UNIVERSITY OF TARTU
Institute of Computer Science
Cybersecurity Curriculum

Lucas Salvatore G Agro

Radio Based Analysis of Wireless Mice and Keyboards using Proprietary Protocols

Master's Thesis (21 ECTS)

Supervisor:
Danielle Melissa Morgan, MSc

Tartu 2025

Radio Based Analysis of Wireless Mice and Keyboards using Proprietary Protocols

Abstract:

Wireless mice and keyboards are widely used for convenience, but many rely on proprietary communication protocols with unknown levels of security. Unlike standards such as Bluetooth or Wi-Fi, these protocols are often undocumented, making it difficult to assess their resistance to eavesdropping or spoofing attacks. This thesis presents a practical methodology to evaluate the security of wireless mice and keyboards using proprietary communication protocols. The approach relies on capturing radio signals with a Software-Defined Radio, demodulating the transmissions, and reverse-engineering the protocol to understand its structure. Exploitation techniques were then implemented using a Crazyradio PA dongle and a custom Python script to sniff or spoof packets. A variety of devices from different brands, all available on the market, were analyzed. The results show that most of these peripherals are vulnerable to either sniffing, spoofing, or both. This work highlights a real and often overlooked threat, as many users rely on such wireless devices without being aware of the potential security risks they carry.

Keywords: Reverse-engineering, wireless devices, security analysis, SDR, HackRF, Universal Radio Hacker

CERCS: T120 Systems engineering, computer technology; T121 Signal processing

Raadiopõhine analüüs juhtmevabade hiirte ja klaviatuuride kohta, mis kasutavad patenteeritud protokolle

Lühikokkuvõte:

Juhtmevabad hiired ja klaviatuurid on laialdaselt kasutusel oma mugavuse tõttu, kuid paljud neist kasutavad patenteeritud sidesüsteeme, mille turvalisuse tase on teadmata. Erinevalt Bluetoothist või Wi-Fi-st on need protokollid sageli dokumenteerimata, mistõttu on raske hinnata nende vastupanuvõimet pealtkuulamisele või võltsimisrünnakutele. Käesolev magistritöö esitab praktilise metoodika juhtmevabade hiirte ja klaviatuuride turvalisuse hindamiseks, mis kasutavad patenteeritud sidesüsteeme. Lähenemine põhineb raadiosignaalide püüdmisel tarkvaramääratletud raadio abil, edastuste demoduleerimisel ja protokolli pöördprojekteerimisel, et mõista selle ülesehitust. Seejärel viidi läbi ärakasutamise tehnikaid kasutades Crazyradio PA USB-donglit ja spetsiaalselt loodud Python-skripti, millega oli võimalik pakette nuhkida või võltsida. Analüüsiti mitmeid turul saadaval olevaid erinevate kaubamärkide seadmeid. Tulemused näitavad, et enamik neist seadmetest on haavatavad kas pealtkuulamisele, võltsimisele või mõlemale. See töö toob esile reaalse ja sageli tähelepanuta jäetud ohu, kuna paljud kasutajad loodavad sellistele juhtmevabadele seadmetele, olles teadmatuses võimalikest turvariskidest.

Võtmesõnad: pöördprojekteerimine, juhtmevabad seadmed, turvaanalüüs, SDR, HackRF, Universal Radio Hacker

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; T121 Signaalitöötlus

Acknowledgements

I would like to sincerely thank my supervisor, Danielle Melissa Morgan, for her continuous support, responsiveness, guidance, and encouragement throughout the development of this thesis. Her knowledge played an important role in the achievement of this work. I am also thankful to Gerhard Klostermeier for his guidance in the interpretation of data after demodulation.

Table of Contents

1. Introduction

1.1. Motivation

A computer is typically controlled using a mouse and a keyboard. While wired peripherals have been used for decades, they often lead to cable clutter, which can be unaesthetic and inconvenient. To address this issue, manufacturers introduced wireless peripherals, eliminating the need for physical connections and allowing for more organized workspaces with an acceptable amount of latency. However, new security vulnerabilities have emerged with the transition to wireless technology, posing significant risks to users and organizations.

Wireless peripherals can use the Bluetooth protocol to transmit wireless data to the computer. The problem is that not every computer possesses a Bluetooth adapter, so constructors mainly provide a USB dongle along with the peripherals. The latter sends radio frequency signals over the air that are received, demodulated, and interpreted by the USB dongle; the resulting action is then transmitted to the computer like a normal wired peripheral would do. The structure of the radio frequency (RF) signals depends on unstandardized proprietary protocols created by the manufacturers and can be different across the brands.

While most of those proprietary protocols are designed to be efficient, security is often not the primary focus. Most of them lack transparency, making it difficult to assess their security measures. As a result, attackers may exploit weaknesses in these protocols to intercept keystrokes, inject malicious commands, or take control of a system remotely. Those security risks seriously affect individuals, businesses, and government organizations relying on those wireless peripherals for daily tasks. Despite the many discussions on these risks [1], there is currently no established methodology to test these devices and identify potential security flaws.

1.2. Contribution

This thesis aims to analyze wireless keyboards and mice that have not been previously tested and use a proprietary protocol. This process involves reverse-engineering the communication protocols of these wireless devices to understand how they work through radio-based analysis. Once the protocol is understood, it is possible to assess the presence of any security mechanisms, such as encryption, encoding, or authentication. Since decryption is outside the scope of this research, an in-depth analysis was not performed on devices using encryption

mechanisms. Once the protocol was fully understood for the remaining devices, a Python script was written to capture keystrokes and inject custom inputs [2].

The main objectives of this thesis are as follows:

- **Present a comprehensive methodology for testing wireless devices.** This methodology will detail a systematic approach for reverse-engineering and analyzing wireless communication protocols, enabling the readers of this thesis to test various other wireless devices.
- **Document the results of the devices tested.** Each entry will indicate whether the communication protocol has been successfully reverse-engineered and whether the device contains vulnerabilities.
- **Create a tool that enables attacks on vulnerable devices.** The tool will propose attacks on all vulnerable devices while allowing others to follow the methodology and extend the tool to target new devices. The code will be written to be easy to understand and easily modifiable.

1.3. Research questions

The research questions of this thesis are the following:

- [RQ1] Are wireless devices using proprietary protocols, still available on the market, vulnerable to security attacks?

  Such security attacks include:

  - [RQ1.1] Sniffing: Can the signals transmitted by these devices be intercepted and analyzed to reveal sensitive information, such as keystrokes?
  - [RQ1.2] Spoofing: Is injecting malicious keystrokes into these devices possible?
- [RQ2]: Can a methodology be defined to aid in the process of testing the security of wireless mice and keyboards?

1.4. Scope

The scope of this thesis is as follows:

- The analysis will focus on devices that are available for purchase in stores. All devices included in the study were bought in Belgian stores during 2024 and 2025.
- Only devices that use USB dongles and proprietary communication protocols will be analyzed; Bluetooth devices are excluded from the thesis.
- The analysis will be radio-based, meaning it will be conducted on the signals emitted by the devices, without the use of logic analyzers or firmware extraction.
- Devices using encryption mechanisms will not decrypted.

1.5. Thesis outline

Chapter 2 provides the background knowledge necessary to understand this thesis. Chapter 3 explores the state of the art in reverse-engineering with a radio-based analysis and exploiting wireless peripherals using proprietary protocols. Chapter 4 presents the methodology used to analyze and exploit the target devices. Chapter 5 applies this methodology to seventeen devices from ten different brands. Chapter 6 concludes the thesis by summarizing the work, answering the research questions, and discussing potential future research.

1.6. Use of Artificial Intelligence

GPT-4o and GPT-4o Mini were used to improve the readability of this thesis by shortening the initial text. Additionally, Grammarly was utilized to correct errors and improve clarity.

2. Background

This chapter provides the background knowledge required to understand the work of this thesis. Chapter 2.1 is focused on explaining how radio waves can be used in the context of wireless peripheral communication. Chapter 2.2 introduces the tools used in the thesis for analyzing and creating wireless signals. Chapter 2.3 takes time to explain how proprietary wireless mice and keyboards usually work and gives a high-level idea about what a proprietary protocol for such devices usually looks like.

2.1. Wireless signal

When using wired keyboards and mice, data is transmitted directly to the computer via a cable. Each key press or mouse movement is instantly converted into an electrical signal and sent through the wire. In the case of wireless peripherals, this role is fulfilled by radio waves. These waves enable information to be exchanged between the device and its receiver without any physical connection.

This chapter covers the key characteristics of radio waves, modulation and demodulation processes, and data encoding methods that govern the principles of wireless peripherals communication.

2.1.1. Radio waves

Radio waves are electromagnetic waves characterized by their wavelength. They are widely used for transmitting information over distances since their relatively long wavelengths allow them to travel far and penetrate various materials. The wavelength also affects the amount of data that can be sent. Figure 1 presents a visual representation of a radio wave.

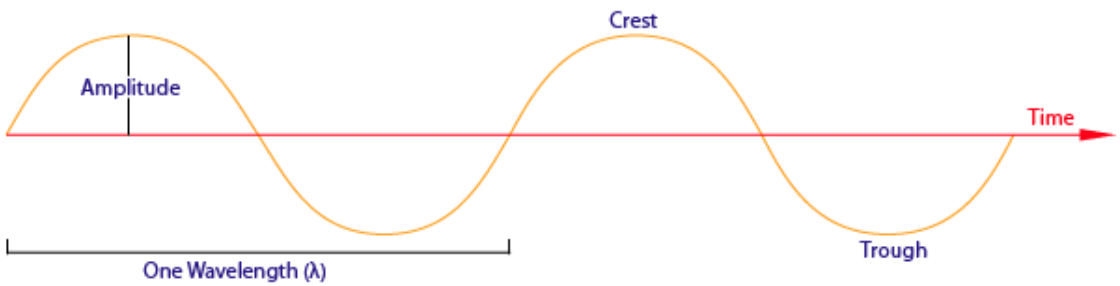


Figure 1. Visual representation of a radio wave [3]

Radio waves are measured in frequency, expressed in Hertz (Hz), which refers to the rate of oscillation of a wave. Waves with higher frequencies have shorter wavelengths and vice versa.

The amplitude of a wave is the distance between the crest or trough and the wave and its rest point. The bigger the amplitude is, the stronger the wave.

2.1.2. Modulation

Modulation is the process of encoding binary data into a periodic wave called a carrier signal. The data is embedded into the carrier signal by altering its characteristics, such as amplitude or frequency, to represent a series of bits.

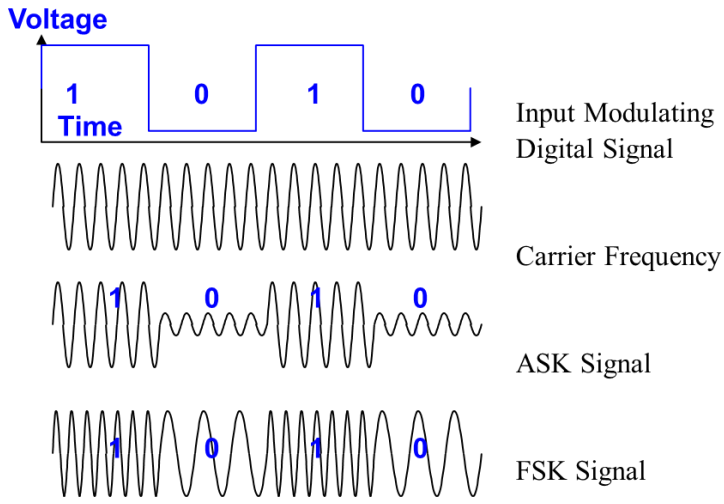


Figure 2. Visual representation of ASK and FSK modulation. Adapted from [4]

As depicted in Figure 2 there are multiple types of modulation techniques:

- **ASK (Amplitude Shift Keying):** Modifies the amplitude of the carrier signal to encode data.
- **FSK (frequency-shift keying):** Modifies the frequency of the carrier signal to encode data.
- **GFSK (Gaussian Frequency Shift Keying):** A refined version of FSK that produces cleaner signals by using a Gaussian filter [5]. This modulation is widely used in wireless keyboards and mice.

There are also other types of modulation, but they are not important for this thesis.

2.1.3. Demodulation

Demodulation is the reverse process of modulation, extracting the original data (bit sequences) from a modulated wave. The process of demodulation can be done manually or automatically using algorithms. Figure 3 presents a wave modulated using FSK, by assuming that the part of the wave with a higher frequency (shorter wavelengths) represents the binary "1" and the wave with shorter frequency (longer wavelengths) represents the binary "0", we get the following bit sequence: "010101100".

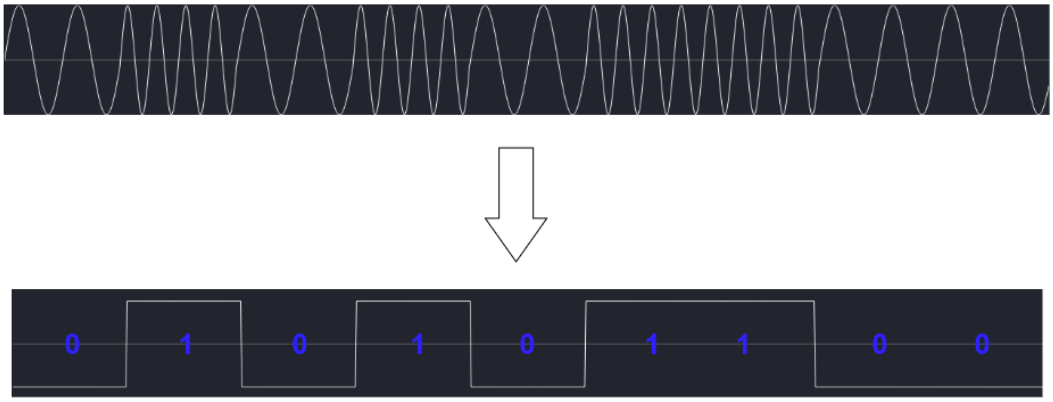


Figure 3. Demodulation of an FSK-modulated radio wave

2.1.4. Data encoding techniques

Before transmitting radio waves, the binary (See *Input Modulating Digital Signal* in Figure 2) is first encoded. Here are some common data encoding techniques:

- **Non-Return-to-Zero (NRZ):** Each bit is directly mapped to high or low voltage levels. This is the simplest and most used type of encoding. (Figure 4 A)
- **Return-to-Zero (RZ):** The signal returns to a baseline voltage between each bit. (Figure 4 B)
- **Manchester Encoding:** Each bit is represented by a transition, making it easier to synchronize data reception. (Figure 4 C)

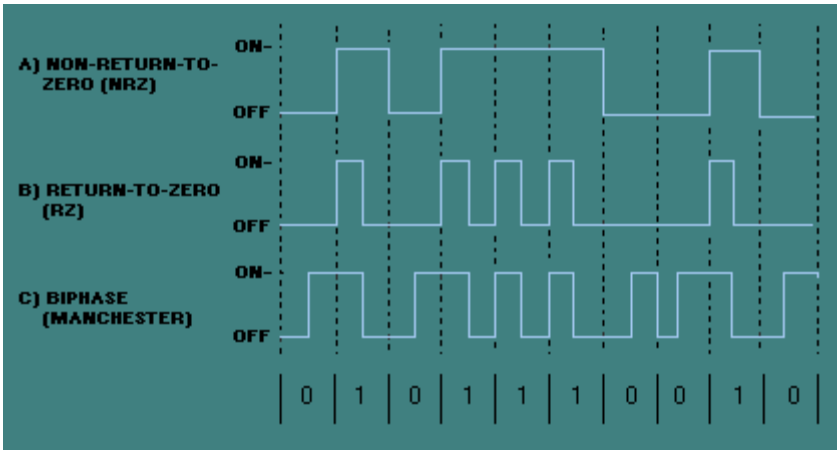


Figure 4. Visual representation of three data encoding techniques: NRZ, RZ, and Manchester [6]

2.2. Analysis of radio waves

This chapter explains the techniques used to analyze radio waves. Chapter 2.2.1 discusses hardware, while Chapter 2.2.2 covers software tools.

2.2.1. Hardware

SDR

SDR stands for Software-Defined Radio, a technology that uses software to process radio signals. When referring to SDR, it usually implies hardware capable of capturing and analyzing radio waves. It is also possible to generate and send signals with an SDR.

Various SDR models exist, such as the BladeRF, USRP, and HackRF One. The HackRF One was used in this study because it is relatively affordable (~$300), supports a broad frequency range, and has extensive documentation [7]. During experiments, it operated with firmware version 2023.01.1 and an antenna ANT700 [8].

Figure 5. Picture of a HackRF One [7]

Dongles

It is also possible to use dongles to analyze radio waves. Those dongles possess a Radio Frequency chip, and their utilization is more straightforward than an SDR. However, they offer less flexibility, as they often have fixed configurations for parameters such as the frequency, the modulation and demodulation, the data rate, and the maximum size of the packets that can be sent. On the other hand, dongles are optimized and are faster to send and receive data than SDRs.

Figure 6. Picture of a Crazyradio PA [9]

The Crazyradio PA was used in this thesis. It is a dongle initially designed to work with a drone. The advantage of this dongle is that it possesses an nRF24 chip, an RF chip widely used in wireless peripherals [9].

2.2.2. Software

GNU radio companion

GNU Radio is a software framework for controlling SDRs. It requires significant knowledge to use it and is not beginner-friendly. However, advanced users can use GNU radio companion to fully manipulate SDR functionalities. As a graphical programming tool, GRC allows real-time signal processing without requiring extensive coding.

Spectrum analyzer

A spectrum analyzer is a software that continuously scans a frequency range to visualize active signals. This is useful when analyzing new hardware and determining which frequencies are used. Many spectrum analyzer tools are compatible with the HackRF One. The HackRF Spectrum Analyzer is one of the simplest and was used in this study.

Universal Radio Hacker

Universal Radio Hacker (URH) is a software that allows the visualization of signals captured by an SDR. This tool is open-source and well-documented. It possesses functionalities to automatically detect the type of modulation used and the wave data. It also supports automatic demodulation, which saves a lot of time when working with signals containing significant amounts of data.

An interpretation tab enables users to examine demodulated bits, identify patterns, and reverse-engineer protocols [10].

2.3. Proprietary protocols of wireless keyboards and mice

Most wireless keyboards and mice operate in the 2.4 GHz ISM (Industrial, Scientific, Medical) band. It is a large band of frequency reserved for applications with short to medium-range communication such as Bluetooth, Wi-Fi and IoT devices.

Proprietary protocols are designed to be compatible with any host, even in the absence of a Bluetooth adapter. A USB dongle is provided by the manufacturers to allow communication between the peripherals and the host, serving as a bridge.

Figure 7 presents a simplified sequence diagram of wireless peripheral communication. Since the protocols are proprietary, there can be some differences. However, the overall communication flow remains largely the same.

