks. Indeed, we have highlighted in subsection 2.2.4 that existing approaches, such as Cross Technology Communications, are not suited for an offensive context because of serious limitations. While these limitations, such as the need of a cooperation between nodes, are not problematic from a functional perspective, they disallow the use of such strategy in an offensive context. In this chapter, we explore this topic from an offensive perspective and demonstrate that implementing cross-protocol pivoting attacks is feasible and must be considered as a serious threats for wireless communication protocols. We introduce *WazaBee*, an attack aiming at diverting Bluetooth Low Energy transceivers to receive and transmit 802.15.4 packets, and discuss the implementation of such pivoting attacks on off-the-shelf devices such as smartphones or connected objects.

The chapter 4 introduce our second offensive contribution, named InjectaBLE, which is a critical low level injection attack targeting the Bluetooth Low Energy protocol. Indeed, we highlighted in 2.2.1 that existing Bluetooth Low Energy attacks present several limitations: for example, hijacking the Slave's role or establishing a Man-in-the-Middle during an established connection is not possible using existing approaches. We demonstrate in this chapter that our injection strategy allows to perform new kind of attacks, solving the mentioned limitations, while being especially difficult to fix as it is linked to a fundamental Bluetooth Low Energy low level mechanism. We demonstrate its feasibility, evaluate the impact of several parameters on its success and propose some counter-measures allowing to limit its impact.

The third contribution, presented in chapter 5, aims to provide a framework for offensive development, solving issues linked to the current state of offensive tooling in wireless security. Indeed, in 2.1 we presented the wide variety of available hardware tools, exposing various capabilities and APIs, while discussing the limits of existing software tools, such as the use of high level libraries. This situation is problematic, because it generates a chaotic environment with outdated or unmaintained tools impacting the reproducibility of security research in wireless security. We present Mirage, an offensive framework allowing to facilitate the reproducibility of such researches by providing a generic, modular and user-friendly framework, which can be easily extended to add new attacks, protocols or hardware drivers. It also allow the design of complex attack workflows combining several attacks.

The second part is dedicated to the exploration of innovate defensive solutions, especially on intrusion detection and prevention in peer-to-peer wireless communication protocols. In 2.3, we highlighted how difficult it is to exhaustively monitor Bluetooth Low Energy communications using an external probe performing sniffing, and how it impacts existing intrusion detection approaches. In chapter 6, we present an innovative detection approach based on embedding detection modules

directly in Bluetooth Low Energy controllers, resulting in a decentralised IDS design. We highlight the main technical challenges that complicate the deployment of such a system and describe Oasis, a modular and generic framework aiming at facilitating the instrumentation of proprietary Bluetooth Low Energy controllers.

Our second defensive contribution focuses on the development of a firewall dedicated to peer-to-peer wireless communication protocols. Indeed, as far as we know, very few research works have explored this topic. It is indeed especially difficult to filter the wireless traffic in such protocols because of the absence of central point that could be used to efficiently analyze and filter packets. In chapter 7, we explore the use of offensive techniques such as reactive jamming in order to build a generic filtering strategy that could be used to mitigate wireless attacks, and demonstrate its feasibility on Zigbee and Enhanced ShockBurst networks.

Part II

Low-level attacks

Chapter 3

Cross-protocol attacks

Contents

In this chapter, we focus on a specific threat for IoT environments that has been seldom studied from an offensive perspective until now. It takes advantage of similarities in the physical layers of heterogeneous wireless technologies, which may co-exist in the same environment. We investigate the possibility to divert the behaviour of a given device dedicated to a specific radio protocol, to make it communicate through another radio protocol not initially supported by the device, in order to perform malicious activities. The feasibility of such communications

between heterogeneous protocols has been explored in some previous works, commonly called *Cross Technology Communications*. However, existing solutions always assume a cooperation from the surrounding devices to allow such a transmission. This assumption is not realistic from an offensive perspective, which makes such attacks very unlikely. The approach investigated in this chapter does not rely on this assumption, thus increasing its practical feasibility. Specifically, the proposed approach, called *cross-technology pivoting attack*, allows to establish a communication channel between chips supporting *BLE*, and other wireless protocols such as *Zigbee*, *Enhanced ShockBurst* or *Mosart*, in order to perform various types of attacks. The ubiquity and wide deployment of BLE devices make these attacks critical as the attack surface of targeted wireless networks is significantly increased.

3.1 Motivations

In chapter 1, we highlighted several issues and challenges related to the security of wireless communication protocols in the context of *Internet of Things*. One of the main consequences of the expansion of IoT devices is the chaotic deployment of heterogeneous wireless communication protocols. This deployment leads to the co-existence of multiple wireless protocols in the same environments, which may introduce serious security risks. Indeed, even if these protocols are not supposed to interact together, they generally share the same frequency bands and rely on similar technology. Moreover, the context of *Internet of Things* leads to the formation of complex environments, which are decentralized, heterogeneous and dynamic. This situation is problematic because it dramatically increases the attack surface exposed by IoT environments, and opens the opportunity to attackers to set up novel offensive strategies that are difficult to anticipate from a defensive point of view.

This situation leads us to analyze the specific security threats that could be linked to this co-existence, and especially the ones resulting from unexpected interactions between heterogeneous protocols. More precisely, we focus our work on *cross-protocol pivoting attacks*: we explored the feasibility of diverting a specific transceiver dedicated to a given wireless protocol to interact with another wireless protocol, which is not natively supported by the chip. We believe the consequences of such *pivoting attacks*, would be critical, because 1) they open the possibility of new offensive strategies quite difficult to detect, because not considered as for now, and 2) they can be deployed at a large scale because the vast majority of connected objects embed at least one radio technology, that could possibly be remotely diverted.

As an example, such a strategy can be used to perform covert channel attacks or to exfiltrate data to an illegitimate remote receiver by means of a corrupted object, by communicating through a wireless protocol that is not supposed to be monitored in the targeted environment. It can also be used to perform traditional attacks targeting a radio protocol `RP1` (man in the middle, sniffing, spoofing, etc)

from a device supporting another radio protocol `RP2` and that is not considered as a potential source of attack for the `RP1` protocol.

We focused our work on *BLE* chips that are widely deployed in many environments, because they are embedded in the BLE-connected smartphones and smart devices. In a first time, we analyzed how a *Bluetooth Low Energy* transceiver could interact with a *802.15.4*-based protocol such as Zigbee, which is based on a different modulation scheme. While several works in the signal processing community have explored this kind of interactions between heterogeneous protocols, as mentioned in section 2.2.4, we noted that the existing approaches are not suited for an offensive use. Indeed, these techniques have several limitations which strongly restrict their use: they require an active cooperation of other devices, or the modulation of the native protocol and the pivoting protocol must be similar and sometimes depend on the use of specific chips (such as *Nordic SemiConductors nRF24* and *nRF51* chips). While these assumptions are acceptable from a functional perspective, it is not realistic in an offensive context where a compromised node would try to leverage such strategy to attack surrounding devices. Our main contribution, named *WazaBee*, allows to reliably transmit and receive *802.15.4* packets from a diverted *BLE* transceiver without requiring any cooperation from surrounding devices. The attack takes advantage of some characteristics of the *BLE* protocol to allow some *BLE* devices to communicate using *802.15.4*-based protocols not initially supported by these devices. The consequences of this attack are critical because the vulnerability is not specific to some *BLE* chips but is rather related to the design and implementation of the underlying radio protocols. As a consequence, the attack is not implementation dependent and may potentially be used with the majority of *BLE* chips. In addition, the attack can be implemented easily which increases the level of the threat.

Second, we also explored the practical feasibility of implementing such pivoting attacks on off-the-shelf devices, especially mobile devices such as smartphones or connected objects. Indeed, this kind of device is more likely to be compromised because their mobility could be leveraged by an attacker to compromise devices which can't be easily accessed without pivoting attacks. We focused on diverting the proprietary firmware of a *BCM4375* Bluetooth controller embedded in a Samsung Galaxy S20 smartphone, and managed to implement several cross-technology pivoting attacks targeting Zigbee (using WazaBee strategy), but also proprietary protocols used by wireless keyboards such as Enhanced ShockBurst or Mosart. Our proof of concept, named RadioSploit, is a combination of firmware patches allowing to divert the targeted *Bluetooth* controller and an Android application providing a user interface to perform pivoting attacks. From our perspective, this work highlights the practical feasibility of such pivoting attacks and could increase awareness about the need to develop efficient protection mechanisms to prevent and mitigate this type of attack which could have critical consequences on the security of IoT environments.

3.2 Overview of wireless protocols

In this section, we introduce some definitions that are useful to understand the pivoting attack presented in this chapter. Then, we briefly present some background information about the *BLE* and *Zigbee* protocols lower layers.

3.2.1 Digital modulation

Digital modulation is defined as the process of transforming a digital signal (the modulating signal) to adapt it to the transmission channel. This transformation consists in modifying the characteristics of a sine wave, called a carrier, according to the data to be transmitted. The resulting signal is called the modulated signal.

The modulated signal is defined by the following equation:

$$s(t) = A(t)\cos(2\pi f_c t + \varphi(t)) \tag{3.1}$$

where $A(t)$, f_c, and $\varphi(t)$ represent the amplitude, frequency and phase of the signal, respectively.

Formula (3.1) can be also be written as follows:

$$s(t) = I(t)\cos(2\pi f_c t) - Q(t)\sin(2\pi f_c t) \tag{3.2}$$

with:

- $I(t) = A(t)\cos(\varphi(t))$ "In-phase component"
- $Q(t) = A(t)\sin(\varphi(t))$ "Quadrature component"

As a consequence, the state of a modulated signal at a given time can be represented by a vector in the complex plan: the norm of the vector represents the amplitude of the signal, while its argument corresponds to its phase.

Note also that equation (3.2) demonstrates that it is possible to control the instantaneous phase, the instantaneous frequency and the amplitude of a carrier wave by manipulating the amplitude of I and Q signals. This property is the basis of a so-called I/Q modulator.

3.2.2 Bluetooth Low Energy (BLE)

The *Bluetooth Low Energy* protocol, or *BLE*, is a simplified variant of the *Bluetooth* protocol, introduced in version 4.0 of the *Bluetooth* specification [Blu 2019]. In particular, it is optimised for energy saving and is commonly used in IoT networks, due to its low complexity and its wide deployment. It is also supported by default by most smartphones and computers.

In this chapter, we focus on the lower layers of the protocol, notably the physical layer. The physical layer of the protocol (*PHY* layer) describes a single packet format, illustrated in figure 3.1 and composed of the following fields:

- **Preamble:** one byte field corresponding to series of alternating bits (0x55), used to synchronise the receiver at the start of the frame,
- **Access Address:** 4 bytes field, allowing to identify a specific connection or an advertisement,
- **Protocol Data Unit (PDU):** field of variable size made up of a link-layer header (*LL Header*) and the data to be transmitted,
- **Cyclic Redundancy Check (CRC):** 3 bytes field for integrity checking based on cyclic redundancy code.

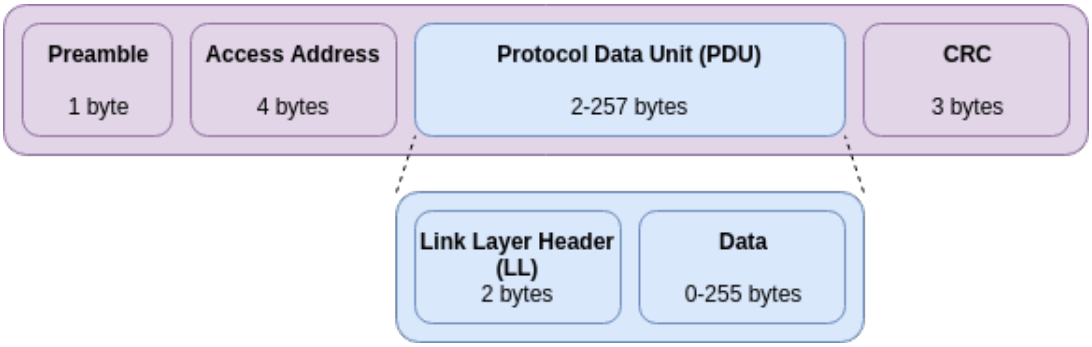

Figure 3.1: Bluetooth Low Energy Link Layer packet format

When a frame is transmitted, the data from the upper layers is prefixed with a header by the link layer (*LL*), and is encapsulated into the PDU field. The corresponding CRC is appended to the PDU. A transformation called *whitening* is then applied, allowing the generation of a pseudo-random sequence, in order to avoid the presence of long repeated sequences of 1 or 0, which could alter the transmission of the modulated signal. Finally, the preamble and the Access Address are included before the PDU and the frame is then processed by the modulator.

The physical layer of the *BLE* protocol is based on a frequency modulation, called *Gaussian Frequency Shift Keying* (*GFSK*), operating in the ISM band (from 2.4 to 2.5 GHz). It is a variant of the *2-Frequency Shift Keying* (*2-FSK*) modulation in which a gaussian filter is applied to the modulating signal to avoid abrupt changes in frequency upon symbol changes.

A *2-FSK* modulation consists in encoding two symbols (0 and 1 for binary data) by two different frequencies defined by the following formulas:

$$F_0 = f_c - \Delta f = f_c - \frac{m}{2T_s} \tag{3.3}$$

$$F_1 = f_c + \Delta f = f_c + \frac{m}{2T_s} \tag{3.4}$$

- f_c is the frequency of the carrier, called central frequency,

- Δf is the modulation deviation (defined as the lag between the frequency encoding the symbol and the frequency of the carrier),
- m is the modulation index (a value between 0 and 1 characterizing the modulation),
- T_s is the symbol duration (the inverse of the data rate).

This modulation provides a modulated signal whose signal envelope amplitude is constant and its phase is continuous over time. In addition, the instantaneous phase $\varphi(t)$ and the instantaneous frequency $f(t)$ are linked as follows:

$$f(t) = \frac{1}{2\pi}\frac{d\varphi(t)}{dt} \tag{3.5}$$

Thus, the variation of instantaneous frequency can be inferred by observing the direction of rotation of the instantaneous phase: an increase in frequency (encoding the value 1) will cause a counter-clockwise rotation of the phase, while a decrease in frequency (encoding the value 0) will cause the phase to rotate clockwise. Such a modulation can thus be represented in the complex plan by observing the direction of rotation of the phase, as illustrated in Figure 3.2.

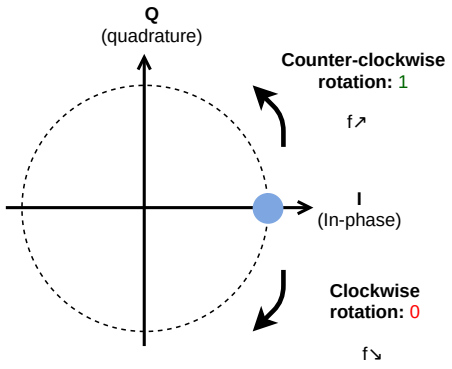

Figure 3.2: I/Q representation of a *2-FSK* modulation

BLE specification states that the modulation index must be set between 0.45 and 0.55. The symbol duration T_s depends on the mode in use. Indeed, the first versions of the specification required a data rate of 1 Mbit/s (i.e., $T_s = 10^{-6}s$). However, version 5 introduced two new operating modes for the physical layer: *LE Coded*, that is out of the scope of this work, and *LE 2M*, operating at 2 Mbits/s (i.e., $T_s = 5 \times 10^{-7}s$).

The central frequency depends on the communication channel. Indeed, the specification proposes 40 communication channels in the ISM frequency band (from 2.4 to 2.5 GHz), each with a bandwidth of 2 MHz, illustrated in figure 3.3. Three of these channels (37, 38 and 39) were initially dedicated to the broadcasting of announcement messages (advertising channels) while the other 37 channels were dedicated for data exchange in connected mode (data channels). However, the

addition of new modes, *LE Coded* and *LE 2M*, introduces the possibility to use data channels as secondary advertising channels. Each channel being identified by a number $k \in [0..39]$. The channels 37, 38 and 39 respectively use the frequencies 2402, 2426 and 2480 MHz. The other channels, from 0 to 36, are spaced of 2MHz from 2404MHz skipping those frequencies.

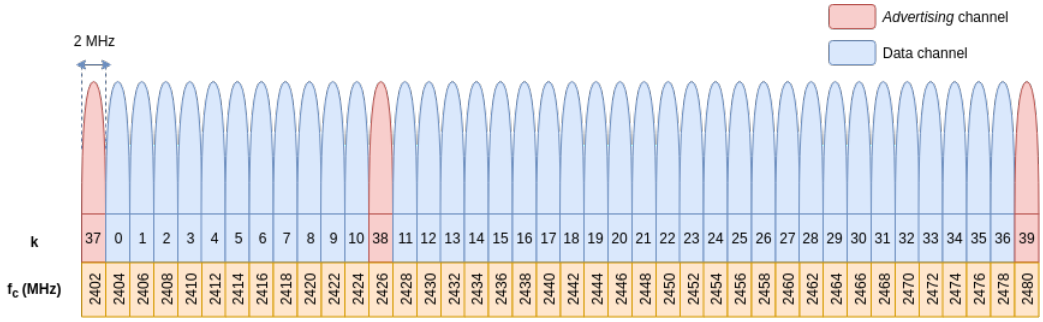

Figure 3.3: Bluetooth Low Energy communication channels

3.2.3 Zigbee

Zigbee is one of the most widespread wireless protocols in *IoT* networks. Its low power consumption, the low cost of radio devices and the ability to build complex topologies make it particularly attractive for IoT systems. It is compliant with the IEEE *802.15.4* standard [IEE 2020] which defines the physical and link layers. Its specification mainly describes the upper layers of the protocol stack (i.e., the network and application layers). In this chapter, we focus on the lower layers, and more specifically on the *802.15.4* standard physical layer. This layer (called *PHY*) defines the format of the frames (named *Physical Protocol Data Unit*, or *PPDU*), as follows (an illustration is provided in figure 3.4):

- **Preamble:** 4 consecutive null bytes field (0x00 0x00 0x00 0x00), used to synchronise the receiver with the beginning of the frame,
- **Start of Frame Delimiter (SFD):** one byte field of value 0x7A, indicating the beginning of the frame,
- **Length (PHR):** one byte field encoding the size in bytes of the Protocol Service Data Unit,
- **Protocol Service Data Unit (PSDU):** field of variable length, encapsulating the frame at link layer (or *MAC*). This frame is composed of a header, (*MHR*), the data to be encapsulated, transmitted by the upper layers, as well as a two bytes field, the Frame Check Sequence (*FCS*), used to check the integrity of the received frame.

Acccording to the *802.15.4* standard, a spread spectrum technique (*Direct Sequence Spread Spectrum* or *DSSS*) is applied to the generated frame before it is

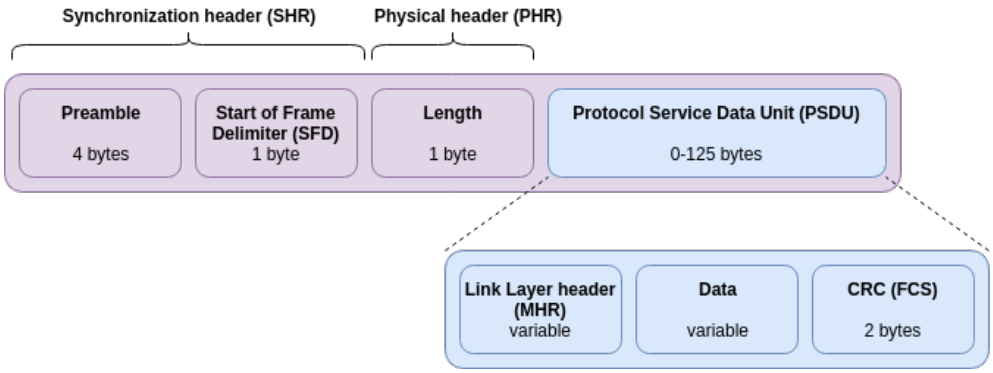

Figure 3.4: Physical Protocol Data Unit format

processed by the modulator. Each byte is split into two blocks of 4 bits, the *Least Significant Bits (LSB)* and the *Most Significant Bits (MSB)*. Each of these blocks is then substituted by a pseudo-random sequence of 32 bits, called *PN* sequence (*Pseudorandom Noise*) according to the correspondences presented in Table 3.1. The bits of this sequence are also called *chips*.

Block ($b_0b_1b_2b_3$)	PN Sequence (c_0c_1 ... $c_{30}c_{31}$)
0000	11011001 11000011 01010010 00101110
1000	11101101 10011100 00110101 00100010
0100	00101110 11011001 11000011 01010010
1100	00100010 11101101 10011100 00110101
0010	01010010 00101110 11011001 11000011
1010	00110101 00100010 11101101 10011100
0110	11000011 01010010 00101110 11011001
1110	10011100 00110101 00100010 11101101
0001	10001100 10010110 00000111 01111011
1001	10111000 11001001 01100000 01110111
0101	01111011 10001100 10010110 00000111
1101	01110111 10111000 11001001 01100000
0011	00000111 01111011 10001100 10010110
1011	01100000 01110111 10111000 11001001
0111	10010110 00000111 01111011 10001100
1111	11001001 01100000 01110111 10111000

Table 3.1: Block/PN sequence correspondence table

PN sequences are then provided as input of the modulator. The physical layer of the *802.15.4* standard is based on a phase modulation called *Offset Quadrature Phase Shift Keying* (or *O-QPSK*) with half sine pulse shaping in the ISM band. This modulation corresponds to a variant of the *Quadrature Phase Shift Keying* phase modulation, which consists in encoding the binary input information by modulating the phase of the carrier. Four phase values are used to transmit four symbols, each symbol being composed of two consecutive bits. In the specific case of *Zigbee* and *O-QPSK* modulation, each symbol is composed of 2 chips.

To generate a *802.15.4* compliant signal, it is necessary to independently control the In-phase and Quadrature components used to modulate the even bits and the odd bits, respectively. The first step consists in transforming the binary message to

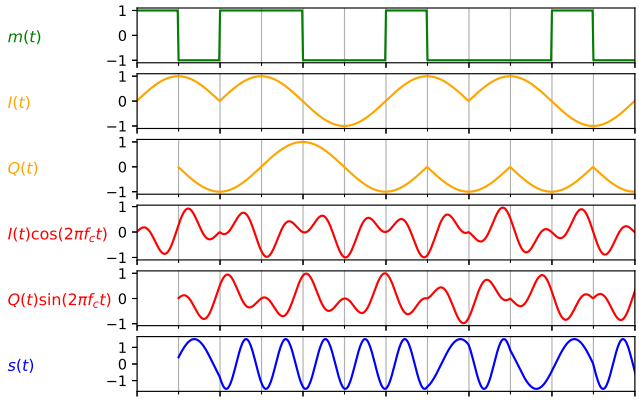

Figure 3.5: Temporal representation of *O-QPSK* modulated signal with half sine pulse shaping

be modulated into two sequences of half sine pulses of duration $T_s = 2T_b$ (where T_b corresponds to half the duration of a symbol): a 1 bit is encoded by a positive half sine pulse while a 0 is encoded by a negative half sine pulse. As a result, $I(t)$ is a sequence of half sine pulses representing the even bits while $Q(t)$ is a sequence of half sine pulses representing the odd bits. The Quadrature component is also temporally delayed of T_b in order to avoid some drawbacks linked to *QPSK* modulation.

Then, the modulated signal $s(t)$ can be generated from the In-Phase and Quadrature signals using formula 3.2. This modulation generates a signal with continuous phase jumps, evolving linearly during the period of a T_b chip: the instantaneous phase of the modulated signal thus becomes continuous as a function of time and the amplitude of the signal's envelope remains constant, as shown in figure 3.5. Thus, at each sampling instant, there are only two possible transitions to the following state: $+\frac{\pi}{2}$ and $-\frac{\pi}{2}$. The transition to be made depends on: 1) the value of the previous bit, 2) whether an even bit or an odd bit is currently modulated, and 3) the current state. For instance, if the current state corresponds to symbol 11 and if one wishes to modulate an odd bit set to 1, one will take the transition to state 01, which will cause a linear increase of $+\frac{\pi}{2}$ in the instantaneous phase during the period T_b. The constellation diagram is represented in Figure 3.6.

The *802.15.4* standard specification indicates a data rate of 2 Mchips/s in the ISM band, which corresponds to $T_b = 5 \times 10^{-7} s$. Consequently, the data rate corresponding to the bits of the *PPDU* before the substitution of the PN sequences corresponds to 250 kbits/s. The carrier wave frequency (called central frequency as in *BLE*) depends on the communication channel used. The *802.15.4* standard proposes use of 16 communication channels, from 11 to 26 with a 2 MHz bandwidth per channel, illustrated in figure 3.7. Two consecutive channels are spaced 3 MHz apart. The following formula gives the relationship between the central frequency

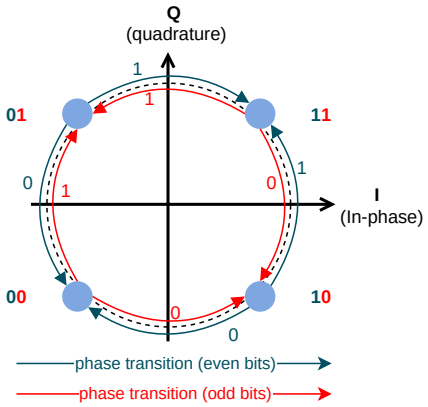

Figure 3.6: I/Q representation of *O-QPSK* modulation with half sine pulse shaping

f_c (in MHz) and the channel number k (from 11 to 26):

$$f_c = 2405 + 5(k - 11) \tag{3.6}$$

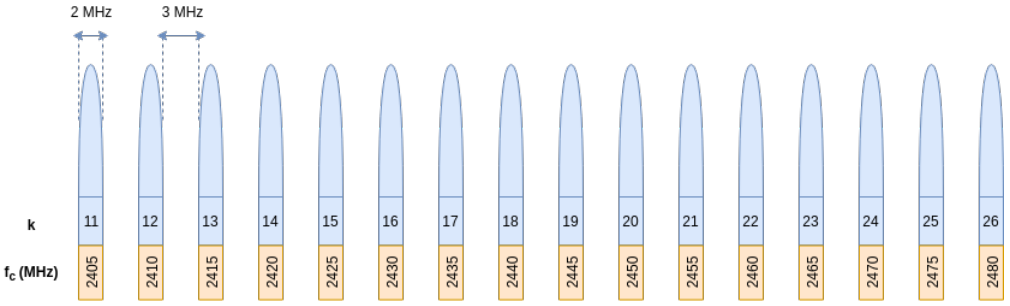

Figure 3.7: 802.15.4 communication channels

3.3 The WazaBee attack

This section describes the *WazaBee* attack and its architecture, which aims to divert the use of the radio device embedded in the *BLE* chip in order to send and receive *802.15.4* frames (in particular *Zigbee* frames). We first describe the attack principle and its theoretical foundations, then we detail the various requirements related to the legitimate operation of the chip that must be taken into account for the attack to be successful and we provide some solutions to fulfil these requirements.

3.3.1 Assumptions

We consider that the attacker has already compromised a *BLE* chip and is able to run arbitrary code on it. This chip compromise may be performed using various techniques, such as network attacks (e.g., attack of an *Over The Air* update process

[Bettayeb 2019]), exploitation of vulnerabilities inherent to the object itself and its firmware allowing some remote code execution [Armis 2017, Armis 2018], or physical attacks allowing to flash the device [Vishwakarma 2018]. This compromise is considered as a prerequisite to the *WazaBee* attack, and is out of the scope of this work.

3.3.2 Attack overview

The *Wazabee* attack relies on the existence of a close relationship between *GFSK* and *O-QPSK*, the modulations used by *BLE* and *Zigbee* protocols. The following subsections explain how to switch from one modulation to another.

3.3.2.1 From GFSK to MSK modulation

As explained in previous sections, *BLE* uses a *Gaussian Frequency Shift Keying* modulation with a modulation index m between 0.45 and 0.55. This characteristic allows us to assimilate the *BLE* modulation to a specific case of *GFSK*, called *GMSK* (*Gaussian Minimum Shift Keying*) with a modulation index $m = \frac{1}{2}$. The signal generated by a *GMSK* modulation has a constant amplitude and a phase evolving continuously over time. Moreover, a *GMSK* modulation is a *MSK* modulation (*Minimum Shift Keying*) whose modulating signal is shaped by a Gaussian filter. If we neglect the effect of the Gaussian filter, *BLE* modulation can be assimilated to *MSK* modulation, which changes linearly and continuously the phase of $-\frac{\pi}{2}$ when modulating a 0-bit and of $+\frac{\pi}{2}$ when modulating a 1-bit.

3.3.2.2 From MSK to O-QPSK modulation

As explained in section 3.2.2, an *O-QPSK* modulation with half sine pulse shaping shares with the *MSK* modulation the property of a constant amplitude and a continuous phase. Moreover, the modulation of each bit generates a $\pm\frac{\pi}{2}$ continuous and linear phase transition. Both *MSK* and *O-QPSK* modulations are thus very close. In a more formal way, the research work of [Pasupathy 1979] shows the theoretical equivalence between *MSK* and *O-QPSK* with half sine pulse shaping, if an encoding strategy is purposely chosen, such as $T_{s(MSK)} = T_{b(OQPSK)}$.

3.3.2.3 From BLE to Zigbee

If we neglect the effect of the Gaussian filter (which will result in more progressive phase transitions), we can make the hypothesis that *BLE* modulation can be approximated by a *MSK* modulation, which is close to the *O-QPSK* modulation used by the *Zigbee* devices. To sum up, we can make the following hypotheses:

- It should be possible to control the input message of a *GFSK* modulator compatible with the *BLE* specification to generate a modulated signal corresponding to a binary sequence that can be interpreted by a *O-QPSK* demodulator (with half sine pulse shaping) compatible with the *802.15.4* standard.

- An arbitrary message modulated by an *O-QPSK* modulator (with half sine pulse shaping) compatible with the *802.15.4* standard should generate a modulated signal corresponding to a binary sequence interpretable by a *GFSK* demodulator compatible with *BLE* specification.

In the following, we outline how these hypotheses can be verified.

3.3.3 Correspondence table generation

The first problem to be addressed consists in establishing a correspondence table between the PN sequences used by the *802.15.4* standard (which results from an interpretation of the signal as a phase modulation, the *O-QPSK* with half sine pulse shaping) and their interpretation by a *MSK* frequency modulation. From this correspondence table, it will then be possible to build a binary sequence to be provided as input to a *BLE* compliant modulator to generate a modulated signal close to the one expected by a *802.15.4* demodulator, but also possible to interpret an *802.15.4* frame as a frequency modulated signal that can be demodulated by a *BLE* demodulator.

The generation of *MSK* sequences consists in encoding each phase transition of the *O-QPSK* modulation with a 1-bit if it corresponds to a counter-clockwise rotation of the vector representing the signal in the complex plan ($+\frac{\pi}{2}$ increase of the instantaneous phase) or with a 0-bit if it corresponds to a clockwise rotation ($-\frac{\pi}{2}$ decrease in the instantaneous phase).

Algorithm 1 illustrates this encoding technique. By applying this algorithm to the 16 PN sequences, it is possible to build the correspondence table 3.2.

Let us note that a sequence of length n encoded in *O-QPSK* has an equivalent of length $n-1$ encoded in *MSK*, because this one represents the transitions between phases.

3.3.4 Requirements

The practical implementation of such an attack requires to take into account a number of requirements, related to *BLE* physical layer characteristics described in previous sections. Our objective is to implement primitives to send and receive *802.15.4* frames on a chip supporting *BLE* 5.0 specification. For that purpose, we have to control the following elements:

- **Data rate:** the duration of one symbol encoded in the *MSK* modulation must be identical to the duration of one bit encoded by the *O-QPSK* modulation, i.e., $T_{s(MSK)} = T_{b(OQPSK)}$. It is thus necessary to configure the modulator and the demodulator used by the chip in order to use a 2 Mbits/s data rate, the same data rate as the 2Mchips/s of the *802.15.4* standard,

- **Central frequency:** *BLE* used channel central frequency must match the frequency of the *Zigbee* channel,

```
   Output: mskSequence[31]
   Input: oqpskSequence[32];
 1 evenStates[4] ← {1, 0, 0, 1};
 2 oddStates[4] ← {1, 1, 0, 0};
 3 currentState ← 0;
 4 i ← 1;
 5 while i < 32 do
 6 |   if i is odd then
 7 |   |   if oqpskSequence[i] = oddStates[(currentState + 1) mod 4] then
 8 |   |   |   currentState ← (currentState + 1) mod 4;
 9 |   |   |   mskSequence[i − 1] ← 1;
10 |   |   else
11 |   |   |   currentState ← (currentState − 1) mod 4;
12 |   |   |   mskSequence[i − 1] ← 0;
13 |   |   end
14 |   else
15 |   |   if oqpskSequence[i] = evenStates[(currentState + 1) mod 4] then
16 |   |   |   currentState ← (currentState + 1) mod 4;
17 |   |   |   mskSequence[i − 1] ← 1;
18 |   |   else
19 |   |   |   currentState ← (currentState − 1) mod 4;
20 |   |   |   mskSequence[i − 1] ← 0;
21 |   |   end
22 |   end
23 |   i ← i + 1;
24 end
```

Algorithm 1: Algorithm of a PN sequence conversion

Block $b_0b_1b_2b_3$	PN Sequence - MSK encoding (m_0m_1 ... $m_{29}m_{30}$)
0000	11000000111011110101110011011 00
1000	10011100000011101111010111001 10
0100	01011001110000001110111101011 10
1100	01001101100111000000111011110 10
0010	11011100110110011100000011101 11
1010	01110101110011011001110000001 11
0110	11101111010111001101100111000 00
1110	00001110111101011100110110011 10
0001	00111111000100001010001100100 11
1001	01100011111100010000101000110 01
0101	10100110001111110001000010100 01
1101	10110010011000111111000100001 01
0011	00100011001001100011111110001 00
1011	10001010001100100110001111111 00
0111	00010000101000110010011000111 11
1111	11110001000010100011001001100 01

Table 3.2: Correspondence table of PN sequences

- **Modulator input:** to implement an emission primitive, it is necessary to control (directly or indirectly) the data sent to the modulator of the chip, in order to be able to provide the PN sequences encoded in *MSK*,

- **Demodulator output:** to implement a reception primitive, it is necessary to detect the reception of a *802.15.4* frame and to retrieve (directly or indirectly) the data output from the demodulator of the chip.

Controlling the data rate is quite easy since the introduction in version 5.0 of a new *LE 2M* mode for *BLE* physical layer, which allows to use a data rate of 2Mbits/s, which perfectly corresponds to our needs. Therefore, it should be possible to satisfy this first requirement on any chip implementing version 5.0 of the Bluetooth specification.

The second requirement is to control the *BLE* central frequency according to the *Zigbee* channel targeted by the attack. Several solutions can be implemented to solve this problem according to the possibilities offered by the chip and the available API. Indeed, most of the chips supporting *BLE* version 5.0 allow to arbitrarily choose a frequency in the 2.4 to 2.5 GHz band, in this case, it is possible to directly select the central frequency of the targeted *Zigbee* channel. If the chip does not allow such a functionality, it is then possible to select a *BLE* channel whose central frequency corresponds to a *Zigbee* channel: only a subset of the *Zigbee* channels will then be available, those which overlap channels defined in the Bluetooth specification.

These channels are indicated in Table 3.3. Such diversion of the use of *BLE* channels is made possible because both *Zigbee* and *BLE* channels share the same characteristics (2MHz bandwidth) and because the *LE 2M* mode allows the use of data channels as secondary advertising channels, thus allowing a direct transmission or reception on the channel via the advertising mode (the connected mode indeed implements a channel hopping algorithm that complicates a lot the implementation of this attack and requires the cooperation of another device).

Zigbee Channels	*BLE* Channels	central frequency (f_c)
12	3	2410 MHz
14	8	2420 MHz
16	12	2430 MHz
18	17	2440 MHz
20	22	2450 MHz
22	27	2460 MHz
24	32	2470 MHz
26	39	2480 MHz

Table 3.3: Zigbee and BLE common channels

The third requirement is to be able to control the data provided as an input to the chip modulator: an arbitrary succession of PN sequences (encoded in *MSK*) must be provided in order to implement a transmission primitive. The main difficulty is related to the *whitening* process, which applies a transformation algorithm on the data to be transmitted, thus modifying the frame before its modulation. This functionality can be disabled on some chips, thus allowing a direct control on the bits transmitted to the modulator. However, even in the absence of this possibility, the *whitening* algorithm is reversible because it is based on a simple linear feedback shift register: it is thus possible to build a sequence of bits which, once the transformation has been applied, corresponds to the PN sequences, by first

applying the de-whitening algorithm on the sequences that must be transmitted. In these two cases, the PN sequences to be transmitted to generate the expected *802.15.4* frame can be encapsulated in the payload of a *BLE* packet, for instance in the advertising data (the *LE 2M* mode allows the transmission of large advertising packets with a payload of up to 255 bytes).

The fourth requirement, which is crucial to build a reception primitive, is to detect *802.15.4* frames and to decode these frames to retrieve the symbols corresponding to PN sequences. For that purpose, the *Access Address* of the *BLE* chip must be configured: this Access Address is used as a pattern to detect a legitimate *BLE* frame. The *Access Address* value can be set with the PN sequence (encoded in *MSK*) corresponding to the 0000 symbol, in order to detect the preamble of a *802.15.4* frame (this preamble is composed of 4 null-bytes, i.e., eight 0000 symbols). The integrity check must be deactivated, because the *802.15.4* frames are not valid *BLE* frames (the chip must allow this deactivation so that a reception primitive can be implemented) and to configure the size of the frame to the maximum available size. At this stage, the *dewhitening* problem has to be solved: it must be ideally disabled, and if this is not possible, a *whitening* algorithm must be applied to the frame in order to extract the output bits of the demodulator. The conversion to the original *Zigbee* symbols can be done very simply by using Hamming distance. Each received packet is split into 31-bits blocks and for each block, a Hamming distance is calculated in order to find which PN sequence encoded in *MSK* fits the best the received block. The use of the Hamming distance allows here to cope with two difficulties: bit errors caused by the approximation presented previously, but also interference due to the channel, that may generate bitflips during transmission.

Figure 3.8 illustrates the WazaBee architecture allowing to fit this requirements:

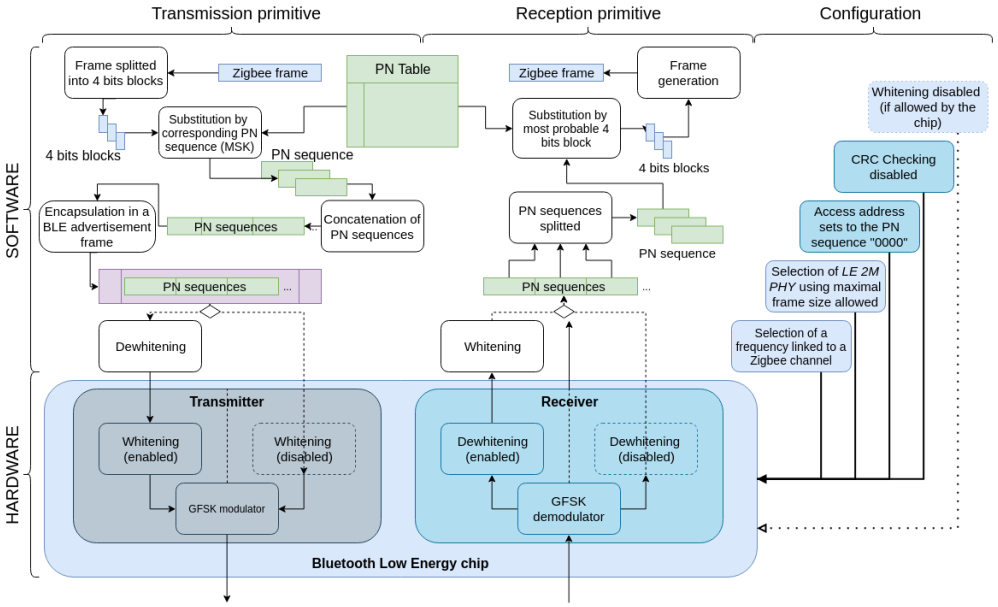

Figure 3.8: WazaBee architecture

Note that the equivalence of *O-QPSK* modulation with half sine pulse shaping and *MSK* modulation should in theory enable a "symmetric" pivoting attack, i.e, to also divert the use of *Zigbee* chips to attack the *BLE* protocol. However, this strategy is quite difficult to implement, because *Zigbee* protocol stack implementation prevents us from finely controlling the *802.15.4* modulator input or demodulator output, mainly due to the *Direct Sequence Spread Spectrum* functionality, which performs the operation of transforming symbols into chip sequences. It would be necessary to be able to control the input of the modulator and the output of the demodulator, which does not seem to be easily achievable with existing devices.

3.4 Benchmarks

Channels	Reception primitive				Transmission primitive			
	nRF52832		CC1352-R1		nRF52832		CC1352-R1	
	valid	corrupted	valid	corrupted	valid	corrupted	valid	corrupted
11	100	0	100	0	98	0	100	0
12	100	0	100	0	100	0	100	0
13	100	0	100	0	95	1	100	0
14	100	0	100	0	97	3	100	0
15	99	1	100	0	100	0	100	0
16	100	0	97	0	90	3	100	0
17	98	1	99	0	94	3	96	0
18	95	2	100	0	91	2	95	0
19	100	0	100	0	97	0	100	0
20	100	0	100	0	100	0	100	0
21	98	2	100	0	100	0	100	0
22	95	2	98	0	100	0	100	0
23	97	0	96	0	100	0	100	0
24	99	1	100	0	100	0	100	0
25	100	0	100	0	100	0	100	0
26	97	2	100	0	98	1	100	0

Table 3.4: Reception and transmission primitives assessment results

It is important to validate the *WazaBee* attack on chips from different manufacturers. We have chosen two different chips, *nRF52832* designed by *Nordic SemiConductors* and *CC1352-R1* designed by *Texas instruments*. Let us note that the attack does not depend on the chips we used, as it only exploits similarities between the physical layers used by the protocols themselves.

Additionally, we are aware that the *TI CC1352-R1* chip natively supports 802.15.4-based protocols, however, of course, we only used its BLE capabilities during our experiments. In this section, we describe the proof of concept implementations, and present the experiments carried out to evaluate the quality of the *Zigbee* communications achieved with *WazaBee*.