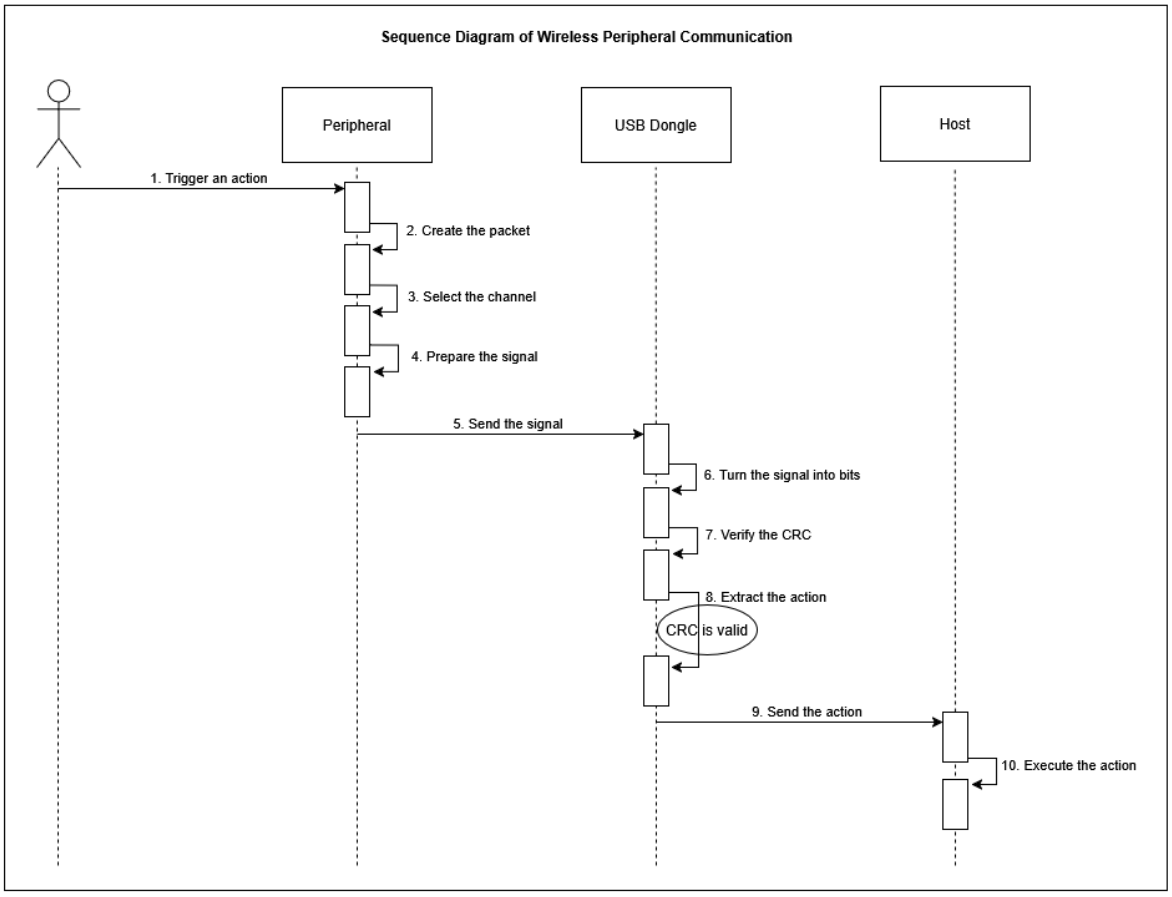


Figure 7. Sequence diagram of wireless peripheral communication

1. **Trigger an action**: The user triggers an action such as pressing a keyboard or moving the mouse.
2. **Create the packet**: The peripheral creates a packet based on its proprietary protocol. Most proprietary protocols follow a similar structure, making them easy to identify. An example is provided in Figure 8.

Preamble	Synchronization	Address	Sequence number	Data	CRC

Figure 8. Example of a structure of a proprietary protocol

- **Preamble:** A series of alternating 0s and 1s that indicate an incoming packet. Preambles often use patterns like 0101 0101... or 1010 1010....

- **Synchronization:** A short sequence of bits sometimes placed between the preamble and the address to avoid confusion when both share similar bit patterns. It helps to identify the start of the address.
- **Address:** A unique identifier for the transmitting device. This is useful when a single dongle handles both a keyboard and a mouse, as they have distinct addresses.
- **Sequence Number:** Helps detect packet loss or duplicates.
- **Data:** Protocol-specific information, such as keypresses or mouse movement.
- **CRC (Cyclic Redundancy Check):** A checksum mechanism that ensures data integrity. The dongle processes the packet through a CRC function, rejecting it if the output does not match the expected value.

3. **Select the channel**: The peripheral chooses a channel to send the packet. The channel tells in which frequency data will be sent. Channel 01 corresponds to 2401MHz, 02 to 2402MHz, and so on. The list can contain channels far from each other (e.g., [05, 14, 38, 52, 64, 73]).
4. **Prepare the signal**: The packet from step 2 is encoded using data encoding technique and embedded into a generated carrier signal using modulation.
5. **Send the signal**: The signal from step 4 is sent over the air to the correct channel and intercepted by the USB dongle.
6. **Turn the signal into bits**: The USB dongle decodes and demodulates the signal to turn it into bits. Those bits correspond to the packet created by the peripheral in step 2.
7. **Verify the CRC**: The packet is inputted without the CRC part to a CRC function; if the output is similar to the CRC part, data integrity is ensured.
8. **Extract the Action**: The USB dongle extracts the action triggered by the user. This corresponds to the Data part in Figure 8.
9. **Send the action**: The action from step 8 is sent to the host using the USB protocol.
10. **Execute the action**: The host executes the action from step 8.

This communication flow is simplified. Usually, the dongle sends an acknowledgment (ACK) whenever it receives a legitimate packet. If the peripheral does not receive any acknowledgment, it resends the message. Figure 7 also does not picture what happens if the peripheral sends the packet to the wrong channel. Typically, the peripheral sends multiple packets to the same channel and switches after X number of messages sent without any ACK

until it finds the correct channel. Those particularities were not depicted in the sequence diagram, as the behavior may change across the different proprietary protocols.

3. State of the art

This chapter mentions the work already made and publicly available using radio-based analysis to reverse-engineer and/or target mice and keyboards using proprietary protocols. The resources are not presented in chronological order.

3.1. Bastille

Bastille is a cybersecurity company that focuses on identifying and mitigating security risks related to wireless devices. The company has released a series of tools and conducted research in the domain of wireless threat detection [11].

3.1.1. MouseJack

MouseJack is a well-known set of vulnerabilities affecting non-Bluetooth wireless keyboards and mice. The researchers have found vulnerabilities in many peripherals from 7 different vendors. They showed that the protocols used by those peripherals are often unsafe. They managed to intercept and eavesdrop on the communication between the targeted peripherals and their dongles, even if strong encryption schemes such as AES were used. It was also shown that it is possible to inject malicious keystrokes, leading to executing malicious actions on unauthorized devices. [12] [13].

3.1.2. KeySniffer

KeySniffer is another set of vulnerabilities identified by Bastille, affecting a variety of non-Bluetooth wireless keyboards from 8 different vendors. Bastille showed that many inexpensive wireless keyboards do not encrypt data, which allows for simple eavesdropping. Where MouseJack focused on devices using the nRF24 chip, KeySniffer works with undocumented RF chips. This means the researchers had to reverse-engineer the protocol before searching for any vulnerabilities [14].

3.1.3. KeyJack

KeyJack is yet another set of vulnerabilities targeting wireless keyboards that use encryption. It allows an attacker to send encrypted keystrokes to a victim's computer without having to know the encryption key. Well-known brands such as Amazon-basics, Dell, Lenovo or Logitech are vulnerable to this attack [15].

3.2. SySS-Research

SySS-Research is a German IT security company. It has a branch that specializes in testing wireless security. The group has been actively analyzing 2.4 GHz wireless input devices, including mice, keyboards, and presentation pointers. They created an open-source tool and proposed multiple webinars to explain how they were conducting their attacks [16] [17].

3.2.1. Keyjector

Keyjector is a fork of a tool developed by Marc Newlin, who was heavily involved in creating MouseJack, KeySniffer, and KeyJack. It is an open-source collection of tools, developed by Matthias Deeg and Gerhard Klostermeirer, targeting previously untested wireless mice, keyboards, and presentation pointers [18].

3.3. Penetration testing wireless keyboards

Niklas Tomsic published a master's thesis in 2022 for the KTH Royal Institute of Technology, which involves pentesting 10 different wireless keyboards using proprietary protocols. Out of the 10 keyboards, he could attack 9 of them, knowing that 4 had the same protocols. The attack consisted either of sniffing the communication between the keyboards and their dongles or injecting malicious keystrokes. The author added his plugins to JackIt (see Chapter 3.4.4) to attack the keyboards [19].

The author showed that most wireless keyboards sold today are still vulnerable. However, even if there are some explanations regarding the methodology used during his pentest, he stays very high-level. A simple reader without advanced knowledge of the topic cannot reproduce his methodology. Concerning the attacks, he shows the result of an attack executed with a keylogger without providing the code, and his plugins added to JackIt do not contain much explanation.

3.4. Other useful resources

3.4.1. KeyKeriki v2.0

KeyKeriki v2.0 is an advanced tool that exploits vulnerabilities in wireless keyboards, particularly those operating over the 2.4 GHz band. Like Keyjack and KeySniffer, KeyKeriki allows attackers to perform keystroke injection attacks, potentially compromising the target system's security. KeyKeriki is part of a broader set of tools demonstrating the risks of insecure

wireless devices. The tool allows attackers to take control of a system or gain access to sensitive data by exploiting flaws in the wireless keyboard's communication protocol [20].

3.4.2. nRF24L01+ promiscuity

The nRF24L01+ is a widely used 2.4 GHz transceiver module. Travis Goodspeed found in 2011 a way to turn that transceiver module into promiscuous mode. In this mode, the module can receive all radio signals in the area, not just those intended for it. This capability makes the nRF24L01+ useful for intercepting and analyzing wireless communication between vulnerable devices, such as keyboards, mice and their dongles [21].

3.4.3. KeySweeper

KeySweeper is a stealthy surveillance device that looks like a USB wall charger. It is designed to target Microsoft wireless keyboards and to intercept the generated keystrokes. It can decrypt and log every keystroke online and locally and transmit this information to the attacker. Finally, it can send SMS if it detects specific keywords, sentences, or URLs. One of the key findings in this project was the realization that the keyboards' encryption is weak. It uses the keyboard's MAC address in a simple XOR cipher, which is predictable and vulnerable [22].

3.4.4. JackIt

JackIt is a tool that seeks to exploit the vulnerabilities disclosed by MouseJack. It has a plugin for each vendor, which tells the program how to identify, interpret, and inject packets from that specific vendor into keyboards. When launching, JackIt scans the 2.4GHz frequency band, seeking devices supported by one of the plugins. The user can inject malicious keystrokes when a vulnerable device is found [23].

3.5. Research gap

The research and tools presented in this chapter highlight the security vulnerabilities of wireless peripherals using proprietary protocols. Bastille's work has demonstrated critical flaws in multiple keyboards and mice, primarily focusing on exploiting weaknesses in nRF24-based protocols and other undocumented RF chips. Other contributions, such as those from SySS-Research and the master's thesis by Niklas Tomsic, have expanded on these findings by testing various commercial devices and developing tools to facilitate their exploitation. However, despite these advancements, several gaps remain in the existing research.

First, most research in this area has focused on previously analyzed NRF24-based protocols. While some studies have explored other brands of RF chips, the coverage remains limited compared to the vast number of proprietary protocols available on the market.

Second, although Niklas Tomsic [16] successfully demonstrated vulnerabilities in wireless keyboards, his methodology lacks sufficient technical details, making it challenging for researchers to reproduce his results.

Lastly, existing open-source tools suffer from poor documentation. There are no clear guidelines on modifying the code to add exploits for new devices, limiting their usability for further research.

In conclusion, this thesis aims to discover, reverse-engineer, and exploit new protocols. It will also provide a detailed methodology to help readers reverse-engineer their devices and develop an open-source tool with clear, extensible code, making it easier to integrate new exploits.

4. Methodology

This chapter presents a methodology for analyzing, reverse-engineering, and exploiting wireless keyboards and mice communicating via a USB dongle. The methodology consists of a series of steps.

4.1. Sniffing USB packets

Although the focus is on analyzing wireless devices, USB communication still occurs between the peripheral's dongle and the computer. Intercepting this communication can provide valuable information that is helpful for further analysis.

On Windows, USBPcap can be used to capture USB traffic. It is a plugin that integrates with Wireshark [24]. Before using USBPcap, selecting the "**Capture from newly connected devices**" option is useful. This ensures that only communication from devices connected after USBPcap starts is captured, reducing noise from other USB devices.

On Linux, USB traffic can be dumped using the *usbmon* kernel module. An explanation on how to set up a Linux or a MacOS-based device for USB sniffing can be found on the Wireshark website [25].

When a USB device is connected, the host requests information as depicted in Figure 9, such as the vendor ID, product ID, and device name. This information is helpful for Chapter 4.2. It also queries the device's configuration, which includes the supported interfaces (e.g., keyboard, mouse, microphone, game controller). This information can be especially useful when considering attacks against a wireless mouse. If the mouse's dongle supports a keyboard interface, like in Figure 10, it may be possible to inject keyboard packets.

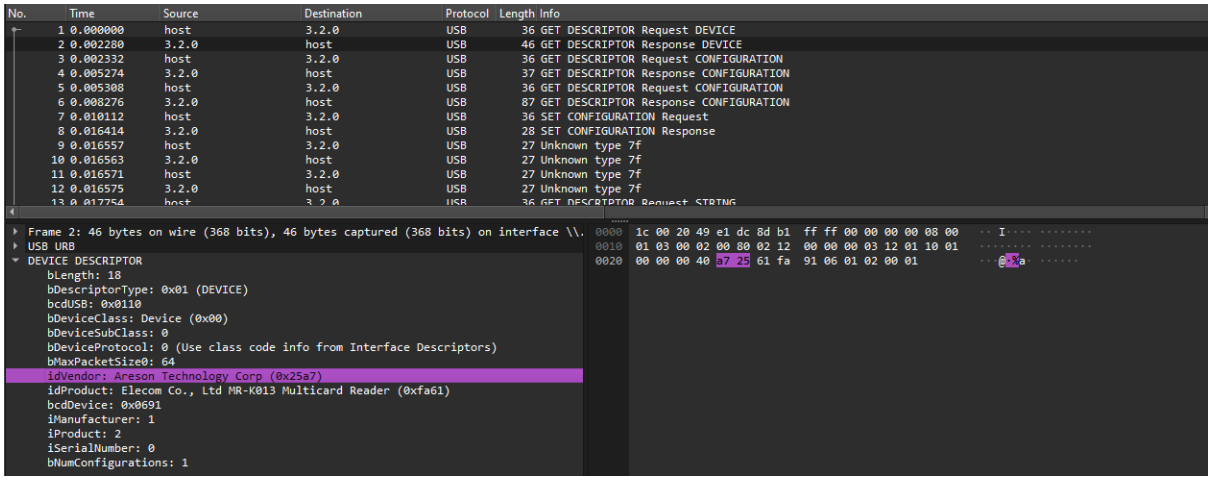


Figure 9. Vendor ID and Product ID sniffed with USBPcap

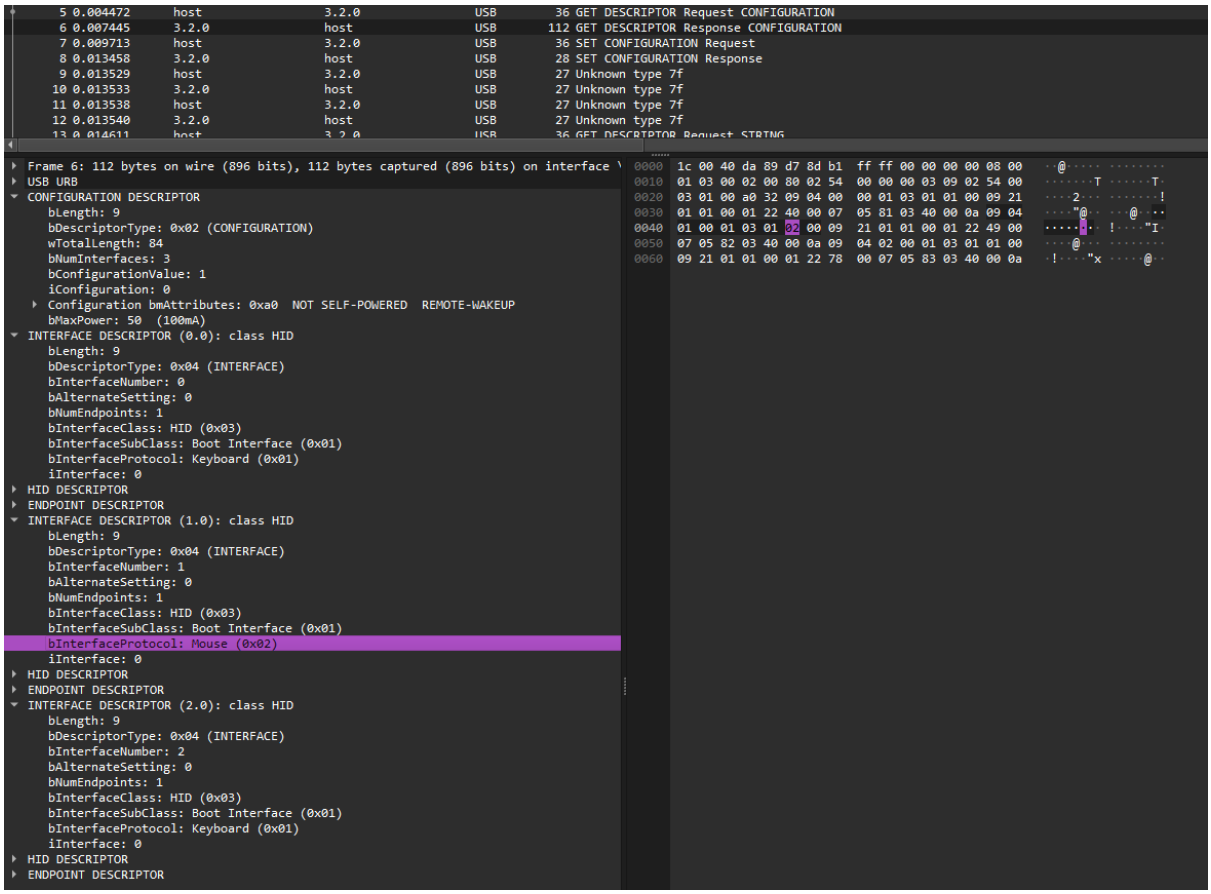


Figure 10. Mouse dongle that also possesses keyboard interfaces

Another good source of information is the HID (Human Interface Device) data. When a mouse is moved, or a keyboard key is pressed, the device transmits HID data to the computer, which describes the performed actions. The contents of HID data such as the number of buttons or

the bit size of variables provide clues about how data is formatted within the protocol and the use of this data for reverse-engineering purposes will be explained in Chapter 4.6.2.

Upon connection, the device sends an HID report (e.g. Figure 11) for each interface, informing the computer how to interpret the incoming HID data.

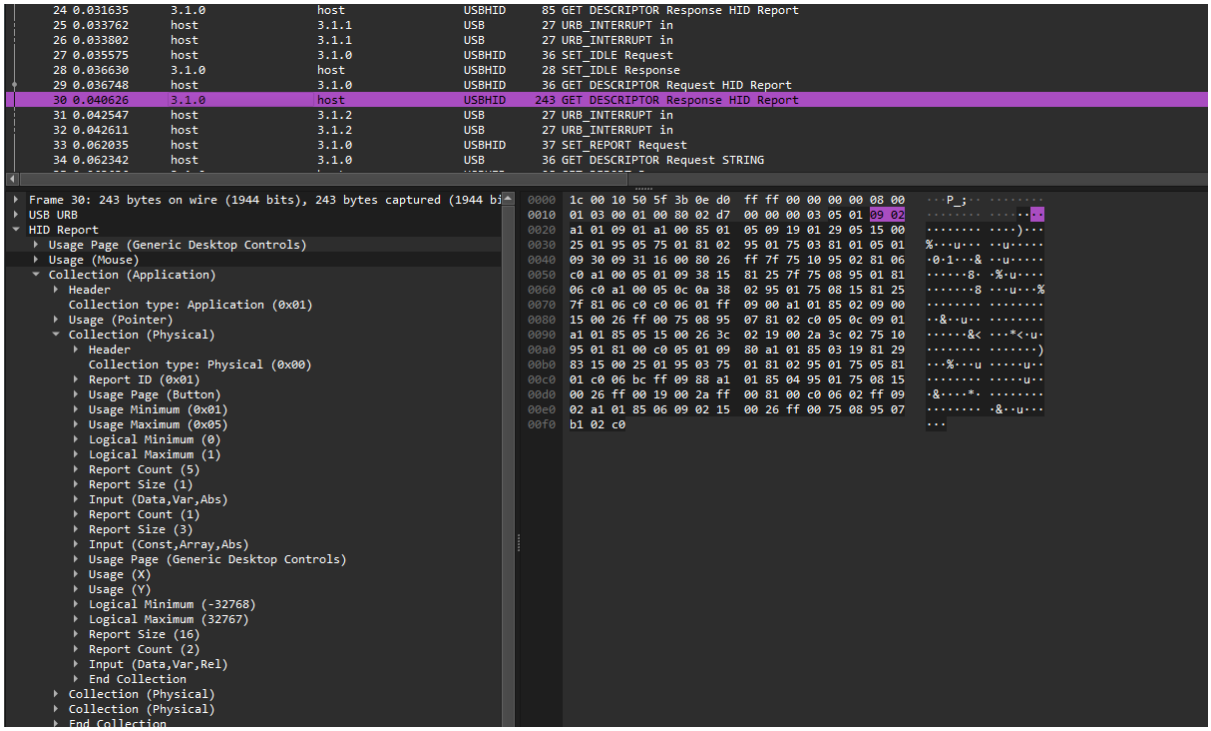


Figure 11. HID report captured with USBPcap

For a mouse, HID data, as seen in Figure 12, typically includes:

- The states of buttons, where a "down" state indicates a button is pressed.
- Movement of the mouse along the X and Y axes.
- The scroll wheel represented as a signed variable (allowing negative values).
- The AC pan indicating horizontal scrolling (left or right movement of the scroll wheel).

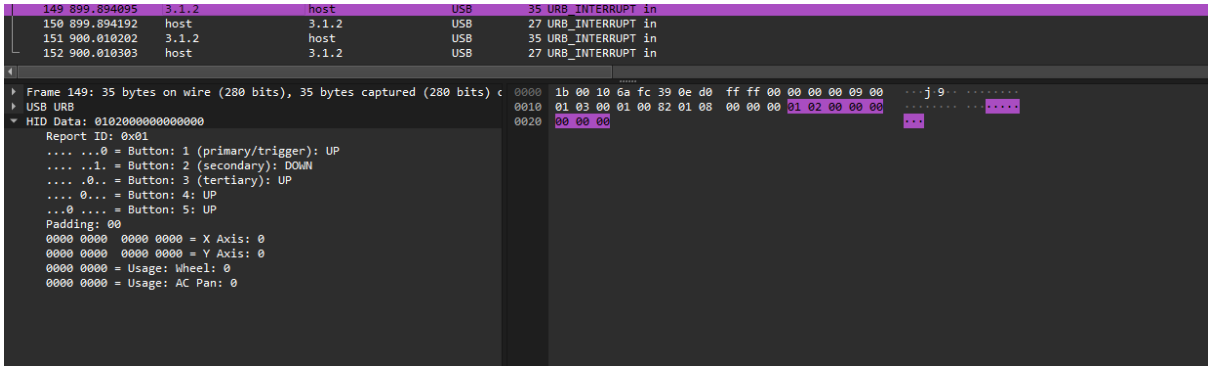


Figure 12. HID data containing a right click sent by a mouse to the host

For a keyboard, the HID data, as seen in Figure 13, generally consists of:

- The states of the modifiers, which refer to keys such as Left or Right Shift, Control, Alt, and GUI (Windows or Command key).
- A six-element array that lists the currently pressed keys, allowing multiple simultaneous key presses. The keys are assigned to scancodes (e.g., the 'a' key corresponds to scancode 0x04, 'b' to 0x05).

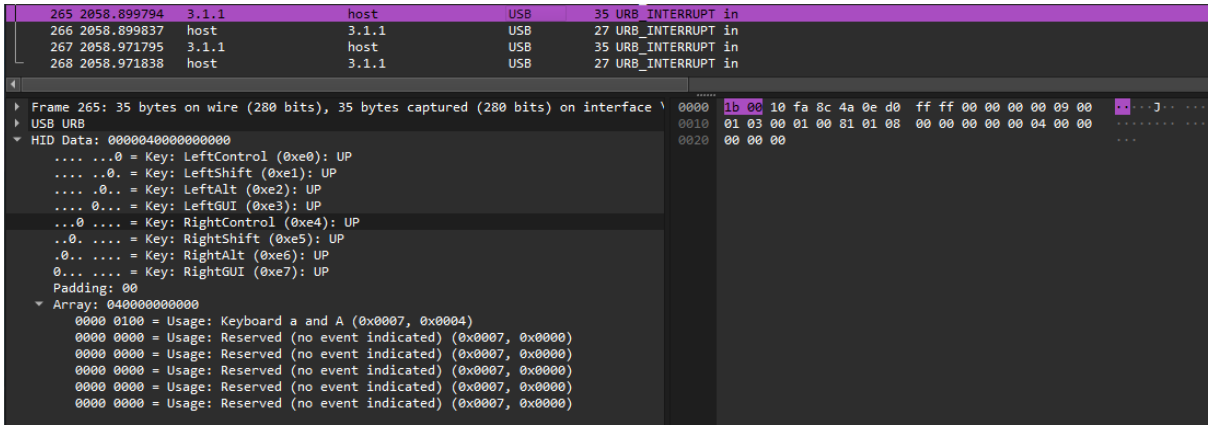


Figure 13. HID data containing the letter ‘a’ sent by a keyboard to the host

It is important to note that HID reports can vary slightly. For example, some mice use 24-bit instead of 32-bit values to represent movement along the X and Y axes.

4.2. OSINT

Once key data such as the Vendor ID and Product ID have been obtained, conducting open-source intelligence (OSINT) research can provide additional information about the devices to make the reverse-engineering process as easy as possible. The Vendor ID and Product ID are used to identify a device uniquely. Various databases, such as [26], can be used to gather

information about these identifiers. In many cases, the Product ID may not be listed, as it often corresponds to the dongle rather than the device itself. However, details about the vendor, its website, and other products manufactured by the company may still be available. Searching for the product there might reveal useful specifications if the vendor has an official website. Alternatively, performing a search engine query with the device name can help locate third-party resellers, who sometimes provide additional details.

When working with lesser-known brands, finding a completely different company name associated with the Vendor ID is not uncommon. This often indicates that the vendor has not configured its device with a unique Vendor ID and instead uses the Vendor ID of the RF chip manufacturer. This information can be valuable, as it may lead to technical documentation describing the operation of the RF chip.

Additionally, many devices have an FCC ID printed on them. The Federal Communications Commission (FCC) requires wireless devices to undergo testing before being marketed in the United States [27]. If an FCC ID is found on the dongle or device, searching for it on databases such as [28] or [29][1] provides access to extensive documentation. This may include internal pictures of the product, revealing the RF chip in use. The databases also contain test reports detailing variables such as modulation type, data rate, and channel usage, though, in practice, the reported channels are often inaccurate.

4.3. Opening the device

This step is optional. If limited information is found in the previous stage, opening the device can provide information about the components used. However, this approach was barely employed in this thesis, as it risks damaging the devices. This is especially true for dongles, which often remain physically compromised even after reassembly.

It is also common for the RF chip to be concealed under a potting compound, such as epoxy. This obfuscation technique protects the chip from dust while also preventing reverse-engineering. Although epoxy can be removed using a heat gun or chemical solutions, these methods may further damage the device.

[1] The fccid.io website seems to be down at specific hours of the day. It might be hosted in the US and automatically down during the night.

Opening the device can also be useful for connecting a logic analyzer or extracting firmware, but this thesis did not apply such techniques.

4.4. Identifying the channels

Before analyzing the wireless signals, it is necessary to determine the frequencies on which they are transmitted. Any spectrum analyzer can be used for this purpose. For this thesis, the HackRF Spectrum Analyzer was utilized. An example of this software can be seen in Figure 14. The waves on the top represent the radio signals captured by the HackRF, with the x-axis being the frequency in MHz and the y-axis being the strength in decibels. Below the main graph is a spectrogram, where time progresses downward. Colors represent signal strength; blue areas indicate low power, while the orange lines show strong signals.

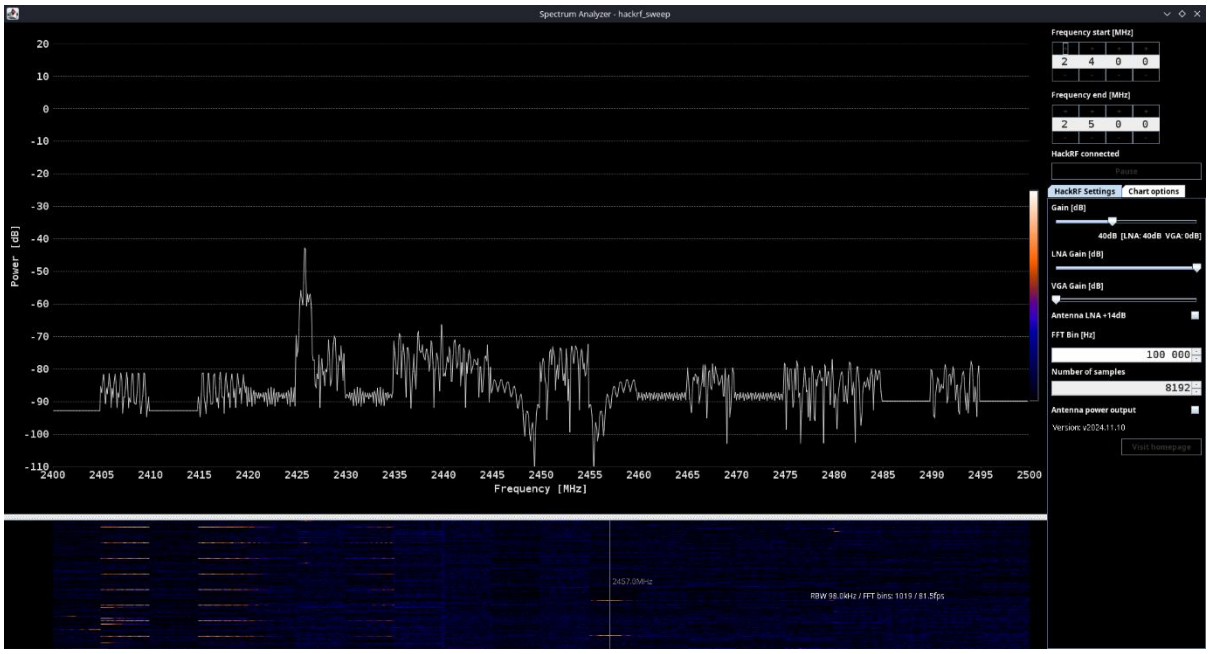


Figure 14. Visualization of the 2.4GHz frequency band with HackRF Spectrum Analyzer

To determine which channels are in use, the target device must be placed close to the SDR, within about 15 cm as done during the experiments. Then, an interaction, such as a key press or mouse movement, is needed to trigger radio transmissions. Using a mouse is preferable, as it typically transmits more signals than a keyboard. While the device is transmitting, new bright lines should appear in the spectrogram, which indicates active frequencies. For example, in Figure 14, a distinct signal can be observed at 2457 MHz whenever the device is triggered.

Wireless peripherals have a frequency hopping mechanism, which can be implemented in various ways. For example, the frequency may change after a set number of packets or when

excessive noise is detected on the current channel. It is, therefore, possible to see multiple bright lines appearing chaotically on different frequencies.

The 2.4GHz band is heavily used by Wi-Fi and Bluetooth, making the discovery of the frequency used by the peripherals difficult. Figure 15 shows this issue; the long orange lines on the left are Wi-Fi packets, while the other isolated lines are Bluetooth packets. To minimize this issue, Bluetooth devices should be put away from the SDR during analysis. If identifying the correct channels remains challenging, a Faraday cage can be used to block external signals. Alternatively, conducting the analysis in a low-interference environment is another option.

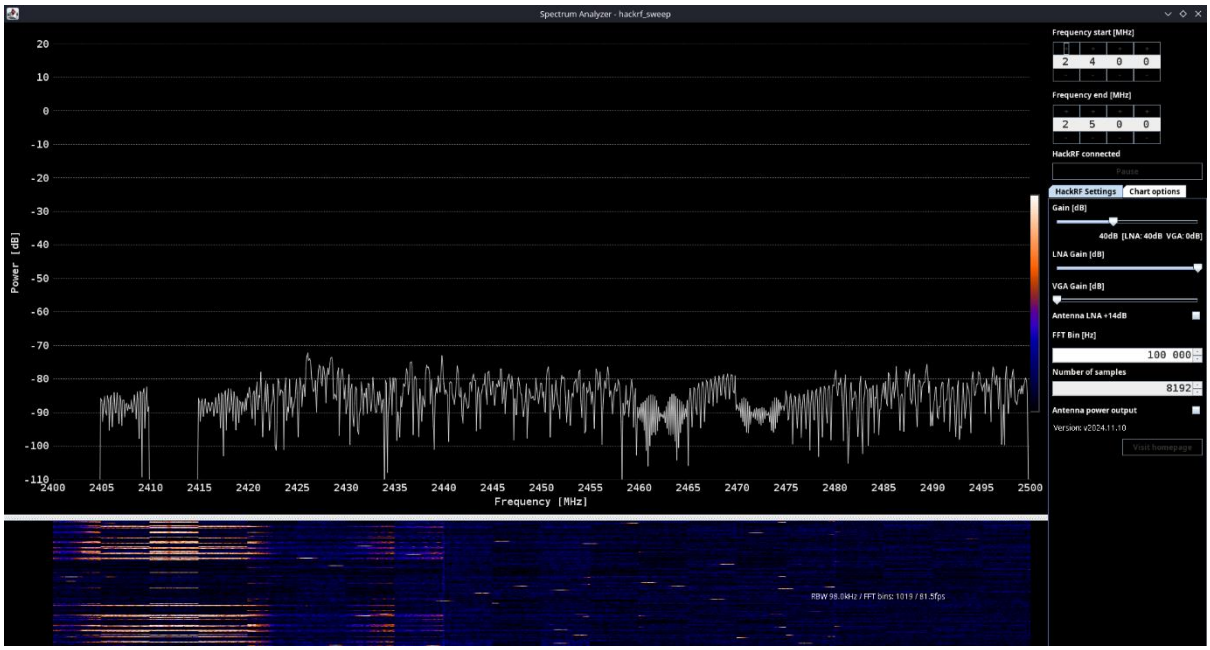


Figure 15. Visualization of the 2.4GHz frequency band with a Bluetooth device near the SDR

The goal of this step is not to identify all used frequencies but rather to find three or four stable channels that allow for efficient signal analysis. Once the device's address is identified, the Crazyradio PA dongle can scan all channels. In this process, the Crazyradio PA is programmed to listen across the entire spectrum and display any channels where packets from the target device are detected.

4.5. Analyzing the waves

Once enough active channels are identified, signals can be captured using an SDR, such as the HackRF One, and analyzed with Universal Radio Hacker (URH).

4.5.1. Project setup

URH allows for the organization of captures into projects[2]. When analyzing multiple devices, maintaining separate projects prevents confusion. Additionally, each project was structured with two subfolders: one for raw captured signals and another for demodulated data. This organization is essential because reopening a signal file requires manually reapplying demodulation settings and noise filters to retrieve the demodulated data, which is time-consuming.

4.5.2. Capture

Proper capture settings are essential to obtain an analyzable wave. Simply tuning the SDR to the identified frequency from Chapter 4.4 may results in a distorted capture as depicted in Figure 16, especially if a HackRF is used.

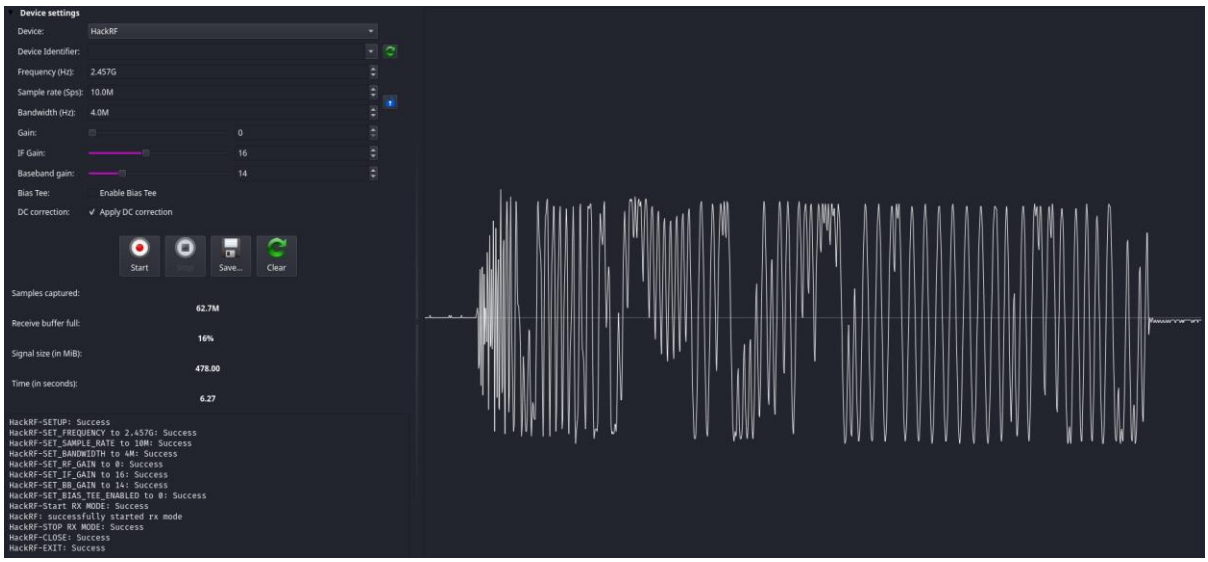


Figure 16. Capture of a distorted radio wave

This happens because of the DC spike generated when the HackRF operates in receiving mode (RX). This DC spike contaminates the captured signals, making analysis more difficult. To mitigate this issue, the listening frequency of the HackRF should be shifted by at least 1 MHz. By offsetting the DC spike from the center of the signal, corruption is avoided. Figure 17 shows a capture with an incremented frequency. URH also has an option *Apply DC correction*, but it was found to be unhelpful.

[2] A new project is created by selecting *File* then *New Project*

Figure 17. Capture of a radio wave without DC spike

Once the listening frequency is adjusted, the signal must remain within the SDR's capture window. This is managed using the **Bandwidth (HZ)** setting. A bandwidth of 4 MHz is generally sufficient, but the exact value can be verified using a spectrum analyzer or can be identified in the documentation potentially found in Chapter 4.2.

The **Sample rate (SPS)** defines the number of samples recorded per second. A higher sample rate increases the capture file size. According to the Nyquist-Shannon theorem, the sample rate must be at least twice the signal bandwidth to accurately capture and reconstruct it [30]. The HackRF One supports up to 20 million samples per second (20 MS/s), but 10 MS/s is typically sufficient for analysis.

The **Gain** setting amplifies weak signals but can also degrade their quality. To maintain signal integrity, the gain should be set to 0 during captures.

All other settings can generally be left at their default values. Ideally, the capture settings, except the **Frequency (Hz)**, should be similar to Figure 18.