The first implementation was carried out on the *nRF52832* chip, which chip offers great flexibility in the configuration of the embedded radio component *BLE 5.0*, and is compliant with the *LE 2M PHY* layer. Its radio *API* is similar to the *nRF51* one. This *nRF51 API* is well known to the security community for having been massively hijacked in recent years in order to develop offensive tools

dedicated to *BLE* and *Enhanced ShockBurst* (*BTLEJack* [Cauquil 2018], *radiobit* [Cauquil 2017a], ...). The prototype was implemented on a development board proposed by *AdaFruit* integrating this chip, the *Adafruit Feather nRF52 Bluefruit LE*. The second implementation was carried out on the *CC1352-R1* chip manufactured by *Texas Instruments*. The main motivation was to test the approach on a chip offering less configuration possibilities than the *nRF52* chip. The chip natively supports several protocols, including *BLE* and *802.15.4*. Obviously, only the Bluetooth API was used for the implementation. This API being common to several chips from *Texas Instruments*, the implementation of the attack should be similar on other systems from the same manufacturer.

The two custom firmwares implemented in the context of these benchmarks are respectively available at:
`https://github.com/RCayre/wazabee_nrf52` and `https://github.com/RCayre/wazabee_ti`.
A simple Command Line Interface has also been developed and can be found here:
`https://github.com/RCayre/wazabee_cli`.

Two experiments were carried out in order to assess the reception and transmission primitives previously described. The first experiment, dealing with reception, consisted in transmitting one hundred *802.15.4* frames with a payload including a counter (incremented with each frame) using a *Zigbee* transmitter (AVR RZUSB-Stick Atmel). The development board implementing the *WazaBee* attack, spaced from the transmitter by a distance of 3 meters, received and decoded the corresponding frames, then calculated the FCS corresponding to the received frame to assess its integrity. For each *Zigbee* channel, the frames were classified into three categories: not received, received with integrity corruption, received without integrity corruption. The results are shown in table 3.4.

It can be seen that the reception primitive of *WazaBee* has a very satisfactory reception rate for the two implementations on all channels, with an average of 98.63 % of the frames received without integrity corruption for *nRF52832* and 99.38 % for *CC1352-R1*. In both cases, there is a slight decrease in the reception rate for channels 17, 18, 21, 22 and 23, which can be explained by the interference with WiFi channels 6 and 11, used in our experimental environment. It can also be observed that the *CC1352-R1* presents a more stable reception than the *nRF52832*, without any integrity corruption of the received frames while the *nRF52832* missed 0.69 % of the frames.

The transmission primitive was assessed under similar conditions: the development board implementing *WazaBee* was configured to transmit one hundred frames including a counter, and a *802.15.4* receiver (the RZUSBStick) was placed 3 meters away. Each transmitted frame could also be classified into three categories: not received, received with integrity corruption and received without integrity corruption. The experiment was performed on all *Zigbee* channels, and the corresponding

results are shown in table 3.4.

In both cases and for all channels, the rate of valid frames received without integrity corruption by the RZUSBStick is very satisfactory, with an average of 97.5% for *nRF52832* and 99.44 % for *CC1352-R1*. We observe a similar phenomenon to the one observed during the assessment of the reception primitive for channels 17 and 18, related to the simultaneous use of WiFi channel number 6 in our experimental environment. The rate of corrupted frames received is also slightly higher for *nRF82832* (with an average of 0.81 % while the *CC1352-R1* did not miss any frame).

3.5 Attack scenarios

In this section, we demonstrate the *WazaBee* attack by describing two attack scenarios we actually carried out. Two main attack scenarios, considering various devices, have been implemented. The first scenario illustrates the implementation of a subset of the *WazaBee* primitives on an unrooted Android phone, using an high level API. The second scenario presents the implementation of *WazaBee* on a commercial *BLE* tracker device in order to perform complex *Zigbee* attacks. We purposely chose these devices in order to illustrate the critical impact of the *WazaBee* attack. Indeed, Android phones and *BLE* trackers are very common devices, that anyone may possess. The successful implementation of *WazaBee* on these devices shows that this attack may actually be deployed easily and massively.

3.5.1 Experimental setup

A main experimental setup is used for the two attack scenarios, based on a simple domotic *Zigbee* network with the *PANID 0x1234*, illustrated in figure 3.9.

This network is composed of two *XBee* (a commercial implementation of *ZigBee*) transceivers. The first one (*16-bits address 0x0063*) is an end device simulating a sensor transmitting an integer (e.g., the temperature) every two seconds while the second one (*16-bits address 0x0042*) is a coordinator which acknowledges the data and displays it on a HTML graph. The channel 14, which matches the 2420 MHz frequency, is used.

3.5.2 Scenario A: injecting 802.15.4 frames using a smartphone

The first attack scenario was the injection of arbitrary *802.15.4* frames into our network, using an unrooted Android smartphone. For instance, an attacker could use a malware installed on an employee's phone to launch such an attack remotely, allowing him to perform multiple active attacks targeting *Zigbee* networks. It could also allow to exfiltrate discreetly sensitive data using a protocol that is not monitored.

As mentioned earlier, implementing the two primitives of the *WazaBee* attack requires the attacker to gain control over the lowest layers of the *BLE* protocol stack. However, the aim of the experiment is to test if an attacker that can only

Figure 3.9: Targeted Zigbee network

interact via an high level API could be able to implement at least a subset of the attack. As a consequence, this scenario was evaluated with the following contraints: 1) the smartphone is unrooted; 2) the attacker has only access to standard Android API with common permissions, and 3) the attack should be compliant with any *BLE* 5-compliant device, without the need to divert specific hardware components (e.g., InternalBlue [Mantz 2019]).

According to the specification, the received frames including a wrong CRC are dropped at the controller level and are not delivered to the host. Therefore, the received *802.15.4* frames are considered as invalid *BLE* frames and are filtered in the controller and not forwarded to the host. As a consequence, the implementation of the reception primitive is not possible without a low-level access allowing to collect invalid frames. The implementation of the transmission primitive is also tricky, because we only have an indirect control over the frequency and the payload content using a high level API. However, the extended advertising feature allows a partial implementation of the transmission primitive. Indeed, this feature has some interesting properties: it allows the transfer of large amount of data, it can use the 37 data channels without the need to initiate a *BLE* connection, it can use the *LE 2M* physical layer and it is based on predictable frame formats.

If the device uses *LE 1M* as primary physical layer and *LE 2M* as secondary physical layer, it initially transmits *ADV_EXT_IND* advertisements at 1 MBits/s on the primary advertising channels (37,38 and 39), indicating on which secondary advertising channel and the offset to the start time the extended advertisement will be transmitted. The channel selection is based on a pseudo-random algorithm named *Channel Selection Algorithm #2* [Blu 2019], and is not directly controllable by the user. Then, the advertiser transmits the extended advertisement embedding the data provided by the user (*AUX_ADV_IND*) at 2 Mbits/s on the selected channel.

Diverting this feature in order to transmit *802.15.4* frames can be achieved using the strategy mentioned above to forge the advertising data. We first need to choose the PN sequences (encoded in *MSK*) corresponding to the frame to transmit. Then, we need to add some padding bytes before the frame (because of the

multiple headers included before the data) and apply the dewhitening function to the resulting data. As this operation depends on the channel, it allows to select a specific *Zigbee* channel: in our case, we want to transmit data at 2420 MHz (Zigbee channel 14), which corresponds to *BLE* channel 8, so we perform the dewhitening operation using this *BLE* channel as input. The output is then cropped to remove the padding bytes, then the result is provided as advertising data. We use a manufacturer data field to encapsulate our forged frame, resulting in a padding size of 16 bytes. Then, the extended advertising can be enabled using the smallest time interval in order to increase the probability that the channel selection algorithm picks our target channel.

We implemented this approach in an android application running on an unrooted OnePlus 6T smartphone, that fully supports the extended advertising feature. We were able to inject forged data packets to our target zigbee network, as illustrated in figure 3.10.

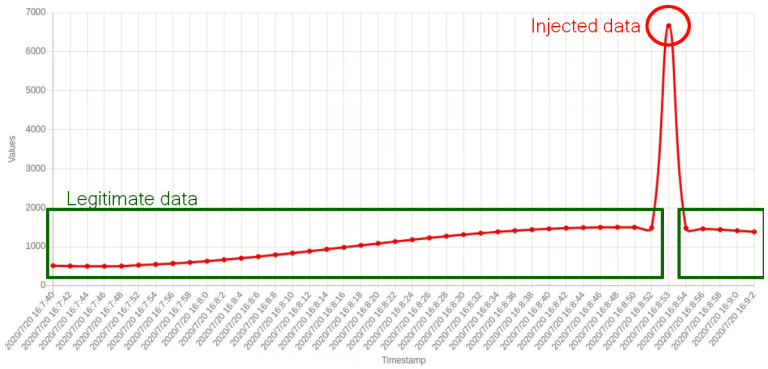

Figure 3.10: Forged data packets injection from a OnePlus 6T smartphone

This approach is entirely compliant with the specification and only uses an high level API, meaning every *BLE* 5 device is able to inject *802.15.4* frames into at least eleven channels (especially those which have common frequencies with *BLE* data channels) in the 2.4-2.5GHz ISM band. As a result, it increases the attack surface of *802.15.4*-based protocols.

As we have chosen to implement the attack on a smartphone with limited permissions, it was not possible to implement the reception primitive. However, let us note that attackers with higher privileges may be able to gain a low level access and easily implement the two primitives. For example, *InternalBlue* [Mantz 2019] allows to patch firmwares of *Broadcom* and *Cypress* controllers, which are common in off-the-shelf devices. If the attackers are able to reverse engineer the target firmware to identify the functions allowing to match the requirements mentioned in 3.3.4, they can easily write malicious patches and add custom code to the firmware implementing *WazaBee* primitives. We demonstrate such an implementation in section 3.6.

3.5.3 Scenario B: performing complex Zigbee attacks from a BLE tracker device

The second attack scenario illustrates the possibility to perform complex *Zigbee* attacks by abusing a *BLE* smart object. The impact of such an attack could be significant, as it may allow an attacker to build complex attacks involving legitimate *BLE* devices, that will not be identified as a potential threat to *802.15.4* networks. For example, an employee's mobile device (e.g., a smart watch, a tracker) could be infected outside the company in order to carry out a complex attack when the device is within range of the company's *Zigbee* network.

Our attack was performed on a commercial *BLE* tracker device, called Gablys Lite, which is based on a *nRF51822* chip. It requires a physical access to the device, as we used some unprotected debug pins providing a *Serial Wire Debug* (*SWD*) to flash a new firmware. Note that a similar attack could be performed using *BLE* vulnerabilities such as OTA updates abuse, which do not require this physical access.

The *nRF51822* chip is similar to *nRF52832*, but it doesn't support *LE 2M*, which is a key requirement of *WazaBee* attack. However, as the *Enhanced ShockBurst* protocol at 2 Mbits/s is supported by the chip, it can be used as an alternative for *LE 2M* physical layer. This solution has a direct impact on the reception quality, but it is sufficient to successfully conduct a complex active attack.

The main goal of this attack is to perform a denial of service targeting the sensor, in order to spoof it and inject fake data into the display interface. We used an existing attack targeting *XBee* nodes in order to perform a denial of service, allowing to inject a new configuration through remote AT commands [Vaccari 2017]. The attack is divided into four main steps:

- **Active scanning:** the device transmits a *Beacon Request* on a channel and waits for a *Beacon* from the coordinator. If no *Beacon* is received before the timeout expires, the device selects the next channel. If a *Beacon* is received, the channel, the PanID and the coordinator's address are collected and saved,
- **Eavesdropping:** the device sniffs the legitimate frames in order to collect the sensor's address,
- **Remote AT command injection:** the device forges a remote AT command, using coordinator's address as source and sensor's address as destination. It allows to force the sensor to use another channel,
- **Fake data injection:** the device transmits fake data frames, mimicking the sensor's behaviour.

This attack was implemented successfully, as illustrated in figure 3.11. This experiment shows that WazaBee's primitives can be combined to conduct complex attack scenarios, and also that a legitimate commercial device can be modified and used to perform this kind of offensive strategies.

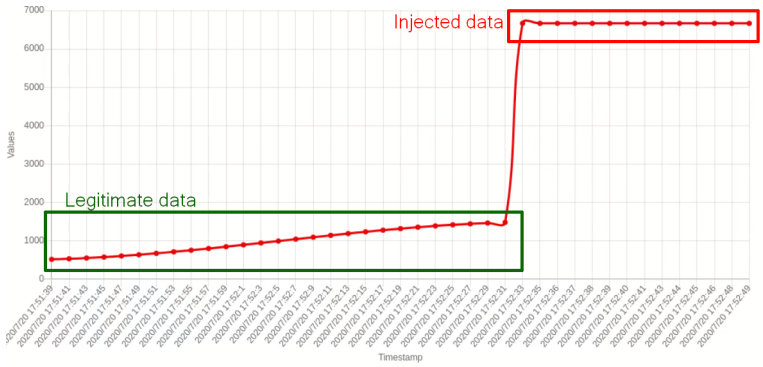

Figure 3.11: Complex attack workflow from a BLE tracker

3.5.4 Conclusion

The two attack scenarios illustrated in the previous subsections are not exhaustive, but they illustrate the critical impact of *WazaBee* attack, and especially the considerable number of legitimate devices that may be diverted in order to attack *802.15.4* networks. Depending on the corrupted device and the privilege level gained, an attacker may be able to implement the two primitives or only a subset of *WazaBee* attack. However, scenario A underlines the fact that even with a partial implementation, an attacker would be able to achieve interesting objectives, such as leaking sensitive data or disrupting legitimate nodes. These offensive strategies could also be combined in order to perform complex attack scenarios. Finally, these two scenarios also underline that the attack is easy to implement on various devices and may be used easily in the wild.

3.6 RadioSploit: implementing pivoting attacks on a recent smartphone

In this section, we explore the feasibility of diverting a *BCM4375* Bluetooth controller embedded in a Samsung Galaxy S20 smartphone to implement WazaBee, resulting in a wide attack surface. We also successfully implemented other cross-technology pivoting attacks targeting Enhanced ShockBurst and Mosart, two proprietary protocols commonly used by wireless keyboards.

3.6.1 Firmware reverse engineering and patching

3.6.1.1 InternalBlue framework

The *InternalBlue* framework [Mantz 2019] takes advantage of some vendor-specific commands and allows to easily dump, analyse and patch firmware embedded in Bluetooth controllers from Broadcom and Cypress, which are common in the wild. First, it allowed us to dynamically instrument the firmware to understand its internals.

Second, we used it to patch some specific functions to integrate our customized receive and send primitives and to add support for new protocols. Note that the use of this framework requires root access on the smartphone as it needs to send arbitrary commands to the Bluetooth controller using the Host-Controller Interface. We also had to replace one of the patched official Broadcom file by an older one, as some of these patches removed support of some specific commands used by *InternalBlue* [Classen 2021].

3.6.1.2 Methodology

We focused our analysis on a recent Bluetooth controller, the *BCM4375* chip from *Broadcom*. This chip is embedded in many smartphones, such as Samsung Galaxy S10 or S20. We have chosen this specific chip for the following reasons: first, it supports Bluetooth 5 and especially the LE 2M physical layer, which is needed to implement Zigbee and Enhanced ShockBurst support. It is also compatible with *InternalBlue*, which considerably facilitates the process of firmware reverse engineering and patching. We also analysed the firmware of the CYW20735 IoT develoment board, illustrated in figure 3.12. Indeed, the symbols associated to this firmware are already known, allowing to easily identify the main functions and to understand their behaviour.

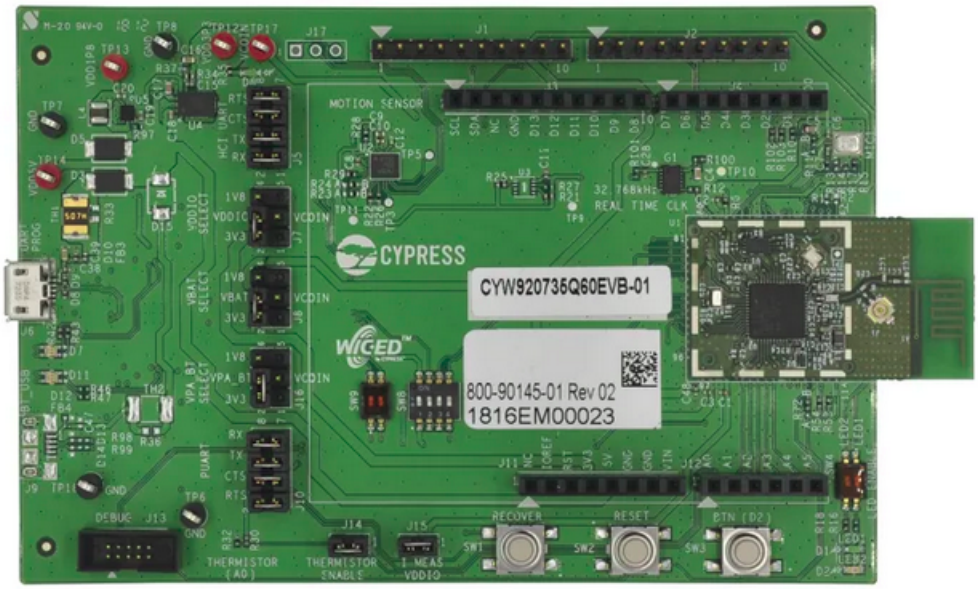

Figure 3.12: CYW20735 IoT development board

We have partially reverse engineered these firmwares, especially the functions linked to the RF hardware configuration and to BLE scanning and advertising tasks. This process was conducted using both static and dynamic analysis with *IDA Pro* and *InternalBlue*. We also identified several common functions that are present

in both firmwares: this allowed us to take advantage of the known symbols of the CYW20735 firmware to infer relevant information about the BCM4375 firmware.

3.6.1.3 Diverting scanning and advertising tasks

We have focused our work on the Bluetooth Low Energy stack, and more particularly on the features related to advertisements, such as scanning or advertising. Indeed, these features do not require the establishment of a connection as a prerequisite to send and receive packets. Therefore, they are good candidates to be diverted, in order to implement receive and send primitives for other wireless protocols. They are implemented in the firmware as tasks, consisting of several functions and callbacks.

First, we identified the main functions linked to the configuration of the RF hardware, the reception callback (`extendedScanTaskRxDone`) used by the scanning task and the transmission function (`extendedAdvTaskProgHw`) used by the advertising task. Second, we modified some specific instructions in these functions to redirect the execution flow to our own code stored in RAM, allowing us to 1) configure the RF hardware; 2) gain direct access to the raw demodulator output thanks to a memory mapped register and 3) gain indirect access to the modulator input by storing our complete packet into an advertisement packet payload, using Packet-in-Packet attack [Goodspeed 2011b].

3.6.1.4 RF hardware configuration

In order to implement our reception and transmission primitives to support other wireless protocols, we must be able to perform the following operations: a) choose an arbitrary preamble, b) choose an arbitrary frequency, c) select a 2Mbps data rate, d) receive data from the demodulator output, e) send data to the modulator input.

We mainly identified the configuration function linked to the setup of *LE 2M* physical layer (`le2m_program2MAdvMode`). The BLE *access address* being used as a pattern to match the beginning of a BLE packet, we used it to select an arbitrary preamble to synchronize with packets from other wireless protocols using a 2Mbps data rate. The whitening operation was configured using a specific function (`bcsulp_setupWhitenning`) which has been modified to disable this feature, allowing us to manipulate the demodulator output and the modulator input without requiring additional data processing. Two different registers are used to select the frequency, one being used by the scanning task and the other one by the advertising task. However, both of them allow to select an arbitrary frequency in the 2.4 to 2.5 GHz band by providing an offset from 2402 MHz, specified in MHz (as an example, selecting 2410 MHz implies to write a 8MHz offset to one of these registers).

We were able to implement both a reception and a transmission primitive by diverting these features, allowing us to handle arbitrary GFSK packets using a 2Mbps data rate. We then used these primitives to add support for several non-native protocols, such as Zigbee, Mosart and Enhanced ShockBurst.

3.6.1.5 Host/Controller communication

We introduced these new offensive capabilities directly in the Bluetooth Controller by patching its firmware with *InternalBlue*. However, they have to be handled from the smartphone, also known as Host. Therefore, we had to find a way to establish a communication between the Controller and the Host to build a relevant API.

The Bluetooth specification describes an interface named *Host Controller Interface* (HCI), allowing Host to Controller communication using commands and Controller to Host communication using events. We identified two functions allowing to allocate a buffer describing an event message (`bthci_event_AllocateEventAndFillHeader`) and send it to the Host (`bthci_event_AttemptToEnqueueEventToTransport`) : we mostly used them to send the received packets to the smartphone. We also discovered that HCI commands are handled using an array of function pointers: the command identifier is used to calculate an index, allowing to call the corresponding function into the firmware. We found two unused command identifiers and stored our own functions' addresses at the right places in this array, allowing us to expose a simple API that can be used to control the receiver mode and transmit a given packet.

These modifications allowed us to implement a user-friendly API, which can then be easily manipulated from the smartphone using the HCI. We implemented an experimental Android application, illustrated in figure 3.13 allowing to interact with the controller to trigger the new offensive capabilities we added: we monitor HCI events by parsing in real time the Bluetooth HCI snoop log and we can also send commands to the controller by writing the raw command message directly to /dev/ttySAC1.

Both the patches and the application are released as open source softwares, available at:
`https://github.com/RCayre/radiosploit` (for the Android application)
and at:
`https://github.com/RCayre/radiosploit_patches` (for the controller patches).

3.6.2 Protocols support

3.6.2.1 Implementation

We implemented Zigbee protocol support using WazaBee attack. We added a correspondence table in the firmware, allowing to map each Zigbee symbol to the corresponding GFSK demodulated binary sequence. We also added helper functions allowing to automatically perform this conversion when a Zigbee packet is received or sent. Every Zigbee packet starts with a 4 bytes-long preamble which is composed of zeros: as a consequence, we generated the GFSK bytes sequence cor-

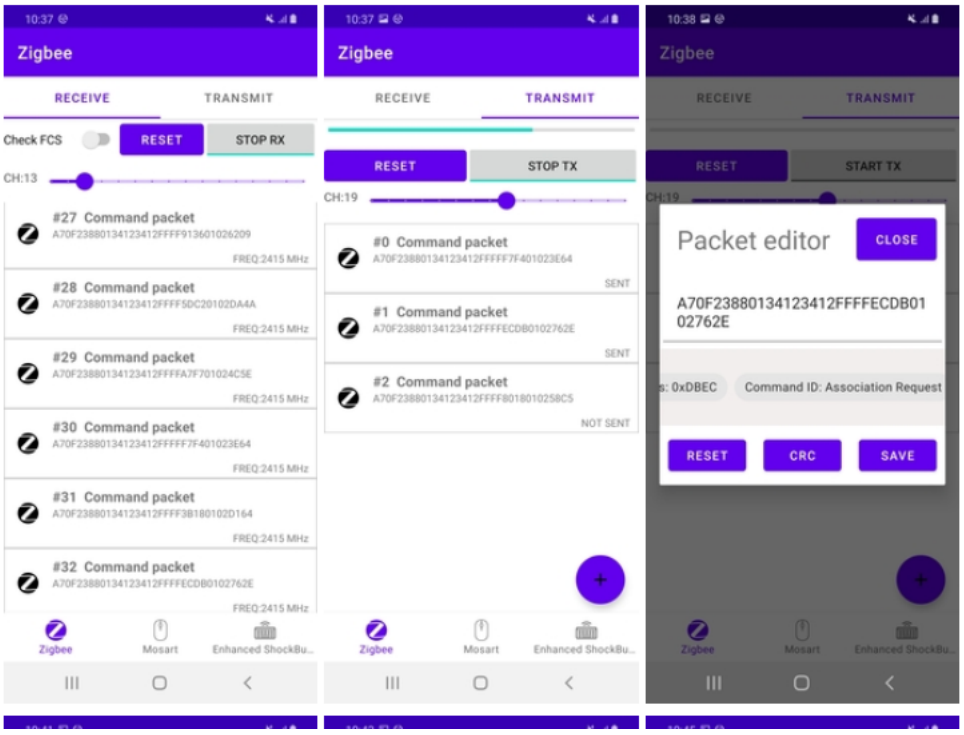

Figure 3.13: RadioSploit interface

responding to the zero symbol and used it as preamble to synchronise the receiver with Zigbee frames. Selecting a specific Zigbee channel is straightforward, as the offset we have to write in the frequency selection register depends directly on the Zigbee channel number.

We also implemented support for proprietary protocols used by wireless keyboards, such as Mosart or Enhanced ShockBurst. Mosart is a proprietary protocol commonly used by wireless mice and keyboards from various manufacturers. It is based on a GFSK modulation scheme using a 1 Mbps datarate. A Mosart packet consists of a 2-byte preamble (0x5555), a 4-byte address, a variable length payload and a 2-byte CRC. One of the major issues we encountered in implementing this protocol is related to the RF hardware of BCM4375 chip: even though BLE natively supports a physical layer using 1 Mbps data rate, the firmware does not expose any function to select an arbitrary access address if 1 Mbps data rate is used. We assume that the access address used in LE 1M advertising mode is hard-coded in the RF hardware and cannot be easily changed from the firmware, which complicates the implementation of the reception primitive.

However, as we mentioned in subsection 3.6.1.4, the access address can be chosen arbitrarily if the *LE 2M* physical layer is used. We solved this problem by using *LE 2M* and duplicating every bit: as an example, the 2-byte preamble 0x5555 at 1 Mbps becomes 0x33333333 with 2 Mbps data rate, as illustrated in figure 3.14. We have implemented helper functions to select one bit over two in the demodulator output

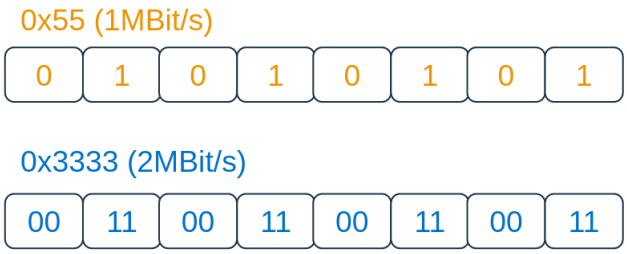

Figure 3.14: Mosart decoding using double bit strategy

to decode a received Mosart packet and to duplicate each bit of the sent packets before their transmission to the modulator input. Other simple transformations are also performed in these functions, allowing to deal with scrambling and endianness.

We also implemented support of *Enhanced ShockBurst* protocol. It is a proprietary protocol using a GFSK modulation at 2Mbps, used by many keyboard or mouse protocols, such as Logitech Unifying. Each packet starts with a preamble of 0xAA or 0x55, followed by a 5-byte address, a payload and a CRC. Since the modulation scheme it uses is identical to the one used in BLE, it is quite easy to implement the primitives described above to communicate using this protocol. Some minor differences, such as the endianness, can be easily solved with a simple transformation applied to the modulator input and the demodulator output. Synchronizing the receiver with Enhanced ShockBurst packets is straightforward if the address is known, as we can use its first bytes as preamble. Without prior knowledge of this address, we first configure our receiver with an arbitrary preamble to get large demodulated buffers, in which we then search for valid packets to extract the embedded addresses.

3.6.2.2 Evaluation

We evaluated our implementation by performing several experiments. We were able to reproduce the attack scenarios mentioned in section 3.5. We first used our primitives to passively monitor traffic to identify the network PanID and nodes' addresses. Then, we injected a fake configuration to perform a denial of service attack targeting a specific sensor and spoofed it by transmitting fake data to the visualizer.

We evaluated the two Mosart primitives by implementing several attacks from *MouseJack* [Newlin 2016a], a set of vulnerabilities targeting wireless keyboards and mice. Indeed, the Mosart protocol does not use encryption, so we were able to implement a wireless keylogger allowing to passively collect keystrokes and inject arbitrary keystrokes to a vulnerable Mosart dongle.

Similarly, we evaluated *Enhanced ShockBurst* primitives by reproducing *MouseJack* vulnerabilities. We could get the address of a Logitech wireless mouse, and then sniff its communications or inject malicious packets to trigger mouse clicks or arbitrary movements. Let us note that M. Newlin identified multiple critical

vulnerabilities in [Newlin 2016a] that could be triggered using these primitives, allowing a fake device to be paired with a dongle without user interaction or to inject unencrypted keystrokes. Most of these issues were supposed to be fixed, but during our experiments, we encountered recent devices which are still vulnerable to some of them.

3.6.3 Conclusion

In this section, we presented a practical implementation of pivoting attacks by diverting a Bluetooth chip embedded in a smartphone to communicate over different wireless protocols, demonstrating the practical feasibility of such attack strategies on a standard mobile phone.We were able to implement *WazaBee* attack we presented earlier in this chapter, but also proprietary protocols such as *Mosart* or *Enhanced ShockBurst*.

We consider that it shows how critical are these attacks from a security perspective, as it does not require any expensive or specific equipment, takes advantage of the ubiquity of Bluetooth devices and is mobile. For example, compromising an employee's smartphone could lead an attacker to pivot on different other protocols used by a company to carry out passive or active attacks, such as injecting keystrokes on a distant computer, or inserting a malicious node in a ZigBee network.

3.7 Counter-measures

WazaBee attack is inherent to the wireless protocols and their modulations, even if some conditions need to be fulfilled to be implemented on some specific chips. As a consequence, we should consider every *BLE* 5 device as potentially vulnerable and the environments exposed to *BLE* devices should be designed and monitored with the assumption that some attacks could potentially be carried out through *802.15.4* networks. Several counter-measures could be investigated either to limit the impact of the attack, or to prevent or detect it.

As explained in section 3.3.4, the practical implementation of *WazaBee* requires controlling some features of the *BLE* chips. Making it difficult or impossible for an attacker to control these features (such as the deactivation of the CRC or the setting of a precise channel frequency), by chip manufacturers, would complicate the task of the attacker, and especially the implementation of the reception primitive. However, such counter-measures should only be considered as a first barrier for an attacker and not as efficient adequate solutions, as illustrated in our scenario A in section 3.5 which only uses an high level API in order to implement the transmission primitive.

Some other common counter-measures, such as cryptographic techniques, that most of the *802.15.4*-based protocols provide, should be systematically used. If these techniques are implemented, even if the *WazaBee* attack is still possible, the task of the attacker would be much more complicated. Unfortunately, the correct implementation of these counter-measures is not trivial and it highly depends on

the protection of the keys. Note that some known attacks [Vidgren 2013a] aiming at breaking the *802.15.4* encryption can be performed using *WazaBee* and also that the attacker can still perform denial of service attacks [Cao 2016].

Finally, some defensive solutions dedicated to the IoT context, to monitor and detect in real time attacks targeting wireless protocols, can also be considered. Indeed, the existence of such offensive strategies motivates the deployment of intrusion and prevention detection systems based on the analysis of radio communications. Such systems could simultaneously monitor multiple wireless protocols (even those which are not deployed in the legitimate environment) such as the solution proposed in [Siby 2017], or could be protocol agnostic, such as the intrusion detection approaches proposed in [Roux 2018, Rajendran 2019]. These intrusion detection systems are designed to monitor the physical layers of communication protocols (by monitoring signal strength on different frequency bands) and are based on the modeling of legitimate communications and therefore detect accidental faults (in [Rajendran 2019]) or malicious activities (in [Roux 2018]) by identifying deviations from legitimate behavior.

More generally, the wireless attacks investigated in this chapter may impact other protocols, depending on the compatibility between their modulations and channel coding, along with the programmability of the underlying hardware. Indeed, if the frequencies overlap, while the modulations are similar enough to be able to control what is received by one protocol from an emission of the other, the two protocols are by design vulnerable to pivoting techniques. In section 3.6, we demonstrated that proprietary protocols such as *Mosart* and *Enhanced ShockBurst* could also be easily targeted by such offensive strategies.

Let us note that evaluating accurately the similarities between two modulations is an open challenge. Therefore, it might be interesting for companies wishing to introduce a new wireless protocol inside their environment to carefully study the possible compatibilities with other protocols while analysing the risks on their infrastructure. It would also be useful for protocol designers to consider such possibilities of cross-protocol interactions when creating new wireless standards, to reduce the risks of pivoting attacks using their protocol as basis.

3.8 Conclusion

In this chapter, we have highlighted a new pivoting attack strategy, called *WazaBee*, allowing the legitimate operation of a chip intended to communicate via *BLE* to be diverted in order to send and receive *Zigbee* communications (actually, our approach is compliant with all 802.15.4 frames). The results obtained during the two evaluations carried out for the transmission and reception primitives proved to be highly stable and reliable. The direct consequence of this attack is a considerable increase in the attack surface, each system communicating via a protocol based on the 802.15.4 standard (*Zigbee*, *6LoWPan* ...) being potentially accessible from a component supporting *BLE*, a particularly widespread technology in

IoT environments. We also performed multiple experiments with different architectures and from different manufacturers demonstrating the practical feasibility of such pivoting attacks, including an implementation on a recent smartphone that has been weaponized to attack both Zigbee (using *WazaBee* attack) and proprietary protocols used by wireless keyboards by diverting its *Bluetooth* controller.

With the rapid expansion of connected objects and the development of multiple wireless communication protocols, the impact of these pivoting attacks on information systems security seems particularly critical. The coexistence of these protocols in the same environments, as well as certain characteristics of connected objects (such as mobility) are aggravating factors and raise questions about the use of this type of offensive strategies in attack scenarios. One can thus consider the use of this type of attack to try to pivot from a compromised system to another more difficult to access, but also in other types of attacks using covert channels, where the attacker could exploit these send and receive primitives to exfiltrate sensitive data via non monitored wireless protocols.

Multiple scientific articles have been published to present this research work, both in national and international conferences:

- Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Geraldine Marconato. **WazaBee : attaque de réseaux Zigbee par détournement de puces Bluetooth Low Energy.** *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2020)*, Jun 2020, Rennes, France. **[FR]** [Cayre 2020]
- Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, et al.. **WazaBee: attacking Zigbee networks by diverting Bluetooth Low Energy chips.** *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2021)*, Jun 2021, Taipei (virtual), Taiwan. **[EN]** [Cayre 2021d]
- Romain Cayre, Florent Galtier. **Attaques inter-protocolaires par détournement du contrôleur Bluetooth d'un téléphone mobile.** *GT Sécurité des Systèmes, Logiciels et Réseaux*, May 2021, En ligne, France. **[FR]** [Cayre 2021a]
- Romain Cayre, Géraldine Marconato, Florent Galtier, Mohamed Kaâniche, Vincent Nicomette, et al.. **POSTER: Cross-protocol attacks: weaponizing a smartphone by diverting its Bluetooth controller.** *14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, Jun 2021, Abu Dhabi, United Arab Emirates. **[EN]** [Cayre 2021e]

Chapter 4

InjectaBLE: injecting malicious traffic into established Bluetooth Low Energy connections

Contents

In this chapter, we focus on the security of Bluetooth Low Energy protocol, which is widely deployed in connected objects, but also smartphones, tablets and computers. More specifically, we analyze the security of the low level mechanisms the protocol relies on, especially those related to Link Layer time management.

Indeed, multiple offensive research works focusing on this protocol have been released in the recent years, as we presented in Section 2.2.1. However, we noted that some attacks that could be relevant from an offensive perspective, such as hijacking the Peripheral role or performing a Man-in-the-Middle attack targeting an established connection, are missing in the literature. In this work, we present a new critical Bluetooth Low Energy attack, named InjectaBLE, allowing to inject arbitrary packets into an established connection. While the injection is critical in itself, it also can be used as a basis to perform complex offensive scenarios that were not possible until now, from Peripheral Hijacking to Man-in-the-Middle targeting an established connection. This attack relies on a race condition that leverages some fundamental clock synchronization features of BLE protocol, making it especially difficult to fix without significantly impacting the specification. We describe this new injection attack, evaluate the impact of several key parameters upon its success and demonstrate its practical feasibility by means of four critical offensive scenarios taking advantage of this new injection strategy.

4.1 Motivations

Several wireless communication protocols have been developed in recent years to implement these services, among them the Bluetooth Low Energy (BLE) protocol. BLE provides a lightweight protocol stack and allows devices to communicate easily and reliably with a minimal energy consumption, which fits perfectly the constraints of connected objects. It is also widely deployed in smartphones, computers and tablets, enabling direct communications without the need for additional gateways in the network. As a result, many IoT devices already rely on BLE to communicate with their environment.

The growing interest for this technology also raises legitimate concerns about the security of BLE. In the recent years, the security of this protocol has been actively studied both from an offensive and a defensive perspective, highlighting serious flaws in its specification [Blu 2019] and in various implementations. Some papers focused on eavesdropping a BLE connection [Ryan 2013a,Cauquil 2017b,Qasim Khan 2019], which is not straightforward because of the use of a channel hopping algorithm, while other papers described active attacks such as jamming [Bräuer 2016], hijacking [Cauquil 2018] or Man-in-the-Middle attacks [Cauquil 2016, Jasek 2016]. However, to our knowledge, all state of the art offensive techniques described so far require the attack to be carried out before the targeted BLE connection is established, or are based on highly invasive techniques such as jamming. Even if some papers mention a theoretical non invasive injection-based attack in an established

connection [Ryan 2013a] or consider it difficult to achieve [Santos 2019], it has never been implemented in practice and the new offensive capabilities provided by this strategy have not been studied.

In this chapter, we demonstrate the practical feasibility of such an attack, which significantly increases the attack surface of the BLE protocol. We present a novel approach named *InjectaBLE* allowing to perform an arbitrary frame injection into an already established BLE connection. We first explain its theoretical foundations, and then present various experiments illustrating its feasibility.

Four critical offensive scenarios that take advantage of this injection attack are investigated: we show that an attacker can use our approach to stealthily trigger a specific feature of a device, hijack any role involved in the targeted connection or perform a Man-in-the-Middle attack during the connection. We demonstrate that most of these scenarios, that were considered unrealistic until now, are in fact quite easy to perform and could have serious consequences on the security of any BLE device compliant to Bluetooth Core Specification irrespective of how it is implemented. We finally discuss the impact of this attack and potential mitigation measures.

4.2 Bluetooth Low Energy

This Section presents a brief overview of the BLE protocol as well as some more detailed descriptions of the Link layer (LL), which are directly related to our injection attack.

4.2.1 Overview

Bluetooth Low Energy is a lightweight variant of Bluetooth, dedicated to devices needing low energy consumption. Its protocol stack is depicted in Figure 4.1.

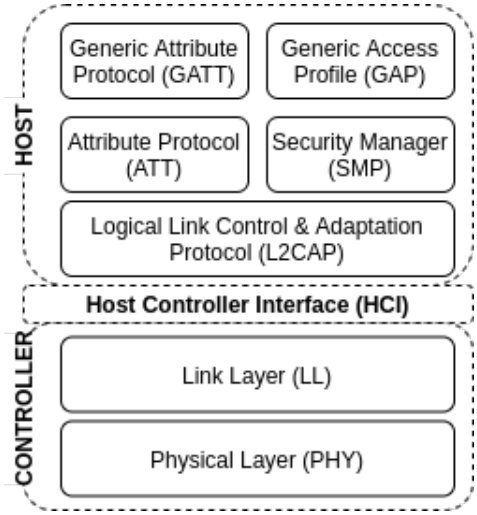

Figure 4.1: Bluetooth Low Energy protocol stack

The stack is split into two major parts: the *Controller* and the *Host*. The lowest layers are included in the *Controller*, while the highest ones are handled by the *Host*.

The physical layer is based on a *Gaussian Frequency Shift Keying* modulation. Three main modes can be used in BLE: an uncoded physical layer with a bitrate of 1 Mbit/s or 2 Mbits/s (respectively called *LE 1M* and *LE 2M*), or a coded physical layer using a 250 kbits/s or 500 kbits/s bitrate (called *LE Coded*). BLE operates in the ISM band from 2.4 to 2.5 GHz, and defines 40 channels, each with a bandwidth of 2 MHz. Three channels (37, 38 and 39) are dedicated to the *advertising mode* (allowing devices to broadcast data using some packets named *advertisements*), while the 37 others channels (numbered from 0 to 36) are dedicated to the *connected mode*, which is used when a connection is established between two devices.

Every BLE-based application using the *connected mode* is built on top of the *ATT* and *GATT* layers. These layers define a client / server model, providing a generic solution to exchange data between devices. An *ATT* server is a database of *attributes*. Each *attribute* is composed of an identifier, a type and a value. An *ATT* client is able to interact with this database using some requests: for example, a *Read Request* allows the client to read a given *attribute*, while a *Write Request* allows to modify the value of an *attribute*. The *GATT* level provides an additional layer of abstraction to define some *services* including *characteristics* and creates generic profiles for a given type of device.

The *Security Manager* provides a set of *pairing* and *bonding* procedures to negotiate multiple keys dedicated to increase the security level of the connection. One of the most important keys is the *Long Term Key*, which allows to establish an AES-CCM encryption over the Link Layer to avoid eavesdropping. The *Generic Access Profile* (*GAP*) introduces four different roles, describing the device's behaviour. Regarding the *connected mode*, two roles are defined. The *Peripheral* role corresponds to a device that can transmit advertisements and is connectable, while the *Central* role corresponds to a device that can receive advertisements and establish a connection with another device. The *Peripheral* is also called *Slave* as it plays a slave role in a BLE connection; the *Central* is called *Master*.

4.2.2 Link layer internals

Our injection-based attack mainly relies on the exploitation of some specific features of the Link Layer. This subsection provides a detailed description of these features.

4.2.2.1 Frame format

Every BLE frame transmitted using the *LE 1M* mode is based on the format described in table 4.1:

Preamble	**Access Address**	**Protocol Data Unit (PDU)**	**CRC**
1 byte	4 bytes	variable	3 bytes

Table 4.1: Frame format for *LE 1M*

Access addr.	CRCInit	WinSize	WinOffset	Interval	Latency	Timeout	Ch. Map	Hop Inc.	SCA
4 bytes	3 bytes	1 byte	2 bytes	2 bytes	2 bytes	2 bytes	5 bytes	5 bits	3 bits

Table 4.2: CONNECT_REQ PDU

The preamble is used by the receiver to detect the start of a BLE frame. The *Access Address* indicates the mode in use, either *advertising mode* or *connected mode*. The *Protocol Data Unit* is a variable field containing the data to transmit. Finally, a 3 bytes CRC is used for integrity checking.

4.2.2.2 Initiating a connection

When a *Peripheral* is not in a connected state, it broadcasts some advertisements on the advertising channels. The payload generally includes some information allowing to identify the device, such as the device name. To establish a connection with a *Peripheral*, the *Central* transmits a dedicated type of advertisement named *CONNECT_REQ* right after the reception of an advertisement from the *Peripheral*. The corresponding *LL PDU*, described in Table 4.2, includes some parameters used during the connection. The *Access Address* field is used by both devices for every frame transmitted during the connection.