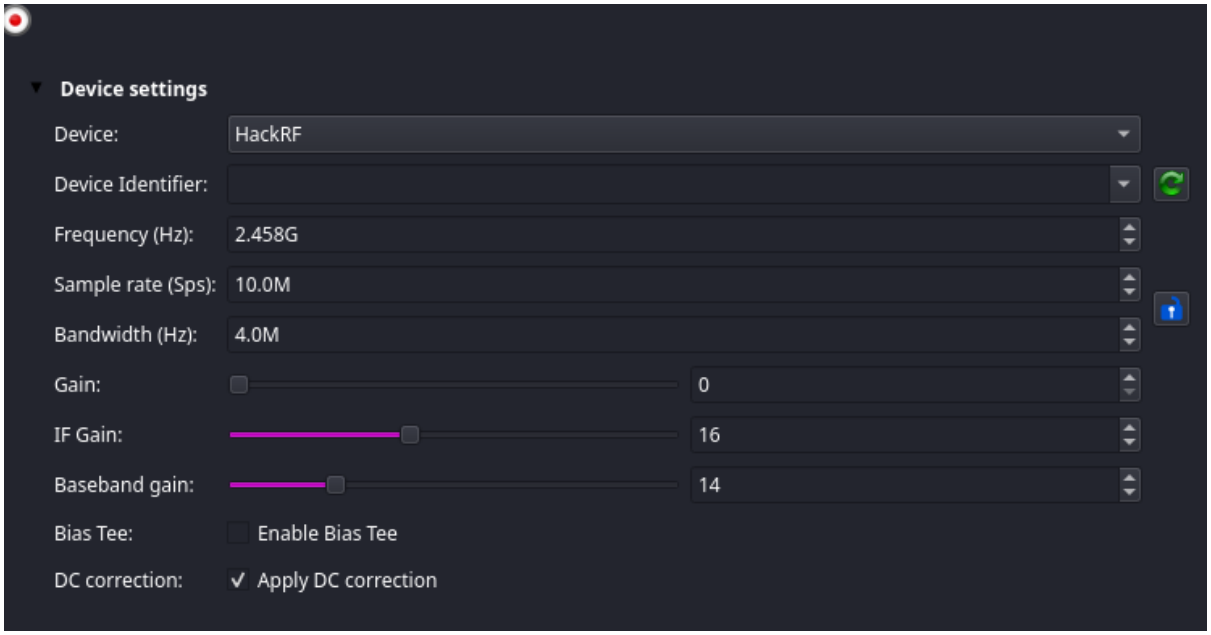


Figure 18. Capture parameters for a radio wave

In most cases, when an input device, such as a keyboard or mouse, transmits a signal, the dongle responds with an acknowledgment (ACK). The device automatically retransmits the signal if a transmitted packet is not received. However, ACK packets are not particularly useful for analysis. To manage noise filtering effectively in later stages, it is necessary to ensure that the dongle's signals are relatively weak compared to the analyzed device. This is achieved by placing the dongle as far as possible from the SDR while keeping the target device close to the SDR's antenna.

4.5.3. Demodulation

After capturing the signal, demodulation is required. While URH provides an automatic tool for detecting demodulation parameters, it is usually better to check them manually.

The first step is to clean the capture file by removing noise, leaving only the relevant signals. This is done in Figure 19 by selecting a noisy segment, right-clicking, and choosing **"Set noise level from Selection."** If the noise threshold is set too high, the peripheral's waves will become impossible to demodulate; this is why it is essential to ensure that the dongle's packet signals have a low amplitude.

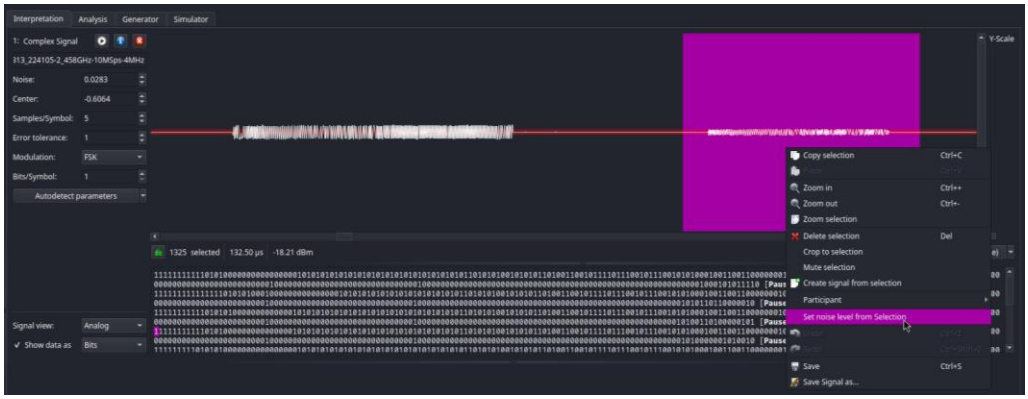


Figure 19. Setting noise level

The **center** parameter distinguishes between binary values. In Figure 20, signals in purple represent binary "1", while signals in green represent binary "0." The center level can be adjusted by dragging the intersection between the two colors. Usually, URH is reliable at autodetecting this parameter if there is no noise, otherwise the center can be manually set to correspond to the middle of the waves.

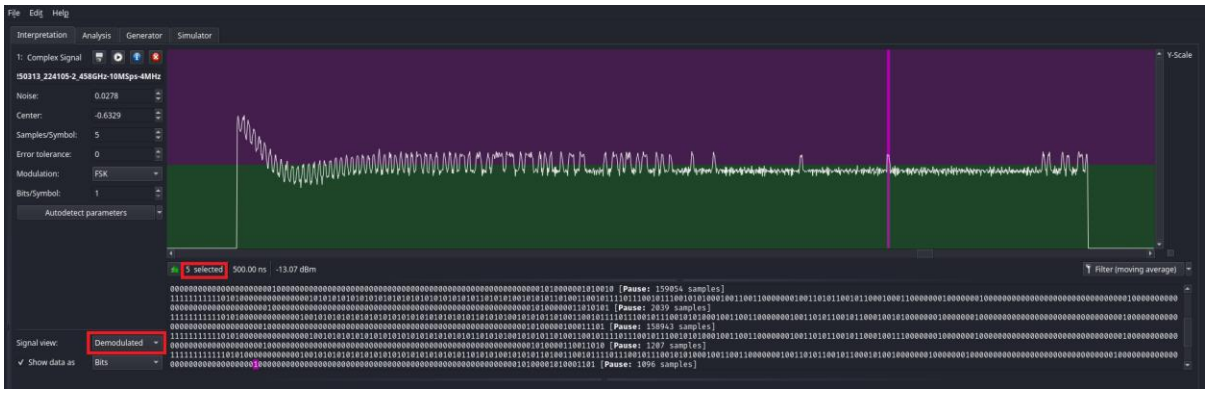


Figure 20. Demodulated view of a radio wave with a samples/symbol of 5 and a data rate of 2Mbps

The **samples/symbol** parameter determines how many samples are used to represent a single bit. This value is found by switching the Signal View to *Demodulated* and selecting the smallest identifiable symbol, as shown in Figure 20. For this, zoom out to view the entire signal and highlight the shortest wave from the two intersections between the wave and the center. URH will indicate the number of samples in the selection. If 5 samples represent a single symbol, then the bit rate of the transmission can be calculated using equation 1 as: data rate = 10M/5 = 2M samples/second (or 2Mbps). The data rate can also be found in the FCC documents from Chapter 4.2.

$$\text{data rate} = \frac{\text{Sample rate}}{\frac{\text{Samples}}{\text{Symbol}}} \quad (1)$$

In most cases, the **Error Tolerance** and **Bits per Symbol** settings can be left at 0 and 1, respectively.

If the entered variables are correct, URH should display a series of bits. Highlighting multiple bits will also highlight their representation in the wave. This allows the demodulation's accuracy to be verified.

4.6. Reverse-engineer the protocol

Once the demodulated data is obtained, the next step is to analyze the protocol to reverse-engineer it. In URH, selecting the **Analysis** tab provides options to view the data in bits or hexadecimal (or ASCII). The bit view is recommended, as it allows for better pattern recognition. Enabling **Mark diffs in protocol** highlights differences in red while keeping a reference message in blue, making it easier to identify changing fields. Figure 21 shows 8 different packets that were captured by pressing the middle button of a mouse multiple times. Even if the same actions were executed, the packets seem to have significant differences.

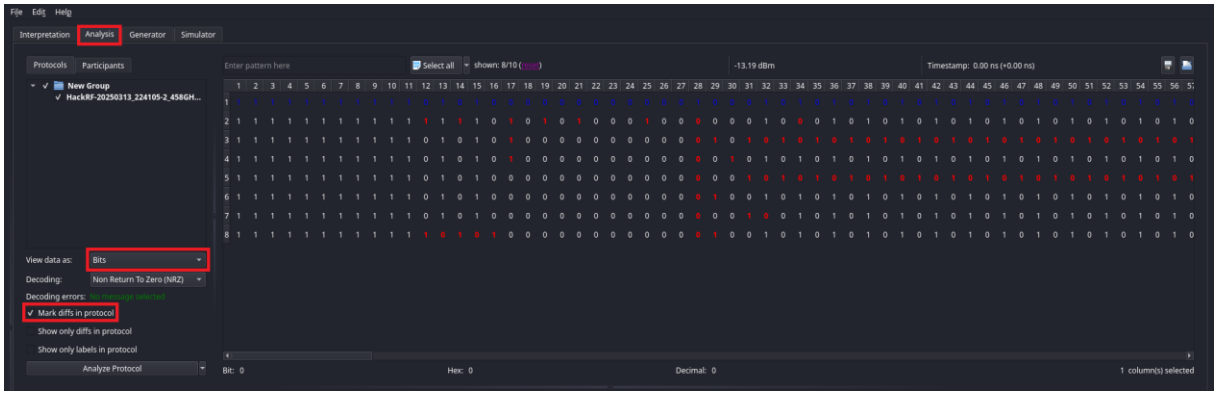


Figure 21. Binary representation of 8 demodulated packets.

4.6.1. Delimiting the packet

The next step is to delimit the packets to find where they start and end. URH is helpful for automatically demodulating signals into bits, but it can make errors. In Figure 21, the eight packets appear different, but this is due to misalignment. URH also sometimes includes incorrect bits at the end of signals and removing these makes the reverse-engineering process easier.

In Figure 20, an initial unstable sequence followed by a stabilization period can be seen. This sequence gets wrongly interpreted by URH as data, leading to packet misalignment. To fix this, the start of a packet, the preamble, must be identified. This is typically a repeating bit pattern found in every packet, often 0b*10* or 0b*01* and ends with a repeating bit. The following bits should remain equal for an extended sequence, generally lasting at least 16 to 20 bits. Figure 22 shows a pattern of 0b*10* which ends with a repeating bit, highlighted in yellow. The following bits are the same in a long sequence.

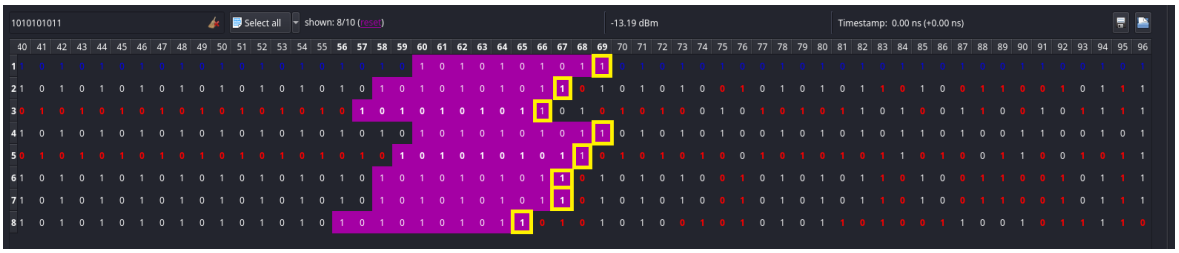


Figure 22. Identification of the end of the preamble

After identifying the end of the preamble, the subsequent bits can be extracted. Additional bits must be included if the selected bit pattern appears before the preamble. This is because the next step involves removing all bits preceding the chosen sequence. If an earlier occurrence of this sequence exists before the preamble, URH may not remove enough bits, and the packets will stay misaligned. Figure 23 shows a bit pattern of 16 bits repeating in every packet after the preamble.

Figure 23. Selected bit pattern

To remove the bits preceding the sequence shown in Figure 23, a new decoding option must be created. This can be done by selecting the **Decoding** dropdown menu and choosing the "**...**" option. On the left, the **Cut before/after** function should be dragged to **Your Decoding**. Then, the **Cut before** option should be chosen, followed by entering the bit sequence and saving it with a preferred name. This process is illustrated in Figure 24.

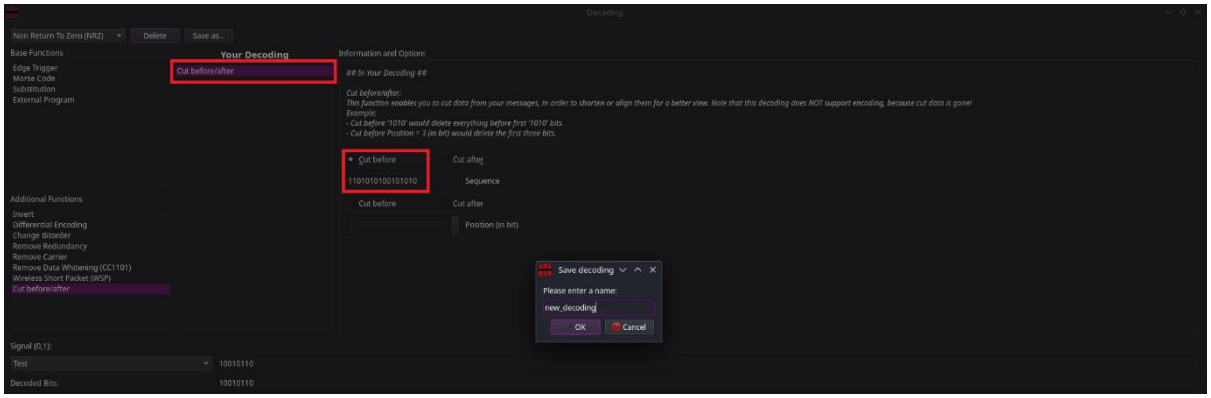


Figure 24. Creation of a new decoding

After closing the decoding window, all packets can be selected, and the new decoding applied. Figure 25 demonstrates the alignment of the 8 packets after the new decoding is applied.

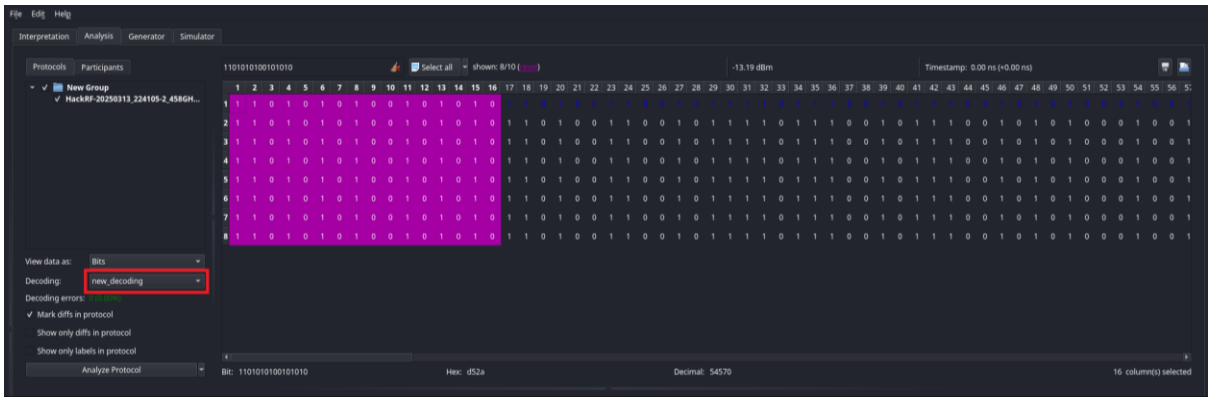


Figure 25. Packets aligned after selecting the new decoding

Next, the end of the packet must be determined. This can be done by capturing multiple identical actions. Pressing the same key generates nearly identical packets, but a section of bits typically increments with each press, as seen in Figure 26. This corresponds to the sequence number and goes from 13 (0b*01101*) to 22 (0b*10110*).

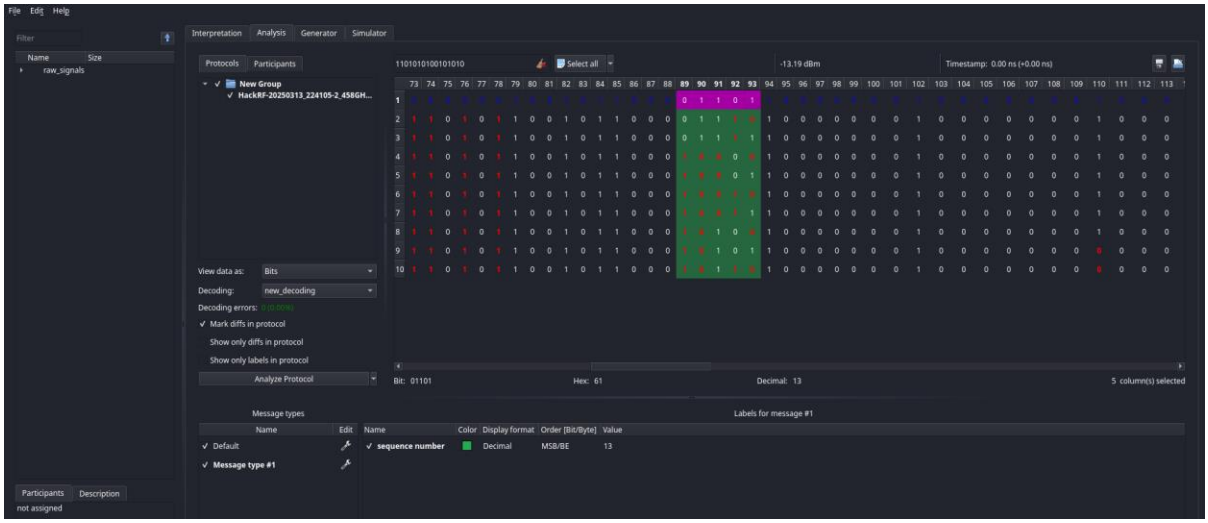


Figure 26. Identification of the sequence number

The coloring in Figure 26 was added by selecting the bit section, right-clicking, and selecting **Create label…**. This is useful to highlight parts of the protocols.

The final portion of the packet contains a CRC (Cyclic Redundancy Check), which ensures packet integrity. CRC values typically consist of 16 bits, meaning the last 16 bits of the packet should change whenever any bit of the packet changes. By having the same packets with different sequence numbers, there should be different CRC values, which makes it easy to find the end of the packet. Figure 27 shows the end of the packets with 17 different bits instead of 16. This usually occurs when URH mistakenly interprets the last wave as a bit, even though it is not. Figure 28 shows that the last wave is too short and should not be interpreted as a bit.