4.2.2.3 Channel selection

The *Channel Map* and *Hop Increment* fields (cf. Table 4.2) are used by the channel selection algorithm. Indeed, a BLE connection uses a channel hopping mechanism to avoid interference with other BLE connections or wireless communication protocols. Two main channel selection algorithms are currently usable: *Channel Selection Algorithm #1* is based on a simple modular addition, while *Channel Selection Algorithm #2* is based on a pseudo-random generator. Both of them can be predicted by an attacker to sniff an established connection (see [Ryan 2013a] and [Cauquil 2019]). In our study we consider *Channel Selection Algorithm #1*, which is the most commonly used algorithm, however the proposed approach can be easily adapted to the second algorithm.

4.2.2.4 Transmit window

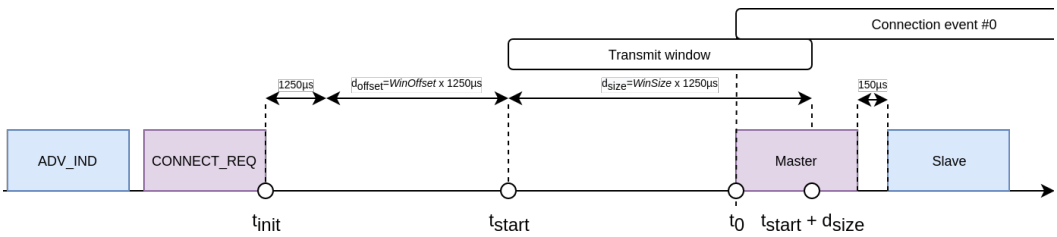

Figure 4.2: Initiation of a BLE connection

Two fields *WinSize* and *WinOffset* (cf. Table 4.2) are used to define the *transmit window*. Indeed, the first frame of the connection is transmitted on the first selected channel by the *Central* to the *Peripheral* at time t_0 during the *transmit window* defined by formula 4.1:

$$\begin{cases} t_{start} \leq t_0 \leq t_{start} + d_{size} \\ t_{start} = t_{init} + 1250\mu s + d_{offset} \end{cases} \tag{4.1}$$

With t_{init} the end of transmission time of the *CONNECT_REQ* frame, $d_{offset} = WinOffset \times 1250\mu s$ and $d_{size} = WinSize \times 1250\mu s$.

t_0 indicates the beginning of the first *connection event*, and is used as a time reference for next *connection events*. This initial phase is illustrated in Figure 4.2.

4.2.2.5 Connection events

Let us consider a *connection event* that starts at the time t_n of frame transmission from the *Central* to the *Peripheral*, called the *anchor point*. t_0 corresponds to the first *anchor point*. When the *Peripheral* receives the frame, it waits during the *inter-frame spacing* ($150\mu s$) before sending a frame to the *Central*. A bit named *More Data* (*MD*) in the header of frames allows to indicate that more data is available and will be transmitted during the *connection event*. If the device does not have data to transmit, it will transmit an empty frame.

The time between two consecutive *anchor points* is given by the *Hop Interval* parameter, according to the formula 4.2:

$$d_{connInterval} = HopInterval \times 1250\mu s \tag{4.2}$$

Each time a *connection event* is closed, the next channel is selected according to the channel selection algorithm in use. Each *connection event* is also identified by a 16-bit unsigned integer named *connection event count*. Figure 4.3 illustrates two typical consecutive *connection events*.

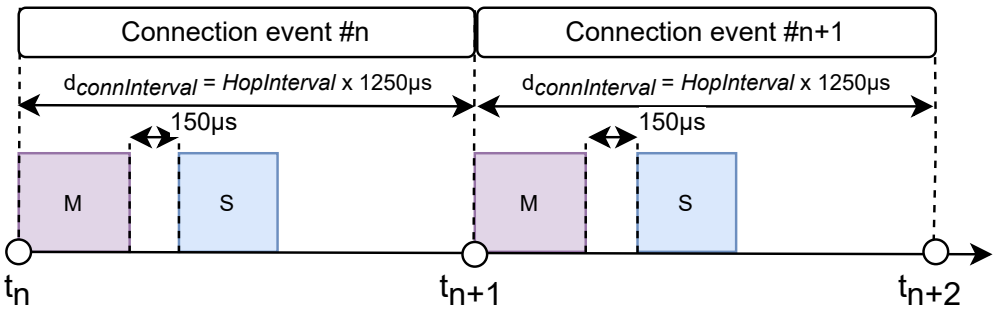

Figure 4.3: Two consecutive *connection events*

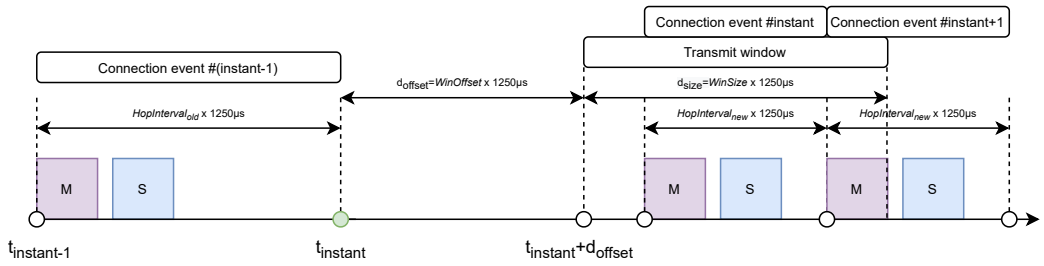

Figure 4.4: Connection update procedure

4.2.2.6 Acknowledgement and flow control

Each BLE frame transmitted during a connection includes two 1 bit fields in the header of the *LL PDU*, indicating respectively the *Sequence Number* (*SN*) and the *Next Expected Sequence Number* (*NESN*). Each device also has two 1 bit counters, respectively named *transmitSeqNum* and *nextExpectedSeqNum*. The *transmitSeqNum* counter is incremented by one (modulo 2) if the previously transmitted data is acknowledged. The *nextExpectedSeqNum* is incremented by one (modulo 2) when the next expected frame has been received.

4.2.2.7 Updating the parameters during the connection

The BLE protocol provides possibilities to update the parameters used by the channel selection algorithm. A *Central* is generally able to manage multiple connections simultaneously, and may need to modify a connection in order to optimise the following of multiple connections. It may also consider a given channel noisy due to high frame loss rate during transmission on that channel and may choose to blacklist it (i.e. mark it as unused). The Link Layer provides two main control frames, *CONNECT_UPDATE_IND* and *CHANNEL_MAP_IND*, to update the *Hop Interval* and the *Channel Map* respectively.

These frames include the new value of the field to update, and a two bytes field named *instant*. When the *Peripheral* receives one, it starts the corresponding procedure, and waits for the time when *instant* equals to *connection event count*. Then:

- In the case of a *connection update*, a *transmit window* similar to the one in the initiation of the connection is computed from the *WinOffset* and *WinSize* values of the *CONNECT_UPDATE_IND* frame. The new interval is then applied to the next *connection events*, as shown in Figure 4.4.

- In the case of a *channel map update*, the new *channel map* is used for next *connection events*.

4.2.2.8 Slave latency

The slave *latency* field (cf. Table 4.2), that is initially proposed by the *Central* in the *CONNECT_REQ* packet and can be updated in a *connection update procedure*, allows the *Peripheral* to avoid entering the listening mode at every *connection event* in order to decrease its energy consumption.

4.3 Adversary model and attack overview

This section presents a novel type of attack targeting BLE protocol, allowing the injection of arbitrary frames into an established connection. As seen in Section 4.2, the BLE protocol provides a *connected mode*, allowing the involved devices to communicate only at some specific time, making injection-based attacks difficult to perform by design. According to the specification [Blu 2019], one of the involved devices can expand the receiving window to compensate clock inaccuracy. However, this also opens the possibility for an attacker to abuse this feature by performing a race condition attack (see Figure 4.5). We focused our work on analysing the feasibility of such an injection, and explored techniques allowing to solve the following technical challenges:

- **(C1) identify when a malicious frame could be injected,**
- **(C2) investigate how to inject a malicious frame without altering the connection state consistency,**
- **(C3) check if the attack is successful or not.**

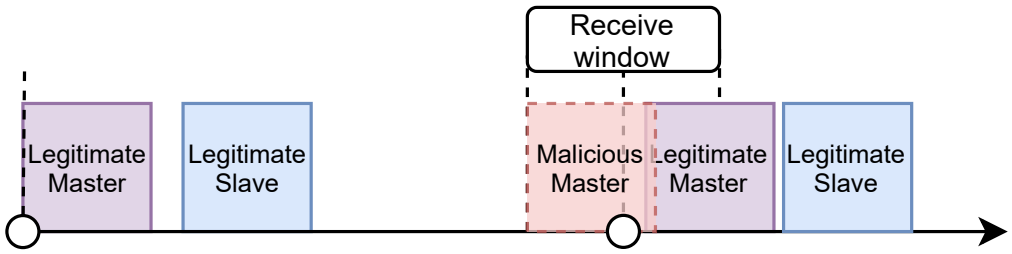

Figure 4.5: Attack overview

From an offensive perspective, the attack presented in this chapter has a significant impact: indeed, although several attacks targeting BLE security have already been investigated in several studies, none of them have made it possible to interfere with an established connection without breaking the communication, at least for one of the concerned devices. The results presented in this chapter show that such an attack is possible and can then be used to perform a wide set of critical offensive scenarios, including an illegitimate use of victim device features and hijacking attacks. We believe that this new offensive capability may consequently impact the

availability, confidentiality and integrity of any BLE communication. Indeed, the vulnerability presented in this chapter is related to the receiving window expansion described in the protocol specification, so any BLE device is potentially vulnerable, independently of its stack implementation. The threat is all the more serious as the attack is straightforward on common BLE chips and can be performed as soon as an attacker is within radio range of the targeted connection. The attack is also compatible with all versions of BLE, from 4.0 to 5.2. The adversary model considered is as follows:

- the attacker must be within the radio range of the target,
- the attacker uses a standard BLE 4.0 or BLE 5.0 device,
- the attacker is capable of passively sniffing the traffic, and actively crafting and transmitting spoofed packets on BLE channels,
- the attacker does not need to exploit any BLE vulnerability on the target devices.

As far as encrypted communications are concerned, the vulnerability being related to the design of the BLE Link Layer, it is independent of the security mechanisms provided by the protocol. Therefore, exploiting the race condition to inject a frame in an encrypted connection remains technically possible. Indeed, even if the attacker cannot obtain the Long Term Key used for encryption by some other mean, he can still inject an invalid packet, leading to a denial of service. As a consequence, enabling the security mechanisms provided by BLE limits the impact of the attack but the vulnerability itself (race condition allowing to inject a frame) remains, with at least an impact on availability.

4.4 InjectaBLE: injecting arbitrary frames in an established connection

In this section, we present the *InjectaBLE* attack, allowing to inject arbitrary frames in an established connection. Performing such an attack requires to identify a specific time when a frame can be successfully injected by the attacker, called the *injection point*. Subsections 4.4.1 and 4.4.2 describe the specific features of the Link Layer that make it possible to find such an injection point (challenge **C1** of Section 4.3). Subsection 4.4.3 describes how to inject the well-formed frame without altering the consistency of the connection state (challenge **C2**) and Subsection 4.4.4 describes how to check whether the injection is successful or not (challenge **C3**).

4.4.1 Clock (in)accuracy

As mentioned earlier, the start of transmission of a *Central* frame in a given *connection event* is used as a time reference, named *anchor point*. Theoretically, given

an *anchor point* t_n, the next *anchor point* should occur at t_{n+1} according to the formula 4.3.

$$t_{n+1} = t_n + d_{connInterval} \tag{4.3}$$

An attacker cannot inject a frame at this specific time, as this frame would collide with the legitimate *Central*'s packet. However, the legitimate devices involved in an established connection use multiple timers based on a clock named *Sleep Clock*. As this clock can introduce a drift in time, the *Peripheral* cannot assume that its *Sleep Clock* is perfectly synchronised with the *Central*'s and should listen for an extra time before and after the timing estimated from the *anchor point*.

4.4.2 Window widening

The specification introduces a concept named *window widening*, which consists in extending the listening time of a given device to compensate clocks inaccuracies. In the specific case of *Peripheral*'s Link Layer receiving the next *connection event*, the *window widening* w is computed using formula 4.4.

$$w = \frac{SCA_M + SCA_S}{1000000} \times (t_{nextAnchor} - t_{lastAnchor}) + 32\mu s \tag{4.4}$$

- SCA_M : *sleep clock accuracy* of *Central*'s LL (in ppm),
- SCA_S : *sleep clock accuracy* of *Peripheral*'s LL (in ppm),
- $t_{nextAnchor}$: predicted next *anchor point* time (in μs),
- $t_{lastAnchor}$: last observed *anchor point* time (in μs).

If the *Peripheral* transmits a frame for every *connection event* (i.e. Peripheral *latency* equals to 0), the formula can be rewritten:

$$w = \frac{SCA_M + SCA_S}{1000000} \times d_{connInterval} + 32\mu s \tag{4.5}$$

A *Slave latency* greater than 0 increases the interval between the last observed *anchor point* and the predicted next *anchor point*, resulting in a larger window. In that case, equation 4.5 can be considered as the minimal *window widening*.

As a consequence, given a predicted *anchor point* t_{n+1}, the *Peripheral* accepts the *Central*'s packet initiating the *connection event* if it is transmitted during the *receive window* from $t_{n+1} - w$ to $t_{n+1} + w$, as illustrated in figure 4.6.

4.4.3 Injecting an arbitrary packet

A frame transmitted in the previously mentioned *receive window* being considered as a *Central* packet by the *Peripheral*, this feature allows a race condition attack, in which an attacker can inject an arbitrary frame in an established connection by transmitting it at the beginning of the *receive window*.

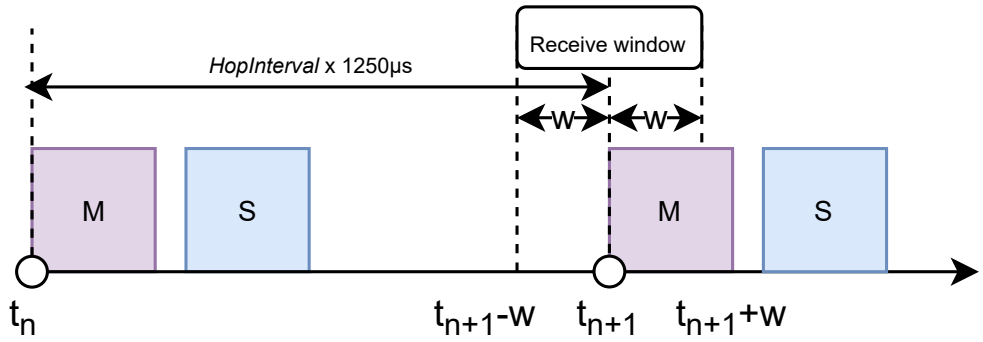

Figure 4.6: *Window widening* for a *Peripheral* receiving the next *connection event*

For this injection to be successful, the attacker has first to be synchronised with the connection: as mentioned in the related work, multiple approaches already exist to passively sniff a connection. Second, the attacker must forge a valid frame to inject. It will be considered as new data by the *Peripheral* if its *Sequence Number* (denoted as SN_a) equals the *Next Expected Sequence Number* counter of the *Peripheral*'s Link Layer (denoted as $NESN_s$). Similarly, the *NESN* bit in the attacker frame (denoted as $NESN_a$) should indicate that the previous frame transmitted by the *Peripheral* (denoted as SN_s) was successfully received. Thus, the attacker should have observed in the *connection event* preceding the injection attempt a frame transmitted by the *Peripheral* and extracted the SN_s and $NESN_s$ bits. The SN_a and $NESN_a$ bits of the injected frame are then set according to the equation 4.6.

$$\begin{cases} SN_a = NESN_s \\ NESN_a = (SN_s + 1) \mod 2 \end{cases} \tag{4.6}$$

Third, the attacker has to calculate the *receive window* to transmit the injected frame as soon as possible during this window. He can use equation 4.5 to estimate the *window widening*. The *Central*'s *Sleep Clock Accuracy* can be extracted from the *CONNECT_REQ* packet or from control packets embedding this information (e.g., *LL_CLOCK_ACCURACY_REQ* or *LL_CLOCK_ACCURACY_RSP*). The *Peripheral*'s *Sleep Clock Accuracy* can be estimated at 20 ppm, which is the worst case from the attacker's perspective.

4.4.4 Checking the injection success

In order to perform various attacks requiring the injection of multiple frames, the attacker must be able to identify whether the injection of each frame is successful or not. This is not straightforward as even a successful injection does not always provoke an observable change in the behaviour of the *Peripheral* receiving the frame. Therefore, we need an heuristic that only relies on the observation of the parameters of the Link Layer, to indicate whether the injection is successful or not.

An injected frame is considered as valid by the *Peripheral* if:

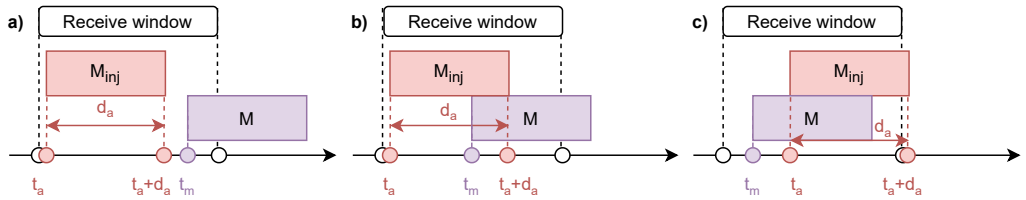

Figure 4.7: Three possible outcomes of an injection attempt

- the injected frame is transmitted before the *Central*'s one during the *receive window*,
- the CRC of the injected frame equals the calculated one.

Let's consider an injection attempt with t_a the start time of the injected frame transmission (i.e., the beginning of the attack), d_a the transmission duration of the injected frame and t_m the beginning of the legitimate *Central*'s frame transmission.

An injection attempt may result in three different situations, as illustrated by figure 4.7:

a. the injected frame is transmitted in the *receive window* before the start of transmission of the legitimate frame ($t_a + d_a < t_m$)

b. the injected frame is transmitted in the *receive window*, but the end of the frame collides with the legitimate frame ($t_a + d_a \geq t_m$)

c. the legitimate frame is transmitted before the injected frame ($t_a \geq t_m$)

In situation a), the injection attempt is successful, because the two conditions are met. Situation b) can result in a successful injection if the collision does not corrupt the injected frame, otherwise the CRC is invalid and the injection attempt fails. Indeed, a collision might not result in a corruption when the power of the injected signal is by far superior to the power of the legitimate signal from the *Peripheral*'s perspective. It can also happen if the modification resulting from the superposition of two signals doesn't change the result of the heuristic used by the demodulator to demodulate the injected signal. This is possible in some configurations, depending on the phase difference between the injected and legitimate signals from the *Peripheral*'s perspective, along with the previously mentioned power difference. Situation c) leads to a failed injection attempt, because the first condition is not fulfilled.

Since an injection attempt may or may not be successful depending on the situation, the attacker can build an heuristic allowing him to know if a given injection was successful. This heuristic is based on the two previously mentioned conditions:

- the injected frame is transmitted before the *Central*'s one during the *receive window*: a direct observation of the legitimate packet transmitted by the *Central* is usually not possible because the attacker transmits its own injected

packet at the same time. However, the *Peripheral*'s response can be used to infer this information indirectly. Indeed, if the injected frame was transmitted before the legitimate one, the *Peripheral* will consider the start of transmission of the injected frame as the new *anchor point*. Consequently, the *Peripheral* will transmit its own frame 150 μs after the end of transmission of the injected frame. If t_s is the start of transmission of the *Peripheral*'s response, this requirement can be expressed as :

$$t_a + d_a + 150 - 5 < t_s < t_a + d_a + 150 + 5$$

We empirically estimated a window width of $10\mu s$, resulting in the $5\mu s$ in the above formula. This estimation has been established by injecting some specific packets that have an observable impact on the *Peripheral* device (e.g., transmitting a response, terminating the connection).

- the CRC of the injected frame equals the computed one: similarly, the attacker cannot directly check if a collision occurs and corrupts the injected frame during the transmission because he cannot listen to the channel during the injection. However, the *Peripheral*'s response can also be used to infer this information, because if the frame was received by the *Peripheral* with a CRC field that does not match the calculated one, the *Peripheral* will not change its *nextExpectedSeqNum* counter to indicate that the last received frame must be transmitted again, resulting in a *NESN* field equal to the one used in the previous frame transmitted by the *Peripheral*. If SN'_s is the *SN* field of *Peripheral*'s response and $NESN'_s$ is the *NESN* field of the *Peripheral*'s reponse, this requirement can be expressed as:

$$((SN_a + 1) \mod 2 = NESN'_s) \wedge (NESN_a = SN'_s)$$

Finally, the global heuristic that allows the attacker to detect the success of the injection can be expressed by the propositional formula 4.7:

$$\begin{aligned} &(t_a + d_a + 150 - 5 < t_s < t_a + d_a + 150 + 5) \wedge \\ &((SN_a + 1) \mod 2 = NESN'_s) \wedge (NESN_a = SN'_s) \end{aligned} \tag{4.7}$$

with t_a the start of the transmission of the injected frame, d_a the duration of the transmission of the injected frame, t_s the start of transmission of the *Peripheral*'s response, SN'_s the *SN* field of the *Peripheral*'s response, $NESN'_s$ the *NESN* field of the *Peripheral*'s response.

4.4.5 Implementation

We have developed a proof of concept in order to easily perform the *InjectaBLE* attack and evaluate it. It has been implemented on a development dongle embedding a *nRF52840* chip from *Nordic Semiconductor*. This chip natively supports

BLE 5.0 and allows a low level access to the *Radio peripheral*, which eases the implementation.

The dongle communicates with the *Host* using a custom USB protocol, allowing to transmit commands to the embedded software. A lightweight BLE sniffer has been implemented, based on previous works [Ryan 2013a, Cauquil 2017b] and [Qasim Khan 2019] on BLE connection eavesdropping. When a new connection is detected by the sniffer, it synchronises with the channel hopping algorithm and transmits the received packets to the *Host*. Then, if a specific command is transmitted to the dongle, it starts the injection process and tries to inject the malicious frame defined in the command:

- before the injection, the *window widening* in use is estimated using formula 4.5.
- the dongle performs an injection attempt as soon as possible during the window previously defined.
- the heuristic defined in formula 4.7 is then used to check whether the injection was successful or not.
- if the injection attempt fails, a new one is prepared.
- if the injection attempt succeeds, a notification is transmitted to the *Host* indicating the number of injection attempts before a successful injection.

Based on this main feature, the dongle also exposes an API allowing to perform the various scenarios described in Section 4.5. A minimal *BLE stack* has also been implemented, to mimic the behaviour of the different roles involved in the connection.

This custom firmware is released as open-source software under MIT license and can be found here: `https://github.com/RCayre/injectable-firmware`.

4.5 Attack scenarios

This Section describes and illustrates four main scenarios allowing an attacker to achieve interesting offensive objectives, such as illegitimately using a device functionality, hijacking any device involved in the connection or performing a Man-in-the-Middle attack during an established connection.

4.5.1 Scenario A: illegitimately using a device functionality

This first attack scenario can be considered as the straightforward application of the injection attack. Indeed, IoT devices based on BLE usually implement the *Peripheral* role, so our injection approach may be used to trigger a specific functionality exposed by the targeted device. More specifically, injecting *ATT Requests* allows

the attacker to interact with the *ATT* server, which is used in BLE as a generic *application layer*. Note that any *ATT* request supported by the target device could be possibly injected.

For example, an attacker could inject a *Read Request* targeting a specific handle: if the injection is successful, the *Peripheral* will generate and transmit a *Read Response* containing the data. It may allow him to extract interesting information from a given *characteristic*: depending on the type of device, this could have a critical impact on confidentiality. Similarly, an attacker could inject a *Write Request* or a *Write Command* to a given device. These *ATT requests* allow to modify the value of a given *characteristic*: as a consequence, the attacker is able to trigger a specific behaviour of the device, which could result in a critical impact on integrity or availability.

To illustrate the impact of this attack scenario, we have performed injection attacks targeting three commercial devices: a lightbulb, a keyfob and a smartwatch. We reverse engineered these devices to identify the type of *ATT requests* and the corresponding payloads used to trigger their main features. We then forged and injected malicious traffic triggering the following features:

- lightbulb: turning the bulb on and off, changing its colour, changing its brightness,
- keyfob: making the keyfob ring,
- smartwatch: transmitting a forged SMS to the watch.

4.5.2 Scenario B: hijacking the *Peripheral* role

This second attack scenario is aimed at hijacking the *Peripheral* role. If this attack succeeds, the *Peripheral* is forced to exit the connection, allowing the attacker to replace it without breaking the connection from the *Central*'s perspective.

This attack scenario is based on the injection of a *Link-layer control* packet: these packets are used by devices to control the connections. More specifically, the attack is based on the injection of a *LL_TERMINATE_IND* packet that is used by a device to indicate to the other one that the connection should be terminated. Since the packet injection is ignored by the *Central* and accepted by the *Peripheral*, it forces the *Peripheral* to exit the connection. However, the *Central* is not aware of the fact that the legitimate *Peripheral* is not present anymore: this situation allows the attacker to imitate the *Peripheral* behaviour in order to hijack the connection. To do so, the attacker must wait during the *inter-frame spacing* (150 μs) after the end of transmission of a *Central*'s packet before transmitting its frame, and carefully set the *SN* and *NESN* fields. This attack scenario is illustrated in figure 4.8.

This scenario has been successfully implemented for the three previously mentioned devices. All of them exposed a *characteristic* corresponding to the *Device Name* which allowed us to transmit a forged value "Hacked" when a *Read Request* targeting this *characteristic* was received. Let us note that such a scenario may

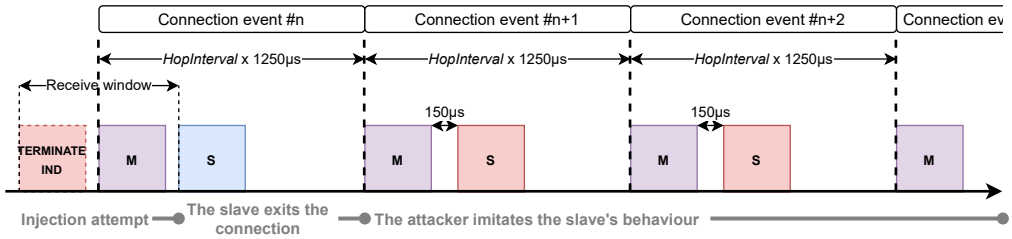

Figure 4.8: Description of the Peripheral hijacking

have critical consequences depending on the type of target: as an example, an insulin pump or a pacemaker could be hijacked, allowing the attacker to transmit fake health data.

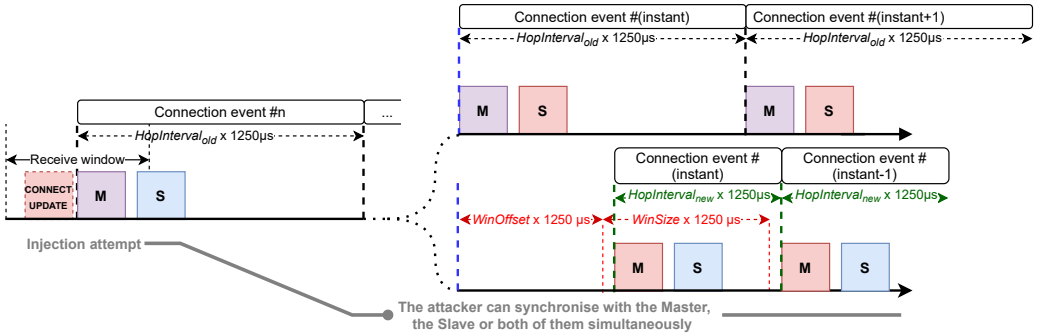

Figure 4.9: Description of the Man-in-the-Middle attack

4.5.3 Scenario C and D: hijacking the *Central*, the *Peripheral* or both of them simultaneously (Man-in-the-Middle attack)

We have explored two other attack scenarios, based on the same approach. The scenario C consists in hijacking the *Central* role. While this kind of hijacking attack was already possible using the *BTLEJack* tool [Cauquil 2018], its strategy is based on jamming and can easily be detected by a monitoring system. Our approach only requires the injection of a single malicious frame, making it more discrete and reliable. Scenario D allowed us to carry out a Man-in-the-Middle attack without interrupting the connection. Indeed, previous approaches to perform Man-in-the-Middle attacks [Cauquil 2016, Jasek 2016] could only be used before the initiation of the connection, which limits drastically their usability. In other words, using our strategy, an attacker could establish a Man-in-the-Middle attack **at any time**, even if a connection is already established between two legitimate devices. This strategy is critical as long term connections are very common in BLE communications, and massively used by devices such as smartwatches or trackers.

These two scenarios use a similar approach, which is based on the injection of a *CONNECTION_UPDATE PDU* as described in Section 4.2.2: it can be used by

the *Central* at any time during the connection in order to modify the parameters of the channel selection algorithm, and especially the *Hop Interval*. The attack relies on a simple idea: the attacker injects a forged *CONNECTION_UPDATE PDU* containing arbitrary parameters, indicating to the *Peripheral* that the connection parameters will change at a given time. When that time is reached, the *Peripheral* waits during the *window offset* specified by the attacker, ignoring the legitimate *Central*'s frame, then uses the new parameters while the *Central* continues to use the old ones, allowing the attacker to synchronise with the *Peripheral* and hijack the *Central* role or to synchronise with both of them, resulting in a Man-in-the-Middle. In the first case (e.g., *Central* hijacking) the legitimate *Central* no longer receives any response after the time at which the parameters are changed, so it leaves the connection due to timeout. Note that this approach is particularly powerful because it could also be used to hijack the *Peripheral* role, in a similar way to scenario B, since the attacker knows both the old and the new parameters. This approach is illustrated in figure 4.9.

We evaluated experimentally the *Central* hijacking using the three previously mentioned devices: with the *Central*'s role successfully hijacked, it allowed us to trigger the same features as in scenario A. Similarly, scenario D was evaluated on our three commercial devices, allowing us to arbitrarily modify the data exchanged between the legitimate devices: for example, a SMS transmitted by the smartphone to the smartwatch has been modified on the fly, or the *RGB* values describing the colour of the lightbulb have also been altered on the fly.

4.6 Sensitivity analysis

We conducted several experiments to validate our attack. The objective was twofold: test its feasibility in a realistic environment and analyse the impact of different parameters upon the attack success rate. We focused on three main parameters that may have a significant impact on the attack success: the *Hop Interval*, the payload size and the distance between the attacker and the target *Peripheral*. One parameter at a time was changed and its impact on the attack success was assessed by monitoring the number of injection attempts before a successful injection.

4.6.1 Experiment 1: Hop Interval

Our first experiment focused on the *Hop Interval* parameter. Indeed, this parameter is directly involved in the estimation of the *window widening* as indicated in equation 4.5. Theoretically, as the attack relies on a race condition based on this window with the legitimate *Central*, the injection should be more difficult when the *Hop Interval* value is lower.

According to the specification, the theoretical *Hop Interval* range is from 6 to 3200. However, we chose to focus on six different values from 25 to 150 for two main reasons:

- We wanted to focus on the worst case of an injection attempt, which occurs when the injected frame collides with the legitimate frame, which means considering low *Hop Interval* values. Since the injected frame used during this experiment was 22 bytes long over the air (i.e., 176 μs of transmission time using the *LE 1M* physical layer), none of the *window widening* values calculated from the tested *Hop Intervals* allowed an injected frame to be entirely transmitted without a collision.

- We wanted to conduct our experiment on target real-life devices and most of them do not allow the use of high *Hop Interval* values, because the resulting connections could be extremely unstable and break quickly. We thus used the *Hop Interval* values in the range supported by a connected lightbulb, which was the commercial device supporting the widest range of *Hop Interval* values we were able to find.

To be able to precisely tune the *Hop Interval* parameter, we used a modified version of the open-source Mirage framework to simulate a *Central* device, because of its capability to access the *HCI* on a low level. This framework is one of our offensive contribution, it is presented in chapter 5.

We reversed the communication protocol built over *GATT* used by this lightbulb, then selected a *Write Request* allowing to turn the light off as our injection frame. The corresponding payload is 14 bytes long, making the entire frame 22 bytes long. We chose a frame with a visible effect on the device to validate our heuristic.

The experimental setup was quite simple: the legitimate *Peripheral* and *Central* devices and the attacker were placed on the three vertices of an equilateral triangle, with 2 meters edges. The *Central* initiates connections with the *Peripheral* repeatedly while the attacker synchronises with these connections and starts the injection attack at a specific *connection event*. The experiment was conducted in a realistic environment, including several other BLE devices and multiple *WiFi* routers. Let us note that synchronising the attack tool with a connection is not trivial, especially in such a noisy environment. For each *Hop Interval* value, we performed 25 injection attacks, and monitored the number of injection attempts required before a successful injection. The results are presented in figure 4.11.

The attack was successful for every tested connection. The variance of the number of unsuccessful attempts decreases quickly between 25 and 100, and stabilises afterwards. Similarly, the median value remains at a low value less than 4. These results show that the injection is always feasible even with small *Hop Intervals*, and the number of injection attempts required before a successful injection is generally low. The experiment confirms that the *Hop Interval* has a significant impact on the injection attack success. However, the injection is more reliable with higher values.

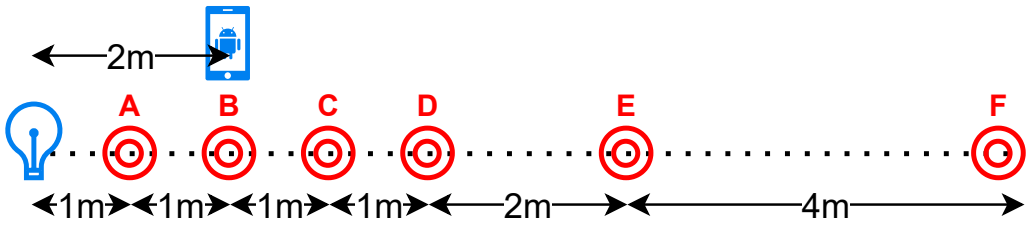

Figure 4.10: Experimental setup

4.6.2 Experiment 2: Payload size

This experiment was focused on the payload size of the injected frame, and was intended to empirically confirm that injecting shorter frames increases the probability of success.

The experimental setup and the environment are similar to the one presented above. We selected four different values of payload size: 4, 9, 14 and 16, which correspond to frames that have an observable effect on the target lightbulb (such as disconnecting it, turning it off, or changing its colour), allowing to confirm the success of an injection attempt independently from our success detection heuristic.

We repeated the experiment 1, this time with a fixed *Hop Interval* of 75, and iterating over the different payload sizes. The results are displayed in figure 4.11.

Similarly to experiment 1, we observe higher reliability when the payload size decreases, which is consistent with the theory as a smaller portion of the injected frame collides. The number of injection attempts required before a successful injection remains very low (less than 3 for the median).

4.6.3 Experiment 3: distance

Our last experiment was conducted to evaluate the impact of the distance between the attacker and the legitimate *Peripheral*. Theoretically, since the distance impacts the signal strength of the injection from the *Peripheral*'s perspective, it may lower even more the success rate when a collision with the legitimate frame occurs. We used the same lightbulb as *Peripheral*, but used a smartphone as legitimate *Central* to get closer to a real-life scenario. The phone was used to establish 25 connections per tested distance, using its default *Hop Interval* value equal to 36. As we chose to only inject the 22 bytes long *Write Request* allowing to turn the bulb off, this *Hop Interval* value doesn't allow for collision-free transmissions.

The experimental setup was slightly different from the one used in experiments 1 and 2: we placed the lightbulb and the phone within two meters of each other, then we tested six different positions for the attacker, from 1 to 10 meters, as illustrated in figure 4.10. This allowed us to evaluate the attack success when the attacker is closer to the *Peripheral* than the legitimate *Central* (position A), when they are at the same distance (position B) and when the attacker is further (positions C,D,E and F).

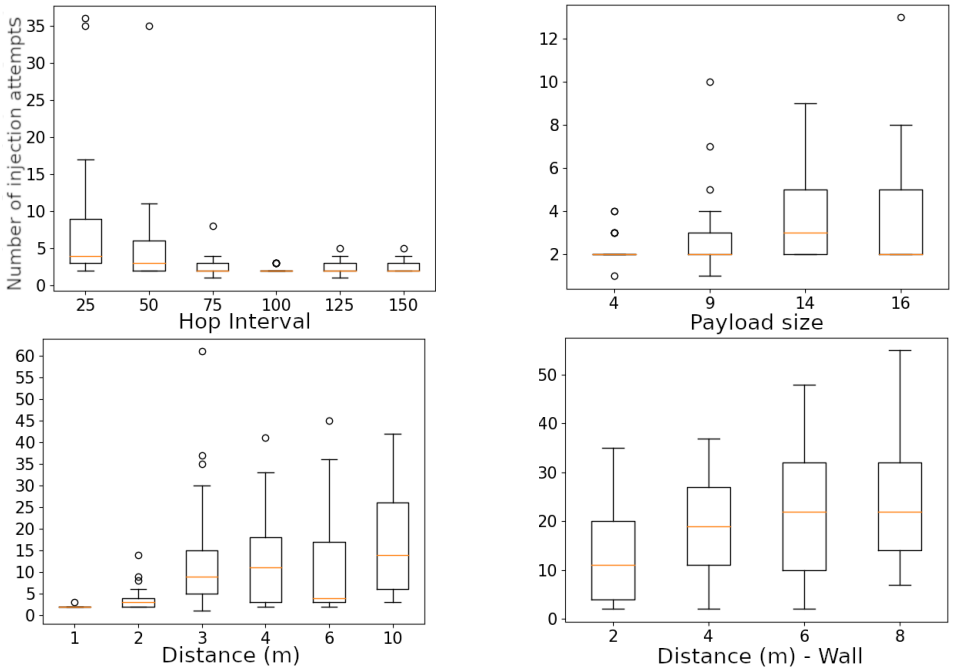

Figure 4.11: Experiment Results

The results are presented in figure 4.11. They show a significant impact of the distance between the attacker and the *Peripheral* on the reliability, as the variance increases when the distance is higher. It validates our assumption that the attacker has an higher probability to quickly perform a successful injection if closer to the target. However, let us note that each tested connection leads to a successful injection: it means that the attacker can perform a successful attack from every location, including position F which is 10 meters away from the *Peripheral*, while the legitimate *Central* is only 2 meters away. This experiment highlights the practical feasibility of the attack, and shows that, even under adverse conditions in a realistic environment, the attack is still possible.

We also tested the attack effectiveness behind a wall, to evaluate the impact of obstacles. The experimental setup was very similar to the distance experiment: the lightbulb (legitimate *Peripheral*) and the phone (legitimate *Central*) were placed within two meters of each other in the same room, while the attacker was located at four different positions behind a wall, from 2 to 8 meters from the *Peripheral*. Similarly to the other experiments, we established 25 connections per tested distance and measured the number of injection attempts needed before a successful injection. Results are presented in figure 4.11: as expected, the presence of a wall increases the number of injection attempts needed to perform a successful injection, and the variance increases with the distance. However, even if the attack requires more attempts, we managed to successfully inject a frame for every connection we tested, even in the worst case from the attacker's perspective. These results show that this attack is realistic and could be carried out even if the attacker is not in the same

room as the target.

4.7 Counter-measures

The *InjectaBLE* attack exploits a vulnerability that is inherent to the BLE protocol specification. As a result, we should consider every BLE communication as potentially vulnerable and the environments exposed to BLE devices should be designed and monitored with the assumption that some attacks could potentially be carried out through legitimate communications. Several counter-measures could be investigated either to limit the impact of the attack, or to prevent or detect it.

As explained in section 4.2.2, the practical implementation of the *InjectaBLE* attack requires injecting arbitrary frames at specific moments. Three solutions could be investigated. Each one requires more or less deep changes in the BLE stack or in the usage of BLE chips. These changes may not be appropriate from the user's point of view in the case of an industrial environment, because of a possibly high number of devices to reprogram and the cost of certification processes.

The first solution deals with some communication time parameters of the stack itself. For example, by reducing the duration of the widening windows the possibility for an attacker to inject a frame at the right time will be mechanically reduced. However it should be noted that such an approach requires changes to the BLE standard which could have side effects on the reliability and stability of the communications. The second solution is slightly less restrictive. Without going as far as modifying the BLE standard, it requires to systematically activate the encryption mechanisms defined in BLE specification. If all frames are correctly encrypted, an attacker will not be able to easily sniff the connection parameters and forge a valid frame. In this specific case, the vulnerability is still present, even if its impact is limited to Denial of Service attacks. While this solution could be straightforward, it is not in reality. It must be noted that the majority of BLE communications are poorly or not at all encrypted today (see [Zuo 2019] for a qualitative study of the percentage of BLE devices activating encryption mechanisms). As a consequence, in most cases, this counter-measure requires end-users to reprogram all their devices, which could be tricky, especially in the context of industrial devices.

During our experiments, we noticed that some manufacturers do not use the native protocol encryption but rather choose to implement their own over the *GATT* application layer. We strongly advise against this solution, since in this case the *LL control frames* will not be encrypted and we have already demonstrated in our attack scenarios that an attacker could achieve interesting objectives by injecting this kind of frames, such as initiating a Man-in-the-Middle and not forwarding the legitimate traffic to perform a denial of service.

The last solution is based on a non intrusive approach. Defensive solutions dedicated to the IoT context could be considered to monitor and detect in real time or not attacks targeting wireless protocols. An *Intrusion Detection System* (*IDS*) designed to monitor BLE Link Layer could be able to detect, at the right instant,

the presence of double frames: the legitimate *Central* frame and the attacker one. For instance, the *IDS* proposed in [Roux 2018] is able to identify deviations from legitimate behaviour by monitoring the radio activity of the wireless environment. F. Galtier et al. also propose, in [Galtier 2020], an *IDS* able to fingerprint legitimate devices (based on physical characteristics of the radio signals) and to detect inappropriate fingerprints related to the attacker frames. Monitoring solutions designed to detect BLE spoofing attacks, such as [Wu 2020b] or [Yaseen 2019], may also detect behavioural anomalies in the communication between the devices, for example variations in the timing between packet emissions or change of BLE profile, and hence detect the injection attempts. Machine Learning oriented solutions can also be relevant, as in [Lahmadi 2020], where A. Lahmadi et al. used Neural Networks to build an attacker model, and detect Man-in-The-Middle attacks.

4.8 Conclusion

In this chapter, we demonstrated the feasibility of a new injection attack named *InjectaBLE* targeting the BLE protocol, allowing to inject malicious frames into an established connection. This attack significantly increases the attack surface of BLE communications, because it exploits a vulnerability of the specification itself independently of the stack implementations, and can be achieved quite easily using common BLE chips. We analysed the impact of multiple factors on the attack success rate and demonstrated that exploiting this weakness could allow an attacker to perform critical attack scenarios that were not realistic until now, such as *Peripheral* hijacking or *Man-in-the-Middle* attack targeting established connections. We also performed sensitivity analyses that showed that this injection always succeeds in various experimental conditions.

Activating the BLE native cryptographic mechanisms can efficiently mitigate this attack. However, in practice, the vast majority of commercial devices do not use encryption, making them vulnerable by design to *InjectaBLE*. The results presented in this chapter clearly highlight the need to generalise the systematic use of encryption in BLE communications.

The new offensive capabilities provided by *InjectaBLE* open opportunities for other critical attack scenarios that need to be carefully investigated. For example, being able to hijack the *Peripheral* role may potentially allow an attacker to transmit an ATT notification indicating that the *ATT* server structure has been modified: it could be used to expose a malicious keyboard profile instead of the original one, and inject keystrokes to the *Central* by implementing *HID over GATT* protocol.

Two scientific articles have been published to describe this research work, both in national and international conferences:

- Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, et al.. **InjectaBLE : injection de trafic malveillant dans une connexion Bluetooth Low Energy**. *Symposium sur la sécurité des technologies de l'information et des communications (SSTIC 2021)*, Jun 2021, Rennes (en ligne), France. **[FR]** [Cayre 2021b]
- Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, et al.. **InjectaBLE: Injecting malicious traffic into established Bluetooth Low Energy connections.** *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2021)*, Jun 2021, Taipei (virtual), Taiwan. **[EN]** [Cayre 2021c]

Chapter 5

Mirage: an offensive auditing framework

Contents

In this chapter, we introduce a new offensive auditing framework named Mirage, that aims at facilitating the development of offensive tools targeting wireless communication protocols. Indeed, we presented in Section 2.1 the wide variety of hardware and software tools present in the wild. A major issue we noted is that these tools provide heterogeneous API and capabilities, while sometimes relying on components and libraries which are not suited for offensive security. This situation

significantly impacts the reproducibility of security research and leads to redundant or poorly-written code, which is difficult to extend, maintain and document.

The main goal of Mirage framework is to provide a generic, modular and user-friendly development environment allowing to easily perform wireless attacks. It significantly facilitates the prototyping and implementation of offensive tools by providing a generic way to interact with underlying hardware, while including several user-friendly and customizable modules implementing various attacks and protocol-specific behaviours. Moreover, it has been designed to facilitate the integration of new hardware tools and protocols. At the time of writing, Mirage supports up to 6 wireless protocols, from Bluetooth Low Energy to Enhanced ShockBurst, and a wide number of hardware tools (e.g., Sniffle, BTLEJack, RFStorm, RZUSBStick) while providing dozens of offensive modules.

5.1 Motivations

In the specific context of IoT, reliable and efficient tools and relevant experimental-based methodologies are needed to analyze and assess the security of connected devices. Many different exploitation frameworks and exploits have been released in recent years, such as Killerbee [Wright 2009] or BTLEJack [Cauquil 2018]. However, conducting security audits of IoT devices is still a challenging task for multiple reasons.

The multiplicity and the heterogeneity of wireless communication protocols used by IoT devices has led to the development of various offensive radio frequency (RF) components (e.g., Ubertooth, Micro:Bit, Yard Stick One) [Spill 2012,Cauquil 2017c, Atlas 2012]: every hardware component has its own specificities and APIs, leading the security analyst to develop time-consuming and low value source code to perform such experimental analyses. Moreover, two distinct types of hardware components are commonly used to analyse wireless communications: *Software Defined Radios* or low cost *Systems On Chip* adapted to RF communications. However, they have various limitations in terms of efficiency. *Software Defined Radios* are interesting because of their genericity and because they are highly customizable. However, they involve a large amount of effort to implement the protocols stacks. On the other hand, the *System on Chips* are generally not designed for security analysis, implying the development of customized offensive *firmware*, often poorly documented and poorly tested.

As a result, security analysts generally make use of high-level libraries that have not been developed with a security perspective in mind, or create their own libraries, resulting in a lack of modularity and flexibility. This generates costly duplicate developments and sometimes leads to poorly written code.

The code of these tools becomes more complex, and as a consequence, more likely to be error-prone. **Simplicity**, **reusability** and **modularity** are three characteristics of modern software development according to McCall's Factor Model [P. Cavano 1978], and the previously mentioned constraints have a significant impact on

these characteristics (e.g., the development of complex architectures or the use of custom firmware).

The aforementioned problems have a big impact on the reproducibility of security audits based on penetration testing techniques, which is a key requirement in this context [Dalalana Bertoglio 2017]. Indeed, the use of non standard libraries and software tools as well as the heterogeneity of hardware components make a systematic approach difficult. It is imperative to provide a tool to efficiently carry out experimental security audits for IoT devices. Unfortunately, to our knowledge, such a tool does not exist yet.

To fill this gap, this chapter presents the design and the implementation of an original open source attack-oriented framework to support the security analysis of IoT devices, named "*Mirage*". This framework, written in Python, targets commonly used wireless IoT communication protocols. The primary objective is to provide a modular and flexible software environment for the development of security assessment tools, similarly to the popular Metasploit framework [MET 2022]. However, providing such a framework is a more difficult task because of IoT specificities (e.g., the large amount of protocols and offensive RF hardware components used).

As a result, the proposed framework is designed to interface with any kind of RF components thanks to a versatile communication architecture. It provides an unified API to analyse the lower layers of various wireless protocols, and can be easily extended to support new protocols. Finally, *Mirage* allows complex attack scenarios to be implemented, by the combination and chaining of different modular software components.

5.2 Key Principles

The proposed attack-oriented framework named *Mirage* is aimed at developing a modular and flexible software environment allowing to address the main constraints inherent to this type of offensive security tools, especially the heterogeneity of RF hardware components commonly used in the IoT world and the lack of low level attack-oriented libraries covering IoT wireless communication protocols. Four main principles, illustrated in figure 5.1 have guided and motivated our work, and are discussed in the following subsections.

5.2.1 Providing an unified API

Nowadays, conducting experimental-based security audits of connected objects implies that security analysts must use at the same time several different software tools, each providing its own API and documentation, and potentially using specific file formats. Moreover, each of these tools has its own limitations.

As a consequence, the analyst has to learn a lot of technical information that is not directly linked to the audit workflow and is generally neither relevant nor reusable. In addition, they must install the different, and potentially dependent,

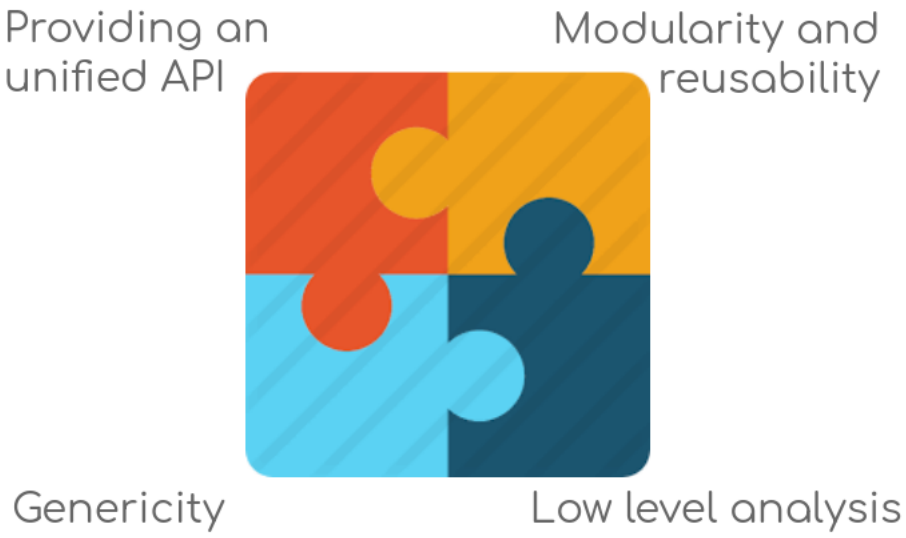

Figure 5.1: Key principles of Mirage framework

tools and libraries, leading to increasingly complex solutions. This situation involves rewriting many non-reusable codes to work together or to integrate specific functionalities.

Our framework is designed to seamlessly integrate the different software components and provide a unified API. Indeed, each software component can be configured via a similar interface and uses the same type of display, logging and output mechanisms. The multiple components used to manipulate the wireless communication protocols closely follow the same lines and expose a common API. In addition, the framework architecture requires developers to follow these guidelines while developing audit modules or implementing new protocols.

This approach facilitates the interactions between different pieces of code while harmonising the use and implementation of attacks.

5.2.2 Modularity and reusability

Modularity is one of the main features of our framework. Indeed, some attacks targeting a specific protocol implement similar elementary actions and behaviours. For example, fuzzing and cloning a *Bluetooth Low Energy* devices involve the same type of actions: the tool must scan the RF environment to find the target device, then connect to that device and discover the services and features, etc. Finally, the specific behaviour (fuzzing the device or simulating a similar one) can be triggered.

To prevent code redundancy and facilitate maintenance, the framework is designed to divide complex attacks scenarios into small functional modules. Therefore, in the aforementioned use cases, attacks could be broken down into small code units: a scanner, a connection module and a discovery module are provided and can be used in both scenarios, avoiding code redundancy. *Mirage* allows existing

modules to be directly reused in a new attack implementation, but also to be executed sequentially using a chaining operator similar to the pipe operator in UNIX environments. This behaviour allows to quickly generate complex attack workflows by controlling the combination of several modules.

This modular approach allows us to cover a large amount of existing attack tools without rewriting them all in our framework. Therefore, a security analyst can really focus on his attack workflow without having to write a lot of irrelevant code.

5.2.3 Genericity

Many different RF hardware components are nowadays used to perform experimental based security audits. However, each offers different features and APIs, and a significant part of an analyst's work is devoted to understanding the functionalities provided by a specific component and implementing the corresponding methods. As a result, one of the most important guidelines for us was to design an architecture able to manage these multiple hardware components while following the aforementioned principles, e.g., by providing a unified API to use them.

Each wireless communication protocol commonly used in IoT devices has its own specificities. However, we have identified similar behaviours. As a consequence, we have chosen to design a generic communication architecture that allows new protocols to be easily integrated or existing ones to be manipulated. Currently, this architecture is functional and we have successfully integrated twelve heterogenous hardware components related to several protocols such as *Bluetooth Low Energy*, *Wifi*, *Zigbee*, *Enhanced ShockBurst* and *Infrared radiation* (IR).

5.2.4 Low level analysis

As mentioned above, many different attack tools use high-level libraries. These libraries are generally not designed for security analysis, and they potentially suffer from constraints and limitations that may have a significant impact on the tools design.

As a consequence, it is necessary to allow the security analyst to work on the lower layers of the communication protocol stacks. To address this problem, we have implemented flexible and modular protocol stacks, allowing to deeply modify the behaviour of the protocols and easily manipulate the lower layers accessible by software.

This approach has been successful in the implementation of *Bluetooth Low Energy Man-in-the-Middle* attacks. Working at a lower level allowed us to avoid the limitations implied by the previously mentioned libraries: our *Man-in-the-middle* implementation does not require multiple operating systems or need to fully clone the GATT layer of the device to work, allowing us to redirect the packets without simulating an entire *BLE* device. As a result, the limitations of BTLE-Juice [Cauquil 2016] and GATTacker [Jasek 2016] tools linked to the use of high-level

libraries have been avoided, considerably simplifying the attack design.

5.3 Architecture overview

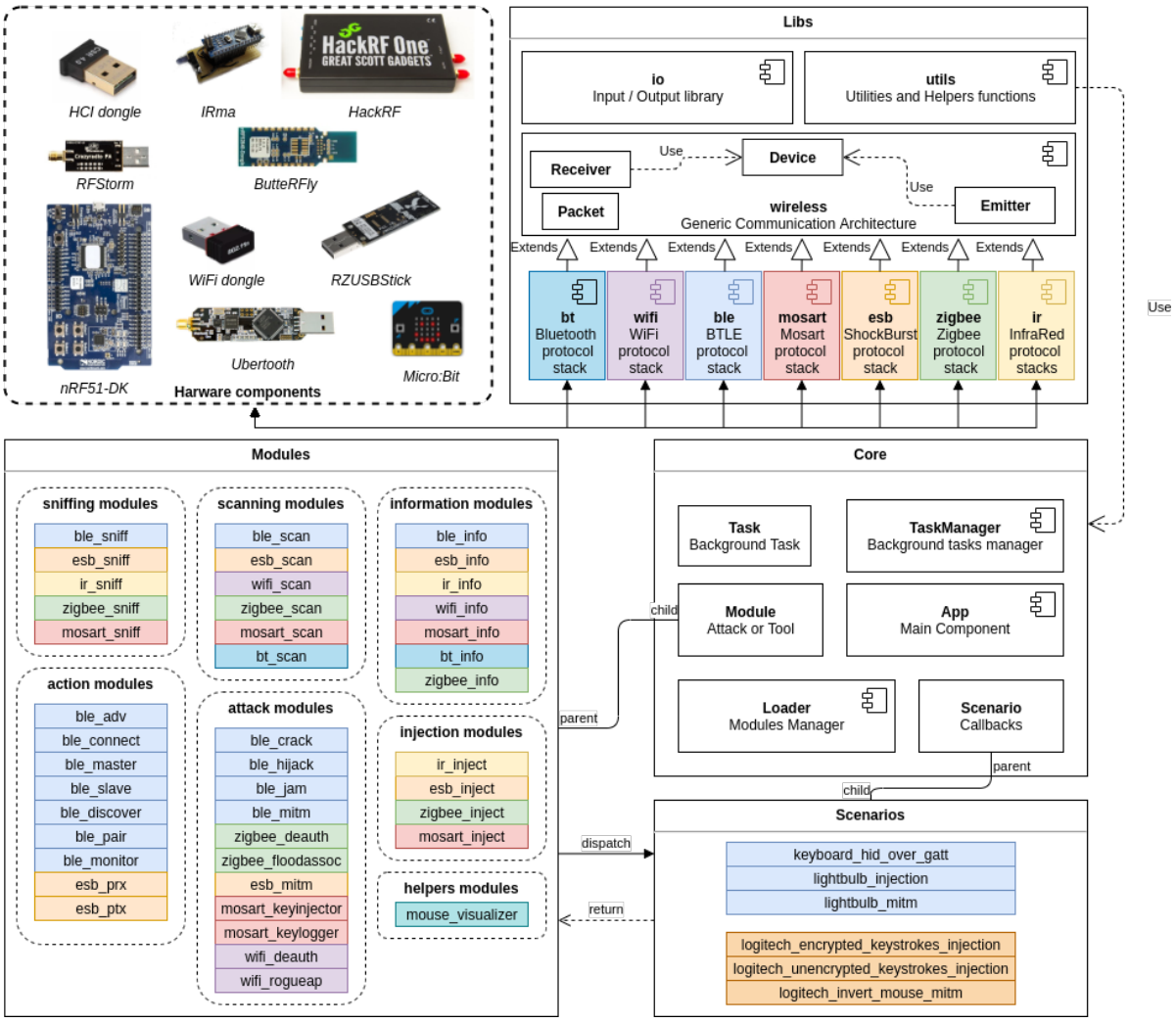

Figure 5.2: Global architecture of *Mirage* framework

This section describes the key features of our framework: the main software components and the generic communication architecture. Then, we focus on the modularity of our framework by presenting the concepts of Modules and Scenarios and introducing the chaining operator designed to execute modules sequentially in a pipe, as does the UNIX shell with commands.

5.3.1 Main software components

Mirage framework is composed of four main components, as shown in Figure 5.2:

a. **The core component ("*core*")**: this component includes the core mechanisms of our framework to load, configure and execute the modules, but also to manage the background tasks, signals and configuration files. It provides an unique entrypoint, allowing the *framework* to be used from a command line interface or directly from the shell environment via the execution of scripts.

b. **The internal libraries ("*libs*")**: this component is in charge of implementing the wireless protocol stacks, and provides a generic communication architecture to easily integrate new protocols. It provides some display and logging mechanisms and some utilities and helpers functions (e.g., modules and background tasks manipulation, time management).

c. **The attacks and tools ("*modules*")**: these software components, called "modules", are independent and implement the attacks and tools provided by the framework. They provide a specific service such as protocol sniffing or active attacks, and can be used independently or sequentially thanks to the chaining operator.

d. **The callbacks ("*scenarios*")**: some modules, such as *Man-in-the-Middle* attacks or *devices* simulation, implement some complex behaviours and provide a standardised API to quickly customize their execution. The *scenarios* are specialised classes composed of bindings methods providing simple APIs.

5.3.2 Generic communication architecture

One key feature of our framework is its generic communication architecture. Indeed, many of the wireless communication protocols commonly used by IoT devices have their own specificities, but several similar patterns can be extracted and have made it possible to design a generic architecture. Moreover, many different RF hardware components can be used to communicate with a given protocol, and our architecture must be flexible enough to integrate them easily.

Our design defines a generic way to handle multiple protocols in order to provide a unified API, but also allows the implementation of the specific behaviour of each protocol, while handling multiple RF devices and their key characteristics.

Mirage communication architecture is composed of three main software components, depicted in Figure 5.3. The **Device** class manages the interfacing with the various RF hardware components. As a result, several classes can inherit this abstract class and implement the main methods for sending or receiving a specific frame as a binary representation, check if the hardware component is connected and ready to use, and initialize the component. However, some devices provide additional features which can be implemented as independent methods. Their method names must be added to **sharedMethods**, a class attribute that is an array listing

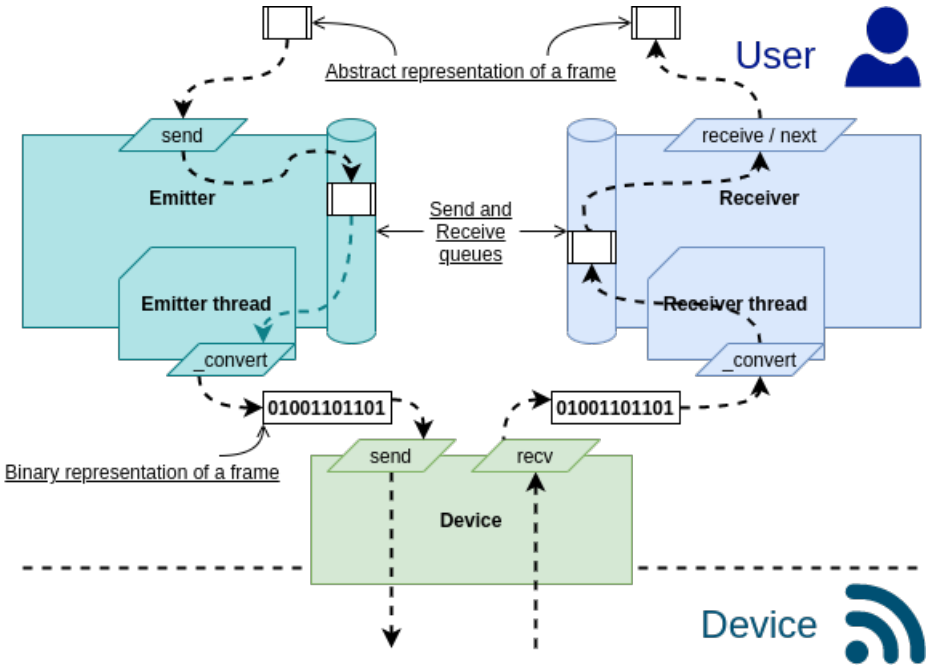

Figure 5.3: Generic communication architecture of *Mirage* framework

the specific features available. An instance attribute called **capabilities** can also be provided and defines some high level capabilities of the device, indicating the available functionalities. This class is not directly available in the modules, but the specific behaviour implemented as shared methods can be directly used.

Another class named **Receiver** can be defined. It is able to instantiate the right **Device** class according to the **interface** parameter, provided by the end user, and communicate directly with it. One main method is needed, called **_convert**. It converts a binary frame provided by the device into an abstract representation. This mechanism allows the programmer to provide a rich interface to user while manipulating frames (e.g., including dissectors, builders or converters). The class **Receiver** exposes two methods to get the received frames, and allows the user to register some callback functions which can be triggered at the reception of a specific frame, at the reception of a given number of frames or at each received frame and can be run in a background thread or in foreground.

The abstract class **Emitter** provides mechanisms similar to **Receiver** for emitting frames. It also includes a **_convert** method, and the child classes have to implement it to convert an abstract frame object into a binary representation. It exposes one main method, **send**, for sending frames from the modules.

The **Emitter** and **Receiver** classes include *First In First Out* data structures (the send and receive queues) to temporarily store the abstract representation of the frames. The **Emitter** class includes a background thread for converting these objects into binary frames thanks to the **_convert** method previously mentioned and transmits the resulting bytes array to the corresponding **Device**'s method

(**send**). Another thread is launched in background by the **Receiver** class and gets the received frames from **Device**'s **recv** method, converts them into their corresponding abstract representations and populates the receive queue.

Finally, it should be noted that end users cannot directly instantiate the **Device**'s classes. According to the design pattern called *Registry*, devices linked to a specific interface are instantiated only once by an **Emitter** or **Receiver** instance, and the same device can be used by multiple emitters or receivers. Methods corresponding to specific behaviours cannot be directly called by the users, but they are provided by the **Emitter** and **Receiver** classes which implement the design pattern called *Proxy*.

This design presents some interesting properties: 1) the most common actions (e.g. receiving and sending frames) are facilitated, and the API provided is the same for each protocol; 2) it allows specific features provided by the hardware components to be easily manipulated, without looking at their respective APIs; 3) the frames are manipulated as an abstract representation, allowing powerful mechanisms such as dissectors to be added.

5.3.3 Modules and scenarios

The modules are the key elements of our framework: they are used to implement the attacks and tools. They inherit and extend a class called **core.Module**, to quickly prototype and develop an offensive strategy.

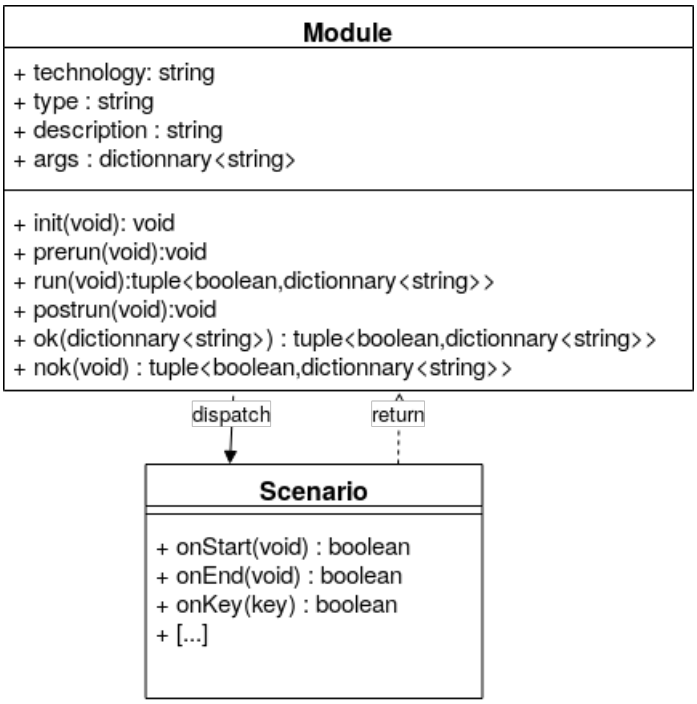

Figure 5.4: Architecture overview of a module

As shown in Figure 5.4, every module must implement the **init** method, allowing

the input parameters to be initialized as a dictionary and providing three main instance attributes, used by the core component to classify them:

- **technology**: this attribute indicates the wireless communication protocol targeted by the module. The corresponding emitters and receivers are automatically selected according to its value.

- **type**: this attribute is used to provide the type of tool implemented by the module. It allows the module to be easily classified.

- **description**: this attribute is a short string describing the role of the module.

The module behaviour can be customized by passing named arguments as inputs. These arguments are defined as the keys of a dictionary called **args**, and the corresponding values are used to provide default values.

The **init** method is called by the constructor when the module class is instantiated. The modules are dynamically loaded and instantiated by the class **Loader**, included in the **core** component. This class lists the files included in the *modules* sub-directory, instantiates the modules classes it founds and classifies them.

This mechanism is automatically executed at the beginning of the execution, allowing the user to focus on the development of the module.

Another main method, called **run**, must be implemented. This is the main method of a module, it contains the code that implements the attack or tool behaviour. This method is called at the beginning of the module execution. It returns a dictionary composed of a boolean value (indicating whether the module execution is successful or not) and a dictionary (providing the potential output parameters), which can be easily generated thanks to two helpers methods, called **ok** (if the execution is successful) and **nok** (if an error occurred during the module execution). If some specific actions need to be performed before or after the module execution, the developer can implement two additional methods named **prerun** and **postrun**.

Finally, some complex modules such as *Man-in-the-Middle* attacks can be highly customized by filling an input argument named "*SCENARIO*". It allows to provide a name corresponding to a child class of **core.Scenario**. It allows to easily customize the behaviour of a module by providing some callback methods, called if the module triggers the corresponding event. An event named "*onKey*" is automatically triggered if a key is pressed, which provides a basic user interface during execution of a module.

This design allows new attacks or tools to be easily prototyped or developed while ensuring a high level of modularity. Indeed, this approach forces the developers to follow the framework guidelines, which leads to a modular software environment. However, it is flexible enough to allow complex developments and the scenarios allow the developer to provide an elegant way to customize the behaviour of this module without changing the corresponding code.

5.3.4 Chaining operator

Another key feature has been added to our framework, to easily combine different modules to set up complex attack workflows: the chaining operator, called "*pipe*". Indeed, several attacks are composed of the same type of actions. For example, cloning a device or launching a fuzzing attack imply the use of similar actions, such as scanning the RF environment or connecting to the device. While it is still possible to use existing modules in a new module, a common need is to sequentially execute existing modules to compose customised attack workflows without writing a module. So, we have included a chaining operator inspired by the pipe operator, commonly used in UNIX environments.

The chaining operator included in our framework operates in a similar manner, allowing a data pipeline to be created between two modules. Every module can be customised by passing named parameters as inputs and can generate named parameters as outputs: as a result, our operator allows to sequentially execute two modules and propagate the outputs from the first module to the inputs of the second one, according to their name. If an output is not used by the next module in the pipeline, it will be stored to be used later by a next module included in the pipeline.

If a module included in the pipeline fails, sequential execution is interrupted. Several modules make use of the classes **Emitter** and/or **Receiver** mentioned previously. Therefore, if a given interface is used by a module, each subsequent module included in the pipeline using an **Emitter** or a **Receiver** based on the same interface does not need to re-instantiate these classes but automatically reuses the existing ones (according to the design pattern called *Registry*). This mechanism allows a complex attack workflow to be divided into simpler actions, leading to a powerful and modular approach. An example of such an execution is shown in Figure 5.5, illustrating the cloning of a *Bluetooth Low Energy* advertiser by combining a scanning module allowing to gather information of the targeted device and an advertising module allowing to transmit similar advertisements.

5.4 Protocols and modules

Several protocols commonly used by IoT devices have been integrated into our framework, and several different modules have been developed. In the following subsections, we present an overview of this work by describing the protocol stacks included in *Mirage* and their corresponding modules. Finally, we underline the development process to integrate a new protocol or add a new attack module.

5.4.1 Bluetooth and Bluetooth Low Energy

A lot of work has been done to integrate *Bluetooth* devices, especially *Bluetooth Low Energy* devices. Indeed, this technology is often used by connected objects because of its low power consumption and its massive integration in smartphones and tablets.

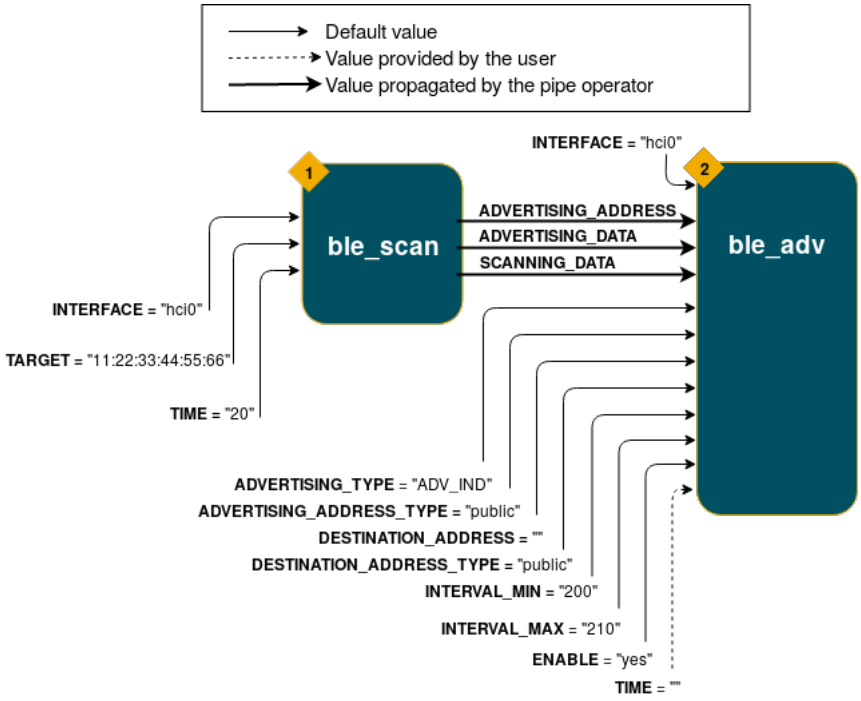

Figure 5.5: Example of sequential execution

A partial implementation of a *Bluetooth Classic* stack is included in *Mirage*. It implements a subset of *Bluetooth* layers (especially those used to inquiry and connect to devices) and works by communicating directly with the *Host Controller Interface* (HCI), without requiring the use of an external library. As a result, it makes it easy to use an HCI device such as *Bluetooth* dongles. Moreover, some providers provide an interesting feature in order to develop attack modules: they include in their hardware design some *vendor-specific* HCI frames allowing to change the unique address used to identify Bluetooth device, so-called *BD* address. As a result, this functionality has been included in *Mirage* and makes it easy to spoof a *BD* address and impersonate the identity of a targeted device. However, a lot of additional work is required to provide a complete stack due to the multiplicity of application layers supported by this technology.

Currently, two main modules related to this technology can be used in our framework: **bt_info** and **bt_scan**. The first module allows useful information about the specified interface to be displayed, while **bt_scan** allows to launch an inquiry scan and identify the visible *Bluetooth* devices in the RF environment.

Many software components have been implemented in our framework, to perform security audits on *Bluetooth Low Energy* devices. Indeed, the *Bluetooth Low Energy* stack inherits from the *Bluetooth Classic* one, allowing to reuse some interesting features (e.g., *BD* address spoofing) while adding an exhaustive *Bluetooth Low Energy* stack implementation. This stack can directly use the *Host Controller Interface* without requiring additional libraries, and it also provides several dissec-

tors and helpers functions to easily analyze and generate data from upper layers, such as the Attribute Protocol (*ATT*) and Generic Attribute Profile (*GATT*) layers. A complete *GATT* server has been implemented, allowing to easily simulate a *BLE* device using the *Peripheral* role.

Multiple sniffers have been implemented in *Mirage* to eavesdrop *Bluetooth Low Energy* communications: *Sniffle*, *nRF sniffer*, *Ubertooth* and *BTLEJack* are supported. We also added support for nRF52840 dongle embedding our custom *ButteRFly* firmware, allowing to perform InjectaBLE attack easily. An experimental *Software Defined Radio* architecture is also implemented, allowing to modulate and demodulate *BLE* packets using an HackRF One. All these hardware components are fully supported by *Mirage* and a unified API to control them is provided. Some additional features have been included in a custom version of *BTLEJack firmware*, allowing to easily sniff and selectively jam *advertisements*, both versions are fully supported by our framework. A PCAP writer is also provided, allowing *Bluetooth Low Energy* sniffed frames to be exported to a PCAP file.

Some of these devices can jam *BLE* communications, and *BTLEJack* or *ButteRFly* are also able to hijack such a communication. These features can be directly used by the framework and the various hijacking attacks can be combined with some active modules commonly used with HCI dongles, to highly customize the attack workflow.

Many different modules, have been included and can be used together to perform complex actions. **ble_info** lists the available interfaces. **ble_mitm**, **ble_hijack** and **ble_jam** allow active attacks to be performed, while **ble_sniff** can be used to passively eavesdrop the communications. Some modules are also provided in order to execute legitimate actions such as **ble_scan**, **ble_adv**, **ble_connect**, **ble_discover**, **ble_pair**, **ble_master** or **ble_slave**: all these modules expose some specific scenarios events, allowing to deeply customize their behaviour. **ble_crack** performs a CRACKLE attack [Ryan 2013b] allowing to break the vulnerable legacy pairing to retrieve the *Long Term Key*, while **ble_monitor** allows to monitor the **HCI** packets generated by a smartphone, facilitating the analysis of Android applications involving *BLE* communications.

5.4.2 Zigbee

Since two security frameworks targeting the *Zigbee* protocol (called *Killerbee* [Wright 2009] and *Secbee* [SEC 2015]) have been published in recent years, a partial implementation of the *Zigbee* stack developed from scratch is included in the framework as a proof of concept. It allows the user to interact with a *RZUSBStick* from *Atmel* using *Killerbee firmware*, but an experimental Software Defined Radio backend has also been implemented, allowing to send and receive 802.15.4 frames from a HackRF One. Six corresponding modules are provided, allowing to display information about an interface (**zigbee_info**), scan the RF environment to identify target networks (**zigbee_scan**), sniff *Zigbee* frames on a given channel (**zigbee_sniff**), inject *Zigbee* frames (**zigbee_inject**) or run a Denial of Service

attack by flooding a *Zigbee* router with *association* frames (**zigbee_floodassoc**) or performing a deauthentication attack (**zigbee_deauth**).

5.4.3 Enhanced ShockBurst and Mosart

Several relevant works focusing on *Enhanced ShockBurst* and *Mosart* protocols have been published in the recent years [Newlin 2016a,Schroeder 2010,LOG 2019]. These protocols are widely used by input devices such as mice or keyboards, and many vulnerabilities targeting this kind of hardware have recently been published, allowing to inject keystrokes or mouse-related frames.

As a result, *Mirage* includes a partial *Enhanced ShockBurst* stack and a complete *Mosart* stack, allowing the user to easily sniff and inject frames. It interacts with a *CrazyRadio PA* dongle or a *Logitech Unifying* dongle embedding the *RFStorm nRF Research firmware* developed and released by *Bastille Networks*, but an experimental SDR backend also allows to sniff and inject Enhanced ShockBurst packets using HackRF One. Several dissectors are provided to analyze mouse movements or keystrokes, and a *Ducky Script* interpreter has been added to facilitate attacks targeting keyboards, which is the *Domain Specific Language* defined to describe keystrokes injection in the *USB Rubber Ducky* hardware [RUB 2021]. Finally, a generic component (named **mouse_visualizer**) allows to generate a graphical view of mouse movements and can be combined with the previously mentioned sniffer.

Several attack modules are provided for both *Enhanced ShockBurst* and *Mosart* protocols. The interfaces can be enumerated thanks to **esb_info** and **mosart_info**, **esb_scan** and **mosart_scan** allow to scan the channels in order to identify devices, **esb_sniff** and **mosart_sniff** are used to sniff the frames on a given channel. Some active modules are also provided, such as **esb_ptx** and **esb_prx** that allow to mimick a specific Enhanced ShockBurst role, and can be customized using scenarios **logitech_encrypted_keystrokes_injection** and **logitech_unencrypted_keystrokes_injection** to perform keystrokes injections targeting Logitech keyboards and mice. Similarly, Mosart attacks can be performed using **mosart_keylogger** (allowing to sniff keystrokes over the air) and **mosart_keyinjector** (allowing to inject keystrokes). Both protocols also include a module dedicated to raw injection, **esb_inject** and **mosart_inject**.

5.4.4 Wifi

Wifi is a well-known technology in terms of offensive security, so we have decided to focus on other protocols more specific to IoT and to implement a minimal stack as a proof of concept. Currently, this stack allows to control management frames such as *deauthentication*, *disassociation* or *probe* frames. As a result, four main modules have been included in our framework: **wifi_info** provides some useful information about the interface used, **wifi_scan** allows to discover access points and stations, **wifi_deauth** allows to run a denial of service attacks by injecting *deauthentication*

or *disassociation* frames while **wifi_rogueap** simulates an access point (without accepting connections).

5.4.5 IR protocols

Infrared Radiations are widely used by manufacturers to control connected objects: it's probably one of the cheapest technology available for short range communications. As a result, we integrate many protocols based on this physical layer in *Mirage* (e.g., *RC5* or *Sony*).

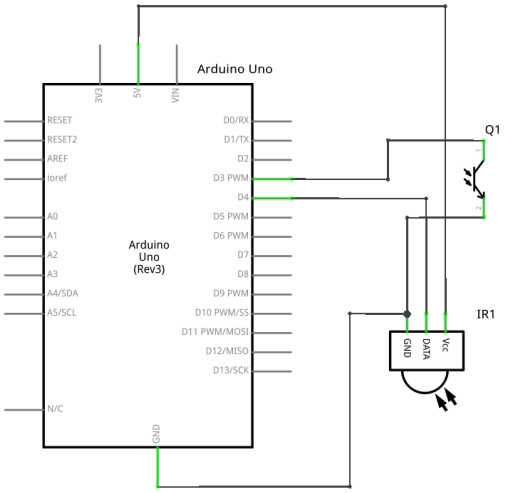

Figure 5.6: IRma hardware schematics

As far as we know, no specific hardware has been released targeting these IR protocols. As a consequence, we have designed and implemented a custom hardware component named *IRma* based on an Arduino, allowing easy sniffing and manipulation of *IR* frames. The corresponding *firmware* and schematics are open source and can be quickly reproduced and improved. The corresponding schematics are illustrated in figure 5.6.

Three main modules are provided by *Mirage* to manipulate these protocols: **ir_info** (displays useful information about an *IR* interface), **ir_sniff** (passively eavesdrop an *IR* frame) and **ir_inject** (injects an *IR* frame).

5.4.6 Adding new protocols and modules

One of the main advantages of our framework is that a new protocol can be easily added. First, the developer has to implement a new child class of **libs.wireless.Device**, allowing interaction with a given specific hardware. This class should provide only four main methods: **a) init** (initializing the hardware component), **b) isUp** (indicating if the hardware can be used), **c) recv** (allowing to receive frames) and **d) send** (allowing to transmit frames). As a result, developing a driver is straightforward. Some specific features can also be added by creating

a new method and appending its name to the array **sharedMethods**, allowing them to be called from a module environment.

At least one of the two child classes of **libs.wireless.Receiver** or **libs.wireless.Emitter** must be implemented. They initialize the device previously defined in their constructor and must implement the **__convert** method, allowing to convert a binary frame into an abstraction or a abstract representation of a frame into an array of bytes. As a result, the protocol is fully integrated and can be used from modules.

Developing an attack targeting this new protocol implies creating a new python file in the *modules* sub-directory. A child class of **core.Module** with the same name will be created into this file, allowing the **core.Loader** component to find this new module. The developer has to integrate an **init** method and provide the main necessary attributes (**technology**, **type**, **description** and **args**). Then, he can instantiate the previously created **Emitter** and **Receiver** and implement his attack by developing the corresponding **run** method.

5.5 Experimentations

In this section, we present two experiments illustrating the efficiency and relevancy of our framework, both to perform security audits and to build complex attack workflows. It also shows how simple it is, using this framework, to perform such security analyses, that have so far been quite complex to carry out, using heterogeneous and sometimes incompatible tools. It should be noted that many other experiments have been performed using our framework, such as security audits of connected objects or the evaluation of an Intrusion Detection System for IoT. The framework also allowed us to discover up to 20 new vulnerabilities targeting various commercial products.

5.5.1 Experiment 1: Auditing a Bluetooth Low Energy smart ligh-bulb

This subsection is dedicated to the presentation of a security audit of a smart connected bulb that we performed with our framework. The main objective of this experiment was first to reverse engineer the communication protocol of the bulb in order to evaluate its attack surface. The bulb is managed, through a *BLE* communication, by an Android application running on a smartphone.

This application allows the bulb user to choose its color, change its brightness, turn it on or off, and update its *firmware*.

5.5.1.1 Information gathering

First of all, it was necessary to use this Android application to register the bulb in the application and activate the various legitimate functionalities of the bulb. During these operations, the framework was used to analyze and understand the

following behaviours: a) change brightness, b) change temperature, c) switch on/off, d) change color and e) update *firmware*.

After this first analysis, the next step consists in identifying the list of the *ATT* server attributes, stored in the bulb, as well as their *GATT* abstractions, under the form of primary, secondary services and characteristics.

To do this, it was necessary to dump the *ATT* and *GATT* databases. The following modules were used:

a. **ble_scan** (in order to scan the environment to identify the *advertisements* of connected objects within radio range),

b. **ble_connect** (in order to establish a connection to a specific object),

c. **ble_discover** (in order to enumerate the services, characteristics and attributes associated to the *ATT*/*GATT* layers of a specific object).

The first step was to launch the **ble_scan** module, whose outputs were the following ones:

```
$ sudo mirage ble_scan
[INFO] Module ble_scan loaded !
[SUCCESS] HCI Device (hci0) successfully instanciated !

Devices found
| BD Address        | Name  | Company                |
|-------------------|-------|------------------------|
| XX:XX:XX:39:8E:07 | Salon | Texas Instruments Inc. |
```

This scan enabled to obtain three interesting information items: the BD address, the manufacturer of the transceiver as well as the name of the object. These information items were extracted from the *advertising* packets. In this study, the name of the smart bulb is "**Salon**", its BD address is XX:XX:XX:39:8E:07 and the manufacturer name is "**Texas Instruments Inc.**".

The next step was to perform a connection to the object and then dump the services, characteristics of the object (at the *GATT* level). The chaining operator integrated in *Mirage* allowed us to easily combine two existing modules (**ble_connect** and **ble_discover**) to obtain the structure of the high level protocol layers, i.e. *GATT* layer. It is also possible to export this information in a .cfg file, by setting the *GATT_FILE* parameter of the **ble_discover** module.

```
$ sudo mirage "ble_connect|ble_discover"
ble_connect1.TARGET=XX:XX:XX:39:8E:07
ble_discover2.GATT_FILE=/tmp/gatt.cfg
```

The output of the module indicates that three services (quite common for most connected object) are available on the smart bulb: **a) Generic Access** (*handles*

0x0001 to 0x000b), **b) Generic Attribute** (*handles* 0x000c to 0x000f), **c) Device Information** (*handles* 0x0010 to 0x001e).