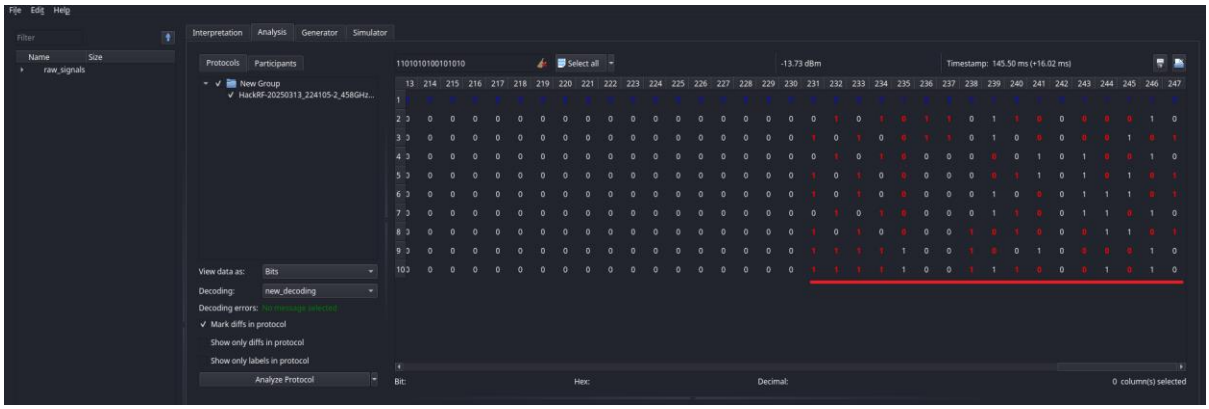


Figure 27. Visualization of the 17 last bits of the packets

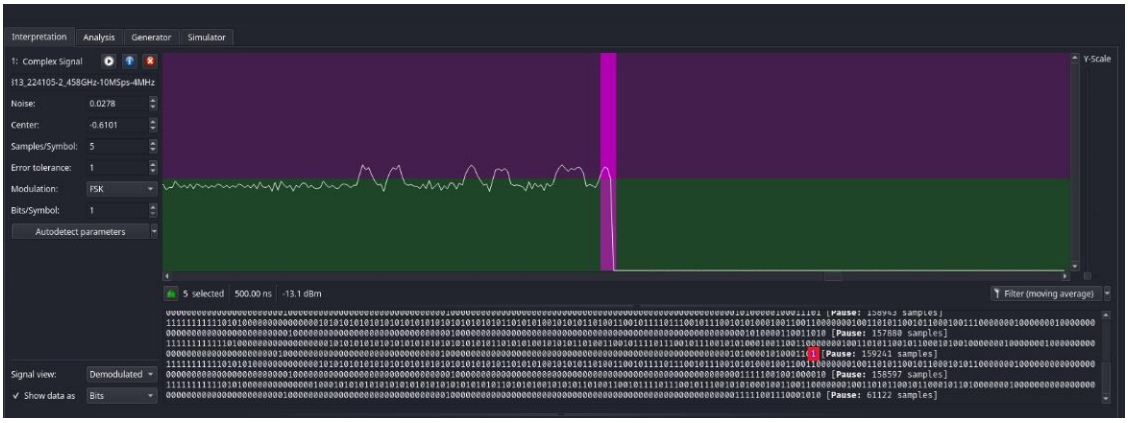


Figure 28. Visualization of the last wave of a packet being interpreted as binary 1

Therefore, it can be assumed that bits 231 to 246 form a 16-bit CRC. To confirm this, the CRC's correctness must be verified. This requires determining the parameters of the CRC function, which include a polynomial and an initialization value (IV). RevEng [31] is an open-source tool that can be used to determine CRC parameters by analyzing the input packet and the corresponding output CRC.

To reverse-engineer a CRC, the following can be done:

1. Select four different packets from the identified beginning, excluding the preamble, to the last bit of the CRC. In this case, take bits 1 to 246.
2. Ensure the packet length is a multiple of 4, as CRC calculations operate on hexadecimal values. If necessary, remove bits from the start to achieve the correct size. In this example, the length is 246, which is not a multiple of 4, but 244 is. Therefore, the first two bits are ignored, selecting bits 3 to 246 as seen in Figure 29.

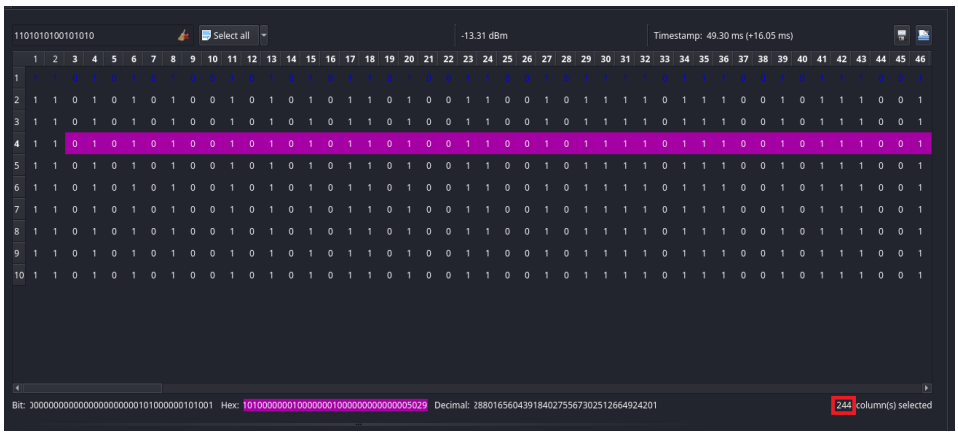


Figure 29. Selection of a series of bits with a size that is a multiple of 4

3. Copying the hexadecimal values of the selected bits and input them into RevEng to determine the CRC parameters. The *w* flag is used to determine the size in bits of the CRC, and the *s* flag is for recovering the used parameters.

Figure 30. Unsuccessful attempt to reverse-engineer a CRC using RevEng

4. If RevEng fails to find a match as seen in Figure 30, it may be necessary to remove the four beginning bits of each packets as these are likely synchronization bits which are typically not included in the CRC calculation. In Figure 31, the hexadecimal value 0x5 was removed which led RevEng to find the CRC parameters .

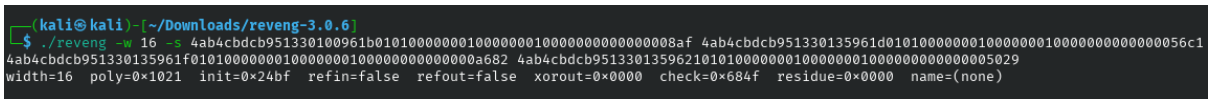


Figure 31. Successful attempt to reverse-engineer a CRC using RevEng

This methodology has consistently allowed the CRC parameters to be identified throughout this thesis. However, manufacturers have the flexibility to implement CRC in different ways. Niklas Tomsic [19] encountered a case where the CRC had to be reversed before being input into RevEng. Additionally, the sequence number portion of the packet may be excluded before being processed by the CRC function. There is no systematic methodology for determining CRC parameters.

A special label can be used to verify the accuracy of the CRC parameters. By naming the label **checksum**, URH automatically enables CRC checking. As shown in Figure 32, an incorrect CRC is highlighted in red. The CRC settings can be edited by right-clicking the label, selecting **Edit label…**, and then entering the identified parameters. It is also crucial to specify the correct start of the CRC input, which begins at the seventh bit in this example, as illustrated in Figure 33. If the parameters are correct, URH will display the CRC label in green, as seen in Figure 34.

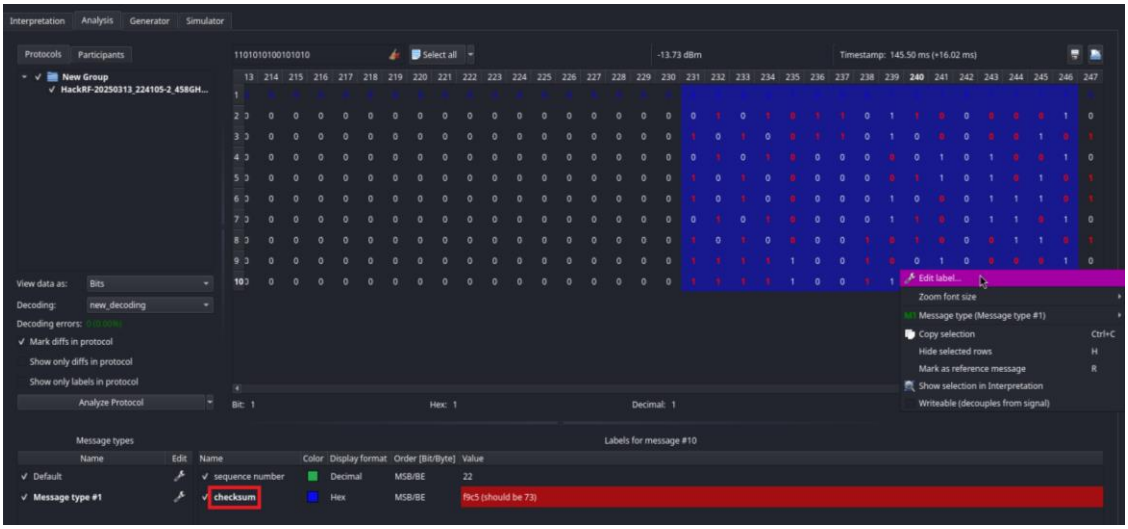


Figure 32. Label **checksum** showing the CRC is incorrect for the selected packet

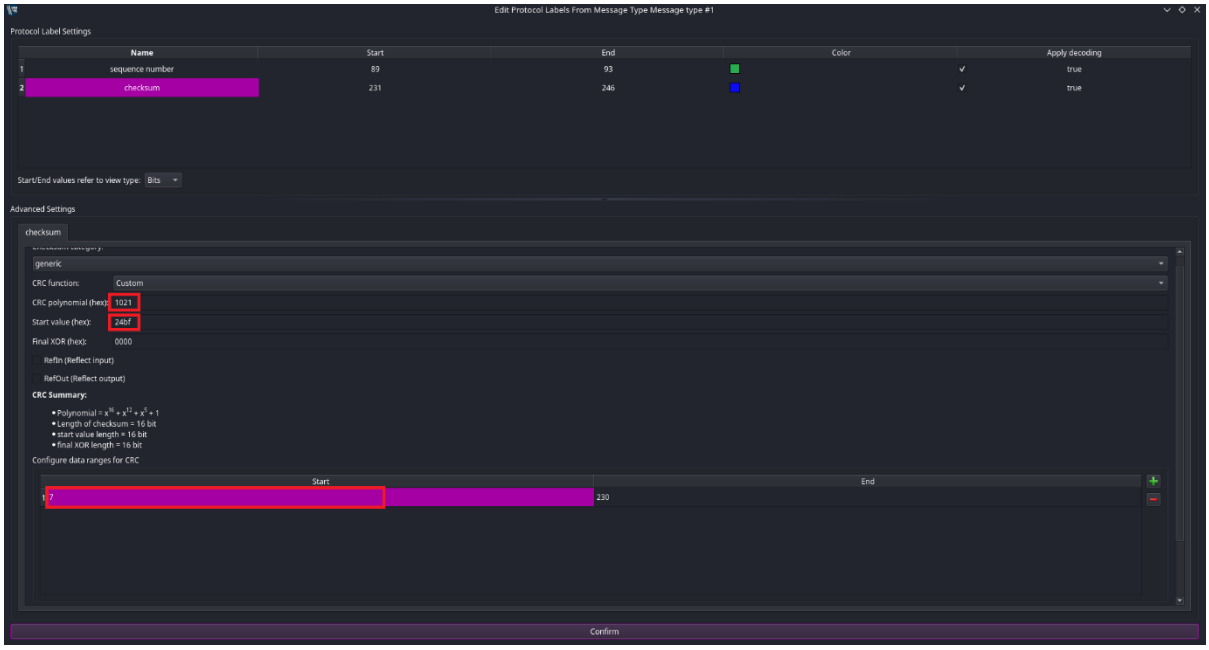


Figure 33. Editing the **checksum** label

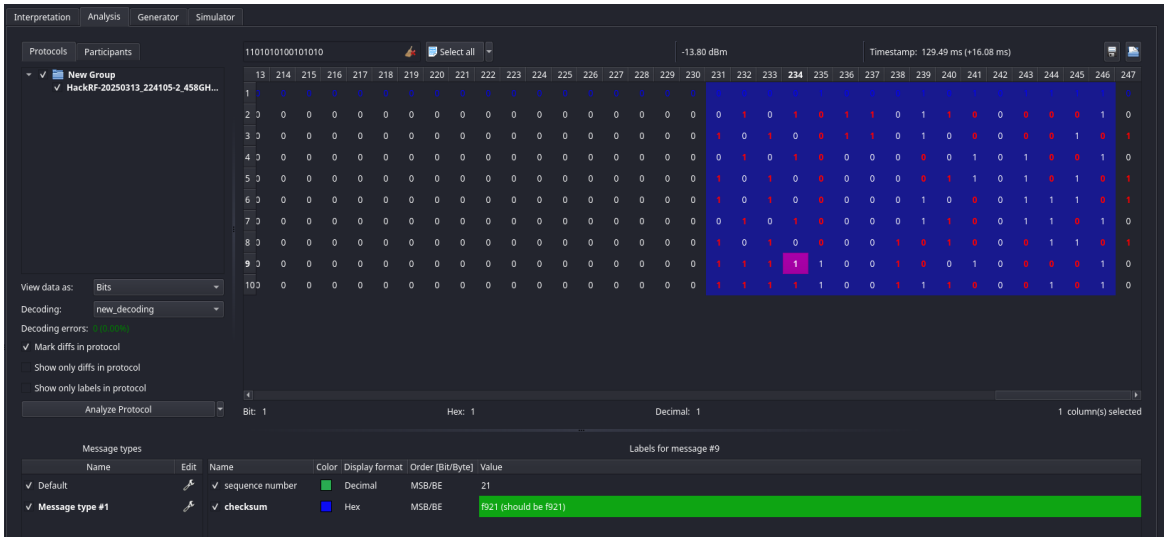


Figure 34. Label **checksum** showing the CRC is correct for the selected packet

Now that the CRC parameters are known, it is confirmed that the signals are correctly demodulated. The packet boundaries are also clearly identified and properly aligned, making the reverse-engineering process much easier.

4.6.2. Understand the protocol

Two approaches were used to understand protocols:

1. Capturing and analyzing signals manually in URH.
2. Fuzzing with the Crazyradio, which automates signal capture and demodulation, making it significantly faster.

Protocol components such as sequence numbers, CRC, and potential authentication mechanisms can be identified by performing the same actions and observing how the packets change. If a simple replay attack (sending back the signal captured) works, it most likely means no authentication mechanism is implemented, and the device is most likely vulnerable. However, the replay attack sometimes does not work even if no authentication mechanism is implemented; this is typically the case because the dongle does not receive the signal as it listens to another frequency. Additionally, a part of the packet may be encrypted. If the same actions result in a large part of the packet being different, it will likely be encrypted.

The "data" part of the protocol can be identified by comparing the results of different actions. For a mouse, the left and right clicks can be pressed sequentially to observe which part of the packet changes. Afterward, the wheel can be scrolled, or the mouse can be moved. The goal is

to perform individual actions to determine what each part of the packet represents. Once these observations are made, multiple actions, such as moving the mouse while clicking, can be performed simultaneously to verify whether the assumptions are correct.

Figure 35 pictures an example of a reverse-engineered mouse protocol with manually created labels:

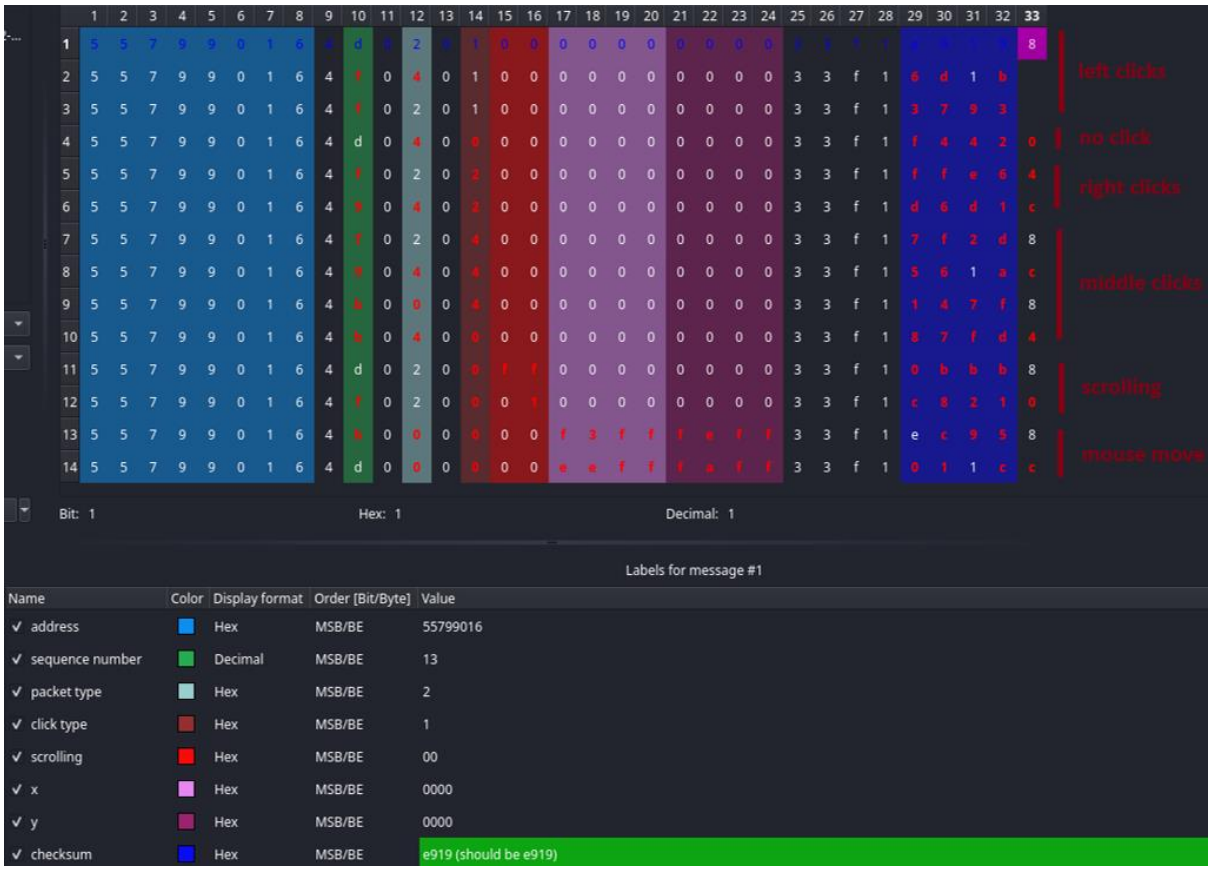


Figure 35. Example of a reverse-engineered mouse protocol in hexadecimal view

- The **packet type** is a variable that was observed to be 0b*01* in the first signal sent, 0b*10* in the second and last signal, and 0b*00* for the rest. This could indicate whether a button is being pressed over an extended period, though this is not certain. In Figure 35, the values are either 0x*0*, 0x*2*, or 0x*4* because the two bits are embedded within a hexadecimal representation, following the format 0*bb*0.
- The **click type** represents the button being pressed. A value of 0b*000* indicates no button is being pressed, 0b*001* represents a left click, 0b*010* a right click, and 0b*100* a middle click. The sum of the individual button presses represents simultaneous clicks. For example, a middle and a left click are represented by 0b*001* + 0b*100* = 0b*101*. As shown in Chapter 4.1, it is possible to have more buttons.

- The **scrolling** is represented by a signed value indicating the direction of the scroll. Figure 35 shows 0x01 for a scroll up, 0xFF for a scroll down, and 0x00 for no scroll.
- The **x** and **y** represent the movement of the mouse along the x and y axes, respectively.

It was also shown in Chapter 4.1 that the protocol supports horizontal scrolling. This feature could not be tested as the mouse physically does not permit it. However, knowing that the field for this action is 8 bits long, columns 11 and 13 are likely linked to it, as they are the only remaining ones initialized to 0x0.

Keyboards are generally easier to reverse-engineer. Similar to mice, it is essential to analyze different actions, such as pressing keys A and B individually or simultaneously or pressing A followed by a modifier like Shift or Ctrl. If the manual of the RF chip was found in Chapter 4.2, it can be used to understand some of the variables.

Figure 36 illustrates an example of a reverse-engineered keyboard protocol:

- The **modifiers** represent the eight flags (left/right control, shift, alt, GUI) such as seen in the HID data in Figure 13.
- **array-1** to **array-6** corresponds to the elements of the array described in Figure 13. The array is usually located just before the CRC.

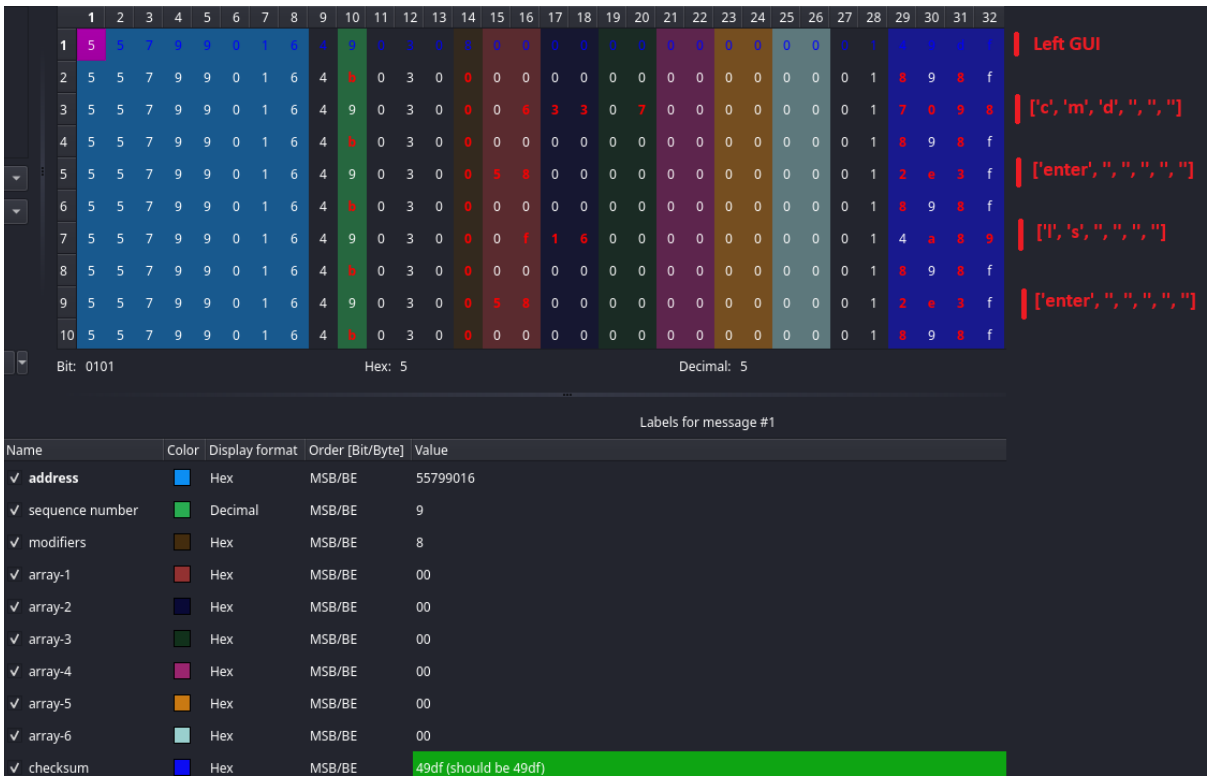


Figure 36. Example of a reverse-engineered keyboard protocol in hexadecimal view

Another way to understand the protocol is by employing the Crazyradio PA to fuzz packets, which significantly saves time. The script created for this thesis can be utilized to do this [2]. The file *main.py* has a method called *quick_sniff*, which is shown commented in Figure 38. To use this method, the identified address, the list of channels, the data rate derived from equation 1, and the packet size in bytes must be provided. The method will then sniff for packets with the specified address across the given channels, display the packets, and highlight the differences. This approach makes it easy to observe the variations when different actions are performed.

The *quick_sniff* method was used in Figure 37 to capture mouse packets from another mouse than the one reverse-engineered in Figure 35. By performing multiple actions such as left clicks, right click, scroll down and scroll up, it can be understood which part of the protocol corresponds to which action. The part in green represents the sequence number, the part in yellow has been identified as the click type, the one in blue is the scrolling, and the part in purple is the CRC. About the click type, two hexadecimal values change: 1 when the left click is pressed, 0 when released, and 2 when the right click is pressed. For the scrolling, when scrolling down, the value is FF; when scrolling up, it is 01. It is also easy to identify the continuously incrementing sequence number and the CRC, which constantly changes even when the data part of the packet is identical.

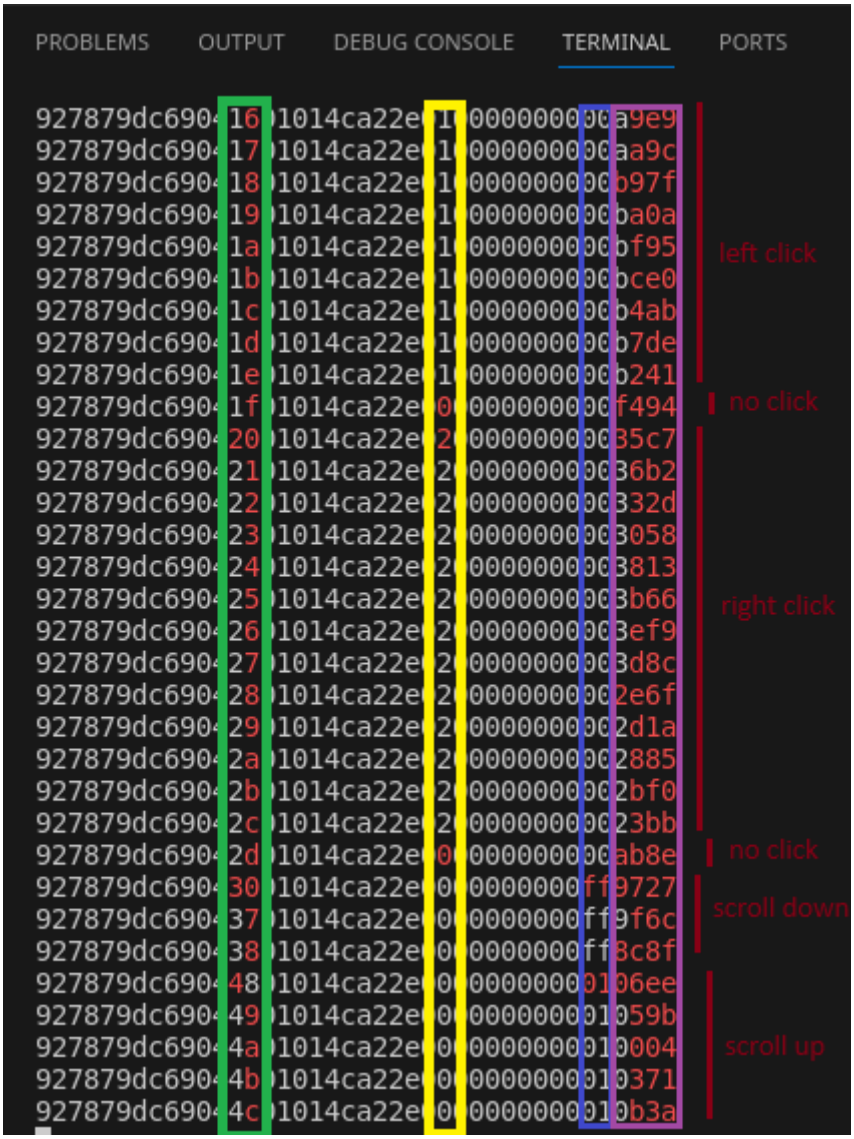


Figure 37. Output of the method *quick_sniff* with different actions captured

If it is difficult to find the channels using the HackRF Spectrum Analyzer, it is also possible to use the Crazyradio PA to fuzz the channels. The more identified channels, the more accurate the sniffing process will be. The method *fuzz_channels* requires the address and data rate, and it will scan the entire 2.4GHz frequency band (2400MHz to 2499MHz), gradually outputting a list of channels as new ones are found. The process may take some time to identify all the channels. It is essential to continuously send signals with the peripheral so that the Crazyradio PA can receive them. Figure 38 shows the use of this method with five identified channels.

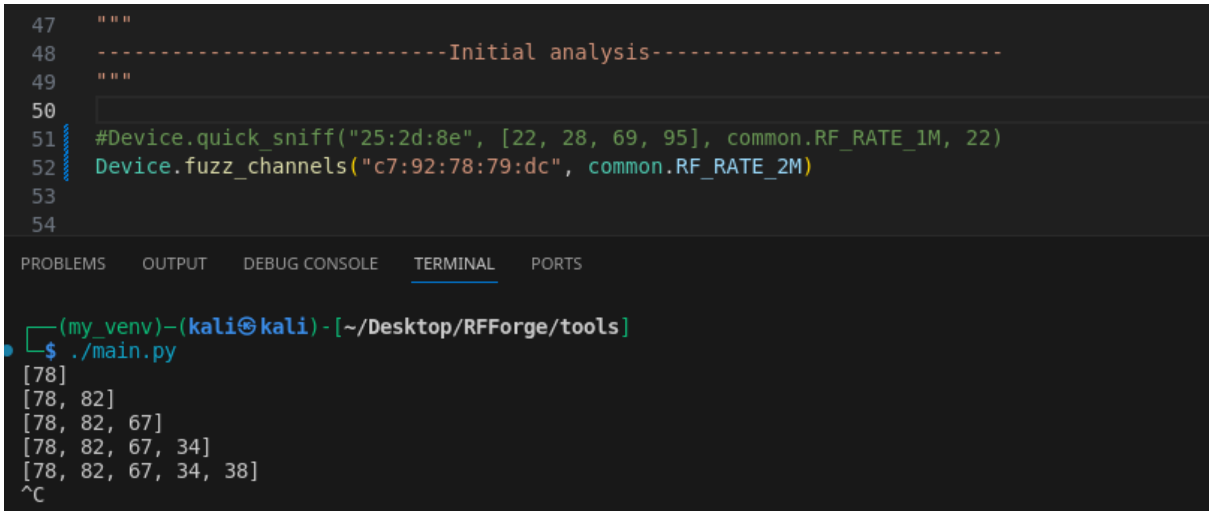


Figure 38. Usage of the method *fuzz_channels* to find channels used by a peripheral

4.7. Exploit the protocol

This chapter explores two attacks on wireless peripherals: sniffing and spoofing. Sniffing consists of intercepting and analyzing packets transmitted between a device and its USB dongle, allowing an attacker to extract sensitive information such as keystrokes. On the other hand, spoofing involves crafting and transmitting malicious packets that the dongle interprets as legitimate inputs, enabling unauthorized control over a target system. By using the Crazyradio PA, both attacks can be implemented.

4.7.1. Sniff the device

The first possible attack is sniffing the device, which involves catching the peripheral signals intended for the USB dongle. Since the protocol structure is known and the data can be extracted from the signals, it is possible to analyze their contents. This attack is illustrated in Figure 39: when a user triggers any action, the peripheral will send signals over the air. Those signals are intended for the USB dongle, but a threat actor can also catch them. If the protocol is known by the threat actor then he can extract the actions triggered by the user. For example, if the user enters the password “qwerty”, the threat actor can catch the signals sent by the peripherals and determine which keys were pressed.

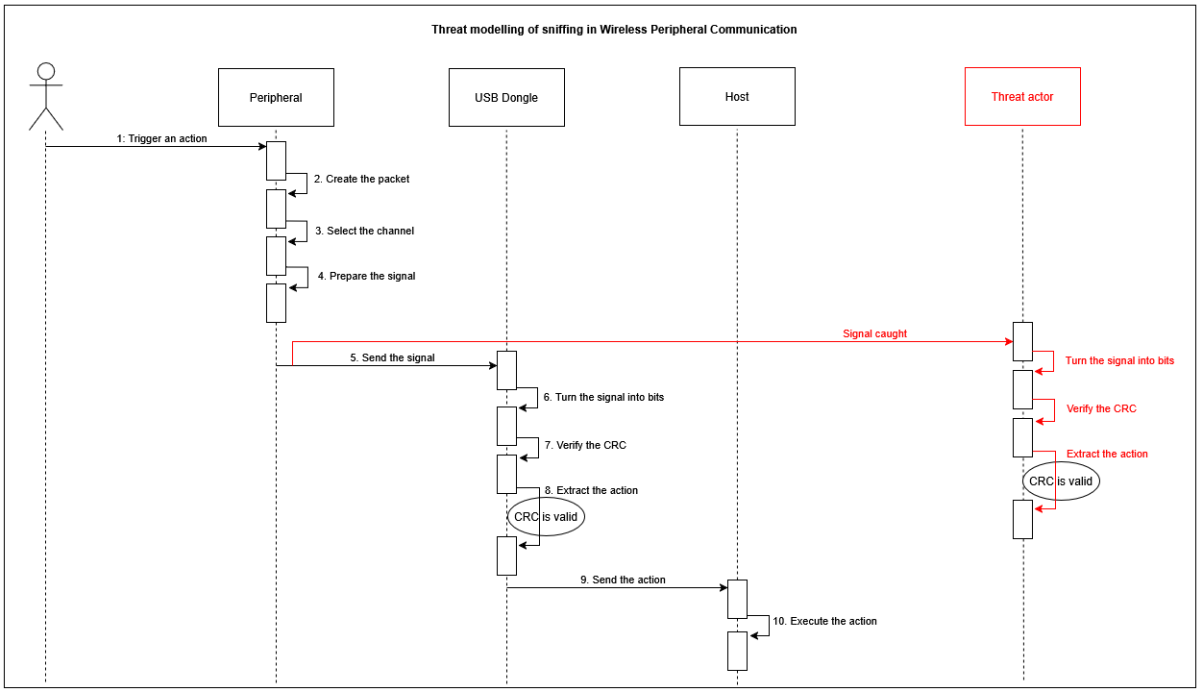


Figure 39. Threat modelling of sniffing in wireless peripheral communication using sequence diagram

This attack can be implemented using the Crazyradio PA. The method *quick_sniff*, shown in Figure 38 with the output in Figure 37, can be used for this purpose. However, the data obtained is in raw form and needs to be interpreted. To achieve this, a new class can be created in the *devices* folder, inheriting from the *Device* class. A template is available in the *devices/Template* folder.

For the sniffing attack, the only additional methods that need to be implemented besides *__init__* are *parse_packet()* and *handle_sniffed_packet()*. These methods are called within the *sniff()* function of the abstract class *Device*.

The *parse_packet()* method receives an array of bytes and should output a dictionary containing the parsed packet. The dictionary structure can vary depending on the protocol, but it should include a key for the CRC and a key for the payload, which will be used as input for the CRC function. Figure 40 illustrates an implementation of this method.

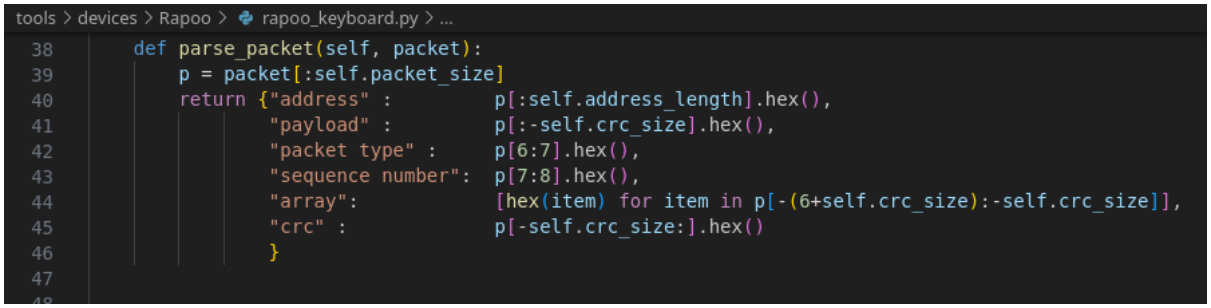


Figure 40. Implementation of *parse_packet* for the Rapoo keyboard

As shown in Figure 41, *handle_sniffed_packet()* typically verifies the CRC and, if valid, displays information about the packet, including the dictionary returned by *parse_packet()*. This method can also be adapted to perform additional checks, such as verifying the *packet_type*, as demonstrated in Figure 41.

Rather than displaying the scancodes directly, implementing a method, *scancodes_to_string()*, is preferable, which converts scancodes into readable characters. Figure 42 presents an implementation of this method using the *SCANCODE_TO_CHAR* dictionary from *keyboard.py*.

```
def handle_sniffed_packet(self, packet, channel):
    if self.check_crc(packet["crc"], packet["payload"]):
        if packet["packet type"] == "06":
            print(f"Rapoo Keyboard Packet\tCHANNEL : {channel}")
            print(packet)
            print(self.scancodes_to_string(packet["array"]))
            return True
    return False
```

Figure 41. Implementation of *handle_sniffed_packet* for the Rapoo keyboard

```
48
49      def scancodes_to_string(self, array):
50          """Convert an list of scancodes into a string.
51
52          Also handles the modifiers (l/r shift and r alt)
53
54          Args:
55              array (list[str]): A list containing USB HID scancode in hexadecimal string.
56
57          Returns:
58              str: A string containing the characters from the list.
59          """
60          modifiers = 0
61          shifted = False
62          for scancode in array: # grab potential modifiers
63              value = int(scancode, 16)
64              if value == KeyboardScancode.KEY_LSHIFT.value or value == KeyboardScancode.KEY_RSHIFT.value and not shifted:
65                  modifiers += 1
66                  shifted = True # avoid incrementing modifier 2 times in case we press L and R shift at the same time
67              elif value == KeyboardScancode .KEY_RALT.value:
68                  modifiers += 2
69          chars = ""
70          for scancode in array:
71              chars += KeyboardScancode.SCANCODE_TO_CHAR.value.get((int(scancode, 16), modifiers), "")
72          return chars
73
74
```

Figure 42. Implementation of a method to turn scancodes into characters

The *sniff()* method, shown in Figure 43, in the *Device* class continuously scans the device's channels to detect transmitted packets. An example of the output of this method is shown in Figure 44, where the keys "a", "b" and "c" are pressed one after the other.

```
def sniff(self):
    """Sniff all the device's channels to retrieve raw packet sent in the air. Display information about the retrieved packets.

    Change channel every dwell_time unless we found a packet, in that case the dwell_time is reset.
    """
    dwell_time = 0.2
    channel_index = 0
    common.radio.set_channel(self.channels[channel_index]) # Set channel here to prevent USBError (somehow)
    common.radio.enter_promiscuous_mode_generic(unhexlify(self.address.replace(':', '')), rate=self.rate)
    last_tune = time.time()

    try:
        while True:
            # Increment the channel after dwell_time
            if len(self.channels) > 1 and time.time() - last_tune > dwell_time:
                channel_index = (channel_index + 1) % (len(self.channels))
                common.radio.set_channel(self.channels[channel_index])
                last_tune = time.time()

            value = common.radio.receive_payload()
            if len(value) >= self.address_length:
                if bytes(value[:self.address_length]) == unhexlify(self.address.replace(':', '')):
                    if self.handle_sniffed_packet(self.parse_packet(bytes(value)), self.channels[channel_index]):
                        last_tune = time.time()
    except KeyboardInterrupt:
        usb.core.find(idVendor=0x1915, idProduct=0x0102).reset()
```

Figure 43. Implementation of the *sniff* method from *Device.py*

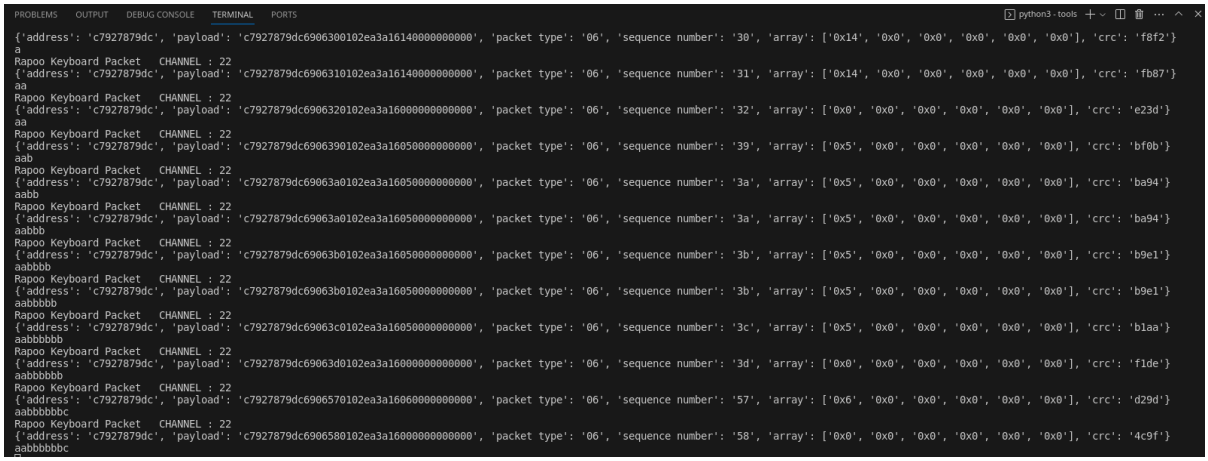


Figure 44. Output of the *sniff* method

However, since no advanced logic is implemented, the Crazyradio PA often misses packets when listening to the wrong channel. As a result, the sniffing tool is low in reliability. This method is a proof of concept and should be further developed to properly handle the device's frequency hopping mechanisms.