Three other services, specific to the bulb, are also available from *handles* 0x001f to 0x002f, 0x0030 to 0x0039 and 0x003a to 0xFFFF.

Furthermore, two interesting features are associated with the first service: **DataTransmit** (*handle* 0x0020) and **DataReceive** (*handle* 0x0023).

5.5.1.2 Reverse-engineering of the communication protocol

In order to accurately identify the behaviour of the object, a *Man-In-The-Middle* attack was performed, while the different functionalities of the bulb were activated thanks to the Android application on the smartphone. The *Man-In-The-Middle* attack allowed us to analyze the traffic corresponding to these functionalities. At first, since no specific scenario was loaded in the *Man-In-The-Middle* module, the default behaviour was applied (redirection and logging of the packets).

```
$ sudo mirage ble_mitm TARGET=XX:XX:XX:39:8E:07 SHOW_SCANNING=no
MITM_STRATEGY=preconnect
```

This attack allowed us to identify the format of the command messages. They are triggered by a **Write Request** to the *handle* of value 0x0021 (which corresponds to the **DataTransmit** characteristics, identified during the previous step of the analysis).

The messages format is as follows:

0x55	identifier – 1 byte	parameter	0x0d 0x0a

Every action is performed using a specific identifier (e.g., 0x10 for switching on or off, 0x13 for color modification ...) and the corresponding parameter. As an example, the messages intended to modify the color include the identifier **0x13**, followed by three bytes corresponding to the hexadecimal *RGB* code of the required color, as shown in Table 5.1.

Table 5.1: Messages format related to color modification

Color modification (Red)	55 13 ff 00 00 0d 0a
Color modification (Green)	55 13 00 ff 00 0d 0a
Color modification (Blue)	55 13 00 00 ff 0d 0a

To confirm our assumptions, we performed a connection to the bulb and executed the **ble_master** module, allowing us to easily replay these messages. Then, another *Man-In-The-Middle* attack was also performed, in which our framework was able to modify on the fly the different commands sent by the smartphone application to the bulb (for instance, the color Red and Green were exchanged, as well as the switch on/off action).

5.5.1.3 Obtaining a *firmware* dump

The last step of our security audit was to analyze the *firmware* update procedure of the bulb. Indeed, when connecting the smartphone application, a dialog box proposes to update the *firmware* of the bulb *over the air*. To analyze this update process, we used the **ble_sniff** module:

```
 $ sudo mirage ble_sniff CHANNEL=37 SNIFFING_MODE=newConnections
INTERFACE=microbit0
```

The data dumped during this sniffing attack allowed us to identify eight different steps of the update process (mostly *Read Requests* and *Write Commands* on different *handles*).

Once these messages are exchanged, the Master starts to write, by means of **Write Commands**, in the *handle 0x0040*, values formatted as follows:

```
000017deffff0500007c42424242ffffffff
0100ffffffffffffffffffffffffffffffff
02000001020304050607080 90a0bffffffff
[...]
```

By analyzing the format of these messages, we were able to understand that the first two bytes represent a counter, followed by the contents of the *firmware*, sent by 16 bytes data blocks. The attack scenario **slave_lightbulb** was then built, (based on the **ble_slave** module) in order to dump the whole *firmware* in the "firmware.bin" file. Creating an identical clone of the bulb can offer many advantages. First, it enables to simulate the behaviour of the object to be audited. It may also be used to perform a denial of service attack of the legitimate object.

Such a strategy could easily be instantiated in our framework, thanks to the following chained execution: **a) ble_scan** (dumping of *advertisement* data),**b) ble_connect** (connection to the bulb), **c) ble_discover** (dumping of GATT services and characteristics in a .cfg file), **d) ble_adv** (sending *advertisements*) and **e) ble_slave** (creation of a *BLE Slave* using the same GATT data and implementing the *slave_lightbulb* scenario). This example illustrates the relevance of introducing a chaining operator, which allows complex attack workflow to be designed without writing a single line of code.

```
$ sudo mirage "ble_scan|ble_connect|ble_discover|ble_adv|ble_slave"
ble_scan1.TARGET=XX:XX:XX:39:8E:07
ble_discover3.GATT_FILE=/tmp/gatt.cfg ble_adv4.INTERFACE=hci1
ble_slave5.SCENARIO=slave_lightbulb
```

After the information collection phase and the creation of the *BLE* slave, the latter was executed and produced the following output:

```
[...]
[SUCCESS] HCI Device (hci1) successfully instanciated !
[INFO] Importing GATT layer datas from /tmp/gatt.cfg ...
[INFO] Scenario loaded !
[INFO] Updating connection handle : 68
[INFO] Master connected : 73:5E:A2:21:C7:9D
[...]
[INFO] Sending notification (1)...
[INFO] Sending notification (2)...
[INFO] Starting firmware recuperation ...
[INFO] Writing 0...[INFO] Write Command :  handle = 0x40 /value =
000017deffff0500007c42424242ffffffff[...]
```

After this operation, the *firmware* was dumped and available in the "/tmp/-firmware.bin" file.

5.5.2 Experiment 2: Attacking a randomized keyboard

This subsection is dedicated to the presentation of a complex attack workflow, aiming at exploiting a Logitech wireless mouse to extract the credentials entered in a banking website. The attack is quite complex as it aims to break a security measure integrated by the website: indeed, it provides a visual keyboard to enter the password, which is randomized every time the page is refreshed. This behaviour is illustrated in figure 5.7.

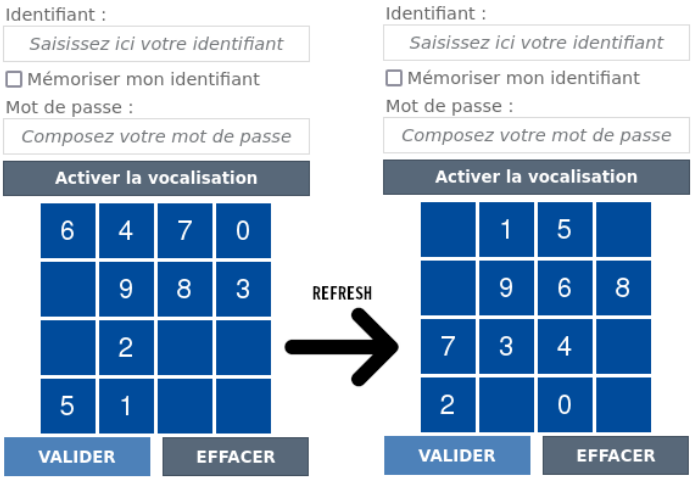

Figure 5.7: Example of randomized keyboard on a banking website

We illustrate here how our framework can be used to perform complex attack scenarios, combining several exploits to build an advanced offensive strategy. The attacker is present within the radio range and targets a victim using a wireless Logitech mouse vulnerable to MouseJack keystrokes injection attack while connecting to its bank account.

5.5.2.1 Information gathering

Logitech Unifying is based on Enhanced ShockBurst: Mirage provides a set of module that can be used to interact with Enhanced ShockBurst-based protocols. First, the attacker performs a scanning operation using **esb_scan** module to discover both the address of the targeted mouse and the channel in use.

```
$ sudo mirage esb_scan
[INFO] Module esb_scan loaded !
```

Address	Channels	Protocol
E8:46:F9:2F:A4	74	logitech

```
[INFO] Mirage process terminated !
```

5.5.2.2 Capturing the mouse movement

Once the address has been identified, the attacker can combine **esb_sniff** and **mouse_visualizer** to sniff the mouse traffic while the victim is entering its password on the wireless keyboard displayed on the banking website. The **mouse_visualizer** module then extracts the speed vector embedded in the captured mouse packets to generate an animated image representing the mouse movement.

```
$ sudo mirage "esb_sniff|mouse_visualizer"
esb_sniff1.TARGET=E8:46:F9:2F:A4 esb_sniff1.TIME=5
esb_sniff1.MOUSE_FILE=/tmp/mouse.capture
mouse_visualizer2.GIF_FILE=/tmp/mouse.gif
[INFO] Module esb_sniff loaded !
[INFO] Module mouse_visualizer loaded !
[INFO] Sniffing mode enabled !
[INFO] Channels: 0-99
[INFO] Looking for an active channel for E8:46:F9:2F:A4...
[SUCCESS] Channel found: 8
[PACKET] [ CH:8 ] << ESB - Logitech Mouse Packet | x=0 | y=-1 >>
[PACKET] [ CH:8 ] << ESB - Logitech Mouse Packet | x=0 | y=-1 >>
[PACKET] [ CH:8 ] << ESB - Logitech Mouse Packet | x=1 | y=-2 >>
[...]
[PACKET] [ CH:8 ] << ESB - Logitech Mouse Packet | x=-1 | y=0 >>
[PACKET] [ CH:8 ] << ESB - Logitech Mouse Packet | x=-1 | y=0 >>
[SUCCESS] Sniffed mice datas are saved as /tmp/mouse.capture (CFG
file format)
[INFO] Importing mice datas from /tmp/mouse.capture ...
[INFO] Mirage process terminated !
```

It produces the capture illustrated in figure 5.8, where the blue line indicates the mouse movement while the red dots indicated the mouse clicks.

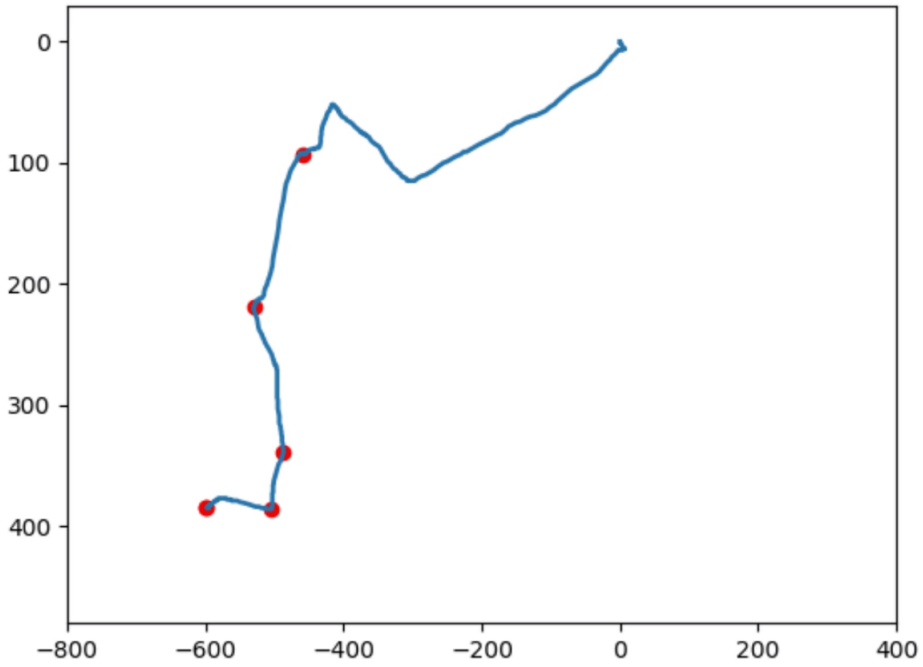

Figure 5.8: Mouse movements and actions extracted from eavesdropped traffic

5.5.2.3 Injecting arbitrary keystrokes

The last step performed by the attacker aims at taking a screenshot of the victim's browser to get the current keyboard layout. To do so, the attacker can exploit a MouseJack [Newlin 2016a] vulnerability allowing to inject unencrypted keystrokes to the dongle. The attack is implemented in a Mirage scenario named **logitech_unencrypted_keystrokes_injection** and can be easily customized by providing a DuckyScript describing the keys sequence to inject.

First, the attacker sets up a netcat listening on port 1324 on a server he owns (in our case, we will use *malicious-server.com*) :

```
$ sudo nc -l -p 1234 > /tmp/victim.png < /dev/null
```

The following DuckyScript, once injected on a Linux machine, opens a terminal, takes a screenshot and sends it to the malicious server. Of course, this script could be adapted to match another operating system:

```
CTRL ALT T
DELAY 500
STRING gnome-screenshot -f /tmp/out.png
```

Figure 5.9: Cropped screenshot indicating the randomized keyboard layout

```
DELAY 500
ENTER
STRING nc malicious-server.com 1234 < /tmp/out.png
DELAY 500
ENTER
DELAY 500
ALT F4
```

The DuckyScript can then be injected using the following Mirage command:

```
$ sudo mirage esb_ptx
SCENARIO=logitech_unencrypted_keystrokes_injection
DUCKYSCRIPT=/tmp/duckyscript TARGET=E8:46:F9:2F:A4
```

Once the injection has been performed, the attacker has received the screenshot and can extract the randomized keyboard layout, illustrated in figure 5.9.

5.5.2.4 Retrieving the credentials

Once the mouse movements have been collected and the screenshot has been received, it is trivial for the attacker to retrieve the credentials entered by the victim by combining the acquired data. Figure 5.10 illustrates how this combination may allow to infer the login (in our example, *1234*) and the password (*4382*).

Figure 5.10: Retrieving the credentials by combining mouse movements and malicious screenshot

5.6 Conclusion

In this chapter, we presented a new security audit and penetration testing framework called *Mirage* dedicated to IoT devices, focusing on the analysis of widely used wireless communication protocols. It offers a flexible software environment for developing new tools and attacks thanks to a modular architecture and the introduction of a chaining operator. We also described a generic communication architecture that allows new protocols to be easily integrated and provides a unified API for multiple technologies. Then, we highlighted multiple protocols and modules already included in *Mirage*, demonstrating that several existing attacks could be easily integrated into our framework and sometimes improved, thanks to a low level architecture. Finally, we described two experiments to demonstrate its usability and efficiency, both to perform security audits or complex attack workflows. It should also be noted that our framework has been successfully used to evaluate an Intrusion Detection System dedicated to IoT. It has been useful in easily automating the generation of intrusion attempts in a smart-homes context, leading to an efficient evaluation process.

The framework is publically available as an open-source project and can be found here: `https://github.com/RCayre/mirage`. The documentation of the project can be found at the following address: `https://homepages.laas.fr/rcayre/mirage-documentation`.

It is actively maintained and regularly extended: we also plan to integrate new protocols such as ZWave, ANT+ or LoRaWAN, and add relevant modules to analyze a complete wireless environment such as for network topology inference. We also consider that such a framework is also relevant from a defensive perspective, especially by exploiting passive modules allowing to efficiently monitor wireless environments.

Two scientific articles describing this contribution have been published, both in national and international conferences:

- Romain Cayre, Jonathan Roux, Eric Alata, Vincent Nicomette, Guillaume Auriol. **Mirage : un framework offensif pour l'audit du Bluetooth Low Energy.** *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2019)*, Jun 2019, Rennes, France. **[FR]** [Cayre 2019b]
- Romain Cayre, Vincent Nicomette, Guillaume Auriol, Eric Alata, Mohamed Kaâniche, et al.. **Mirage: towards a Metasploit-like framework for IoT.** *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, Oct 2019, Berlin, Germany. **[EN]** [Cayre 2019a]

Part III

Intrusion detection and prevention

Chapter 6

OASIS, an Intrusion Detection System embedded in Bluetooth Low Energy controllers

Contents

In this chapter, we focus on the security of one of the most commonly used IoT protocols, BLE, and specifically on the detection of low level attacks. We explore the feasibility of building an embedded Intrusion Detection System for this protocol, as well as detection heuristics suited for these attacks, and to include them inside the BLE controllers. This embedded approach solves multiple technical challenges related to the protocol design, and can be easily deployed on multiple BLE-enabled devices in the wild. We also introduce Oasis, a generic framework allowing to patch various BLE controllers to include these detection heuristics. We describe its modular architecture, how we successfully implemented it on five widely used BLE chips embedding heterogeneous stacks, and how we used it to detect up to six critical low level attacks.

6.1 Motivations

In the past few years, the Bluetooth Low Energy protocol has been widely used in a variety of use cases, due to its massive deployment, low complexity and versatility. As a consequence, the security of this protocol has become a major concern. Multiple critical vulnerabilities [Armis 2018, Garbelini 2020, Armis 2017, Antonioli 2019], have been discovered recently, illustrating the growing interest for this technology. Some of these vulnerabilities [Armis 2018, Garbelini 2020, Armis 2017] are related to the stack implementation and can potentially be patched by the manufacturers, while others [Antonioli 2019, Cauquil 2018, Jasek 2016, Cauquil 2016] are related to the protocol design itself and cannot be easily fixed without modifying the specification [Blu 2019].

This situation highlights the need to develop defensive measures, especially Intrusion Detection Systems (IDS), to detect this kind of wireless attacks. However, building such IDS is a major challenge. Indeed, the BLE protocol design introduces many technical constraints which are difficult to solve. Firstly, the protocol is difficult to monitor by an external probe, mainly because it uses a channel hopping algorithm during connections. As a consequence, a BLE connection may use any of the 40 channels in the 2.4GHz ISM band, requiring an IDS to continuously monitor the wide frequency band in order to comprehensively analyze the traffic, which increases the cost and complexity of such a system. Moreover, the wireless nature of the protocol introduces some issues related to the potential difference of perception between the external probe and the nodes themselves: unlike a wired medium of communication, many factors can impact on the completeness and representativity of the monitored traffic, such as the probe's position in the environment or its sensitivity. As the protocol is also mainly used to establish peer to peer communications, it's not possible to easily monitor the traffic from a central node. Finally, many BLE devices are intended for mobile use, resulting in a dynamic environment in which it is difficult to identify whether the presence of a given node is legitimate or not.

In this context, a relevant approach would be to build an IDS embedded in the devices, allowing to monitor the BLE traffic locally to detect attacks. However, designing such a system is challenging for many reasons: first, a typical BLE stack is splitted into two parts: the Host, which manages the application layers of the protocol, and the Controller, which is in charge of the lower layers. These two components can be integrated into different chips and communicate with each other using a standardized interface named Host Controller Interface (HCI).

Instrumenting the Host side of the stack to detect attacks could be interesting because it is generally straightforward to implement. However, this strategy suffers from serious limitations. First, most low level traffic is hidden from the Host by design: this is problematic because many existing attacks [Cauquil 2018] abuse the lower layers of the stack and cannot be detected effectively by such a system. Another problem is related to the existence of non standard implementations of the stack, that may expose a proprietary API to interact with the Controller instead of

the HCI API: this situation cannot be ignored as it is common in many IoT devices.

According to these constraints, instrumenting the Controller is probably the most powerful way to detect BLE attacks, as it allows to monitor low level traffic to identify low level attacks while being able to detect attacks targeting upper layers. This strategy takes advantage relevant features that the controller can access, such as the RSSI or CRC validity, and is also suited to carry out some low level defensive actions to prevent a detected attack. However, instrumenting the Controller is non trivial. First, controller implementations are generally proprietary and not documented: the only way to understand and instrument their internals is to manually reverse engineer the corresponding firmware, which is a time-consuming, tedious and error-prone process. These controllers are also implemented on many different chips based on heterogeneous architectures. They are also difficult to instrument, as manufacturers generally do not provide any easy way to patch them to include defensive code. Finally, a protocol stack implementation is time sensitive by design, resulting in an optimized code which can be difficult to modify and improve.

In this chapter, we propose a novel approach to design an IDS for BLE attacks by embedding detection mechanisms directly into the BLE controllers, based on the identification of relevant features at the controller level that can be used to successfully characterize the occurrence of attacks. To the best of our knowledge, this is the first solution exploring this direction. We assess the relevance of our detection approach by means of 6 of the main low level structural attacks targeting BLE protocol. We designed 6 intrusion detection modules for these attacks, embedded them into various BLE controllers and successfully detected these attacks with very good false positive and negative rates. In particular, we are the first to propose a detection strategy for critical attacks such as BTLEJack [Cauquil 2018]. We also provide a generic and modular framework named *Oasis*, dedicated to the development of these intrusion detection modules. We have implemented it on several different boards embedding heterogeneous controllers' architectures, including commercial products such as smartphones and IoT devices. To do so, we reverse engineered the internals of three major BLE stacks embedded in various chips and developed a set of automatic reverse engineering tools, allowing to automatically identify the main functions and memory areas required to implement the *Oasis* framework.

6.2 Detection of low level BLE attacks

In this section, we describe six major low level attacks related to the protocol design and discuss how these attacks can be characterized and detected using appropriate features at the BLE controller level. Finally, we list the requirements needed to embed detection heuristics in the controller and how they motivated the development of our detection framework *Oasis*.

6.2.1 Detection strategies

In this subsection, we briefly present six major low level attacks targeting the BLE protocol, and present our detection strategy for each one. We focus on attacks that are related to the protocol design itself and cannot be easily fixed without changing the specification.

6.2.1.1 GATTacker and BTLEJuice: Man-in-the-Middle attacks

Attacks presentation: Two main strategies have been developed in order to perform a Man-in-the-Middle attack targeting a BLE connection. They are both based on a spoofing strategy targeting the advertisements transmitted by a *Peripheral* before the initiation of the connection, even if they adopt different approaches to perform this operation. *GATTacker* [Jasek 2016] exploits the fact that a *Central* node trying to initiate a connection with a *Peripheral* node transmits its *Connection Request* just after the reception of an advertisement packet transmitted by the *Peripheral*. As a consequence, the *GATTacker* approach advertises spoofed advertisements packets faster than the legitimate *Peripheral* to maximize the probability of receiving the *Connection Request* before the legitimate device. Once the *Central* is connected to the attacker fake *Peripheral*, the attacker initiates a connection using a second dongle with the legitimate *Peripheral* to establish the Man-in-the-Middle attack. *BTLEJuice* [Cauquil 2016] approach is based on the fact that a *Peripheral* stops transmitting advertisements if it is involved in a connection: the attacker uses a first dongle to establish a connection with the target *Peripheral*, forcing him to stop transmitting its advertisements. Then, the attacker uses a second dongle to expose a spoofed *Peripheral*, waiting for a *Central* to initiate a connection.

Detection strategies In advertising mode, a device which is able to transmit advertisements follows a hopping pattern along the three advertising channels and broadcasts its frames. The time between each advertising event (an advertising event being defined by a complete cycle of hopping along the three channels) is defined by the *advInterval*, which is an integer chosen by the device and multiple of 0.625ms, and by a random delay named *advDelay* between 0 and 10ms, which is automatically generated by the *Link Layer* for each advertising event. After every transmission, the advertising device briefly listens to the channel, for possible *Scan Request* or *Connection Request* transmitted by another device.

Our strategy to detect *GATTacker* is based on the idea that a *Peripheral* transmitting advertisements must follow this specific channel hopping pattern. If an attacker is transmitting advertisements simultaneously, a node monitoring the advertising channels as a *Scanner* or a *Central* should receive both the legitimate and the spoofed advertisements and be able to detect that the received packets are not compliant with the protocol specification, indicating the presence of a malicious node.

In order to detect this situation, we first estimate the *advInterval*, in absence of attacks, for each device transmitting advertisements. This estimation is based

on a sliding window which is filled with the duration between two consecutive advertisements from the same device received on the same channel: once the window has been completely filled, the minimum value in the window is considered as our *advInterval* estimate (keeping the minimum value allows us to minimize the impact of the random *advDelay* parameter). Then we set a detection threshold to the *advInterval* value minus the maximum *advDelay* value, which represents the worst legitimate case. Each time a new packet is received, a new estimate is calculated with this method, and if the calculated value is lower than the detection threshold, an alert is raised indicating the presence of a malicious node.

The *BTLEJuice* attack is more difficult to detect because a node monitoring the advertising channel has no guarantee to observe the *Connection Request* transmitted by the attacker. Therefore, we choose to adopt another strategy, allowing the target *Peripheral* to detect its own spoofing by an attacker. When a connection is established, the *Peripheral* simultaneously maintains the connection and also scans the advertisements. During this scan operation, the *Peripheral* checks if its own address is included in the advertisements packets. If this situation is detected, it means that an attacker is trying to perform a *BTLEJuice* attack and an alert is raised.

While these strategies provide effective detection, they have some limitations that should be highlighted. The *GATTacker* detection must be able to correctly estimate the legitimate *advInterval* before it can detect an attacker node: as a consequence, the detection requires that the monitoring device has been able to fill its sliding window to estimate the interval before the attack begins. This learning phase could probably be reduced or removed in a controlled environment, where the monitoring devices could use pre-defined *advInterval* values for each monitored *Peripheral*. Likewise, the *BTLEJuice* detection requires that the target *Peripheral* is able to simultaneously maintain a connection and scan the advertisements. Most controllers should be able to perform these operations simultaneously, but it may be problematic for some specific controllers that only implement a subset of *BLE* roles (as an example, some *BLE stacks* from Nordic SemiConductors only implement the *Peripheral* role and cannot perform scanning).

6.2.1.2 Continuous Jamming attack

Attack presentation: A common security issue observed when using a wireless communication protocol is related to the fact that the medium is open by design, allowing an attacker to impact the availability by attacking the link itself. One of the simplest strategies for performing such a *Denial of Service* attack is to transmit a *jamming* signal, that interferes with legitimate traffic and introduces integrity errors, resulting in invalid CRCs that force legitimate nodes to drop the corrupted frames.

While multiple jamming strategies [Bräuer 2016, Shintani 2020] are available, we focus here on a simple jammer design aiming at attacking the advertising channels by transmitting a strong signal on the corresponding frequencies. Advertising

channels are indeed an interesting target for an attacker, as they are both used to indicate the presence of devices and to initiate connections: as a result, a continuous jammer targeting these channels can have a serious impact on the nodes present in the environment by disrupting any attempt to initiate a connection or scan. This offensive strategy is also interesting from a cost perspective, because it doesn't require following the channel hopping pattern of a connection and targets pre-defined channels, considerably reducing both the cost and complexity of the jammer.

Detection strategy: An obvious solution to detect such an attack would be to analyze the physical layer to detect the jamming signal. However, this solution cannot be easily used by an Intrusion Detection System embedded in legitimate nodes because most existing BLE controllers do not allow a direct access to the physical layer. It would also require complex analysis because the attacker has very few constraints regarding the jammer design, and is not forced to be compliant with the protocol specification. Another way to detect this attack is to monitor its consequences at the *Link Layer* level: indeed, a successful jamming attack causes packet corruptions, resulting in invalid CRCs. Since any device compliant with the specification is able to check the CRC validity, our detection strategy is based on this verification: every second, the nodes implementing the *Scanner* or the *Central* roles (i.e. being able to scan advertisements) compute the number of packets received without integrity corruption per second on a given channel, the frames with an invalid CRC being ignored. If this value is equal to zero during more than a predefined threshold number of computations (set to 5 in our experiments) for a given channel, we consider that the channel is being jammed and an alert is raised.

Note that this strategy detects an environment without any traffic as a false positive: although this situation rarely occurs, it should be taken into account from a defensive perspective. One way to distinguish this legitimate situation from an attack would be to estimate both the number of corrupted and non-corrupted packets per second, and to raise the alert only if the number of non-corrupted packets per second is equal to zero while the number of corrupted packets per second is not equal to zero. However, this variant could lead to false negatives if the attacker jams the preamble of the packets, causing the embedded IDS to not receive them at all and not raise any alerts. If the environment can be controlled, inserting a non-connectable *Advertiser* device could be a good compromise, allowing the first strategy to be applied without the risk of false positives.

6.2.1.3 BTLEJack attack

Attack presentation: Another attack that may have a significant impact on availability is named *BTLEJack* [Cauquil 2018]. This attack, presented by D. Cauquil, is a jamming approach allowing to jam an established connection or to hijack the Master role under certain circumstances. The attacker first synchronizes with an established connection, then transmits a jamming signal when the *Slave* sends a reply to the *Master* at each connection event. The attack exploits a counter

mechanism to detect a link loss by incrementing the counter value every missed or invalid packet. When this counter reaches a predefined threshold, the *Master* considers the connection as lost and exits, allowing the attacker to interrupt it or, in the worst case, to hijack the *Master* role if the *Slave* does not disconnect immediately after the *Master* disconnection.

Detection strategy: From a *Central* node perspective, detecting this attack can be performed easily: unlike a normal connection loss, the *Central* receives frames including an invalid CRC on multiple consecutive connection events during an attack, instead of receiving no packets in a legitimate scenario. This situation has a very low probability of occurrence in a legitimate situation, as the channel hopping algorithm ensures the use of multiple channels distributed along the wide ISM band. The detection strategy consists in raising an alert when the consecutive received frames with integrity corruption counter reaches the value of the connection counter minus one.

6.2.1.4 KNOB attack

Attack presentation: The *KNOB* attack, presented by D. Antonioli et al [Antonioli 2019], allows an attacker performing a Man-in-the-Middle attack to inject a low entropy value during the pairing process. Indeed, the pairing process includes a protocol for entropy negotiation, allowing each involved device to indicate how many entropy bytes can be used during key generation. As a result, an attacker can perform an entropy downgrade attack, by setting this number of bytes to 7 bytes instead of 16 in the case of BLE. As a consequence, the key can easily be bruteforced, which compromises the security of the future communications between the involved devices.

Detection strategy: This attack can be detected by both a *Central* or a *Peripheral* using a simple passive strategy. When a Pairing Request (i.e. the packet type used to negotiate the entropy value) is received, the entropy value field is extracted from the packet payload and an alert is raised if the value is less than 10 bytes of entropy. Even if the protocol technically allows such a lower value to be used legitimately, considering that a device should not be allowed to use an entropy value low enough to allow a bruteforce seems a reasonable assumption from a security perspective. The *Pairing Request* being transmitted by the *Controller* to the *Host*, let us note that this detection strategy could be implemented at the *Host* level.

6.2.1.5 InjectaBLE attack

Attack presentation: The last attack we focus on is our new injection attack targeting BLE communications named *InjectaBLE*, that we presented in chapter 4. This attack abuses a feature that allows two devices communicating together to

compensate potential clock drift: when a *Peripheral* enters reception mode to receive a packet from the *Central* during a connection, it listens during a short window (named *window widening*) after and before the theoretical instant, allowing an attacker to exploit a race condition and inject a malicious packet before the legitimate *Central* node. This attack is critical, especially if the connection is not encrypted, as it allows any role of the communication to be hijacked or a Man-in-the-Middle to be performed by injecting carefully chosen frames.

Detection strategy: This attack can be detected by the targeted *Peripheral* itself by monitoring the interval between two consecutive received packets. We are able to detect if a packet is injected by comparing the last interval to the legitimate connection interval: if the interval is less than the theoretical interval minus an empirically estimated threshold, we consider the frame as malicious and raise an alert. This strategy may lead to false positives if the devices use clocks with an important drift.

6.2.2 Detection requirements

The previously mentioned detection strategies give us an overview of the requirements needed to implement detection heuristics at the controller level.

To embed efficient detection mechanisms at this level, we need to instrument:

- **the packet reception mechanisms:** most of our detection strategies require access to link layer packets, especially the received ones. Both advertisements and data packets must be collected, with some relevant metadatas such as CRC validity or RSSI.
- **the time management mechanisms:** we need to collect timestamps as accurately as possible in order to estimate intervals between packets, as expected by *GATTacker* or *InjectaBLE* detection modules. We also need to be able to run code regularly, independently of packet reception, for example to compute the number of valid packets per second (e.g., Continuous Jamming detection).
- **the connection and device management mechanisms:** some of our detection strategies need to use some data handled by the controller related to connections (e.g., the connection interval for *InjectaBLE* detection) or to the local device (e.g. the BD address for *BTLEJuice* detection). As some of our detection strategies are restricted to specific roles, we also need to be able to know in real time what role our instrumented device is currently using, and trigger the execution of certain code when an event occurs (e.g., connection initiation).
- **the high level operations:** we need to be able to instrument the high level operations of the controller, for example to trigger the scan mode once a connection is established for *BTLEJuice* detection.

Obviously, the implementation of these mechanisms can be very heterogeneous depending on the stack used: to avoid developing multiple detection modules depending on the stack, this motivates the development of a generic framework with wrappers allowing to instrument the stacks and exposing a homogeneous API.

6.3 Framework design

In this section, we describe the design of *Oasis*, a generic and modular framework allowing to patch controllers in order to embed intrusion detection mechanisms. We first introduce the main guidelines that guided its development. Then, we describe its global architecture and the structure of the generated code that is used to patch the controllers. Finally, we briefly describe the implementations of its main components and a typical use case of this framework.

6.3.1 Main guidelines

In subsection 6.2.2, we highlighted the minimal requirements needed to embed the previously mentioned detection strategies into controllers. However, many controller implementations are proprietary and not documented: the direct consequence of this situation is that it is not possible to instrument the source code. It is therefore necessary to find a way to interact with the BLE stack by patching the firmware binary while running our own code without disrupting the legitimate behavior of the stack.

This situation motivated the development of a framework generating a detection software, that must be able to run independently from the controller, without altering its normal behavior. This implies carefully choosing the hooked functions to avoid adding delays in time sensitive components, but also finding a way to inject our code and data in memory without impacting the controller execution. Our framework must also be **user-friendly**, i.e., allows a developer to easily implement a new detection module without requiring a deep understanding of the underlying controller architecture. To this end, it provides an user-friendly environment to allocate memory, collect features or react to a specific event.

The controllers are also heterogeneous, and cannot be instrumented without writing specific code for each of them. However, a detection module implements a logic which is independent of the underlying implementation, and the corresponding code has to be written only once. As a consequence, every target-specific wrapper must expose an homogeneous API, facilitating the development of target-agnostic components. As a consequence, one of the key principles that has guided our framework design is the **genericity**.

Moreover, some controllers only implement a subset of the BLE specification: as an example, some IoT oriented controllers only implement the *Peripheral* role. As some of our detection strategies only work if the device uses a specific role, only a subset of the generated code by the framework needs to be embedded. Given this situation and the strong constraints in terms of time and memory associated with

the embedded approach, **modularity** should be a fundamental design guideline of our framework. Similarly, extending the framework to add a new target or a new detection module should be as straightforward as possible.

6.3.2 Embedded detection software

Oasis framework allows to generate an embedded detection software ready to be loaded into the chip memory. This embedded software instruments the target controller by patching specific functions to extract relevant features, then forwards these features to the selected detection modules, that analyze them and potentially raise an alert if an attack is detected. Although the software interacts with the BLE stack, it is designed to run without interfering with the legitimate behavior: as a consequence, the software uses and manages its own memory space, independently from the main firmware.

The detection software is composed of three main components, as illustrated by Figure 6.1: a *target-specific wrapper*, a *core* and a *set of detection modules*. They are described in the following sections.

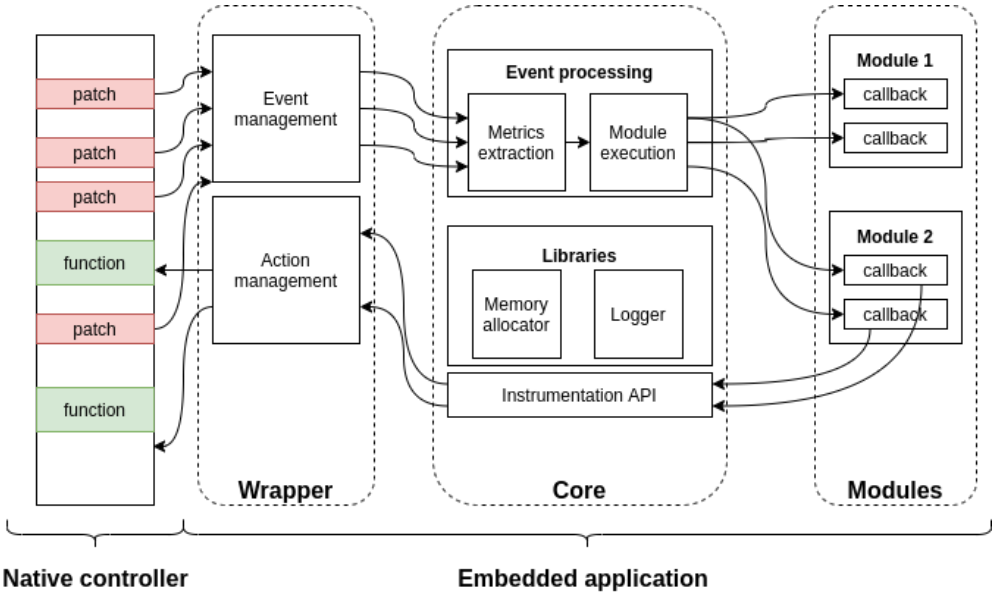

Figure 6.1: Embedded detection software overview

6.3.2.1 Target-specific wrapper

The *wrapper* is the target-specific component for interacting with the controller. It is composed of two main systems: 1) an *event management system* to react to specific events (e.g., packet reception, packet transmission, connection initiation) and to extract all available low level features from the controller, and 2) an *action management system*, to trigger specific actions in the controller (e.g., sending an event to the Host, entering a specific state).

The *event management system* is composed of a set of wrapper functions corresponding to the monitored events. It instruments the controller by patching some specific instructions of the BLE stack to redirect the execution flow to a trampoline function, that saves the context and calls the corresponding wrapper function. Once the wrapper function has been executed, the trampoline function restores the context, executes the instruction altered by the patch and redirects the execution flow to the next instruction in the stack. This mechanism allows to call the corresponding wrapper function when a specific event occurs. Then, the wrapper function extracts all available features and propagates them to the *event processing system* implemented in the Core component.

The *action management system* is composed of a set of functions to trigger a specific action in the instrumented controller. Depending on the instrumented stack, it can make a function call, mimic an HCI command transmitted by the Host or modify a variable in the controller memory.

This component is the only one that depends on the target: therefore, each implemented wrapper exposes a similar API, allowing the target-independent components to interact with the controller in a standardized and unified way.

6.3.2.2 Core

The Core is the central component of the detection software. It is composed of an *event processing system*, a *set of libraries* and an *instrumentation system*.

The *event processing system* handles the different events monitored by the detection software. When the wrapper generates a specific event, the Core collects the features extracted by the wrapper and possibly infers some complementary features from the extracted ones (e.g., the Core can infer the *advInterval* used by an *Advertiser* or a *Peripheral* from the timestamps of the advertisements received from that device). Then, the *event processing system* propagates the event and a structure containing the collected features to the loaded detection modules by executing the corresponding callbacks.

The Core also exposes an *instrumentation system*, that can be used by the detection modules to interact with the controller. This system propagates the function calls to the wrapper, allowing to enter a specific state or trigger an action in a generic way. It also provides various *libraries* facilitating the modules development. The Core exposes a custom memory allocator, allowing to dynamically allocate and release memory without interfering with the native memory management (the embedded detection software manages its own independent memory), a hashmap implementation and a logging system, allowing to send detection alerts to the Host.

6.3.2.3 Detection modules

The detection modules implement the detection strategies: they are generally responsible for analyzing the features provided by the Core component to detect attacks. They can declare a set of callbacks that are executed when a specific event

occurs, for example when a packet is received or a connection is initiated. They also have access to features collected and inferred using a specific structure, and can trigger various behaviors using the instrumentation API.

Each module is independent, and can be considered as a small embedded detection software. With this design, the user can choose which modules to include in the embedded detection software. A system of dependencies also allows to compile and flash only a subset of the framework features, depending on the selected modules requirements. This is particularly relevant given the constraints in time and memory inherent to an embedded approach.

6.3.3 Architecture of the *Oasis* framework

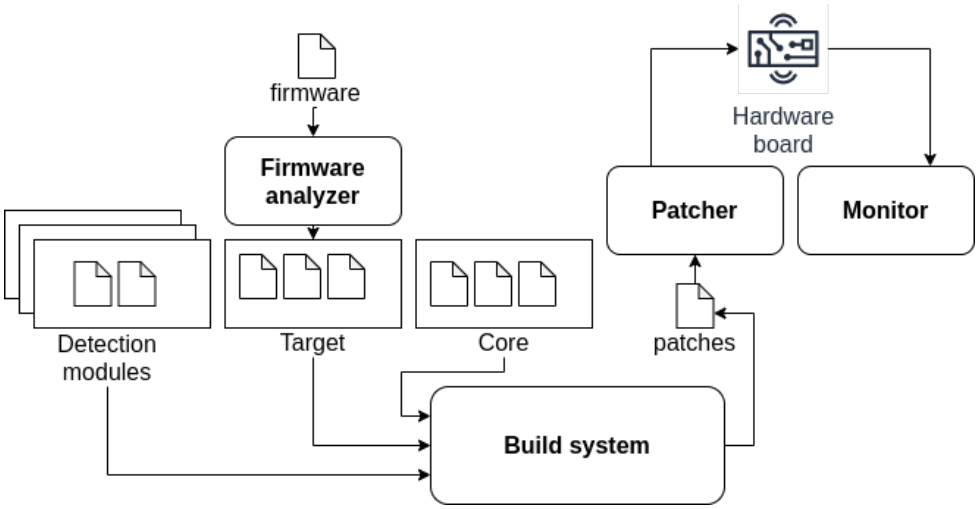

Figure 6.2: *Oasis* Framework architecture

The *Oasis* framework allows to generate the aforementioned embedded detection software and to inject it into the memory. The framework is composed of four main components, as shown in Figure 6.2: the *Firmware analyzer*, the *Build system*, the *Patcher* and the *Monitor*. They are described in the following section.

6.3.3.1 Firmware analyzer

To instrument a specific controller, the framework relies on a set of source code and configuration files describing a target, including the wrapper source code, linker scripts and configuration files. These files describe all the information needed by the framework to patch the controller firmware, inject the detection software code into the memory and interact with the controller.

Identifying the information needed to generate these files generally requires reverse engineering of the controller firmware, most of them being proprietary and not documented. Since this process is tedious and error-prone when performed

manually, the role of the *firmware analyzer* is to automate this reverse engineering task and the generation of the corresponding target-specific files.

The process is divided into two main steps. The first one is dedicated to reverse engineering the provided firmware while the second one uses the collected information to generate the source and configuration files describing the target. The reverse-engineering step is mainly based on an automated static analysis of the firmware binary which tries to identify the relevant functions, variables and structures by means of regular expressions describing specific instruction patterns or values and leveraging some empirical knowledge about code structure. It exploits the fact that different firmwares may share many similarities because of code reuse, which allows us to automate the analysis of several firmwares sharing the same controller architecture.

Once the firmware is analyzed, the tool generates the target's configuration and source code files needed to instrument it. It automatically disassembles the functions linked to a specific event to identify instructions to patch, allowing to build the list of firmware instructions to patch. The wrapper source code, linker files and configuration files are then automatically generated from the previously extracted information.