4.7.2. Spoof the device

Spoofing involves crafting packets and transmitting them wirelessly. Figure 45 illustrates the attack process: A threat actor crafts his own packets and sends them over the air on the identified dongle's channels. The USB dongle interprets these packets as legitimate, accepts them, and executes the corresponding actions on the computer. This attack is similar to a BadUSB attack [32] but does not require physical access to the computer.

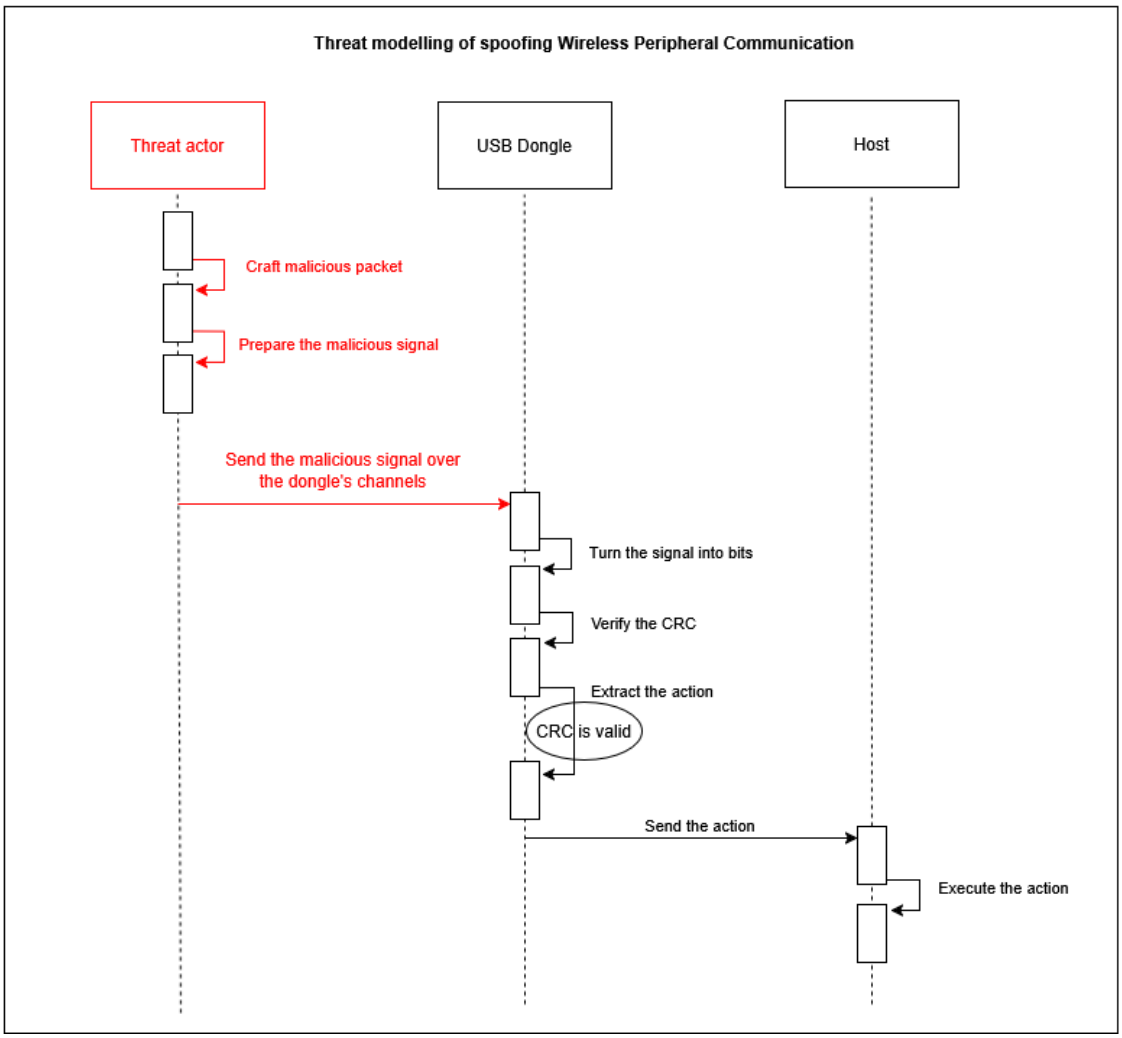


Figure 45. Threat modelling of spoofing in wireless peripheral communication using sequence diagram

To implement this attack with the Crazyradio PA, a method for crafting malicious packets needs to be implemented. In the script created for this thesis, this method is called *build_packet*. Figure 46 shows an implementation of this method. It takes a table of *KeyboardScancode* from the *keyboard.py* file as input.

```
def build_packet(self, scancodes=[]):
    """Build a raw packet based on the scancodes given.

    Args:
        scancodes (list[KeyboardScancode]): A list of KeyboardScancode to include in the packet.

    Returns:
        bytes: A raw packet in bytes format (it does not contain the preamble).
    """
    address = unhexlify(self.address.replace(':', ''))
    sequence_number = self.sequence_number.to_bytes(1, "big")
    self.sequence_number = (self.sequence_number + 1) % 255
    padding = b"\x01\x02\xea\x3a\x16"
    array = self.build_array(scancodes)
    crc = self.calculate_crc(address+sequence_number+padding+array)

    return address+sequence_number+padding+array+crc
```

Figure 46. Implementation of *build_packet*

Figure 47 demonstrates how to use the *build_packet* method to create a payload. Modifiers such as Shift, Ctrl, Alt, or GUI can be included in the payload to simulate key presses. Additionally, delays can be added between packets to ensure the host has enough time to complete an action before receiving the following command. The payload in Figure 47 opens the Linux/Windows menu using the Left GUI key, types "CMD" followed by Enter to launch a command line and executes the *ls* command.

```
rapoo_keyboard = Rapoo_Keyboard("c7:92:78:79:dc:69:06", 0x11021, 0xefdf)

attack_rapoo_keyboard = [
    rapoo_keyboard.build_packet([KeyboardScancode.KEY_LGUI]),
    lambda: time.sleep(1),
    rapoo_keyboard.build_packet([KeyboardScancode.KEY_C, KeyboardScancode.KEY_M, KeyboardScancode.KEY_D]),
    lambda: time.sleep(1),
    rapoo_keyboard.build_packet([KeyboardScancode.KEY_KEYPAD_ENTER]),
    lambda: time.sleep(1.5),
    rapoo_keyboard.build_packet([KeyboardScancode.KEY_L, KeyboardScancode.KEY_S]),
    rapoo_keyboard.build_packet([KeyboardScancode.KEY_KEYPAD_ENTER])
    ]

rapoo_keyboard.spoof(attack_rapoo_keyboard)
```

Figure 47. Creation of a keyboard payload using the *build_packet* method

The payload created in Figure 47 is a simple proof of concept example, more complex payloads could include downloading and executing malware to establish a backdoor.

5. Validation

In this chapter, the methodology described in Chapter 4 is applied to reverse-engineer and pentest ten different brands. Chapter 5.1 presents the hardware tested during this thesis, Chapter 5.2 presents the pentest results of each devices, and Chapter 5.3 summarizes the findings.

5.1. Hardware tested

All devices tested in this thesis were purchased in 2024/2025, either from large retail stores or secondhand shops. As stated in Chapter 1.4, the goal is to test devices that are currently available, at the time of writing, on the market. When purchasing the hardware, different brands were selected to ensure diversity. Due to a budget limit, devices costing more than 30€ were not tested.

A list of the devices tested can be found in Table 1. The column 'Brand' indicates the manufacturer brand, the second column contains the devices' model number, the third column indicates whether a mouse, keyboard or both were tested and the fourth indicates where the devices were purchased.

Table 1. Hardware tested

Brand	Model number	Type	Acquisition
Trust	ODY-II	Keyboard and Mouse	Hypermarket
Poss	PSKEY530BK	Keyboard	Hypermarket
Rapoo	X1800s (E1050 + M10)	Keyboard and Mouse	Hypermarket
Edenwood	963716	Keyboard and Mouse	Hypermarket
Qware	Nottingham wireless combo (QW PCB-238BL)	Keyboard and Mouse	Hypermarket
Think Xtra	Ms6-TXn-wh	Mouse	Second hand
Hama	AKMW-100	Keyboard and Mouse	Second hand (also seen on hypermarket)
Omega	OM08WBL	Mouse	Second hand

Cherry	DW5100	Keyboard and Mouse	Hypermarket
HP	230	Keyboard and Mouse	Hypermarket

5.2. Pentest

This chapter presents the pentest results for all tested devices using the methodology described in Chapter 4 and indicates whether sniffing and/or spoofing was successful. To keep the content concise, the full analysis (such as identifying the preamble, etc.) has been omitted.

5.2.1. Trust

The Trust ODY-II set consists of the Ody-II silent wireless keyboard and the Yvi+ silent wireless mouse, as seen in Figure 48. The results of the USB sniffing performed on the devices' dongle can be found in Table 2.

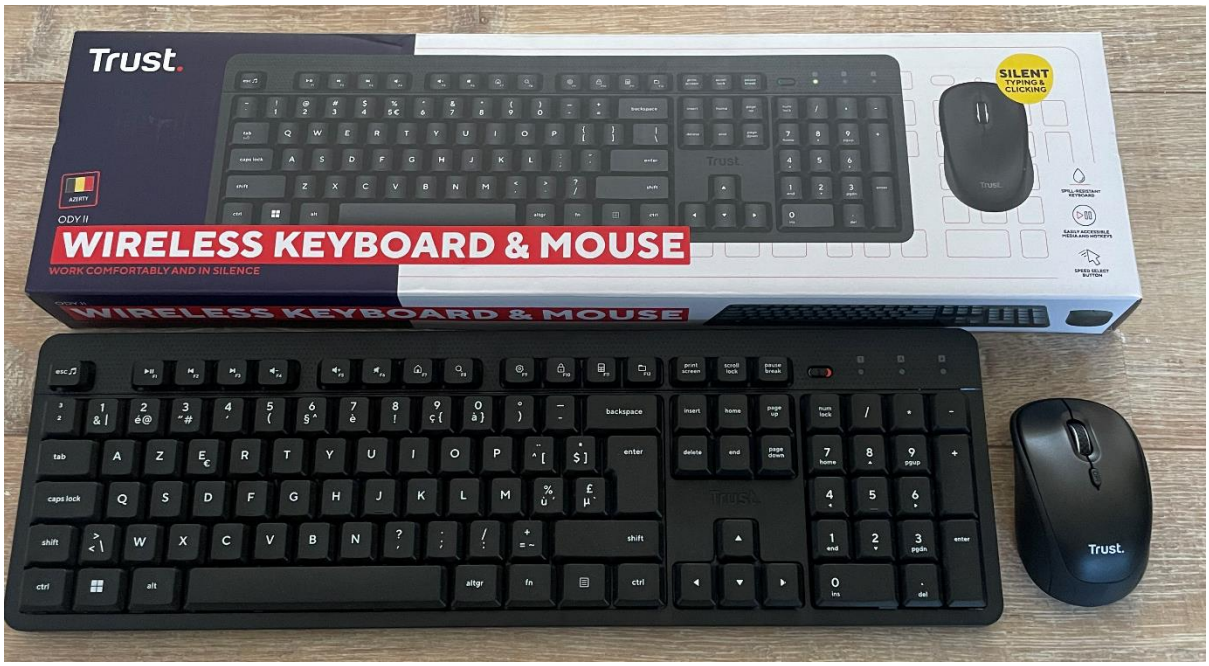


Figure 48. Trust ODY-II set

USB sniffing

Table 2. Data found with USBPcap for the Trust dongle

Vendor ID	**0x145F** (Trust)

Product ID (dongle)	**0x0317** (Unknown)
String descriptor	Trust Deskset
Interfaces	Has Mouse and Keyboard interfaces
HID Data	For mouse: 5 buttons, 16 bits for X/Y, 8 bits wheel For keyboard: 8 bits modifiers, 6-elements array

OSINT

Trust has a website, which contains a page for the Trust ODY-II [33]. The products are still available for sale. Little useful information was found on that website. No FCC ID was found for those peripherals. The Trust company has an FCC ID of "VNK" but there is not entry for the targeted devices.

The devices were opened to inspect their RF chips and gather more details. As seen in Figure 49, the keyboard's RF chip was covered with epoxy resin, preventing identification. However, the mouse's RF chip in Figure 50 was visible and identified as the *TLSR8516EP*.

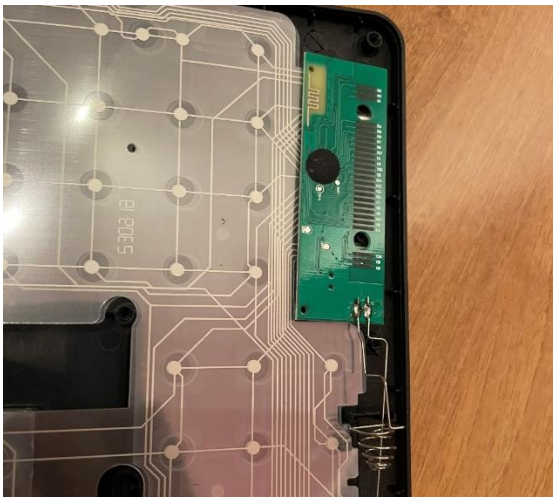


Figure 49. PCB of the Trust keyboard

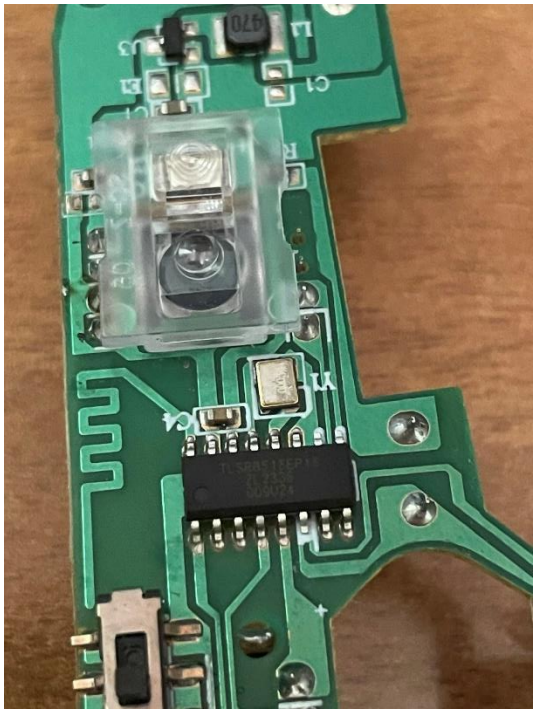


Figure 50. PCB of the Trust mouse

The datasheet for the TLSR8516EP was found [34], but it was in Chinese. After translation, the following key details were noted:

- 8 communication channels
- 2 Mbps data rate
- Possible EMI test mode

No specific information about the communication protocol was found.

Reverse-engineer

Trust keyboard

Multiple “7” presses were captured. Figure 51 shows that FSK can demodulate the signals, and the samples/symbol is 5. As the sample rate of the capture is 10M, the data rate is 10M/5 = 2M. The center was manually tweaked to have coherent demodulation data.

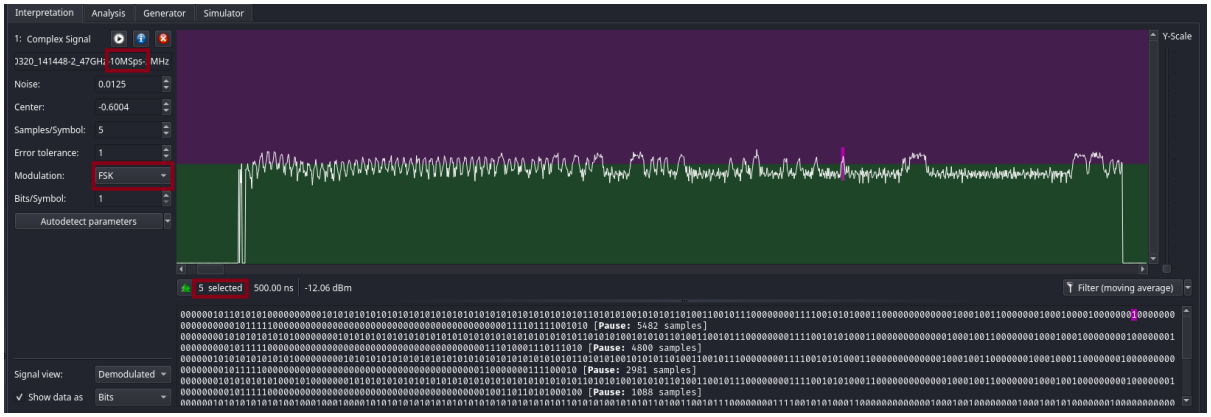


Figure 51. Demodulation of a “7” signal of Trust keyboard

By using the methodology from Chapter 4.6.1, the preamble was found, and a new decoding was created to align the packets. Instead of cutting the packet before the end of the preamble, the preamble was kept to better understand the protocol. RevEng was used to find the CRC parameters as shown in Figure 52.

```
┌──(kali㉿kali)-[~/Downloads/reveng-3.0.6]
└─$ ./reveng -w16 -s b54ab4cb807951800226022101000 5f00000000001ef2 b54ab4cb807951800226022201010 05f0000000000e8ee  b54ab4cb807951800226022301000
5f0000000000c078 b54ab4cb807951800226022401010 05f00000000009b51
width=16  poly=0×1021  init=0×efdf  refin=false  refout=false  xorout=0×0000  check=0×0127  residue=0×0000  name=(none)
```

Figure 52. Identification of the CRC parameters of the Trust keyboard using RevEng

Figure 53 shows the partially reverse-engineered protocol. Index 40 to 51 was identified as being the array. Index 40-41 shows the hexadecimal 0x5F which corresponds to the scancode “7”. However, when the key was released, the scancode was still in the array. It was deduced that index 37 is a Boolean that represents if a key is pressed.

“left shift + 7” and “left control + 7” were captured to identify the modifiers. In Figure 53, the hexadecimal value at index 39 is changing. As there are eight modifiers, and each modifier is represented by a Boolean bit (0 or 1), it can be assumed that the hexadecimal value at index 38 is also part of the modifiers. Additionally, two more fields were found but couldn’t be identified.