6.3.3.2 Build system

Once generated, the target is provided as input to the *build system*, with the target-agnostic software components (e.g., the Core and the selected detection modules). The *build system* is composed of a set of scripts for generating the final list of instruction patches and binary blobs that will be injected into the memory, using common tools such as the GNU gcc compiler and assembler.

The build system performs the following steps:

- **Detection modules compilation:** each selected module is compiled without linking, which allows to generate the corresponding binary blobs.
- **Modules callbacks generation:** for each selected module, the build system lists the callbacks needed by the module. Then, a glue C source code including the module callbacks as function pointers for every event is generated, allowing the Core to redirect the execution flow to the right module callback when the event occurs.
- **Trampoline functions generation:** for each patch required to instrument the target, the build system generates a trampoline function to save the context, restore it and execute the removed instruction.
- **Compilation and linking:** the whole embedded software (including the core, the target wrapper, the detection modules, the glue code and the trampoline functions) is compiled and linked. A dependency mechanism allows to compile only the required software components if the selected modules do not use some components.

- **Symbols extraction:** each symbol contained in the compiled binary is extracted from the binary and stored in a temporary file containing the symbol name, its address and its content.

- **Patches generation:** the final list of patches and binary blobs is generated, by combining the symbols previously extracted from the binary and the patches that must be applied to the controller firmware to instrument it.

6.3.3.3 Patcher and monitor

Finally, once the list of patches has been generated, the framework can inject them into the memory using the *patcher* system. Depending on the type of controller used, it may use a different backend to execute the patching process: for example, the Broadcom and Cypress stacks are patched using InternalBlue while Zephyr and Nordic Semiconductors stacks are patched using openOCD.

Similarly, the framework re-uses these backends to facilitate debugging and to monitor logs and detection alerts. Using the list of patches, the *monitor* is able to map a given symbol to its address in memory, thus allowing user-friendly debugging of the embedded software.

6.3.4 Framework usage

The framework can be easily used or extended thanks to the different components previously mentioned. A typical workflow is composed of the following steps:

- **Generating the target-specific files (optional):** if the target-specific files have not been previously generated (the framework includes a set of pre-generated files for various targets), the users can dump the firmware and use the **firmware analyzer** to automatically perform the reverse engineering process and generate the corresponding target-specific files.

- **Selecting detection modules:** users can easily select the modules they want to include in the final embedded detection software, or write their own modules using standard C code. Other software components do not require any modifications if the existing collected features are sufficient to perform the detection.

- **Building and patching the embedded detection software:** the users can then execute the **build system** to build the corresponding embedded detection software, then they can inject it into the memory using the **patcher**.

- **Monitoring the embedded detection software:** the users can debug the embedded detection software or monitor the generated logs and detection alerts using the **monitor**.

The source code of Oasis is available as open-source software under MIT license. It can be downloaded from the following repository: `https://github.com/RCayre/oasis`

6.4 Controllers instrumentation

We focused our work on three heterogeneous and widely used BLE stacks: the *Broadcom/Cypress stack*, embedded in a large number of Bluetooth chips from these manufacturers, the *SoftDevice* from Nordic SemiConductors, embedded in their BLE-enabled chips (e.g., nRF51 and nRF52 families), and the BLE stack included in the open-source OS named Zephyr.

In this section, we briefly present the internals and the instrumentation methodology we applied to the two proprietary stacks, illustrated in Figure 6.3, as they required a significant reverse engineering effort. For each stack analyzed, we performed a partial reverse engineering targeting a representative set of firmwares implementing the stack. This allowed us to identify the underlying software architecture, the implementation of the features listed in our detection requirements 6.2.2 and the memory mapping.

6.4.1 Broadcom and Cypress Bluetooth controllers

Bluetooth-enabled chips from Broadcom and Cypress use a proprietary stack based on a real time Operating System named *ThreadX*. The chips involved, based on a ARM Cortex M3 processor, are common in the wild and can be embedded in various types of devices, such as smartphones (e.g., Nexus 5, Samsung Galaxy S20), computers (e.g., Raspberry Pi) or IoT devices (e.g., FitBit Charge). These chips are poorly documented, but several works [Classen 2019, Mantz 2019] have partially documented their internals.

The BLE features are implemented as *tasks*, representing a specific state (e.g., *connection*, *scan*). A task is described by a set of functions linked to a specific event (initialisation, packet reception, packet transmission,...) and listed in a specific callbacks table. The tasks are managed by a software component named *Bluetooth Core Scheduler*, which allows to start, stop and schedule them. We hooked the initialisation and packet processing functions linked to each BLE task, allowing us to analyze the received and transmitted packets in real time while being able to detect the active GAP role. We also extracted from some radio configuration functions the structures used to store relevant features such as connection parameters or BD address.

A specific thread handles the high level operations, especially the HCI management. Every HCI command is processed by the thread and leads to the execution of a specific function, which is stored in a table of function pointers indexed by the command opcode. The HCI events are generated using an allocation function that allocates and initializes the event buffer while another function allows their transmission. We hooked both the commands processing thread, allowing us to inject

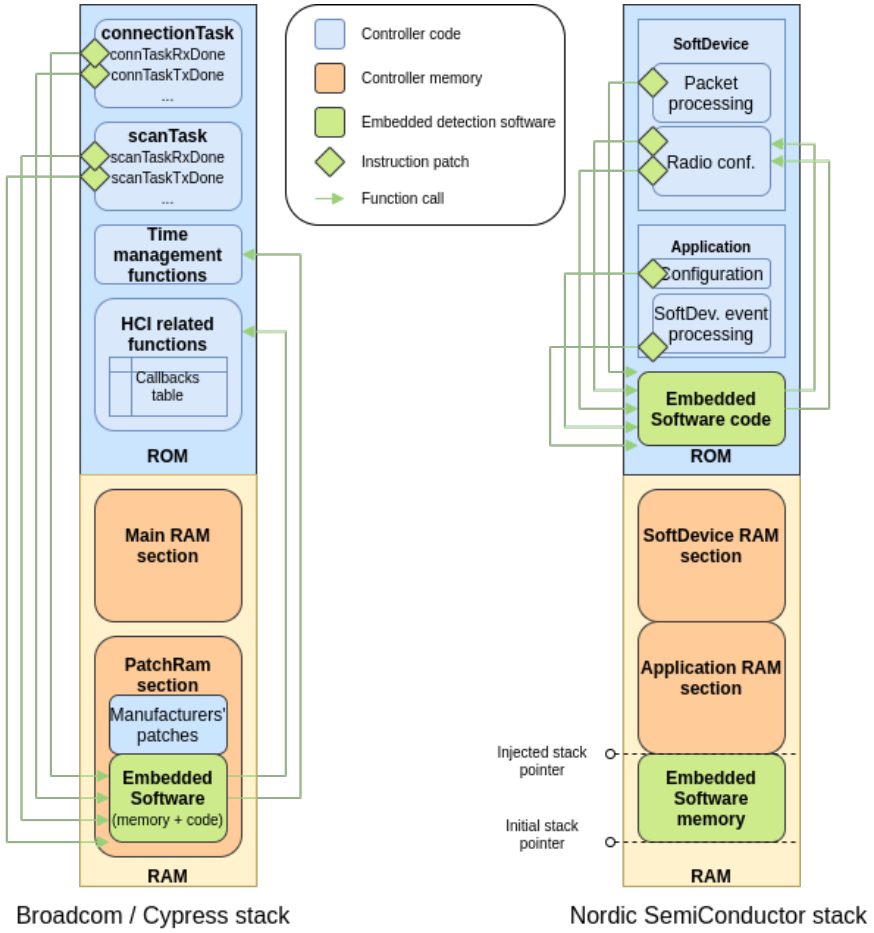

Figure 6.3: Embedded detection software integration in proprietary stacks

arbitrary commands to trigger high level operations, and the HCI events functions, used to build our logging system by passing detection alerts to the Host.

The firmware is stored in ROM, but the manufacturers have included a mechanism named *PatchRam* for updating it: a specific memory area in RAM can be used to store a limited number of ROM changes. Manufacturer patches are written in a dedicated RAM area, then a specific ROM instruction of the original firmware is patched using *PatchRam* to redirect the execution flow to the updated function in RAM. These mechanisms are triggered using vendor-specific HCI commands, allowing us to easily divert them to patch the existing firmware and inject our own code into memory. The embedded detection software code and data are stored in the manufacturer's patch section of RAM, while the *patchRam* mechanism can be used to modify the firmware instructions in ROM to setup our hooks. *InternalBlue* tool makes this process much easier, so it is used as a backend by our framework to patch and monitor these chips.

6.4.2 Nordic SemiConductors SoftDevice

Nordic SemiConductors designed a custom proprietary controller for its BLE-enabled chips (e.g., nRF51 and nRF52 families, based on ARM processors), named *SoftDevice*. These chips are commonly used in IoT devices, and multiple versions of the *SoftDevice* can be found in the wild.

The *SoftDevice* is provided by the manufacturer as a binary blob, which is loaded in the lowest parts of the ROM. The user application is flashed in the upper part of the ROM, and communicates with the *SoftDevice* using a non-standard proprietary API based on supervisor calls. A typical application initializes the *SoftDevice*, configures it to enable the needed BLE features, then monitors the events transmitted by the controller by calling a specific function in an infinite loop. The *SoftDevice* manages the low level operations: a single packet processing function is called by the radio interrupt when a packet is received or transmitted, which is able to identify the current GAP role and the current radio operation using a set of internal variables and structures. We also identified a set of configuration functions aiming at storing relevant features such as connection parameters in the internal structures.

We mainly hooked packet processing and configuration functions in the *SoftDevice* component, and extracted various features from the internal structures we identified. The function used by the application to collect the *SoftDevice* events has also been hooked, allowing us to generate the right supervisor call when we need to trigger a high level action. Similarly, we hook the entry point of the application to execute our initialization routine, allowing us to initialize the memory and configure a timer to facilitate time-management operations.

The strategy to patch the firmware and inject our code into the memory is based on the modification of the firmware binary. The firmware instructions to patch are altered in the binary itself, then the code and the memory of our detection software are appended at the end of the firmware. We also inject a decreased stack pointer initialization value in the interrupt vector, allowing us to set aside a specific zone of the RAM to avoid conflicts between the memory used by the SoftDevice, the application and our detection software. This modified firmware is flashed into the chip's ROM using openOCD, then the memory zone is copied from the ROM to the reserved RAM zone by our initialization hook.

6.5 Experiments

We performed several experiments to evaluate our detection strategies. For each attack, we flashed the corresponding detection module on multiple chips and generated both legitimate and malicious traffic in a realistic environment to estimate the detection performance. Each experiment was performed under similar conditions, with all the detection boards connected to a central gateway that monitors detection logs while regularly generating attacks and legitimate traffic.

6.5.1 Experimental setup

	Targets				
	Ra	Ne	Ga	D_1	D_2
GATTacker	✓	✓		✓	✓
BTLEJuice			✓	✓	✓
Jamming	✓	✓		✓	✓
KNOB			✓	✓	✓
InjectaBLE	✓			✓	✓
BTLEJack		✓		✓	

Table 6.1: Targets used for each experiment

Our experiments were conducted on five different targets: BCM4345C0 embedded on Raspberry Pi 3+ board, BCM4339 embedded on a Nexus 5 smartphone, Gablys, a smart keyfob with an nRF51822 controller, CYW20735 embedded on an IoT development kit, nRF51422 embedded on a nRF51 Development kit embedding various SDK examples (e.g., *Scanner* and *Peripheral*), respectively called Ra, Ne, GA, D_1 , D_2 in Table 6.1. For each experiment, the targets were selected according to their support of the roles required by our detection modules.

6.5.1.1 Experiment 1 - Gattacker

The attacks were carried out using two *HCI* dongles and the *Mirage* offensive framework presented in chapter 5 (*ble_mitm* module with *flood* strategy). The attacks targeted a connected lightbulb, located at two meters from the detection boards.

We performed 250 attacks, with a duration between 10 and 30 seconds randomly generated for each attack. Every attack was followed by a 30 second period without attack, resulting in 250 periods of legitimate traffic. Since the detection is based on the *Scanner* role, each detection board was configured to perform a scanning operation during the whole experiment.

6.5.1.2 Experiment 2 - BTLEJuice

We performed the attacks using two *HCI* dongles and the *Mirage* offensive framework (*ble_mitm* module with *pre-connect* strategy) targeting the detection boards themselves. Similarly, we generated legitimate connections representing legitimate traffic using the *ble_master* module of *Mirage*.

We performed 250 attacks and 250 legitimate connections for each detection board. Each attack duration was also randomly picked up between 10 and 30 seconds, while each legitimate connection was conducted during 5 seconds. Since detection is based on the Peripheral role being able to simultaneously maintain the connection and scan the environment, we selected targets supporting these constraints.

6.5.1.3 Experiment 3 - Jamming

The attack was carried out using an *HackRF one* transmitting random data on the frequency used by one of the three advertising channels (*hackrf_transfer* utility).

We performed 250 attacks, targeting a randomly selected advertising channel with a random duration between 10 and 30 seconds. Each attack was followed by 30 second period without attack, corresponding to the legitimate traffic phases. A connected lightbulb was present during the whole experiment in the environment. Since the detection strategy is based on the *Scanner* role, each target was configured to perform a scanning operation during the whole experiment.

6.5.1.4 Experiment 4 - KNOB

As far as we know, there is no implementation of this attack over the air, the *Proof of Concept* presented in the paper [Antonioli 2019] being implemented as an *InternalBlue* patch mimicking the behavior of the attack. We developed our own over-the-air implementation by modifying the *Mirage* framework to allow the transmission of a *Pairing Request* with an arbitrary *maxKeySize* field.

We conducted 250 attacks (i.e. 250 connections with the transmission of a *Pairing Request* using a *maxKeySize* of 7 bytes) and 250 legitimate connections (i.e. 250 connections with the transmission of a *Pairing Request* using a *maxKeySize* of 16 bytes). Each target simulated a *Peripheral* role.

6.5.1.5 Experiment 5 - InjectaBLE

The attack requires sniffing a connection, which is a non-trivial task [Ryan 2013a, Cauquil 2019], and can sometimes fail due to sniffer desynchronization: as a result, performing a fully automated experiment could lead to invalid results (e.g., an attack failure being considered as a false negative) and we chose to manually monitor the experiment. This allowed us to control the injection success but had an impact on the number of attacks that could be performed in a reasonable amount of time.

We performed 100 attacks (i.e. 100 successful injections during a connection) and mimicked 100 legitimate behaviors (i.e. 100 legitimate packets transmission during a connection, with different packet types and lengths) per target. This sample is statistically large enough to derive relevant conclusions about the efficiency of our approach to detect such attacks.

6.5.1.6 Experiment 6 - BTLEJack

Similarly to *InjectaBLE*, *BTLEJack* attack requires sniffing a connection and relies on a jamming strategy, introducing a serious risk of desynchronization or failure. As a consequence, we also chose to manually monitor the experiment to control the success of the attack.

We performed 100 attacks for each target, an attack being defined as a connection which has been successfully disrupted by BTLEJack. We also performed 100

legitimate connections per board (i.e. a connection without attack). Each target simulated a *Central* role, connecting repeatedly to the connected lightbulb.

6.5.2 Experiment Results

For each experiment performed, we compute the number of true positives (i.e. detection alert raised during an attack sequence, noted TP), false positives (i.e. detection alert raised during a legitimate sequence, noted FP), true negatives (i.e. no detection alert during a legitimate sequence, noted TN) and false negatives (i.e. no detection alert during an attack sequence, noted FN) by target. We also compute the Recall and the Precision using the following formulas:

$$Recall = \frac{TP}{TP + FN} \qquad Precision = \frac{TP}{TP + FP}$$

The results for each experiment are listed in Table 6.2. Multiple observations can be made from these results. First, we can emphasize that our detection strategies are relevant to successfully detect attacks, as illustrated by the very good Recall values we obtained (ranging from 0.94 and 1.0). Moreover, these experiments have been conducted in realistic conditions, using standard offensive hardware and software: from our perspective, these experiments and the associated results can be considered as representative of a real attacker using standard tooling.

Similarly, the high Precision values, all between 0.87 and 1.0, show that our detection strategies generate only a very small amount of false positives. In addition, four of our six experiments have a precision value equal to 1.0 for every tested target. The detection strategies that rely solely on advertisements passive monitoring (e.g., GATTacker and Jamming) generate slightly more false positives: this can be explained by the fact that they have to compute some estimates that may be impacted by some environment changes inherent to these intensively used channels.

Finally, we can point out that the results of a given experiment are globally homogeneous for each tested target. This shows that our detection modules are, as expected, independent of the underlying wrapper implementations. Even if some of our strategies cannot be systematically implemented on every target due to role requirements, these experiments also demonstrate that these defensive detections can be implemented on various types of devices, including a smartphone, a Raspberry Pi and a commercial connected object with very limited resources.

6.6 Discussions

In this chapter we focused our work on low level attacks, which are difficult to detect and mitigate by design. However, from our perspective, our approach could be easily applied to any kind of active attacks targeting the Bluetooth Low Energy protocol. Indeed, implementing the detection at the lowest level accessible by software allows to detect low level attacks as illustrated by this work, but is also

Experiment	Target	TP	FP	TN	FN	Recall	Precision
GATTacker	Ra	250	0	250	0	1.0	1.0
	Ne	250	0	250	0	1.0	1.0
	D_1	250	0	250	0	1.0	1.0
	D_2	250	19	231	0	1.0	0.93
BTLEJuice	Ga	245	0	250	5	0.98	1.0
	D_1	239	0	250	11	0.96	1.0
	D_2	250	0	250	0	1.0	1.0
Jamming	Ra	238	9	241	12	0.95	0.96
	Ne	250	13	237	0	1.0	0.95
	D_1	247	13	237	3	0.99	0.95
	D_2	250	39	211	0	1.0	0.87
KNOB	Ga	247	0	250	3	0.99	1.0
	D_1	250	0	250	0	1.0	1.0
	D_2	249	0	250	1	0.99	1.0
InjectaBLE	Ra	99	0	100	1	0.99	1.0
	D_1	100	0	100	0	1.0	1.0
	D_2	94	0	100	6	0.94	1.0
BTLEJack	Ne	95	0	100	5	0.95	1.0
	D_1	98	0	100	2	0.98	1.0

Table 6.2: Experimental results

relevant to detect attacks targeting the upper layers or being linked to a specific vulnerable implementation, as this approach gives access to the entire traffic received and transmitted by the node. More importantly, we also consider that our approach is generic enough to be extended to other wireless communication protocols that are commonly used by IoT devices, such as Zigbee or ShockBurst. Indeed, the constraints related to this type of protocols, such as the dynamicity of the environment or the absence of a central node, are effectively solved by an embedded detection performed locally by the nodes themselves. Similarly, the instrumentation of the lowest layers can provide access to a large number of features, allowing to build effective detection modules for different types of attacks. We believe that the methodology applied to build our detection modules, based on the analysis of the impact of the attack on the low level features accessible by the embedded detection software, can also be generalized to other wireless technologies. The fact that we have successfully implemented such an approach for the Bluetooth Low Energy protocol, which provides many features and makes use of complex mechanisms such as channel hopping, seems encouraging to implement such a strategy on simpler wireless protocols.

Some limitations and open challenges of this approach can also be highlighted. First, implementing the detection on local nodes complicates the collection of alerts, especially if those alerts are to be handled by a centralized SOC.

However, this problem can be solved by establishing a secure communication

channel dedicated to alert reporting between a central monitoring node and the local nodes detecting the malicious traffic. Such a channel could also be used to exploit the decentralized nature of our embedded approach, allowing nodes to share knowledge about the detected threats or to coordinate more complex detection algorithms involving multiple devices. From the perspective of generalizing this detection design to other wireless protocols, Cross Technology Communications could be a promising solution to establish a secure communication channel between local devices embedding heterogeneous wireless protocols.

Another limitation comes from the need to write target specific code to instrument heterogeneous stacks. When the implementation is proprietary, which is common in practice, one must also reverse engineer the target stack to understand and instrument its internals. Note, however, that the growing number of open source wireless stack implementations (e.g., Zephyr, NimBLE) may limit the impact of this issue in the future. Some manufacturers may also choose to integrate some detection modules in their proprietary stack, as we demonstrated that our embedded detection approach is lightweight and can even be implemented in small IoT devices with very limited resources.

6.7 Conclusion

In this chapter, we presented a new embedded detection approach for the Bluetooth Low Energy protocol, based on the instrumentation of the controller to take control over the lowest layers of the stack. We demonstrated the feasibility and relevance of this local embedded detection approach by conducting several experiments under realistic conditions on various targets, including smartphones and IoT devices with limited resources, representative of heterogeneous devices embedding this wireless technology. We were able to detect up to six critical low level attacks, including various attacks targeting the connected mode which were difficult to detect with existing strategies.

We also provide a modular, generic and user friendly framework for instrumenting various Bluetooth Low Energy controllers, suitable for collecting low level detection features and released as open-source. We consider this framework to be an important contribution to the security community, as it provides a simple way to instrument Bluetooth Low Energy controllers, and could facilitate research work in various areas such as vulnerability research or intrusion detection.

This research work has been presented in the following scientific article:

- Romain Cayre, Clément Chaine, Guillaume Auriol, Vincent Nicomette, Géraldine Marconato. **OASIS: un framework pour la détection d'intrusion embarquée dans les contrôleurs Bluetooth Low Energy.** *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2022)*, Jun 2022, Rennes, France. **[FR]** [Cayre 2022]

Chapter 7

Reactive-jamming based firewall

Contents

Packet filtering is a key requirement for various defensive systems, such as firewall or Intrusion Prevention Systems. However, the deployment of peer-to-peer wireless communication protocols significantly complicates the use of such techniques, because a node can communicate with another node without requiring the use of a central point, which is traditionally instrumented to perform filtering. In this chapter, we explore a novel approach to solve this issue and implement packet filtering in this kind of wireless networks. We combine reactive jamming techniques, allowing to corrupt the unfiltered packet on the fly to prevent its reception before its analysis, and a correction algorithm allowing to recover the initial data from the corrupted one to analyze its content and classify the packet as malicious or

legitimate. We describe the key components of our firewall architecture and perform several preliminary experiments, showing promising results to block malicious traffic in Zigbee and Enhanced ShockBurst networks.

7.1 Motivations

The most efficient approaches to preventing network level intrusion attempts mostly rely on packet filtering. From Intrusion Prevention Systems to firewalls, the ability to analyze the traffic in real time to identify malicious traffic and block it on the fly is a key requirement for many defensive mechanisms. While these capabilities are easy to implement in wired networks and most WiFi networks, which generally rely on routers and central points that can be easily instrumented to filter the traffic, they are particularly difficult to deploy on most wireless communication protocols commonly used by the Internet of Things.

Indeed, most of these protocols allow nodes to perform peer-to-peer communications, and do not require the use of central points: this leads to decentralized environments where any node can communicate with any other node, which leads to wireless topologies that do not rely on central points and are especially difficult to analyze and instrument to perform packet filtering. Because the wireless packet is transmitted directly to its destination without routing mechanisms or an intermediary by design, implementing packet analysis and filtering is not trivial and becomes a significant challenge to solve in order to efficiently mitigate wireless attacks targeting these protocols.

To implement packet filtering on such wireless technologies, we explored the use of various reactive jamming techniques from a defensive perspective. Indeed, even if the design of these protocols makes packet filtering non-trivial due to the lack of central points, the alteration of traffic reception in a wireless network has already been partially covered by offensive research works. Several offensive techniques can be used to block or alter traffic on the fly (e.g., Man-in-the-Middle attacks, Overshadowing). While some of these techniques are protocol-dependent, a jamming-based approach is interesting because it can be performed on most wireless protocols, as they generally implement an integrity checking mechanism, providing this approach a genericity which is relevant in the context of heterogeneous protocols we analyzed. Indeed, since our strategy relies on the capability to correct the corrupted frames, we need to exploit a way to check the validity of our correction.

While some previous works already explored the use of friendly jamming [Vilela 2011, Shen 2013], most of them covered confidentiality protection and targeted WiFi networks. In this work, we explore the use of defensive jamming from a different perspective, targeting packet filtering capabilities that could be used both for Intrusion Prevention systems and firewalls.

We propose to build a flexible and generic jammer that can be implemented on low cost off-the-shelf transceivers, allowing to efficiently filter wireless packets from various protocols. Moreover, we describe the design of a simple hardware

and an embedded software allowing to perform our defensive jamming approaches, and demonstrate their feasibility by carrying out several preliminary experiments protecting Zigbee and Enhanced ShockBurst communications.

7.2 Context and prerequisites

In this section, we present the context and introduce some pre-requisites that are needed to fully understand our approach. We first describe the threat model we consider, then we present the various types of jammers described in the literature. Finally, we present the objectives and challenges linked to this approach.

7.2.1 Threat model

In this work, we consider an attacker who is able to perform active attacks by transmitting malicious packets to a given wireless network. The attacker has both transmission and reception capabilities. The transmitted packets have to be compliant with the packet format of the targeted protocol, but the attacker has no time constraints (for example, starvation attacks by ignoring the Clear Channel Assessment are allowed as he can transmit at any time without being compliant with the protocol) and has control over any field composing the transmitted packet (allowing Link-Layer level spoofing attacks).

7.2.2 Jamming taxonomy

Jamming strategies mainly focus on attacking the availability of a wireless communication by transmitting an arbitrary signal on the channel in use during the packets transmissions involved in the targeted communication. The main objective is generally to corrupt the packet content by transmitting simultaneously, and prevent the legitimate nodes from receiving the targeted packet, or to force the receiver to drop it at reception (which is common when a CRC is used to check packet integrity). While the attacker can use a packet transmission to force a collision with the packet and generate some bitflips on the receiver side, he can also transmit a signal which is not compliant with the targeted protocol (e.g., noise or arbitrary waveform). Several jammer designs have been discussed in the literature in the recent years, leading to the following taxonomy (presented by W. Xu et al in [Xu 2005] and illustrated in figure 7.1):

- **Constant jammer:** a constant jammer continuously transmits an arbitrary radio signal on the targeted channel.
- **Deceptive jammer:** a deceptive jammer continuously transmits packets on the targeted channel.
- **Random jammer:** a random jammer transmits an arbitrary radio signal during a random duration on the targeted channel, then stops transmitting during a random duration.

- **Reactive jammer:** a reactive jammer only transmits an arbitrary radio signal when a packet transmission is detected on the targeted channel.

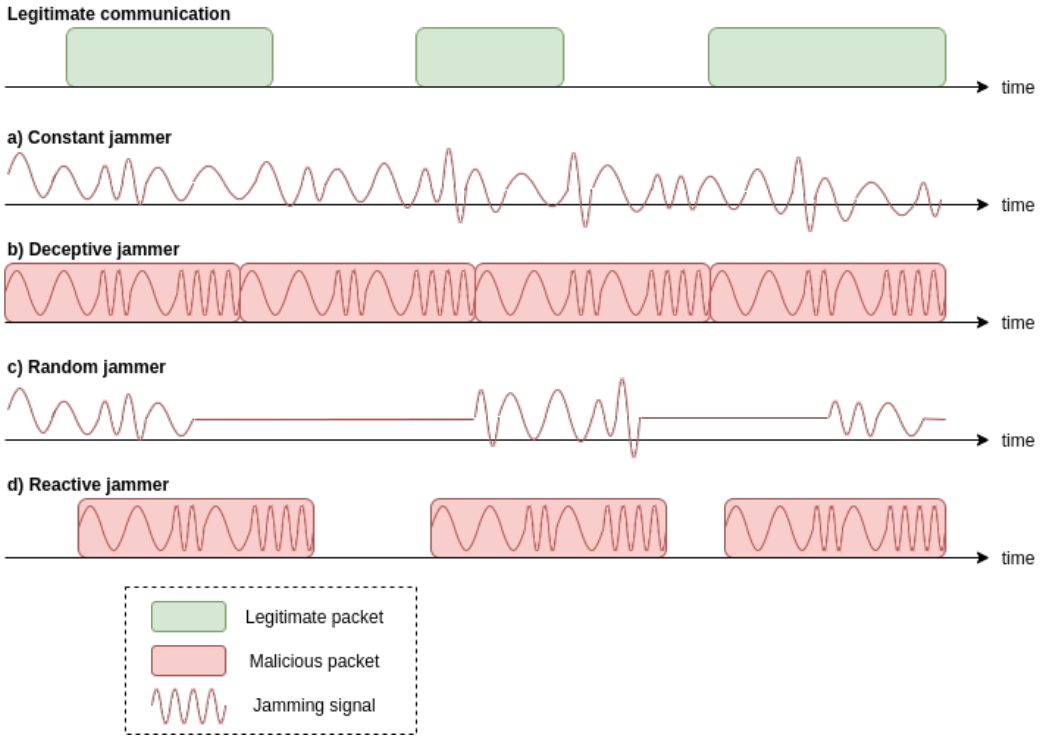

Figure 7.1: Jammers taxonomy

While constant jammers and deceptive jammers are efficient to exhaustively disrupt the traffic and present a simple design, they also imply a significant power consumption as they continuously transmit the jamming signal even if the legitimate nodes are not transmitting. The random jammer design aims to decrease the energy consumption by alternating transmission phase during a random period and sleep phase. The most efficient design is obviously the reactive jammer as it only transmits the jamming signal when a legitimate packet has been detected, leading to a minimized energy consumption and a more discrete attack. However, it also implies to increase the jammer complexity while relying on real time requirements as the jammer must quickly switch from receive to transmit mode to corrupt the detected packet before the transmission was terminated. In this work, we focus on a specific reactive jamming approach, relying on packet transmission.

7.2.3 Objectives and challenges

Our objective is to evaluate the feasibility of performing packet filtering on wireless peer-to-peer communications by using reactive jamming techniques. Ideally, the defensive system we aim to build should allow the following operations for multiple wireless protocols:

- prevent legitimate nodes within the defensive systems from receiving packets that have not been analyzed by the system (O_1),
- process the received packets to perform a security analysis in real time (O_2),
- make a decision based on the previous analysis to forward the analyzed packet to legitimate nodes or block it (O_3).

Achieving O_1 is not trivial because of the nature of the wireless medium and the peer-to-peer topology used by the protocols we consider in this work, especially if we want to build a generic approach that could be used to prevent intrusions on multiple heterogeneous protocols. While Man-in-the-Middle strategies seem suited for this operation, they generally rely on exploiting the characteristics of a specific protocol.

Jamming techniques are interesting because they can prevent the packet from being received (by corrupting the preamble or the synchronization pattern) or prevent the received packet from being processed by the receiver if an integrity checking mechanism is implemented (by corrupting any field implied in the CRC computation). Jamming is particularly interesting because of the genericity of such an approach, which can be applied to most of wireless protocols. Indeed, our main goal is to force the initial packet to be dropped, allowing us to analyze it before it is received and to forward it to the surrounding receivers only if it is considered as legitimate by our firewall.

However, even if jamming a packet can efficiently prevent its reception or its processing by the surrounding receivers, it is also a destructive operation: by design, a packet which has been jammed is partly corrupted, raising a new challenge for the defensive system. Indeed, O_2 implies that the defensive system must be able to correctly receive the packet to analyze it: as a consequence, a solution must be found to recover the packet content and in particular the corrupted parts that have been impacted by the jamming. Our approach relies on a correction algorithm that allows to recover the initial data from the corrupted packet by exploiting some knowledge about the jamming process and the targeted protocol.

Similarly, if the defensive system classifies the packet as legitimate, we need to forward it to the legitimate receiver without jamming it a second time. As a consequence, once the decision has been taken to forward a packet, the jammer must be temporarily disabled to allow the transmission of this legitimate packet to perform O_3. Such an operation is obviously sensitive from a security perspective, as it should not allow the transmission of an unknown packet during the jammer inactivity. Our approach aims to minimize this window by implementing this mechanism at a very low level.

Such an approach is promising from a defensive perspective because it allows packet filtering on peer-to-peer wireless protocols without implying any modification of the monitored devices or requiring changes to the protocol specification. However, it also raises significant challenges that must be overcome to make the approach practical:

- The jamming process must efficiently corrupt the packet to prevent its reception by surrounding nodes while allowing the correction algorithm to recover the corrupted part with a high success rate in a minimal amount of time,
- The packet filtering process must minimize the timing overhead to stay compliant with the timing constraints of the filtered protocols and have a minimal impact on the communication stability and availability,
- The firewall must ensure that the traffic is exhaustively blocked and analyzed.

In the next sections, we present the design of our approach, its implementation and the preliminary experiments we carried out to evaluate its feasibility.

7.3 Approach overview

In this section, we present our approach and its implementation. First, we introduce a high level overview of the architecture. Then, we describe the key components that are combined in the global architecture, from the reactive jamming to the correction algorithm.

7.3.1 Global architecture

Our approach relies on the combination of two transceivers (so-called nodes in the next sections). The first node implements the jamming mechanism while the second one is dedicated to the reception of the corrupted frames, the correction and the decision mechanism.

Figure 7.2 describes the two states machines representing the behaviour of each node and the interactions between them:

- **The jammer node** implements a reactive jamming mechanism, which is synchronized on a specific pattern indicating the start of a packet transmission (e.g., a preamble). When this pattern is received, the node performs a transition from reception to transmission mode to transmit a jamming signal during a short amount of time, aiming at corrupting the transmitted packet to prevent its reception by surrounding receivers.
- **The correction node** implements the analysis and decision process, allowing to classify a packet as malicious or legitimate. First, it receives the frame which has been corrupted by the jammer, and relies on a correction algorithm aiming at recovering the initial data (before its corruption by the jammer node). Once the packet has been corrected, it is analyzed according to the firewall rules to classify it as a legitimate packet or a malicious one. If the packet is considered as legitimate, the correction node disables the jammer node temporarily and retransmits the packet to the surrounding receivers.

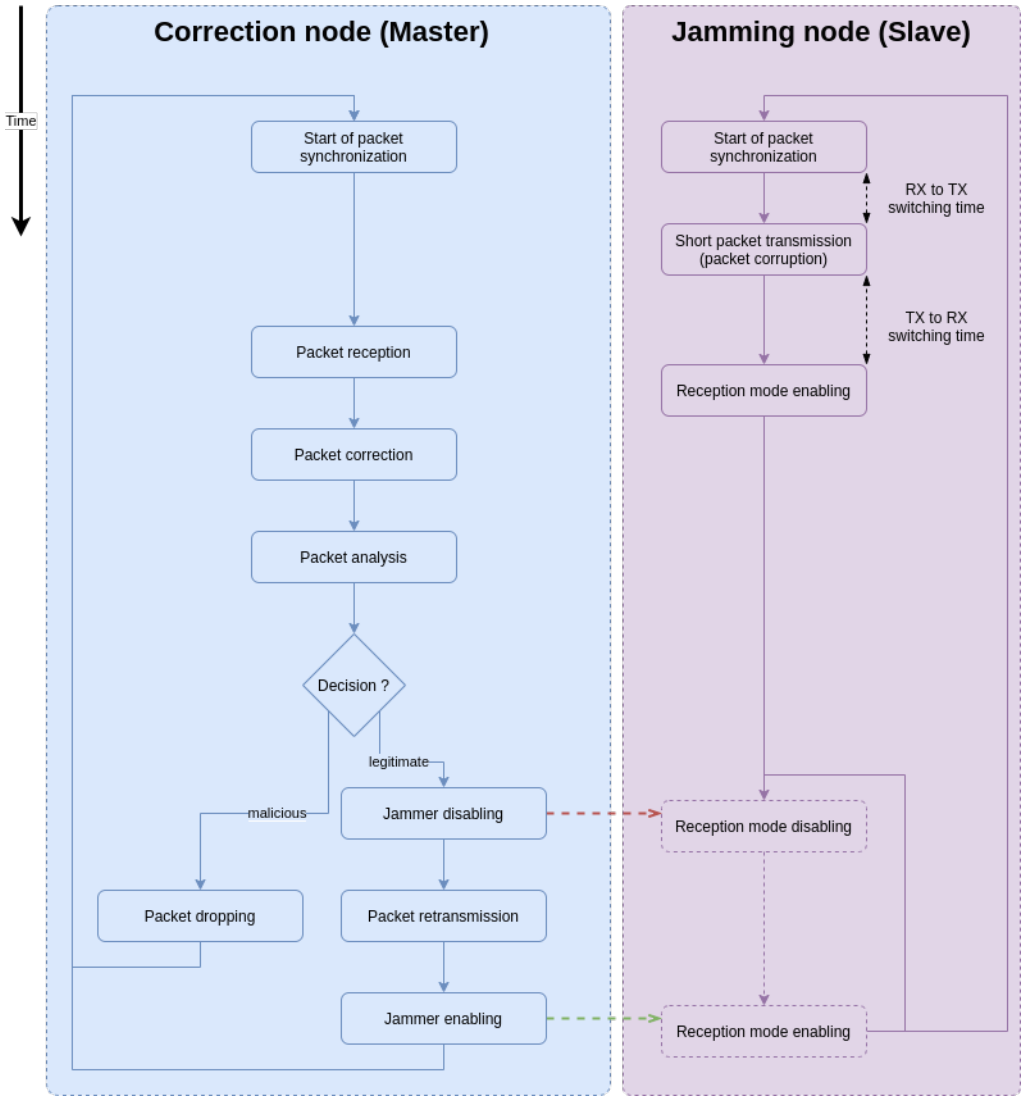

Figure 7.2: Global architecture overview

7.3.2 Reactive jamming

7.3.2.1 Requirements and design

While several jamming techniques could be used to efficiently prevent the reception of a packet by the surrounding receivers, our approach requires the ability to recover the initial data from the corrupted one in a minimum amount of time. As a result, the jamming node must meet the following requirements:

- The jamming signal must be destructive enough to efficiently corrupt every transmitted frame,
- The jamming signal must be short enough to preserve most of the received frame,

- The jamming signal must target a predictable portion of the packet to allow a fast correction by the correction node.

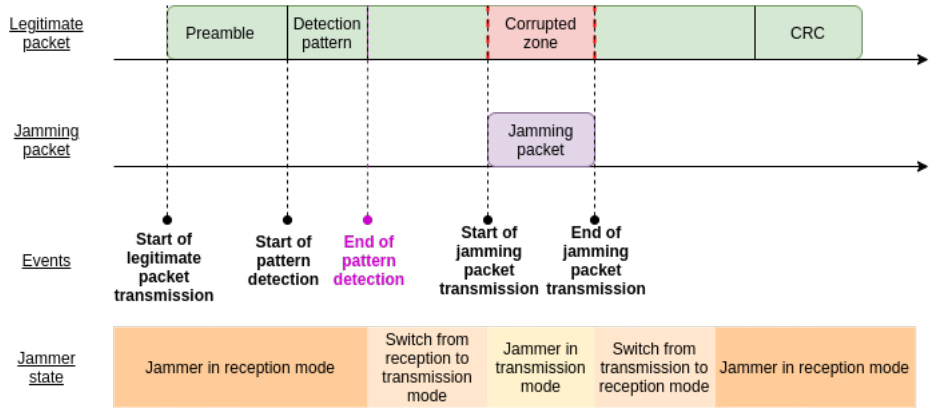

Figure 7.3: Reactive jamming operations

As a result, we need to find the right balance between these two opposite requirements, and ensure that we are able to target a specific portion of the frame. These requirements justify the use of a reactive jammer, which is less destructive than most of other jammer types and can achieve high precision for matching a specific portion of the packet because of the synchronization mechanism.

The reactive jammer we designed implements the following operations in a loop, as illustrated by figure 7.3:

- it waits for the synchronization pattern on the targeted channel,
- it switches from reception to transmission mode, when the synchronization pattern is detected,
- it transmits an arbitrary packet during a specific amount of time,
- it switches to reception mode.

7.3.2.2 Evaluation

We empirically evaluated our reactive jammer implementation to check that the previously mentioned requirements could be achieved. To perform this experiment, we programmed three nRF52840 chips (respectively named C_{JAM}, C_{RX} and C_{TX} to match the following experimental setup, illustrated in figure 7.4:

- C_{JAM} implements the previously mentioned reactive jamming algorithm. The synchronization pattern is set to **0x11 0x22 0x33 0x44**. The considered chip is able to configure the interval needed to switch from RX mode to TX mode (fast or normal) and the data rate (1Mbit/s using GFSK modulation or 2Mbit/s using GFSK modulation). We tested the following configurations:

a) fast mode, 1Mbit/s, b) normal mode, 1Mbit/s, c) fast mode, 2Mbit/s and d) normal mode, 2Mbit/s.

- C_{TX} transmits 1000 frames (using GFSK modulation at 1Mbit/s or 2Mbit/s) using the following (arbitrary) format: **0x11 0x22 0x33 0x44 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29**. A frame is transmitted every 0.5s at 2430MHz.
- C_{RX} receives the frames transmitted by C_{TX} and corrupted by C_{JAM} (using GFSK modulation at 1Mbit/s or 2Mbit/s) at 2430MHz. For each received frame, every bit composing the frame is compared to the original version: if a bitflip occurs at position n, a counter linked to bit n is incremented by one.

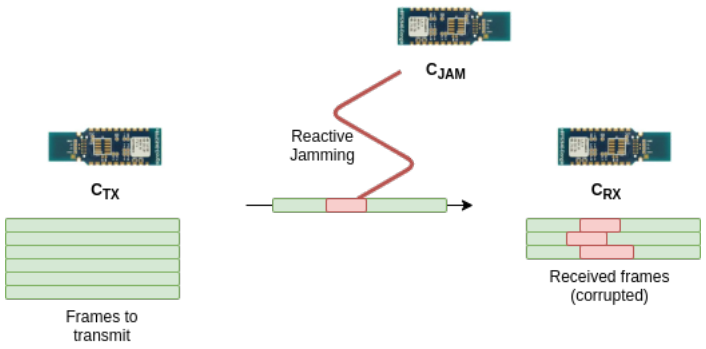

Figure 7.4: Reactive Jamming experimental setup

We obtained the following results, illustrated in figure 7.5.

The results show that our jammer implementation seems promising to meet the previously mentioned requirements.

First, we can note that our jammer managed to corrupt a very large number of frames in all modes: only 8 frames out of 1000 have been received without corruption for 1Mbit/s in fast mode and 1 uncorrupted frame out of 1000 for 1Mbit/s in normal mode, while every frame has been corrupted at 2Mbit/s in both evaluated modes. These results seem promising to ensure that no frame can be transmitted without being corrupted when the reactive jammer is enabled, which is a key requirement for our defensive system.

Second, we can clearly observe that for each tested mode, the majority of the corruption occurs in a specific zone of the frame, especially for the 2Mbit/s data rate where each corrupted bit is included between the 96th bit and the 113th bit (for the fast mode) and between the 273th and the 347th bit (for the normal node). This property is particularly interesting for us, as it shows that using a reactive jamming strategy allows to achieve high timing precision and to corrupt a predictable zone of the packet.

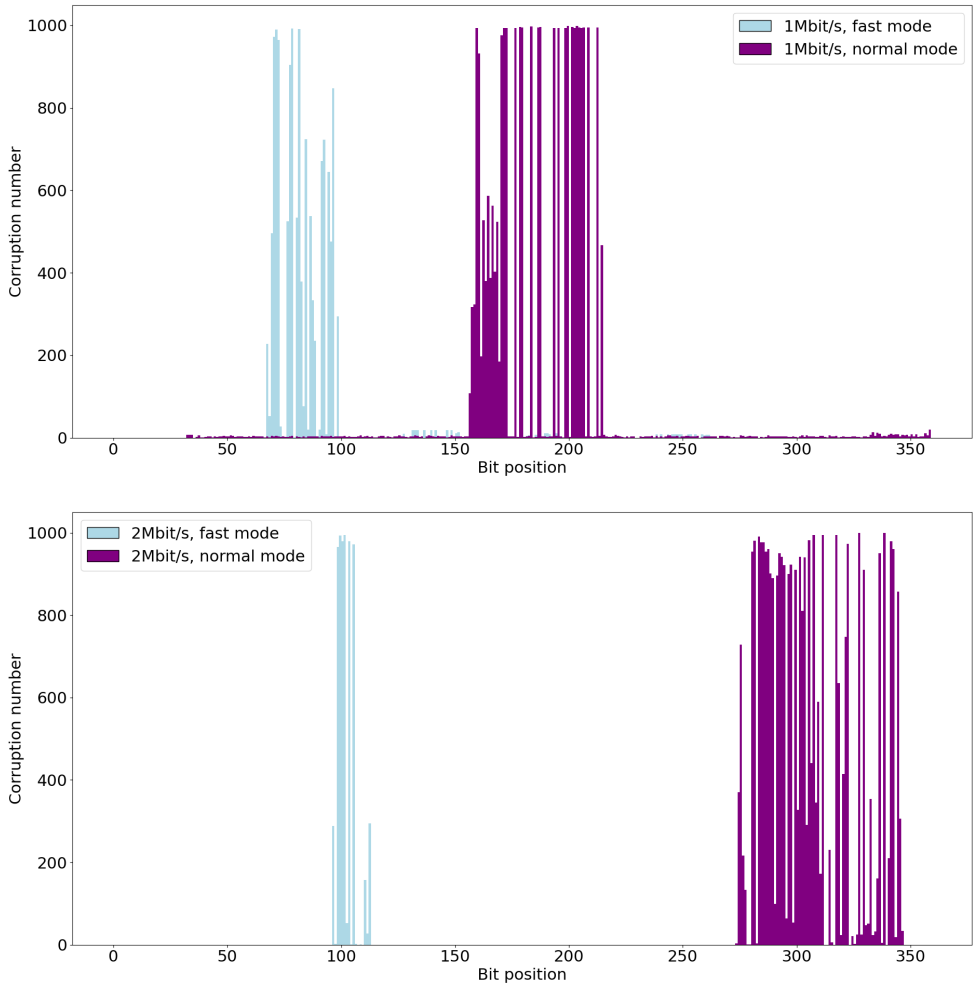

Figure 7.5: Reactive Jamming experiment results

Finally, we can observe that the jamming duration directly depends on the mode used. The fast mode allows to quickly switch from reception mode to transmission mode and from the transmission mode to reception mode, allowing to significantly reduce the duration of radio transmission to the minimum time needed to transmit the packet. At 2Mbit/s, the duration of jamming is 8.5us in fast mode and 37us in normal mode, leading to the corruption of 17 bits and 37 bits respectively.

We can conclude that the requirements we previously mentioned can be achieved using this strategy: it shows that it is possible to both prevent surrounding receivers from receiving a valid frame while limiting the corrupted zone to a small amount of time, where its position in the frame can be predicted to allow for potential correction. Since our experiment showed better results for 2Mbit/s, we focused on wireless protocols which are compatible with this data rate to build our prototype.

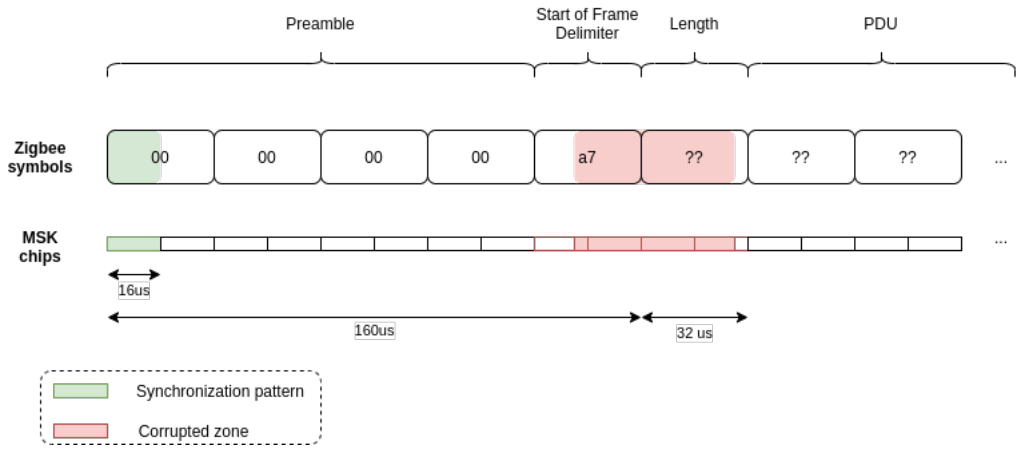

Figure 7.6: Zigbee packet corrupted by the reactive jammer

7.3.3 Correction algorithm

In the previous section, we highlighted that our reactive jamming approach could be used to target a specific zone of the transmitted frame, and could efficiently corrupt this zone without impacting the rest of the frame, especially when the data rate is set to 2Mbit/s. Thanks to this property, we were able to build two correction algorithms, implemented on a nRF52840 chip acting as receiver and allowing to recover the initial data from the corrupted one. The first algorithm is dedicated to Zigbee protocol while the second one allows the correction of Enhanced ShockBurst frames.

7.3.3.1 Zigbee

Our Zigbee receiver implementation relies on WazaBee attack, presented in chapter 3. Indeed, we showed that 802.15.4 can be received using 2Mbit/s MSK, which can be implemented on nRF52840 chips.

Both the receiver and the jammer have been synchronized on the chip sequence equivalent to 0 symbol. It allows to synchronize them on the first 4 bits composing the preamble of a 802.15.4 frame. Then, the jammer is configured to use normal mode: this configuration allows to corrupt a specific zone of the packet, located between the start of frame delimiter and the length field, as illustrated by figure 7.6.

This configuration corrupts the Start of Frame Delimiter, a 1-byte long field using a constant value (0xA7), and the 1-byte long length field. Indeed, even if the switching time is higher using normal mode than fast mode, it matches these two fields and corrupt only a small portion of the frame because of the chips sequences used by 802.15.4-based protocols. This situation is ideal because these fields are both critical to correctly analyze the frame, forcing the surrounding receivers to drop it if they are corrupted, and also trivial to correct because they can be inferred easily from the uncorrupted part of the packet.

The correction algorithm first corrects the Start of Frame Delimiter: it compares

the field value received with 0xA7. If the comparison fails, the correction algorithm considers that the field has been corrupted and sets it to 0xA7.

Then, it reconstructs the length field, by applying an iterative process checking the integrity of the frame. Initially, the field value is set to zero, then a FCS integrity check is performed according to this assumption. If the check fails, the field value is incremented by one and the checking is performed again until a matching FCS value is found.

To evaluate our correction algorithm, we transmitted 255 Zigbee frames embedding a counter value and compared the corrected frames with the transmitted ones. Our results show that only two frames could not be corrected, leading to a success rate of 99.21%.

7.3.3.2 Enhanced ShockBurst

Similarly, we built another correction algorithm for Enhanced ShockBurst protocol, and mainly focused our work on Logitech Unifying, the protocol used by Logitech for their wireless keyboards and mices.

This protocol uses a Gaussian Frequency Shift Keying modulation at 2Mbit/s, which is supported by our nRF52840 chip. A typical Enhanced ShockBurst packet is composed of a 1-byte long preamble, a 5-bytes long address, a 9-bits long header, a payload with a variable length and a 2-bytes long CRC. By configuring the jammer to use the first three bytes of the targeted address as the synchronization pattern, we observed that the jamming signal corrupts the two least significant bits of the 11th byte, the 12th byte and the three most significant bits of the 13th byte.

Multiple observations can be made in this situation. First, if the frame has a null payload length, the time needed to switch from reception to transmission mode is too long to corrupt the frame. While it may be theoretically problematic, this situation is uncommon for regular frames and only the acknowledgment frames use such small payloads: as a result, our approach is still relevant to prevent the reception of packets containing significant data. Second, because of the varying length of the packets, the jamming signal can corrupt the CRC, the payload or both. Finally, the jamming signal corrupts up to 13 bits of the received frame: as a consequence, it is not trivial to correct the frame in a short amount of time.

Our correction algorithm aims to minimize the time needed to perform the correction. As a consequence, we first use the length embedded in the frame to check the position of the corruption zone in the packet. If it only occurs in the CRC, correcting the frame is trivial because we only need to calculate a new CRC from the unaltered data. If it occurs in the CRC and in the payload or in the payload only, we perform a bruteforce algorithm by iterating over possible values for the corrupted bits until we found a valid CRC check. As this operation is obviously time consuming and introduces a significant overhead, we tried to minimize its use by saving the corrected frames in an hashmap indexed by the CRC value. Indeed, the Logitech Unifying protocol transmits some frames regularly, containing the same data and the same CRC. As a result, this approach allows us to correct the

frame without performing the costly bruteforce computation if it has previously been observed and corrected by exploiting the payloads saved in the hashmap.

7.3.4 Decision and transmission

Thanks to the reactive jammer and the receiver previously described, we can prevent surrounding receivers from receiving frames from Zigbee and Enhanced ShockBurst transmitters. We are also able to correct the received frames to recover the initial data from the corrupted one. However, we need to implement the decision mechanism checking whether the frame is malicious or not and whether it must be blocked or transmitted to legitimate receivers.

In this work, we mainly focus on evaluating the feasibility of a filtering mechanism targeting packets transmitted over the air. As a consequence, we have only implemented a basic proof of concept decision algorithm relying on pattern matching. For every frame, the decision to classify it as malicious or legitimate is performed by checking a set of basic filtering rules. Typically, a rule evaluates if the packet contains a specific value, by analyzing the bytes at a specific position or analyzing the full packet. Obviously, this decision algorithm could be significantly improved by implementing a more expressive set of rules, allowing to process a stream of packets instead of analyzing them one by one (making the resulting firewall stateful). However, our implementation allows us to focus on the filtering mechanism itself and allows to perform basic filtering operations efficiently.

If the decision algorithm classifies the frame as legitimate, it must be transmitted again without being corrupted by the jammer. However, such operation is sensitive from a security perspective as it may open a window to an attacker to transmit unfiltered packets: as a consequence, the window must be short enough to prevent this kind of attack. We implemented a Master/Slave architecture, allowing to minimize the duration of this transmission window. The correction node acts as master, and can disable the reactive jammer (acting as slave) temporarily by toggling the state of a GPIO pin. When a frame is considered as legitimate, the master disables the jammer by setting the pin to zero, transmits the legitimate frame and sets the pin to one to enable the jammer again.

We implemented this hardware architecture by combining six nRF52840 dongles and connecting the three pairs of chips together (one chip acting as a Master while the other acts as a Slave), as illustrated in figure 7.7. We implemented three firewalls in parallel to enable packet filtering on three channels simultaneously, as we plan to work on a future implementation targeting the Bluetooth Low Energy advertising channels.

7.4 Experiments

We implemented multiple experiments to evaluate the relevance of this packet filtering approach. We performed two experiments for each evaluated protocol, ZigBee and Enhanced ShockBurst, that are described in the following subsections.

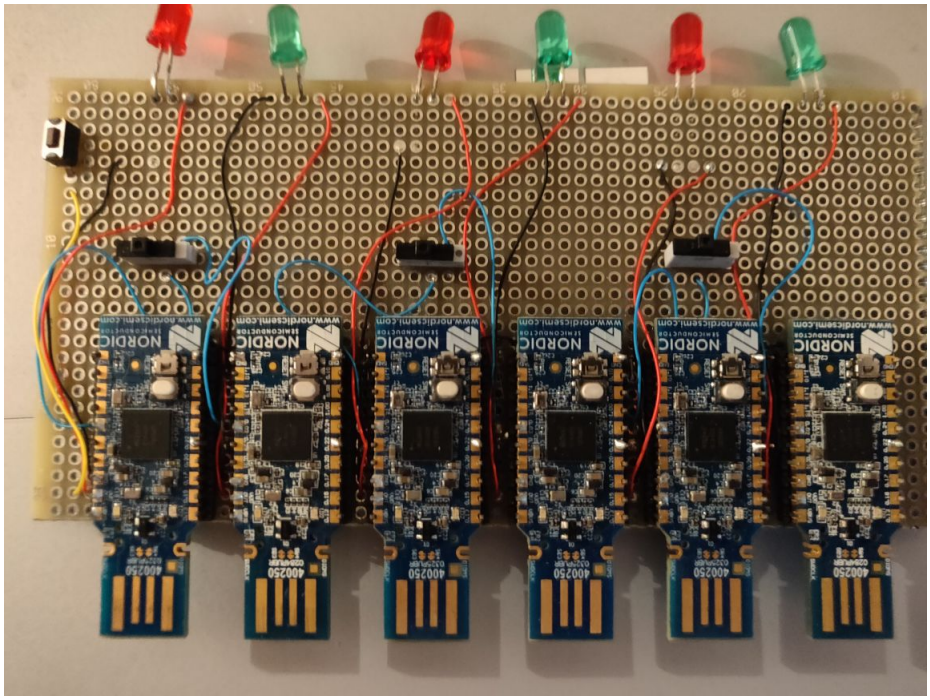

Figure 7.7: Hardware architecture

7.4.1 Experiment 1: Zigbee, basic filtering

This first experiment has been carried out on channel 10 (2410 MHz) to evaluate our approach on Zigbee. We configured an XBee node as transmitter, dedicated to the transmission of 802.15.4 beacons. Every beacon includes a counter value, incrementing on each frame. We also installed two meters away a RZUSBStick receiving the packets using the **zigbee_sniff** module included in Mirage framework (this contribution is presented in chapter 5).

The firewall was also installed two meters away from the two other nodes, and configured to consider as legitimate only the frame containing a zero in the least significant byte of the counter. We transmitted 500 packets and checked that the only ones being received by the RZUSBStick were the allowed ones. As expected, we observed that every received frame with a valid FCS matches the filtering rules.

7.4.2 Experiment 2: Zigbee, attack filtering

Our second experiment was performed using a similar setup, but two XBee nodes were configured to communicate together while the RZUSBStick was configured as a transmitter to mimic an attacker. The main goal of this experiment was to check if our approach can prevent a Zigbee attack. We focused on a remote AT injection attack [Vaccari 2017]: indeed, XBee nodes are vulnerable to a configuration injection attack, where an attacker can spoof the address of another node to inject a malicious configuration using remote AT commands, leading to a denial of service

or a Man-in-the-Middle in the worst case.

We configured our firewall to allow every frame except if it includes the *"AT"* string (0x4154 in hexadecimal), which is an indicator of a remote AT configuration packet. We performed ten attacks and managed to block them without altering the legitimate traffic between XBee nodes.

7.4.3 Experiment 3: Enhanced ShockBurst, basic filtering

Similarly to experiment 1, we evaluated Enhanced ShockBurst packet filtering by setting up a basic Enhanced ShockBurst network. The Enhanced ShockBurst network was simulated by two nRF24 devices embedding the RFStorm nRF Research firmware [Newlin 2016b], allowing to receive and transmit Enhanced ShockBurst packet. The first transceiver acted as a transmitter by sending Enhanced ShockBurst frames including a counter value, while the second one acted as a passive receiver.

Our firewall was installed two meters away from the Enhanced ShockBurst nodes and configured using the same rules we used in experiment 1. We also managed to block every unauthorized frames, every frame being received by the receiver with a valid CRC matching the filtering rules.

7.4.4 Experiment 4: Enhanced ShockBurst, attack filtering

This last experiment was dedicated to the filtering of a known Logitech Unifying attack, the unencrypted keystrokes injection presented in MouseJack [Newlin 2016a]. We installed a wireless keyboard from Logitech and the paired dongle as our legitimate Enhanced ShockBurst network, and performed the attack using *Mirage* framework 5 and a nRF24 chip embedding RFStorm firmware.

Some modifications were needed to implement this attack filtering: indeed, Logitech Unifying implements a lazy channel hopping algorithm. When a packet transmitted by the keyboard is not acknowledged by the dongle in a short period after the transmission, it considers the channel as noisy and enables the channel hopping algorithm to find another channel. Similarly, the dongle needs to regularly receive Keep Alive packets embedding a timeout value: if no packet has been received when the timer reaches the last timeout value received, the dongle enables channel hopping mode. To prevent legitimate devices from enabling this channel hopping algorithm, we modified our firewall implementation to acknowledge every frame transmitted by the keyboard before correcting the received frames: as the acknowledgment frames use zero length payload, this operation does not require to disable the jammer. Similarly, every Keep Alive packet transmitted by the keyboard is modified to increase the timeout value, compensating the overhead linked to the correction algorithm.

The firewall was configured to detect and block every unencrypted keystroke-related packet by checking that the packet type is equal to 0xC1. The legitimate keystrokes transmitted by the keyboard are not affected by the rule because they

are encrypted and use a different packet type to transmit the encrypted payload. We performed ten injection attacks, and managed to block them systematically without affecting the traffic linked to legitimate keyboard behaviour. However, we noted a significant latency for each legitimate keypress, which is probably linked to the overhead induced by our correction algorithm. While the experiment can be considered as successful, this algorithm must be significantly optimized to reduce this latency and makes it usable in a realistic context.

7.4.5 Experimental conclusion

While the results we observed during these experiments seem promising, we consider that further work is needed to consider the approach as realistic. First, we noted that it introduces some latency, especially in Enhanced ShockBurst experiments, suggesting that the correction algorithm must be significantly improved to reduce the time cost. Similarly, we only performed preliminary experiments at a small scale, and the approach may be evaluated in other environments composed of a high number of nodes interacting at a higher speed. We also consider that a sensitivity analysis is needed to evaluate the impact of several parameters, such as the firewall position or the TX power.

7.5 Discussion and Limitations

In this section, we discuss the limitations and perspectives of our firewall approach. We first analyze the genericity provided by the approach and underline some challenges that need to be solved to extend it to other protocols. We also discuss the impact of its deployment on the performance of wireless networks and the challenges that must be solved to allow its deployment in critical environments.

7.5.1 Genericity and extension to other protocols

By implementing and evaluating a proof of concept of our approach to perform packet filtering on Zigbee and Enhanced ShockBurst networks, we demonstrated the feasibility of such an approach. The main advantage of this approach is that it doesn't require any modification of the concerned devices or networks, and packet filtering can be performed transparently by deploying the firewall in the targeted environment.

A few clarifications must be made to avoid misinterpretation of this work. Obviously, an attacker passively monitoring the environment could apply a similar strategy to correct the frames himself: as a result, this approach is not designed to protect the confidentiality of wireless traffic. Let us note that the firewall is only intended to prevent the reception of unfiltered and potentially malicious traffic by legitimate surrounding receivers, while being able to recover the initial data from the corrupted one: only active attacks can be efficiently mitigated by such an approach.

We can also note that the correction node requires the presence of an integrity checking mechanism (e.g., CRC or FCS) to check the validity of the corrected data. While the presence of such mechanism is common in wireless communication protocols, we can note that some basic protocols (e.g., OOK-based communications in the 433MHz frequency band) do not provide any solution to check the packet integrity. As a result, such an approach could not be deployed to filter packets transmitted using these simple technologies.

We can also highlight that the approach can be considered as generic as it relies on a low level approach which could theoretically be applied to any wireless protocol providing an integrity checking mechanism. However, let us note that a minimal amount of knowledge about the targeted protocol is needed, especially the over the air packet format and the CRC computation algorithm used. Similarly, some specific modulation schemes (e.g., OFDM) that transmit multiple symbols at the same time could significantly complicate the deployment of this approach in a way allowing both the jamming and an effective correction process.

Another significant issue is linked to potential timing requirements linked to the targeted protocol. Indeed, if the monitored protocol implies the transmission of packets at specific times, the fact that the approach introduces a latency by design could lead to timing issues. As an example, the Bluetooth Low Energy protocol relies on strict timing requirements: if a connection request is received, the reception instant is used as a timing reference for the resulting connection. If the connection request is filtered by the firewall but considered as legitimate and retransmitted, it could introduce a desynchronization between the two concerned nodes because of the latency.

Finally, another issue could be related to the protocol design itself. While some attacks can be easily characterized using Link Layer level features that can be collected by the firewall, some basic protocols that do not provide any security features could be targeted by attacks that cannot be distinguished from legitimate traffic. For example, a keystrokes injection attack targeting a Mosart network cannot be easily separated from a legitimate keystroke packet transmitted by the keyboard. The approach could potentially use complementary information inferred from the physical layer level to identify malicious traffic (e.g., exploiting RF fingerprinting techniques), but the impact of jamming upon the physical signal must be evaluated carefully to make such a solution practical.

7.5.2 Performance issues

One of the significant issues introduced by the approach is the latency induced by the correction, decision and retransmission algorithm. We observed such a behaviour in the legitimate keystrokes retransmission in experiment 4 7.4.4, where the latency was induced by the costly bruteforce approach. As a result, minimizing the complexity and time cost of the correction and decision algorithms is a key requirement to make the approach practical.

Several solutions could be used to decrease the cost of the correction algorithm.

First, the zone corrupted by the reactive jammer could be minimized to reduce the number of bits to correct, leading to a faster computation. Our experiments were limited by the nRF52840 hardware, which is a half duplex architecture that systematically introduces a specific switching time from reception to transmission mode: implementing the reactive jammer in a dedicated hardware (e.g., FPGA, SDR) could allow the reduction of the jamming duration. A full duplex architecture would be probably more effective to minimize the duration between the synchronization pattern detection and the jamming signal transmission. Similarly, implementing the reactive jammer on a dedicated hardware could also allow to target more precisely a predictable zone of the packet, which could be corrected with a low complexity algorithm (e.g., jamming a length field or a static field). Finally, acquiring more knowledge about the protocol could facilitate the correction by providing information about specific fields formats, allowing to exploit this knowledge to check in priority some values: for example, a counter value could be easily guessed from the previous received packets without exhaustively bruteforcing the field.

Another significant timing issue is linked to the duration between the synchronization pattern detection and the jamming signal transmission, which is deeply linked to the transceiver architecture. We observed in experiment 4 7.4.4 that this interval can be greater than the packet transmission duration for short frames, resulting in a jamming signal transmitted lately after the end of transmission of the packet. While this situation was not critical in this case, it could have a greater impact if the protocol intensively uses short packets. Reducing this interval probably requires the use of a dedicated hardware with a very low level control over the transceiver, increasing the cost of the solution.

7.5.3 Critical environments

We already noted that the approach may induce some latency, but it also artificially generates a central point that can be considered as a bottleneck for the network or a single point of failure. Moreover, the impact of this firewall deployment in a complex network has not been exhaustively explored, and the potential attack surface remains unclear. As a result, we consider at the time of writing that the approach is not suited for critical environments, especially those that require high availability.

Further experiments are needed to validate the practicality of the approach in a realistic environment. We plan to explore the impact of multiple parameters by performing a sensitivity analysis. For example, the firewall position may have a significant impact on the jamming success: as the security of the approach mainly relies on this jamming mechanism, we must provide some guarantees of its efficiency under various context. Similarly, evaluating the impact of a high number of nodes and communications on the system is mandatory to ensure the scalability of the approach.

Other parameters could also be interesting to evaluate. We explored the deploy-

ment of a single firewall in the environment, but the cooperation of several packet filtering probes deployed at different locations could also be a relevant direction to improve the coverage and minimize the number. It implies to analyze the impact of the coexistence of multiple jammers in the same environment, and check the impact upon the correction algorithm performance. Similarly, such an approach could also be deployed in a mobile context, where the firewall aims to protect a single user embedding the defensive system.

7.6 Conclusion

In this work, we explored the feasibility of exploiting offensive techniques, especially reactive jamming, to perform over the air packet filtering in wireless communication networks. In the context of Internet of Things, where most of protocols allow peer to peer communications, implementing such a system is not trivial while remaining a key component of an intrusion prevention strategy or firewalling strategy.

We have demonstrated the feasibility of preventing surrounding receivers from receiving unfiltered and potentially malicious traffic, while being able to recover the initial data from the corrupted one to analyze it and classify it as malicious or legitimate. We implemented this solution and were able to block in real time some attacks targeting Zigbee and Enhanced ShockBurst protocol. While we consider that the approach must be significantly improved and evaluated in depth to become practical in a realistic environment, we have chosen to include these preliminary results as they seem promising and could motivate the exploration of related topics. Moreover, we also think that the techniques we have explored could be of interest in other contexts, both from a defensive and offensive perspective.

Conclusion and future work

Conclusion

In this PhD thesis, we explored several dimensions of the security of peer-to-peer wireless communication protocols in the context of the Internet of Things, both from an offensive and defensive perspective. The main guideline that motivated this research work was to understand in depth the peculiarities of this new context, how these new wireless protocols significantly impact security of modern devices and lead us to rethink both our defensive approaches and the threats we need to anticipate. In our opinion, the massive expansion of connected devices must be considered as a fundamental game changer in the computer science, especially from a security perspective, and we hope that this research work could contribute to a better understanding of this new situation, by highlighting the new types of threats taking advantage of wireless communications used in the Internet of Things and potential suitable defensive solutions to detect and prevent these threats.

During this research work, we tried to always keep in mind several ideas, that deeply impacted our analysis of the context and our approach. First, we take very seriously the reproducibility of our scientific work, which is a key scientific requirement from our perspective. As a result, we tried to be as comprehensive and transparent as possible when describing our methodology, our experimental setups or the tools we used. We also published and documented the source code of the tools we developed during this PhD thesis as open-source software, in an effort to facilitate both the reproducibility of our own work and allow other researchers to explore new directions thanks to the reuse, modification and distribution of these tools. We have also actively tried to explore this research theme in a cross-disciplinary and interdisciplinarity perspective. This led us to analyze multiple works from other fields, from signal processing to electronics, that brought us new insights and perspectives during our own research work. We also worked on this research theme both from an offensive and a defensive perspective and exploited the dialectical relationship between these two complementary points of view. Our offensive perspective constantly feeds and challenges our defensive analysis, and both aspects have enriched the other in stimulating ways. Finally, we have tried to give a solid experimental dimension to this work, by privileging experiments in realistic environments and using commercial off-the-shelf devices, while minimizing the use of simulations, with the aim of staying as close as possible to the complexity of real world scenarios.

Our offensive contributions have been mainly focused on the new threats raised by the specific context of Internet of Things, especially the ones linked to the way the lowest layers of wireless protocols are designed. In chapter 3, we analyzed the similarities between the modulation schemes that are commonly used by wireless protocols. We demonstrated that an attacker could take advantage of these similar-

ities to divert a transceiver dedicated to a given protocol to perform cross protocols attacks targeting another protocol which is not natively supported, or perform covert channel attacks exploiting this situation. The attack strategy we discovered, named WazaBee, showed the practical feasibility of diverting a Bluetooth Low Energy transceiver to perform attacks targeting Zigbee. We also illustrated the impact of such attacks by implementing and evaluating this strategy on a recent and popular smartphone, while extending it to proprietary protocols used by wireless keyboards and mice. It allowed us to perform several critical attack scenarios, such as sensor spoofing or wireless keylogger. With the co-existence of heterogeneous wireless protocols in the same environments and the mobility provided by multiple connected objects, we consider that these threats are realistic and should be seriously considered in the attack surface of wireless devices.

Our second offensive contribution, presented in chapter 4, focused on the lowest layers of the Bluetooth Low Energy protocol, which is massively deployed in most smartphones, laptops and connected objects. We highlighted a serious structural weakness in the connected mode provided by the protocol, allowing to inject malicious traffic into an established connection. The vulnerability we discovered being the first one to provide injection capabilities to the attacker, it allows for a significant number of critical scenarios leveraging these capabilities, such as hijacking any node involved in the connection or establishing a Man-in-the-Middle attack. We performed a sensitivity analysis to evaluate the impact of three key parameters on the success of the attack, and demonstrated that the injection could be easily exploited in a realistic context. We also implemented a custom firmware to execute this attack from a cheap nRF52840 chip, and made its source code available as open-source software to facilitate future offensive research works aiming at analyzing the low layers of Bluetooth Low Energy stacks, which are particularly difficult to analyze and secure.

Finally, our third offensive contribution (introduced in chapter 5) was motivated by the current situation of wireless security tooling, which relies on a large amount of hardware devices and software codes providing various capabilities and exposing heterogeneous APIs. We developed a generic and modular framework named Mirage, to harmonize both the use of heterogeneous hardware and facilitate the development of offensive modules. It also allows to easily combine multiple attack modules to build complex attack workflows. It provides a generic architecture, allowing the support of multiple protocols, from Bluetooth Low Energy to Zigbee, and the development of up to 20 offensive modules. The development of this tool allowed us to improve significantly the implementation of some existing offensive strategies while being a valuable resource to analyze wireless protocols and evaluate our other contributions. It has also been used by other researchers to analyze the security of Bluetooth Low Energy [Claverie 2021, von Tschirschnitz 2021], facilitating research related to wireless security.

While highlighting these new types of threats taking advantage of the lower layers of wireless communication protocols, we also worked on the detection and prevention of wireless attacks. Our defensive contributions have been dedicated

to exploring innovative designs for Intrusion Detection and Prevention Systems adapted to this new context. Our first defensive contribution was to explore the feasibility of implementing an Intrusion Detection System directly embedded in Bluetooth Low Energy controllers, allowing to collect and analyze multiple low level indicators to detect wireless attacks. We present this contribution in chapter 6. We designed an embedded detection software that hooks into multiple functions of the controller's firmware to detect the occurrence of specific events and instrument the controller to detect intrusion attempts. This embedded approach allowed us to avoid the significant issues associated with the monitoring of Bluetooth Low Energy communications with an external probe, which is a non trivial task because of the use of a channel hopping algorithm in connected mode. In this research, we designed a lightweight and modular detection framework named Oasis, allowing to facilitate the development of embedded detection modules in popular Bluetooth Low Energy controllers, while automating some difficult tasks such as the reverse engineering of firmware, the compilation of embedded detection software or the patching process. Using this approach, we successfully implemented our detection software on heterogeneous Bluetooth Low Energy stacks embedded in various devices from smartphones to connected objects to detect up to six critical low level attacks, such as BTLEJack [Cauquil 2018] or KNOB [Antonioli 2019]. We evaluated the approach in realistic conditions, achieving very good recall and precision values.

Our second defensive contribution, presented in chapter 7, aims to explore the feasibility of packets filtering in peer-to-peer wireless communication protocols by leveraging offensive techniques. We developed an experimental approach based on reactive jamming, allowing to prevent surrounding receivers from receiving and processing an unfiltered packet by corrupting it using a jamming signal, while allowing our defensive system to receive the corrupted frame, correct it to recover the initial data and analyze it. Once analyzed, the defensive system retransmits the packet to surrounding receivers if it is classified as legitimate, or raises an alert if the packet is considered as malicious. We successfully implemented a proof of concept of such packet filtering system for ZigBee and Enhanced ShockBurst protocols, and were able to block wireless attacks such as remote configuration injections or keystroke injections on the fly while allowing legitimate communications. While additional experiments are needed to accurately assess the impact of such a filtering scheme on the stability and availability of wireless communications, these preliminary results seem promising and contribute to extending defensive capabilities to prevent intrusion attempts targeting wireless networks.

Future work

We believe that this research highlights new challenges for wireless security and opens new perspectives for this research field. We were able to demonstrate the practical feasibility of cross-protocol pivoting attacks for several protocols, by di-

verting Bluetooth Low Energy transceivers from off-the-shelf devices. While this obviously highlights a new attack surface that could be considered critical in the context of mobile devices deployment and co-existence of heterogeneous wireless protocols, the impact of this type of attacks remains unclear for now. We consider that formalizing the similarities between the physical layers used by wireless protocols is a key requirement to anticipate this new type of threats, and we plan to generalize our analysis to other protocols by listing and evaluating protocol characteristics that could be leveraged to attack another technology which is not natively supported by a specific transceiver. Modeling the factors that lead to similar attacks and the proximity between two physical layers can efficiently anticipate the risk involved by these similarities and discover new cross-protocol attacks. We also plan to investigate the use of some physical properties, such as harmonics, to interact with protocols using a different frequency band. Moreover, let us note that even if some protocols cannot be easily imitated from a specific transceiver, they could be vulnerable to other offensive cross-protocol strategies such as jamming: such scenarios must also be investigated to comprehensively analyze such threats. Finally, some transceivers may implement debugging or calibration features that could potentially be diverted to manipulate the transmitted signal at a low level: investigating this type of mechanisms also seems relevant to develop a better understanding of the threat.

Similarly, new perspectives emerge from InjectaBLE attack, which allows to inject malicious traffic into an established Bluetooth Low Energy connection. We have demonstrated several critical scenarios that take advantage of this vulnerability to acquire new offensive capabilities, such as Slave Hijacking for example. The existence of such scenarios could lead to the discovery of new risks that are not anticipated for now. Since Bluetooth Low Energy is a complex protocol, it provides a large number of complex features that could potentially be exploited or diverted. For example, the HID over GATT specification defines the behaviour of application layers when communicating with input devices such as a mouse or a keyboard, which are obviously dangerous: an attacker could use the privileged position acquired by hijacking a trusted connected object to force the use of a new GATT profile describing an HID device, allowing critical scenarios such as keystrokes injection. Investigating the feasibility of such complex attack scenarios combining several structural vulnerabilities and affecting various layers is especially relevant in our perspective to improve the security of Bluetooth Low Energy protocol. We also observed that multiple other wireless protocols rely on similar features, allowing the transmission of frames only at specific times: a better understanding of the internals of these protocols could potentially lead to similar race condition based injection techniques. As future work, we plan to formalize the low level reception and transmission mechanisms used by wireless protocols and their timing requirements in order to automate the analysis and discovery of such vulnerabilities.

The development of Mirage was a key aspect of this PhD thesis, as it provided us with a simple and flexible way to implement wireless attacks and build offensive scenarios. This capability was valuable when exploring defensive solutions, as it

allowed us to easily evaluate our approaches in realistic conditions. Obviously, we plan to extend the framework by including new protocols and complementary modules, while improving the support of various hardware components. We also wish to include new modules facilitating the analysis of physical layers and the reverse engineering of proprietary protocols: the experimental Software Defined Radio support we already implemented could be improved to perform real time low level analysis, that could be relevant both from an offensive and defensive perspective. We consider that this evolution could enable the development of hybrid monitoring approaches leveraging indicators extracted from various layers, that could be relevant in a defensive perspective to detect spoofing attacks.

On the defensive side, we have shown the relevance of deploying an embedded approach on the nodes themselves to detect intrusion attempts, and explored associated tools to facilitate the implementation of such strategy. We plan to extend this strategy to other wireless protocols, as we believe that this Intrusion Detection System design is particularly relevant in the context of IoT. We also want to explore an extension of such design, by establishing a secure wireless communication between these local detection nodes to allow the implementation of a decentralized and distributed Intrusion Detection System, that could take advantage of the cooperation between multiple local detection nodes to detect more complex attacks or perform distributed computations, allowing to implement detection algorithms requiring more resources. Let us note that the deployment of this approach to multiple protocols could potentially take advantage of cross-technology communication techniques to allow the establishment of a secure wireless communication channel between heterogeneous protocols. Similarly, embedding the detection software at the lowest level that can be accessed by software in the node itself may allow the implementation of prevention techniques. The embedded software could interact with the stack to trigger specific behaviours to prevent the intrusion attempt after an alert was raised by the detection algorithm: we plan to explore this new direction in a future work.

Finally, the research we have initiated with our reactive-jamming based firewall has provided promising preliminary results, that motivate complementary work in this direction. First, we consider that a substantial optimization work is needed to make the approach practical, allowing to minimize the timing overhead generated by the correction and decision algorithm. We also plan to explore the use of a dedicated full duplex hardware embedding a custom algorithm (e.g., SDR with an implementation at FPGA level) to avoid the problems associated with the current hardware architecture, that relies on simple transceivers. Moreover, the use of reactive jamming techniques could also allow to filter some specific packets without requiring the use of a correction algorithm if the malicious packet signature can be anticipated: exploring these complementary strategies could significantly reduce the performance issues and allow an extension to protocols with strong timing requirements. Finally, combining this packet filtering approach with a physical layer analysis (e.g., signal-based fingerprinting techniques) could allow to block malicious traffic transmitted by a spoofer even if it cannot be identified as malicious at a Link

Layer level: obviously, such a hybrid approach requires accurate assessment of the impact of jamming on the transmitted signal and the design of lightweight and efficient physical layer analysis algorithms.

Bibliography

[Adafruit 2014] Adafruit. *Bluefruit LE Sniffer tutorial by AdaFruit*, 2014. Available at `https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-sniffer`. (Cited in page 30.)

[Albertini 2013] Ange Albertini. *Polyglottes binaires et implications.* In Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2013), 2013. (Cited in page 42.)

[ANT 2016] *ANT Security Resources GitHub repository*, 2016. Available at `https://github.com/sghctoma/antfs-poc-defcon24/`. (Cited in page 31.)

[Antonakakis 2017] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas and Yi Zhou. *Understanding the Mirai Botnet.* In Proceedings of the 26th USENIX Security Symposium, 2017. (Cited in pages 11 and 15.)

[Antonioli 2019] Daniele Antonioli, Nils Ole Tippenhauer and Kasper B Rasmussen. *The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR.* In 28th USENIX Security Symposium (USENIX Security 19), pages 1047–1061, 2019. (Cited in pages 30, 35, 132, 137, 149, and 177.)

[Antonioli 2020] Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen and Mathias Payer. *BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy*, 2020. (Cited in page 35.)

[Armis 2017] Armis. *Blueborne Technical White Paper.* `https://go.armis.com/hubfs/BlueBorneTechnicalWhitePaper.pdf`, 2017. (Cited in pages 16, 35, 59, and 132.)

[Armis 2018] Armis. *BleedingBit Technical White Paper.* `https://go.armis.com/hubfs/BLEEDINGBIT-TechnicalWhitePaper.pdf`, 2018. (Cited in pages 16, 35, 59, and 132.)

[Atlas 2012] Atlas. *SubGHz or Bust*, 2012. Available at `https://media.blackhat.com/bh-us-12/Briefings/Atlas/BH_US_12_Atlas_GHZ_Workshop_Slides.pdf`. (Cited in pages 31 and 104.)

[Bachy 2015] Y. Bachy, F. Basse, V. Nicomette, E. Alata, M. Kaâniche, J. Courrège and P. Lukjanenko. *Smart-TV Security Analysis: Practical Experiments.* In 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages 497–504, June 2015. (Cited in page 40.)

[Bettayeb 2019] Meriem Bettayeb, Qassim Nasir and Manar Abu Talib. *Firmware Update Attacks and Security for IoT Devices: Survey.* In Proceedings of the ArabWIC 6th Annual International Conference Research Track, ArabWIC 2019, New York, NY, USA, 2019. Association for Computing Machinery. (Cited in pages 13, 16, and 59.)

[BLE 2018] *Bleno Library GitHub repository*, 2018. Available at `https://github.com/noble/bleno/`. (Cited in page 33.)

[BLE 2019] *BLEAH GitHub repository*, 2019. Available at `https://github.com/evilsocket/bleah/`. (Cited in page 33.)

[BLU 2000] *BlueZ Website*, 2000. Available at `http://www.bluez.org/`. (Cited in page 33.)

[Blu 2019] Bluetooth SIG. *Bluetooth Core Specification*, 2019. (Cited in pages 10, 11, 34, 52, 67, 80, 86, and 132.)

[Bojovic 2019] Petar D Bojovic, Ilija Basicevic, Milos Pilipovic, Zivko Bojovic and Milena Bojovic. *The rising threat of hardware attacks: USB keyboard attack case study.* 2019. (Cited in page 14.)

[Bratus 2016] Sergey Bratus, Travis Goodspeed, Ange Albertini and Debanjum S. Solanky. *Fillory of PHY: Toward a Periodic Table of Signal Corruption Exploits and Polyglots in Digital Radio.* In 10th USENIX Workshop on Offensive Technologies (WOOT 16), Austin, TX, August 2016. USENIX Association. (Cited in page 42.)

[Brik 2008] Vladimir Brik, Suman Banerjee, Marco Gruteser and Sangho Oh. *PARADIS : Physical 802 . 11 Device Identification with Radiometric Signatures.* 2008. (Cited in page 43.)

[Bräuer 2016] S. Bräuer, A. Zubow, S. Zehl, M. Roshandel and S. Mashhadi-Sohi. *On practical selective jamming of Bluetooth Low Energy advertising.* In 2016 IEEE Conference on Standards for Communications and Networking (CSCN), pages 1–6, 2016. (Cited in pages 17, 34, 80, and 135.)

[Camurati 2018] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes and Aurélien Francillon. *Screaming Channels: When Electromagnetic Side Channels Meet Radio Transceivers.* In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, page 163–177, New York, NY, USA, 2018. Association for Computing Machinery. (Cited in page 14.)

[Camurati 2022] Giovanni Camurati and Aurélien Francillon. *Noise-SDR: Arbitrary modulation of electromagnetic noise from unprivileged software and its impact on emission security.* In IEEE, editor, S&P 2022, 43rd IEEE Symposium on Security & Privacy, 22-26 May 2022, San Francisco, CA,

USA, San Francisco, 2022. © 2022 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. (Cited in page 41.)

[Cao 2016] Xianghui Cao, Devu Manikantan Shila, Yu Cheng, Zequ Yang, Yang Zhou and Jiming Chen. *Ghost-in-ZigBee: Energy Depletion Attack on ZigBee-Based Wireless Networks*. IEEE Internet of Things Journal, vol. 3, no. 5, pages 816–829, 2016. (Cited in pages 36 and 77.)

[Cauquil 2016] Damien Cauquil. *BtleJuice: The Bluetooth Smart MiTM framework*. In DEF CON, volume 24, 2016. (Cited in pages 17, 29, 33, 35, 80, 94, 107, 132, and 134.)

[Cauquil 2017a] Damien Cauquil. *Radiobit, a BBC Micro:Bit RF firmware*, 2017. `https://github.com/virtualabs/radiobit`. (Cited in pages 39, 41, and 65.)

[Cauquil 2017b] Damien Cauquil. *Sniffing BTLE with the Micro:Bit*. PoC or GTFO, vol. 17, pages 13–20, 2017. (Cited in pages 17, 30, 34, 41, 43, 80, and 92.)

[Cauquil 2017c] Damien Cauquil. *Weaponizing the BBC Micro:Bit*. In DEF CON, volume 25, 2017. Available at `https://media.defcon.org/DEFCON25/DEFCON25presentations/DEFCON25-Damien-Cauquil-Weaponizing-the-BBC-MicroBit-UPDATED.pdf`. (Cited in pages 30, 41, and 104.)

[Cauquil 2018] Damien Cauquil. *You'd better secure your BLE devices or we'll kick your butts !* In DEF CON, volume 26, 2018. Available at `https://media.defcon.org/DEFCON26/DEFCON26presentations/DEFCON-26-Damien-Cauquil-Secure-Your-BLE-Devices-Updated.pdf`. (Cited in pages 32, 34, 65, 80, 94, 104, 132, 133, 136, and 177.)

[Cauquil 2019] Damien Cauquil. *Defeating Bluetooth Low Energy 5 PRNG for fun and jamming*. In DEF CON, volume 27, 2019. Available at `https://media.defcon.org/DEFCON27/DEFCON27presentations/DEFCON-27-Damien-Cauquil-Defeating-Bluetooth-Low-Energy-5-PRNG-for-fun-and-jamming.PDF`. (Cited in pages 17, 34, 43, 83, and 149.)

[Cayre 2019a] Romain Cayre, Vincent Nicomette, Guillaume Auriol, Eric Alata, Mohamed Kaâniche and Geraldine Marconato. *Mirage: towards a Metasploit-like framework for IoT*. In 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), Berlin, Germany, October 2019. (Cited in page 127.)

[Cayre 2019b] Romain Cayre, Jonathan Roux, Eric Alata, Vincent Nicomette and Guillaume Auriol. *Mirage : un framework offensif pour l'audit du Bluetooth Low Energy.* In Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2019), pages 229–258, Rennes, France, June 2019. (Cited in page 127.)

[Cayre 2020] Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette and Geraldine Marconato. *WazaBee : attaque de réseaux Zigbee par détournement de puces Bluetooth Low Energy.* In Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2020), Rennes, France, June 2020. (Cited in page 78.)

[Cayre 2021a] Romain Cayre and Florent Galtier. *Attaques inter-protocolaires par détournement du contrôleur Bluetooth d'un téléphone mobile.* In GT Sécurité des Systèmes, Logiciels et Réseaux, En ligne, France, May 2021. (Cited in page 78.)

[Cayre 2021b] Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche and Géraldine Marconato. *InjectaBLE : injection de trafic malveillant dans une connexion Bluetooth Low Energy.* In Symposium sur la sécurité des technologies de l'information et des communications (SSTIC 2021), Rennes (en ligne), France, June 2021. (Cited in page 101.)

[Cayre 2021c] Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche and Géraldine Marconato. *InjectaBLE: Injecting malicious traffic into established Bluetooth Low Energy connections.* In IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2021), Taipei (virtual), Taiwan, June 2021. (Cited in page 101.)

[Cayre 2021d] Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche and Géraldine Marconato. *WazaBee: attacking Zigbee networks by diverting Bluetooth Low Energy chips.* In IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2021), Taipei (virtual), Taiwan, June 2021. (Cited in page 78.)

[Cayre 2021e] Romain Cayre, Géraldine Marconato, Florent Galtier, Mohamed Kaâniche, Vincent Nicomette and Guillaume Auriol. *POSTER: Cross-protocol attacks: weaponizing a smartphone by diverting its Bluetooth controller.* In 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Abu Dhabi, United Arab Emirates, June 2021. Prix du meilleur poster à la conférence WiSec 2021. (Cited in page 78.)

[Cayre 2022] Romain Cayre, Clément Chaine, Guillaume Auriol, Vincent Nicomette and Géraldine Marconato. *OASIS: un framework pour la détection d'intrusion embarquée dans les contrôleurs Bluetooth Low Energy.* In

Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2022), Rennes, France, June 2022. (Cited in page 153.)

[CC2 2019] *CC2652R Data Sheet*, 2019. `http://www.ti.com/lit/ds/symlink/cc2652r.pdf`. (Cited in page 40.)

[Celosia 2019] Guillaume Celosia and Mathieu Cunche. *DEMO: Himiko: A human interface for monitoring and inferring knowledge on Bluetooth-Low-Energy objects*. WiSec 2019 - 12th Conference on Security and Privacy in Wireless and Mobile Networks, May 2019. Poster. (Cited in page 34.)

[Celosia 2020a] Guillaume Celosia and Mathieu Cunche. *DEMO: Venom: a Visual and Experimental Bluetooth Low Energy Tracking System*. WiSec 2020 - 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, July 2020. Poster. (Cited in page 34.)

[Celosia 2020b] Guillaume Celosia and Mathieu Cunche. *Discontinued Privacy: Personal Data Leaks in Apple Bluetooth-Low-Energy Continuity Protocols*. Proceedings on Privacy Enhancing Technologies, vol. 2020, pages 26 – 46, July 2020. (Cited in page 34.)

[Chebrolu 2009] Kameswari Chebrolu and Ashutosh Dhekne. *Esense: Communication through Energy Sensing*. In Proceedings of the 15th Annual International Conference on Mobile Computing and Networking, MobiCom '09, page 85–96, New York, NY, USA, 2009. Association for Computing Machinery. (Cited in page 40.)

[Chiasserini 2003] C.F. Chiasserini and R.R. Rao. *Coexistence mechanisms for interference mitigation in the 2.4-GHz ISM band*. IEEE Transactions on Wireless Communications, vol. 2, no. 5, pages 964–975, 2003. (Cited in page 12.)

[Classen 2019] Jiska Classen and Matthias Hollick. *Inside Job: Diagnosing Bluetooth Lower Layers Using off-the-Shelf Devices*. In Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks, WiSec '19, page 186–191, New York, NY, USA, 2019. Association for Computing Machinery. (Cited in page 145.)

[Classen 2021] Jiska Classen. *InternalBlue Android documentation*, 2021. Available at `https://github.com/seemoo-lab/internalblue/blob/master/doc/android.md`. (Cited in pages 31 and 71.)

[Claverie 2021] Tristan Claverie, Nicolas Docq and José Lopes-Esteves. *Analyse des propriétés de sécurité dans les implémentations du Bluetooth Low Energy*. In Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2021), 2021. (Cited in page 176.)

[Cui 2013] Ang Cui, Michael Costello and Salvatore Stolfo. *When firmware modifications attack: A case study of embedded exploitation.* 2013. (Cited in page 15.)

[Dalalana Bertoglio 2017] Daniel Dalalana Bertoglio and Avelino Zorzo. *Overview and open issues on penetration test.* Journal of the Brazilian Computer Society, vol. 23, 12 2017. (Cited in page 105.)

[Dixon 2019] Brad Dixon. *How to cheat at virtual cycling using USB hacks.* In DEF CON, volume 27, 2019. Available at `https://media.defcon.org/DEFCON27/DEFCON27presentations/DEFCON-27-Brad-Dixon-Cheating-in-eSports-How-to-cheat-at-virtual-cycling-using-USB-hacks.pdf`. (Cited in page 38.)

[Dorsemaine 2015] Bruno Dorsemaine, Jean-Philippe Gaulier, Jean-Philippe Wary, Nizar Kheir and Pascal Urien. *Internet of Things: A Definition amp; Taxonomy.* In 2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies, pages 72–77, 2015. (Cited in page 8.)

[Dyn 2014] DynaStream. *ANT Specification*, 2014. (Cited in pages 10 and 29.)

[Ell 2021] Ellisys. *Ellisys Bluetooth Explorer datasheet*, 2021. Accessed: 2022-03-21. (Cited in page 30.)

[FitzPatrick 2016] Joe FitzPatrick. *The Tao of hardware, the Te of implants.* Black Hat, USA, 2016. (Cited in page 14.)

[Fragkiadakis 2012] Alexandros Fragkiadakis, Sofia Nikitaki and Panagiotis Tsakalides. *Physical-layer intrusion detection for wireless networks using compressed sensing.* In 2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pages 845–852, 2012. (Cited in page 42.)

[Galtier 2020] Florent Galtier, Romain Cayre, Guillaume Auriol, Mohamed Kaâniche and Vincent Nicomette. *A PSD-based fingerprinting approach to detect IoT device spoofing.* In 25th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2020), Perth, Australia, December 2020. (Cited in pages 27, 43, and 100.)

[Gandolfi 2001] Karine Gandolfi, Christophe Mourtel and Francis Olivier. *Electromagnetic Analysis: Concrete Results.* In Çetin K. Koç, David Naccache and Christof Paar, editors, Cryptographic Hardware and Embedded Systems — CHES 2001, pages 251–261, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. (Cited in page 14.)

[Garbelini 2020] Matheus E. Garbelini, Chundong Wang, Sudipta Chattopadhyay, Sun Sumei and Ernest Kurniawan. *SweynTooth: Unleashing Mayhem over Bluetooth Low Energy.* In 2020 USENIX Annual Technical Conference

(USENIX ATC 20), pages 911–925. USENIX Association, July 2020. (Cited in pages 16, 35, and 132.)

[Genkin 2016] Daniel Genkin, Lev Pachmanov, Itamar Pipman and Eran Tromer. *ECDH Key-Extraction via Low-Bandwidth Electromagnetic Attacks on PCs.* In CT-RSA, 2016. (Cited in page 14.)

[Ghugar 2018] Umashankar Ghugar, Jayaram Pradhan, Sourav Bhoi, Rashmi Sahoo and Sanjaya Panda. *PL-IDS: physical layer trust based intrusion detection system for wireless sensor networks.* International Journal of Information Technology, vol. 10, pages 1–6, 04 2018. (Cited in page 42.)

[GNU 2021] *GNU Radio website*, 2021. Available at `https://www.gnuradio.org/`. (Cited in page 31.)

[Goodspeed 2011a] Travis Goodspeed. *Promiscuity is the nRF24L01+'s Duty.* Available at `http://travisgoodspeed.blogspot.com/2011/02/promiscuity-is-nrf24l01s-duty.html`, 2011. (Cited in pages 29 and 41.)

[Goodspeed 2011b] Travis Goodspeed, Sergey Bratus, Ricky Melgares, Rebecca Shapiro and Ryan Speers. *Packets in packets: Orson Welles' in-band signaling attacks for modern radios.* pages 7–7, 08 2011. (Cited in pages 17, 41, and 72.)

[GQR 2013] *GQRX Website*, 2013. Available at `https://gqrx.dk/`. (Cited in page 31.)

[greatscottgadgets 2022] greatscottgadgets. *GitHub repository of HackRF One.* `https://github.com/greatscottgadgets/hackrf`, 2022. Accessed: 2022-03-21. (Cited in pages 27 and 32.)

[GRN 2016] *gr-nordic GitHub repository*, 2016. Available at `https://github.com/BastilleResearch/gr-nordic/`. (Cited in page 31.)

[Gutierrez del Arroyo 2017] Jose Gutierrez del Arroyo, Jason Bindewald, Scott Graham and Mason Rice. *Enabling Bluetooth Low Energy Auditing through Synchronized Tracking of Multiple Connections.* Int. J. Crit. Infrastruct. Prot., vol. 18, no. C, page 58–70, sep 2017. (Cited in page 44.)

[Hall 2005] Jeyanti Hall, Michel Barbeau and Evangelos Kranakis. *Radio Frequency Fingerprinting for Intrusion Detection in Wireless Networks.* IEEE Trans. Dependable Secure Comput., 2005. (Cited in page 43.)

[Helluy-Lafont 2020] Étienne Helluy-Lafont, Alexandre Boé, Gilles Grimaud and Michaël Hauspie. *Bluetooth devices fingerprinting using low cost SDR.* In Sixth International Workshop on Internet of Things: Networking Applications and Technologies, Paris, France, June 2020. (Cited in page 43.)

[Hu 2006] Yih-Chun Hu, Adrian Perrig and David B Johnson. *Wormhole attacks in wireless networks.* IEEE journal on selected areas in communications, vol. 24, no. 2, pages 370–380, 2006. (Cited in page 17.)

[IEE 2020] *IEEE Standard for Low-Rate Wireless Networks.* IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015), pages 1–800, 2020. (Cited in page 55.)

[INS 2021] *Inspectrum GitHub Repository*, 2021. Available at `https://github.com/miek/inspectrum`. (Cited in page 32.)

[IOT 2018] *B-L475E-IOT01A Data Brief*, 2018. `https://www.st.com/resource/en/data_brief/b-l475e-iot01a.pdf`. (Cited in page 40.)

[JAC 2020] *JackIt GitHub Repository*, 2020. Available at `https://github.com/insecurityofthings/jackit`. (Cited in page 33.)

[Jasek 2016] Sławomir Jasek. *Gattacking Bluetooth Smart Devices.* In BlackHat USA, 2016. Available at `http://gattack.io/whitepaper.pdf`. (Cited in pages 17, 29, 33, 35, 80, 94, 107, 132, and 134.)

[Jiang 2017] Wenchao Jiang, Zhimeng Yin, Ruofeng Liu, Zhijun Li, Song Min Kim and Tian He. *BlueBee: A 10,000x Faster Cross-Technology Communication via PHY Emulation.* In Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery. (Cited in pages 18 and 41.)

[Jiang 2018] Wenchao Jiang, Song Min Kim, Zhijun Li and Tian He. *Achieving Receiver-Side Cross-Technology Communication with Cross-Decoding.* In Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, MobiCom '18, page 639–652, New York, NY, USA, 2018. Association for Computing Machinery. (Cited in pages 18 and 41.)

[Jokar 2016] Paria Jokar. *Intrusion Detection and Prevention for ZigBee-Based Home Area Networks in Smart Grids.* IEEE Transactions on Smart Grid, vol. PP, pages 1–1, 08 2016. (Cited in page 43.)

[Kamkar 2015] Samy Kamkar. *Drive It Like You Hacked It.* In DEF CON, volume 23, 2015. Available at `https://samy.pl/defcon2015/2015-defcon.pdf`. (Cited in page 39.)

[Khelif 2021] Mohamed Amine Khelif, Jordane Lorandel and Olivier Romain. *Non-invasive I2C Hardware Trojan Attack Vector.* In 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pages 1–6, 2021. (Cited in page 14.)

[Kim 2015] Song Min Kim and Tian He. *FreeBee: Cross-Technology Communication via Free Side-Channel.* In Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, MobiCom '15,

page 317–330, New York, NY, USA, 2015. Association for Computing Machinery. (Cited in page 40.)

[Koch 2022] Luke Koch, Sean Oesch, Mary Adkisson, Sam Erwin, Brian Weber and Amul Chaulagain. *Toward the Detection of Polyglot Files.* arXiv preprint arXiv:2203.07561, 2022. (Cited in page 42.)

[Kocher 2019] Paul C. Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Michael Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz and Yuval Yarom. *Spectre Attacks: Exploiting Speculative Execution.* 2019 IEEE Symposium on Security and Privacy (SP), pages 1–19, 2019. (Cited in page 14.)

[Köse 2019] Memduh Köse, Selçuk Taşcioğlu and Ziya Telatar. *RF Fingerprinting of IoT devices based on transient energy spectrum.* IEEE Access, vol. 7, pages 18715–18726, 2019. (Cited in page 43.)

[Krivtsova 2016] Irina Krivtsova, Ilya Lebedev, Mikhail Sukhoparov, Bazhayev Nurzhan, Igor Zikratov, Aleksandr Ometov, Sergey Andreev, Pavel Masek, Radek Fujdiak and Jiri Hosek. *Implementing a Broadcast Storm Attack on a Mission-Critical Wireless Sensor Network.* pages 297–308, 05 2016. (Cited in page 37.)

[Krzysztoń 2020] Mateusz Krzysztoń and Michał Marks. *Simulation of watchdog placement for cooperative anomaly detection in Bluetooth Mesh Intrusion Detection System.* Simulation Modelling Practice and Theory, vol. 101, page 102041, 2020. Modeling and Simulation of Fog Computing. (Cited in page 44.)

[Lacava 2021] Andrea Lacava, Emanuele Giacomini, Francesco D'Alterio and Francesca Cuomo. *Intrusion Detection System for Bluetooth Mesh Networks: Data Gathering and Experimental Evaluations.* In 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), pages 661–666, 2021. (Cited in page 44.)

[Lahmadi 2020] Abdelkader Lahmadi, Alexis Duque, Nathan Heraief and Julien Francq. *MitM Attack Detection in BLE Networks using Reconstruction and Classification Machine Learning Techniques.* In MLCS 2020-2nd Workshop on Machine Learning for Cybersecurity, 2020. (Cited in pages 44 and 100.)

[Lai 2008] Jung-Ying Lai, Jain-Shing Wu, Shih-Jen Chen, Chia-Huan Wu and Chung-Huang Yang. *Designing a taxonomy of web attacks.* In 2008 International Conference on Convergence and Hybrid Information Technology, pages 278–282. IEEE, 2008. (Cited in page 15.)

[Le 2008] Thanh-Ha Le, Cécile Canovas and Jessy Clédiere. *An overview of side channel analysis attacks.* In Proceedings of the 2008 ACM symposium on

Information, computer and communications security, pages 33–43, 2008. (Cited in page 14.)

[Li 2017] Zhijun Li and Tian He. *WEBee: Physical-Layer Cross-Technology Communication via Emulation.* In Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking, MobiCom '17, page 2–14, New York, NY, USA, 2017. Association for Computing Machinery. (Cited in page 40.)

[Lipp 2018] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul C. Kocher, Daniel Genkin, Yuval Yarom and Michael Hamburg. *Meltdown: Reading Kernel Memory from User Space.* In USENIX Security Symposium, 2018. (Cited in page 14.)

[Lo 2017] Owen Lo, William J. Buchanan and Douglas Carson. *Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA).* Journal of Cyber Security Technology, vol. 1, no. 2, pages 88–107, 2017. (Cited in page 14.)

[LOG 2019] *LogiTacker GitHub Repository*, 2019. Available at `https://github.com/RoganDawes/LOGITacker`. (Cited in pages 39 and 116.)

[LoR 2017] LoRa Alliance, Inc. *LoRaWan Specification*, 2017. (Cited in page 10.)

[Mantz 2019] Dennis Mantz, Jiska Classen, Matthias Schulz and Matthias Hollick. *InternalBlue - Bluetooth Binary Patching and Experimentation Framework.* Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services, Jun 2019. (Cited in pages 29, 30, 67, 68, 70, and 145.)

[MET 2022] *Metasploit website*, 2022. Available at `https://www.metasploit.com`. (Cited in page 105.)

[Miettinen 2016] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N. Asokan, Ahmad-Reza Sadeghi and Sasu Tarkoma. *IoT Sentinel: Automated Device-Type Identification for Security Enforcement in IoT.* CoRR, vol. abs/1611.04880, 2016. (Cited in page 43.)

[Millian 2015] Michael C. Millian and Vibhu Yadav. *Packet-in-packet Exploits on 802 . 15 . 4.* 2015. (Cited in page 41.)

[MyriadRF 2022] MyriadRF. *MyriadRF website (Lime SDR manufacturer).* `https://myriadrf.org/`, 2022. Accessed: 2022-03-21. (Cited in pages 27 and 32.)

[Newaz 2020] AKM Iqtidar Newaz, Amit Kumar Sikder, Leonardo Babun and A. Selcuk Uluagac. *HEKA: A Novel Intrusion Detection System for Attacks*

to Personal Medical Devices. In 2020 IEEE Conference on Communications and Network Security (CNS), pages 1–9, 2020. (Cited in page 44.)

[Newlin 2016a] Marc Newlin. *MouseJack : White Paper*. In DEF CON, volume 24, 2016. Available at `https://github.com/BastilleResearch/mousejack/blob/master/doc/pdf/DEFCON-24-Marc-Newlin-MouseJack-Injecting-Keystrokes-Into-Wireless-Mice.whitepaper.pdf`. (Cited in pages 17, 29, 30, 33, 39, 41, 75, 76, 116, 124, and 169.)

[Newlin 2016b] Marc Newlin. *RFStorm nRF24LU1+ Research Firmware GitHub repository*, 2016. `https://github.com/BastilleResearch/nrf-research-firmware`. (Cited in pages 28, 29, 30, 32, 33, and 169.)

[NOB 2018] *Bleno Library GitHub repository*, 2018. Available at `https://github.com/noble/noble/`. (Cited in page 33.)

[NRF 2022] *NRF Connect presentation webiste*, 2022. Available at `https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-desktop/`. (Cited in page 33.)

[Nuand 2022] Nuand. *Nuand website (BladeRF manufacturer)*. `https://www.nuand.com/`, 2022. Accessed: 2022-03-21. (Cited in page 27.)

[Olawumi 2014] Olayemi Olawumi, Keijo Haataja, Mikko Asikainen, Niko Vidgren and Pekka Toivanen. *Three practical attacks against ZigBee security: Attack scenario definitions, practical experiments, countermeasures, and lessons learned*. In 2014 14th International Conference on Hybrid Intelligent Systems, pages 199–206, 2014. (Cited in page 37.)

[OPE 2015] *OpenSesame website*, 2015. Available at `https://samy.pl/opensesame`. (Cited in page 39.)

[Ossmann 2017] Michael Ossmann, Dominic Spill and Great Scott Gadgets. *What's on the wireless? Automating rf signal identification*. Technical Report, Tech. rep, 2017. (Cited in pages 27 and 32.)

[P. Cavano 1978] Joseph P. Cavano and James A. McCall. *A framework for the measurement of software quality*. ACM SIGSOFT Software Engineering Notes, vol. 3, pages 133–139, 11 1978. (Cited in page 104.)

[Pa 2015] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama and Christian Rossow. *IoTPOT: Analysing the Rise of IoT Compromises*. In 9th USENIX Workshop on Offensive Technologies (WOOT 15), 2015. (Cited in page 15.)

[Pasupathy 1979] S. Pasupathy. *Minimum shift keying: A spectrally efficient modulation*. IEEE Communications Magazine, vol. 17, no. 4, pages 14–22, July 1979. (Cited in page 59.)

[Philips 2022] Philips. *Philips Hue website*. `https://www.philips-hue.com`, 2022. Accessed: 2022-03-21. (Cited in page 11.)

[Pohl 2018] Johannes Pohl and Andreas Noack. *Universal Radio Hacker: A Suite for Analyzing and Attacking Stateful Wireless Protocols*. In 12th USENIX Workshop on Offensive Technologies (WOOT 18), Baltimore, MD, August 2018. USENIX Association. (Cited in page 32.)

[POT 2021] *PothosFlow GitHub repository*, 2021. Available at `https://github.com/pothosware/PothosFlow/`. (Cited in page 31.)

[PYB 2015] *PyBT GitHub repository*, 2015. Available at `https://github.com/mikeryan/PyBT`. (Cited in page 34.)

[Qasim Khan 2019] Sultan Qasim Khan. *Sniffle: A sniffer for Bluetooth 5 (LE)*, 2019. Available at `https://hardwear.io/netherlands-2019/presentation/sniffle-talk-hardwear-io-nl-2019.pdf`. (Cited in pages 30, 32, 34, 80, and 92.)

[Rajendran 2019] Sreeraj Rajendran, Wannes Meert, Vincent Lenders and S. Pollin. *Unsupervised Wireless Spectrum Anomaly Detection with Interpretable Features*. IEEE Transactions on Cognitive Communications and Networking, vol. PP, pages 1–1, 04 2019. (Cited in page 77.)

[Ray 2012] Donald Ray and Jay Ligatti. *Defining code-injection attacks*. Acm Sigplan Notices, vol. 47, no. 1, pages 179–190, 2012. (Cited in page 15.)

[Raza 2013] Shahid Raza, Linus Wallgren and Thiemo Voigt. *SVELTE: Real-time intrusion detection in the Internet of Things*. Ad Hoc Networks, vol. 11, no. 8, pages 2661–2674, 2013. (Cited in page 43.)

[Research 2022] Ettus Research. *Ettus Research website (USRP manufacturer)*. `https://www.ettus.com/`, 2022. Accessed: 2022-03-21. (Cited in page 27.)

[REV 2022] *RevEng Website*, 2022. Available at `https://sourceforge.net/projects/reveng/`. (Cited in page 32.)

[RFC 2021] *RFCat GitHub Repository*, 2021. Available at `https://github.com/atlas0fd00m/rfcat`. (Cited in page 32.)

[Ronen 2017] E. Ronen, A. Shamir, A. Weingarten and C. O'Flynn. *IoT Goes Nuclear: Creating a ZigBee Chain Reaction*. In 2017 IEEE Symposium on Security and Privacy (SP), pages 195–212, May 2017. (Cited in page 37.)

[Roux 2018] Jonathan Roux, Eric Alata, Guillaume Auriol, Mohamed Kaâniche, Vincent Nicomette and Romain Cayre. *Radiot: Radio communications intrusion detection for iot-a protocol independent approach*. In 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), pages 1–8. IEEE, 2018. (Cited in pages 27, 42, 77, and 100.)

[RTL-SDR 2022] RTL-SDR. *RTL-SDR website*. `https://www.rtl-sdr.com/`, 2022. Accessed: 2022-03-21. (Cited in page 27.)

[RUB 2021] *Ducky Script Quick Reference*, 2021. Available at `https://docs.hak5.org/usb-rubber-ducky-1/the-ducky-script-language/ducky-script-quick-reference`. (Cited in page 116.)

[Ryan 2013a] Mike Ryan. *Bluetooth: With Low Energy Comes Low Security*. In 7th USENIX Workshop on Offensive Technologies (WOOT 13), Washington, D.C., August 2013. USENIX Association. (Cited in pages 17, 30, 34, 43, 80, 81, 83, 92, and 149.)

[Ryan 2013b] Mike Ryan. *How Smart is Bluetooth Smart ?* 2013. (Cited in pages 35 and 115.)

[Santos 2019] Aellison Santos, José Filho, Avilla Silva, Vivek Nigam and Iguatemi Fonseca. *BLE injection-free attack: a novel attack on bluetooth low energy devices*. Journal of Ambient Intelligence and Humanized Computing, 09 2019. (Cited in page 81.)

[SCA 2022] *Scapy Website*, 2022. Available at `https://scapy.net`. (Cited in page 32.)

[Schroeder 2010] Thorsten Schroeder and Max Moser. *Keykeriki resources*, 2010. Available at `http://www.remote-exploit.org/articles/keykeriki_v2_0__8211_2_4ghz/`. (Cited in pages 31, 38, and 116.)

[Schulz 2017] Matthias Schulz, Efstathios Deligeorgopoulos, Matthias Hollick and Francesco Gringoli. *Demonstrating reactive smartphone-based jamming: demo*. Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks, 2017. (Cited in pages 17 and 30.)

[Schulz 2018a] Matthias Schulz, Jakob Link, Francesco Gringoli and Matthias Hollick. *Shadow Wi-Fi: Teaching smartphones to transmit raw signals and to extract channel state information to implement practical covert channels over Wi-Fi*. In Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services, pages 256–268, 2018. (Cited in page 28.)

[Schulz 2018b] Matthias Schulz, Daniel Wegemer and Matthias Hollick. *The Nexmon firmware analysis and modification framework: Empowering researchers to enhance Wi-Fi devices*. Computer Communications, vol. 129, pages 269–285, 2018. (Cited in pages 28, 29, and 30.)

[SEC 2015] *SecBee GitHub repository*, 2015. Available at `https://github.com/Cognosec/SecBee/`. (Cited in pages 36 and 115.)

[Shen 2013] Wenbo Shen, Peng Ning, Xiaofan He and Huaiyu Dai. *Ally friendly jamming: How to jam your enemy and maintain your own wireless connectivity at the same time.* In 2013 IEEE Symposium on Security and Privacy, pages 174–188. IEEE, 2013. (Cited in page 156.)

[Shintani 2020] Aiku Shintani. *The Design, Testing, and Analysis of a Constant Jammer for the Bluetooth Low Energy (BLE) Wireless Communication Protocol.* PhD thesis, California Polytechnic State University, San Luis Obispo, 06 2020. (Cited in pages 17 and 135.)

[SHO 2009] *shodan.io: Search Engine for the Internet of Everything*, 2009. Available at `https://www.shodan.io/`. (Cited in page 15.)

[Siby 2017] Sandra Siby, Rajib Ranjan Maiti and Nils Ole Tippenhauer. *IoTScanner: Detecting Privacy Threats in IoT Neighborhoods.* In Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security, IoTPTS '17, page 23–30, New York, NY, USA, 2017. Association for Computing Machinery. (Cited in pages 43 and 77.)

[Singh 2019] Ankit Singh, Aditi Sharma, Nikhil Sharma, Ila Kaushik and Bharat Bhushan. *Taxonomy of attacks on web based applications.* In 2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), volume 1, pages 1231–1235. IEEE, 2019. (Cited in page 15.)

[SOA 2022] *Soapy SDR GitHub Repository*, 2022. Available at `https://github.com/pothosware/SoapySDR`. (Cited in page 32.)

[Spill 2007] Dominic Spill and Andrea Bittau. *BlueSniff: Eve Meets Alice and Bluetooth.* In First USENIX Workshop on Offensive Technologies (WOOT 07), Boston, MA, August 2007. USENIX Association. (Cited in page 30.)

[Spill 2012] Dominic Spill. *Ubertooth One website*, 2012. `http://ubertooth.sourceforge.net/`. (Cited in pages 32 and 104.)

[Sung 2016] Yunsick Sung. *Intelligent Security IT System for Detecting Intruders Based on Received Signal Strength Indicators.* Entropy, vol. 18, no. 10, pages 1–16, October 2016. (Cited in page 44.)

[Szakaly 2016] Tamas Szakaly. *Help, I've got ANTs !!!* In DEF CON, volume 24, 2016. Available at `https://media.defcon.org/DEFCON24/DEFCON24presentations/DEFCON24-Tamas-Szakaly-Help-I-got-ANTS.pdf`. (Cited in pages 31 and 38.)

[Timmers 2016] Niek Timmers, Albert Spruyt and Marc Witteman. *Controlling PC on ARM Using Fault Injection.* In 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pages 25–35, 2016. (Cited in page 14.)

[Timmers 2017] Niek Timmers and Cristofaro Mune. *Escalating Privileges in Linux Using Voltage Fault Injection.* In 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pages 1–8, 2017. (Cited in page 14.)

[Tournier 2020] Jonathan Tournier, François Lesueur, Frédéric Le Mouël, Laurent Guyon and Hicham Ben-Hassine. *IoTMap: A protocol-agnostic multi-layer system to detect application patterns in IoT networks.* In 10th International Conference on the Internet of Things (IoT 2020), Malmö, Sweden, October 2020. (Cited in page 43.)

[Tsoutsos 2018] Nektarios Georgios Tsoutsos and Michail Maniatakos. *Anatomy of memory corruption attacks and mitigations in embedded systems.* IEEE Embedded Systems Letters, vol. 10, no. 3, pages 95–98, 2018. (Cited in page 15.)

[Ur Rehman 2012] S. Ur Rehman, K. Sowerby and C. Coghill. *RF fingerprint extraction from the energy envelope of an instantaneous transient signal.* In 2012 Australian Communications Theory Workshop (AusCTW), pages 90–95, 2012. (Cited in page 43.)

[Vaccari 2017] Ivan Vaccari, Enrico Cambiaso and Maurizio Aiello. *Remotely Exploiting AT Command Attacks on ZigBee Networks.* Security and Communication Networks, vol. 2017, pages 1–9, 10 2017. (Cited in pages 37, 69, and 168.)

[Vanhoef 2014] Mathy Vanhoef and Frank Piessens. *Advanced Wi-Fi Attacks Using Commodity Hardware.* In Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC '14, page 256–265, New York, NY, USA, 2014. Association for Computing Machinery. (Cited in page 30.)

[Vidgren 2013a] N. Vidgren, K. Haataja, J. L. Patiño-Andres, J. J. Ramírez-Sanchis and P. Toivanen. *Security Threats in ZigBee-Enabled Systems: Vulnerability Evaluation, Practical Experiments, Countermeasures, and Lessons Learned.* In 2013 46th Hawaii International Conference on System Sciences, pages 5132–5138, 2013. (Cited in page 77.)

[Vidgren 2013b] Niko Vidgren, Keijo Haataja, José Luis Patiño-Andres, Juan José Ramírez-Sanchis and Pekka Toivanen. *Security Threats in ZigBee-Enabled Systems: Vulnerability Evaluation, Practical Experiments, Countermeasures, and Lessons Learned.* In 2013 46th Hawaii International Conference on System Sciences, pages 5132–5138, 2013. (Cited in page 37.)

[Vilela 2011] Joao P Vilela, Matthieu Bloch, Joao Barros and Steven W McLaughlin. *Wireless secrecy regions with friendly jamming.* IEEE Transactions on Information Forensics and Security, vol. 6, no. 2, pages 256–266, 2011. (Cited in page 156.)

[Vishwakarma 2018] Gopal Vishwakarma and Wonjun Lee. *Exploiting JTAG and its mitigation in IOT: A survey.* Future Internet, vol. 10, page 121, 12 2018. (Cited in pages 13 and 59.)

[von Tschirschnitz 2021] M. von Tschirschnitz, L. Peuckert, F. Franzen and J. Grossklags. *Method Confusion Attack on Bluetooth Pairing.* In 2021 2021 IEEE Symposium on Security and Privacy (SP), pages 213–228, Los Alamitos, CA, USA, may 2021. IEEE Computer Society. (Cited in pages 35 and 176.)

[Wilhelm 2012] Matthias Wilhelm, Jens B Schmitt and Vincent Lenders. *Practical message manipulation attacks in IEEE 802.15. 4 wireless networks.* In MMB & DFT 2012 Workshop Proceedings, pages 29–31, 2012. (Cited in page 17.)

[Wright 2009] Joshua Wright. *KillerBee: Practical ZigBee Exploitation Framework*, 2009. Available at `http://www.willhackforsushi.com/presentations/toorcon11-wright.pdf`. (Cited in pages 28, 29, 32, 36, 104, and 115.)

[Wu 2020a] Jian-Liang Wu, Yuhong Nan, V. Kumar, Dave Tian, Antonio Bianchi, M. Payer and D. Xu. *BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy.* In WOOT @ USENIX Security Symposium, 2020. (Cited in page 35.)

[Wu 2020b] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Mathias Payer and Dongyan Xu. *BlueShield: Detecting Spoofing Attacks in Bluetooth Low Energy Networks.* In 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020), pages 397–411, San Sebastian, October 2020. USENIX Association. (Cited in pages 44 and 100.)

[Xu 2005] Wenyuan Xu, Wade Trappe, Yanyong Zhang and Timothy Wood. *The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks.* In Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '05, page 46–57, New York, NY, USA, 2005. Association for Computing Machinery. (Cited in pages 17 and 157.)

[Yang 2019] Hojoon Yang, Sangwook Bae, Mincheol Son, Hongil Kim, Song Min Kim and Yongdae Kim. *Hiding in plain signal: Physical signal overshadowing attack on {LTE}.* In 28th USENIX Security Symposium (USENIX Security 19), pages 55–72, 2019. (Cited in page 17.)

[Yaseen 2019] Muhammad Yaseen, Waseem Iqbal, Imran Rashid, Haider Abbas, Mujahid Mohsin, Kashif Saleem and Yawar Abbas Bangash. *MARC: A Novel Framework for Detecting MITM Attacks in eHealthcare BLE Systems.* Journal of Medical Systems, vol. 43, no. 11, page 324, 2019. (Cited in pages 44 and 100.)

[Yuniati 2019] Yetti Yuniati, Ardian Ulvan and Sitronella Nur Fitriani. *Signal analysis of remote control (RC) UAV used software defined radio (SDR) HackRF One*. Query: Journal of Information Systems, vol. 3, no. 01, pages 62–68, 2019. (Cited in page 27.)

[Zig 2015] Zigbee Alliance. *ZigBee Specification*, 2015. (Cited in page 10.)

[Zillner 2015] T. Zillner. *ZigBee Exploited: The good , the bad and the ugly.* In BlackHat, 2015. (Cited in page 36.)

[Zuo 2019] Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin and Yinqian Zhang. *Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps.* In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pages 1469–1483, 2019. (Cited in pages 11 and 99.)

Abstract

In the recent years, a new kind of devices, so-called connected objects, has been actively deployed, spreading everywhere in our daily life. These devices aim to interact with the physical world while providing a connectivity which is generally based on a new generation of wireless communication protocols such as Zigbee, Bluetooth Low Energy or Thread. The rapid and massive deployment of these new wireless technologies in the context of Internet of Things introduces new challenges from a security perspective. These challenges are mainly linked to the heterogeneity of these protocols, the decentralized and dynamical environments where they are deployed, and their co-existence in the same environments.This PhD thesis is a contribution to the security of wireless communication protocols, both from an offensive and a defensive perspective. We especially focus on the lower layers of the protocol stacks, which are very difficult to analyze.

The first offensive contribution of this work highlights some critical vulnerabilities which are linked to the design of the protocols and can't be fixed easily without significantly modifying the specification. We present the InjectaBLE vulnerability allowing an attacker to inject arbitrary packets into an established Bluetooth Low Energy connection. We also show that exploiting this vulnerability may allow the attacker to divert some low level mechanisms in order to perform complex attacks, such as hijacking or man-in-the-middle attacks.

We also worked on the security risks linked to the co-existence of heterogeneous wireless communication protocols in the same environments. Our second offensive contribution demonstrates the feasibility of diverting a Bluetooth Low Energy transceiver in order to interact with other protocols such as Zigbee or Enhanced ShockBurst, which are not natively supported by the chip. We highlight the practical feasibility of implementing such a strategy on multiple devices, including smartphones and connected objects, and we show that this offensive strategy allow pivoting attacks or covert-channel attacks, which are especially difficult to anticipate and mitigate.

The existence of such offensive strategies which are linked to the low level internals of the wireless communication protocols, requires the development of efficient mitigations, especially intrusion detection and prevention strategies. However, designing such mitigations remains a complex challenge because of the decentralization and dynamicity of wireless environments where connected objects are deployed. Our defensive contributions introduce two innovative defensive approaches, facilitating the deployment of Intrusion Detection Systems and Intrusion Prevention Systems in such environments. Our first contribution, named Oasis, demonstrates the feasibility of embedding detection mechanisms directly into the connected objects. We mainly show that such a defensive strategy allows the extraction of low level indicators which can be analyzed to perform a reliable detection of the main protocol attacks targeting the Bluetooth Low Energy protocol. Our second defensive contribution focuses on the intrusion prevention challenge, and introduces an approach based on reactive jamming to efficiently filter malicious traffic. We show the genericity of these prevention strategy by implementing it in practice on two wireless protocols commonly used in IoT: Zigbee and Enhanced ShockBurst.

Keywords— IoT, Security, Internet of Things, Protocols, Wireless Networks

Résumé

Ces dernières années, nous avons pu assister à l'émergence d'un nouveau type de systèmes informatiques, nommés objets connectés. Ces systèmes sont caractérisés par leur capacité à interagir avec le monde physique et par leur connectivité, celle ci étant généralement basée sur une nouvelle génération de protocoles de communication sans fil tels que Zigbee, Bluetooth Low Energy ou Thread. Le déploiement rapide et massif de ces nouvelles technologies de communication sans fil dans le contexte de l'Internet des Objets introduit de nouveaux défis pour la sécurité, liés à l'hétérogénéité des protocoles, la nature dynamique et décentralisée des environnements dans lesquels ils sont déployés ainsi que leur co-existence dans les mêmes environnements. Cette thèse se concentre sur la sécurité de ces nouveaux protocoles de communication sans fil, tant d'un point de vue offensif que défensif, et notamment sur les problématiques liées aux couches inférieures des piles protocolaires, dont l'analyse est particulièrement complexe.

La première contribution offensive de ces travaux met en évidence l'existence de vulnérabilités critiques liées au design des protocoles eux mêmes, qui ne peuvent être corrigées sans modifier significativement la spécification. Nous présentons notamment la vulnérabilité InjectaBLE, permettant à un attaquant d'injecter des paquets arbitraires au sein d'une communication Bluetooth Low Energy établie. Nous démontrons également que l'exploitation de cette vulnérabilité permet à l'attaquant de détourner des mécanismes bas niveau du protocole pour mettre en place des attaques plus complexes, telles que des usurpations d'identité ou la mise en place d'une attaque de l'homme du milieu.

Nous nous sommes également intéressé aux problématiques de sécurité résultant de la co-existence de protocoles sans fil hétérogènes au sein des mêmes environnements. Notre seconde contribution offensive démontre la possibilité de détourner le fonctionnement d'une puce Bluetooth Low Energy afin d'interagir avec des protocoles non nativement supportés par celle ci, tels que Zigbee, Enhanced ShockBurst ou Mosart. Nous montrons notamment que ce détournement est réalisable en pratique sur de nombreux équipements incluant des smartphones et des objets connectés, et qu'il peut permettre la mise en place d'attaques pivots et d'attaques par canaux cachés, difficiles à anticiper et corriger.

L'existence de telles stratégies d'attaque liées aux fonctionnement même des couches inférieures des protocoles sans fil nécessite la mise en place de contre-mesures efficaces, et notamment de stratégies de détection et de prévention adaptées. Cependant, leur mise en place reste aujourd'hui particulièrement complexe, notamment du fait de la nature décentralisée et dynamique des environnements concernés. Nos contributions défensives proposent deux approches défensives innovantes, destinées à faciliter le déploiement de système de détection et de prévention d'intrusion dans de tels environnements. Notre première contribution, Oasis, démontre la faisabilité d'embarquer des mécanismes de détection directement au sein des objets connectés. Nous montrons notamment que cette approche permet la collecte d'indicateurs bas niveau dont l'analyse permet une détection efficace des principales attaques protocolaires visant le protocole Bluetooth Low Energy. Notre seconde contribution défensive s'intéresse à la problématique de la prévention d'intrusion, et propose une approche basée sur une stratégie de brouillage réactif permettant de filtrer efficacement le trafic malveillant. Nous illustrons notamment la généricité de cette stratégie de prévention en l'implémentant sur deux protocoles communs de l'IoT: Zigbee et Enhanced ShockBurst.

Mots clés— IoT, Sécurité, Internet des Objets, Protocoles, Réseaux sans fil