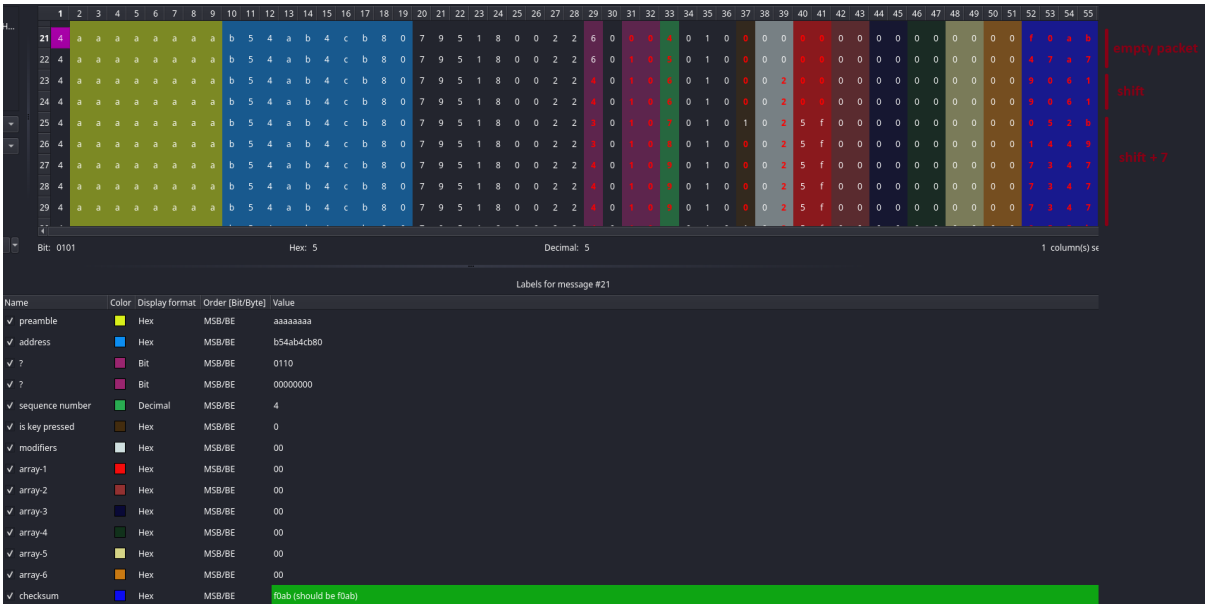


Figure 53. Partially reverse-engineered protocol of the Trust keyboard

Trust mouse

Multiple mouse actions were captured, such as left-click, right-click, scrolling, and movement. The demodulation parameters are the same as those of the Trust keyboard, as shown in Figure 54.

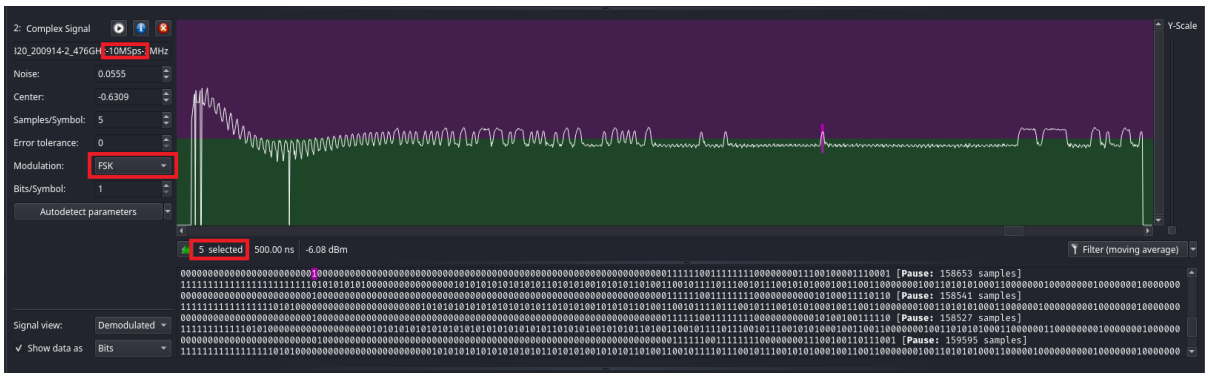


Figure 54. Demodulation of a Trust mouse signal

After aligning the packet, the CRC parameters were found with RevEng. The Trust mouse uses the same parameters as the Trust keyboard.

Multiple actions were performed, and many fields were identified, such as the click type, the x and y movement, and the scrolling. However, other fields were found and could not be identified. An attempt at using the Crazyradio PA to fuzz the different actions was made, but

the field still could not be understood. Figure 55 shows the partially reverse-engineered protocol.

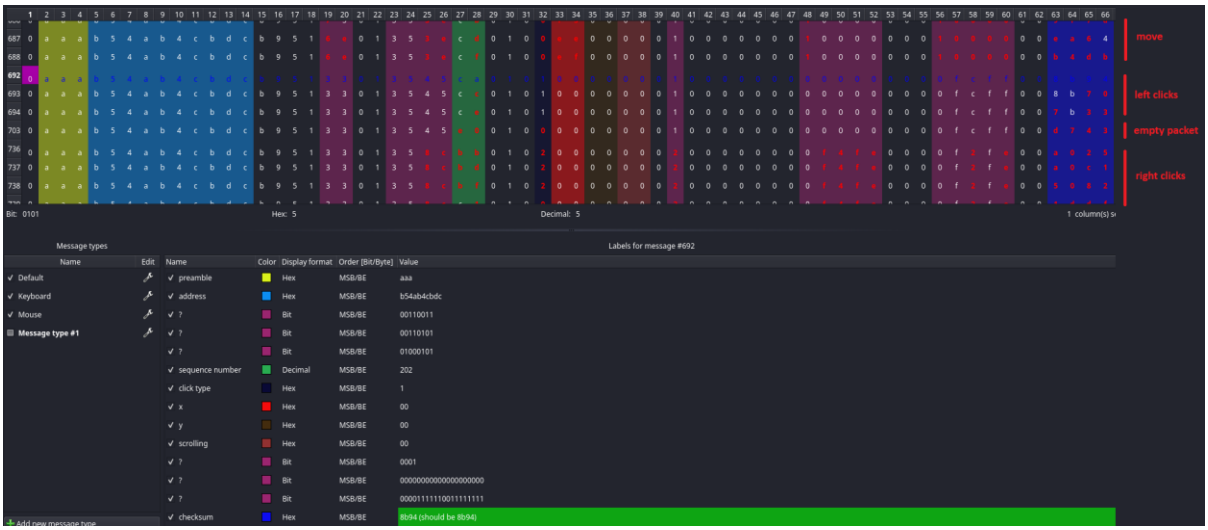


Figure 55. Partially reverse-engineered protocol of the Trust mouse

Results

The sniffing for both the mouse and the keyboard is functional. However, the protocols were not fully reverse-engineered. It is possible to spoof the keyboard despite having two unidentified fields, but the mouse spoofing does not work. The implementation of the exploits is available in the *devices/TLSR85* folder [2].

5.2.2. Poss

The Poss PSKEY530BK consists of one keyboard, as seen in Figure 56. The results of the USB sniffing performed on the device's dongle can be found in Table 3.

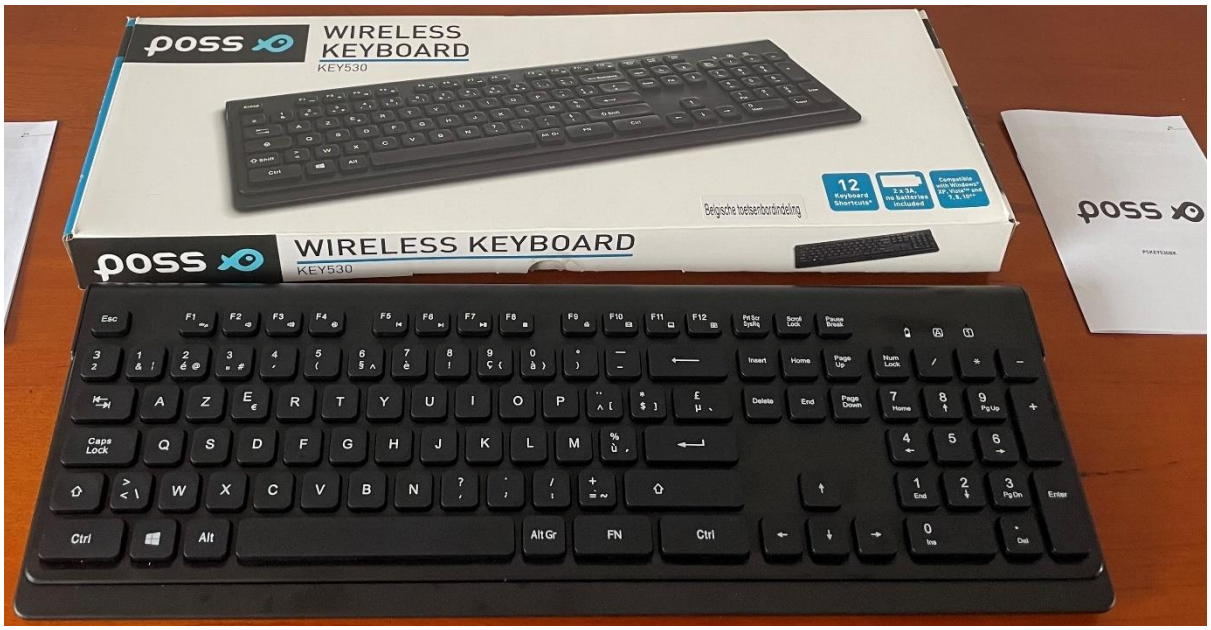


Figure 56. Poss PSKEY530BK keyboard

USB sniffing

Table 3. Data found with USBPcap for the Poss dongle

Vendor ID	**0x8510** (Unknown)
Product ID (dongle)	**0x13AB** (Unknown)
String descriptor	Wireless receiver
Interfaces	Has Mouse and Keyboard interfaces
HID Data	8 bits modifiers, 6-elements array

OSINT

The vendor ID corresponds to Telink, the Chinese company that manufactured the RF chip inside the Trust mouse. No information was found on the Telink website [35]. No website for the Poss company was found. No FCC ID was found either.

The keyboard was opened to see the RF chip, which is obfuscated with epoxy resin, as seen in Figure 57. By looking at the vendor ID, it can be assumed that Telink manufactures the chip.

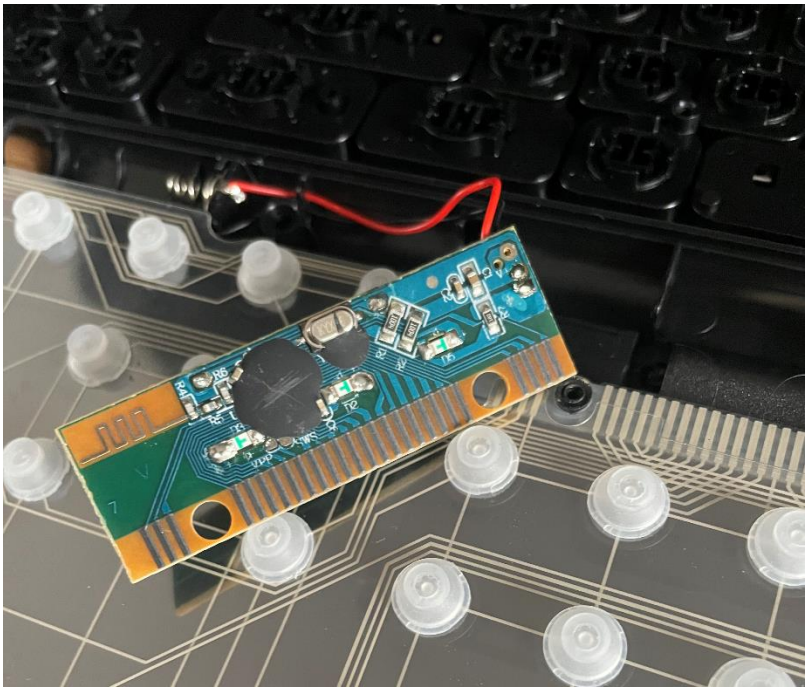


Figure 57. PCB of the Poss keyboard

Reverse-engineer

Multiple “7” presses were captured. Figure 58 shows that the FSK option was used to demodulate the signals, and the samples/symbol is 5. As the sample rate of the capture is 10M, the data rate is 10M/5 = 2M. The center was manually tweaked to have coherent demodulation data.

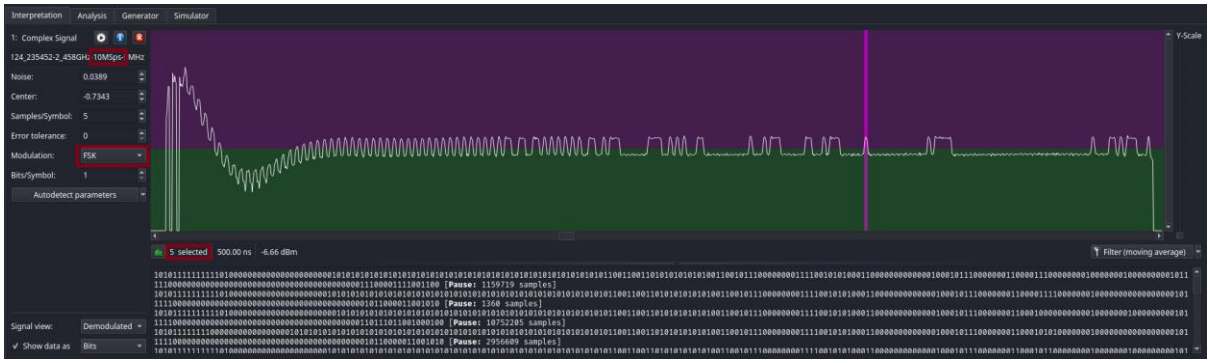


Figure 58. Demodulation of a Poss keyboard signal

After aligning the packets, RevEng was used to find the CRC parameters. As shown in Figure 59, those are the same as the ones used for the Trust devices.

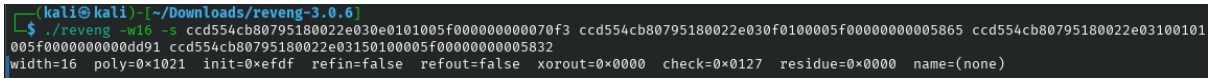


Figure 59. Identification of the CRC parameters of the Poss keyboard using RevEng

After comparing multiple inputs, the same protocol for the Trust keyboard was identified. Figure 60 shows the partially reverse-engineered protocol.

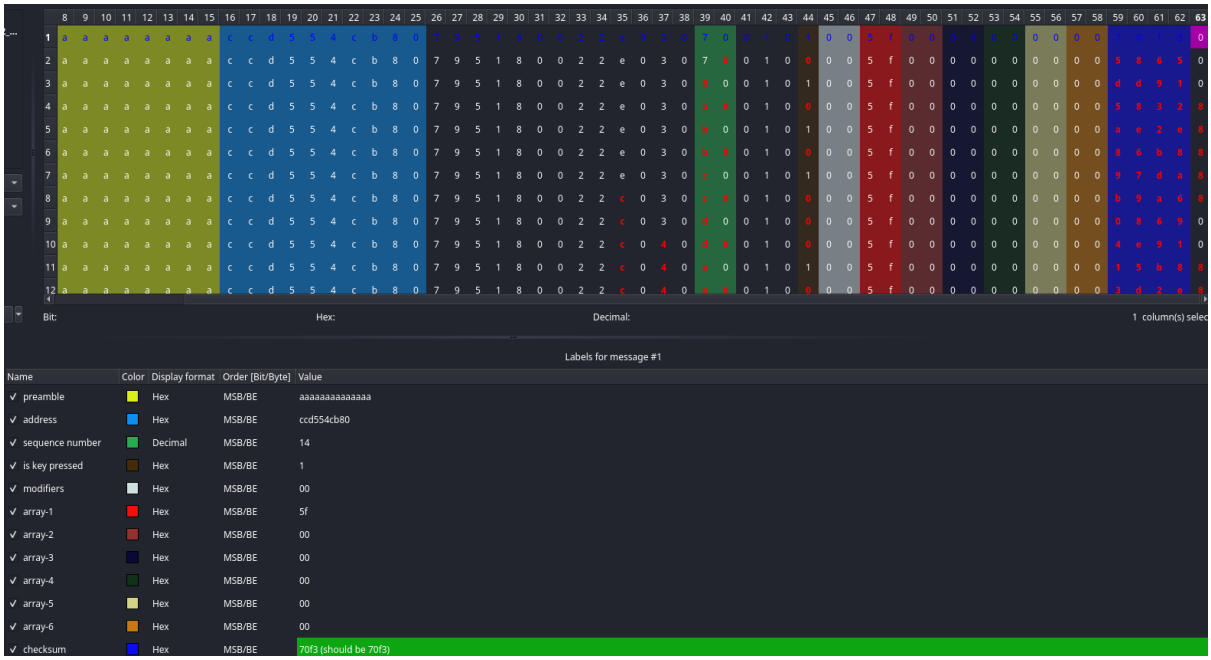


Figure 60. Partially reverse-engineered protocol of the Poss keyboard

Result

The protocol is the same as the one used by the Trust keyboard; therefore, sniffing and spoofing are possible. The implementation of the exploit is in the *devices/tlsr85* folder [2].

5.2.3. Rapoo

The Rapoo X1800 set consists of the E1050 keyboard and the M10 mouse, as seen in Figure 61. The results of the USB sniffing performed on the devices' dongle can be found in Table 4.

Figure 61. Rapoo X1800S set

USB sniffing

Table 4. Data found with USBPcap for the Rapoo dongle

Vendor ID	**0x24AE** (Shenzhen Rapoo Technology Co., LTD.)
Product ID (dongle)	**0x2015** (Unknown)
String descriptor	Rapoo 2.4G Wireless Device
Interfaces	Has Mouse and Keyboard interfaces
HID Data	For mouse: 3 buttons, 32 bits for X/Y, 8 bits wheel For keyboard: 8 bits modifiers, 6-elements array

OSINT

The dongle possesses an FCC ID: **PP203060C**; however, the entry does not exist in the FCC databases. However, documents were found for the mouse and keyboard FCC ID (**PP2M10**

[36] and **PP2E1050** [37]). 16 channels are used, the data rate is 1 Mbps and the modulation used is GFSK. The internal pictures reveal the RF used chip: “R24-M M3258F” for the mouse and “R24-K Q12197F” for the keyboard. However, no technical information about these chips was found. Additionally, the test reports were written in 2012 and 2013. The devices are still available on the manufacturer’s website [38].

Reverse-engineer

Rapoo keyboard E1050

A capture of multiple “7” presses was first analyzed. The FSK option can be used to demodulate the signals, and, as seen in Figure 61, the data rate is 2Mbps, contrary to the information from the test report.

Figure 62. Demodulation parameters for the E1050 keyboard

After aligning the packets, RevEng was used to find the CRC parameters. As shown in Figure 63, the parameters are the same that were used for the Trust and Poss devices.

```
┌──(kali㉿kali)-[~/Downloads/reveng-3.0.6]
└─$ ./reveng -w 16 -s c7927879dc6906400102ea3a165f0000000000d5f0 c7927879dc6906410102ea3a165f0000000000d685 c7927879dc6906420102ea3a165f00000000
00d31a c7927879dc6906430102ea3a1600000000000065f8
width=16  poly=0×1021  init=0×efdf  refin=false  refout=false  xorout=0×0000  check=0×0127  residue=0×0000  name=(none)
```

Figure 63. Reverse-engineering of the CRC function of the E1050 keyboard using RevEng

Figure 64 shows the reverse-engineered protocol. The hexadecimal value 0x5F corresponding to the scancode “7” can be seen in index 32-33. When pressing Left Shift, it was found that the first element of the array is modified. This means that unlike for the other protocols, there is no special field for the modifiers.

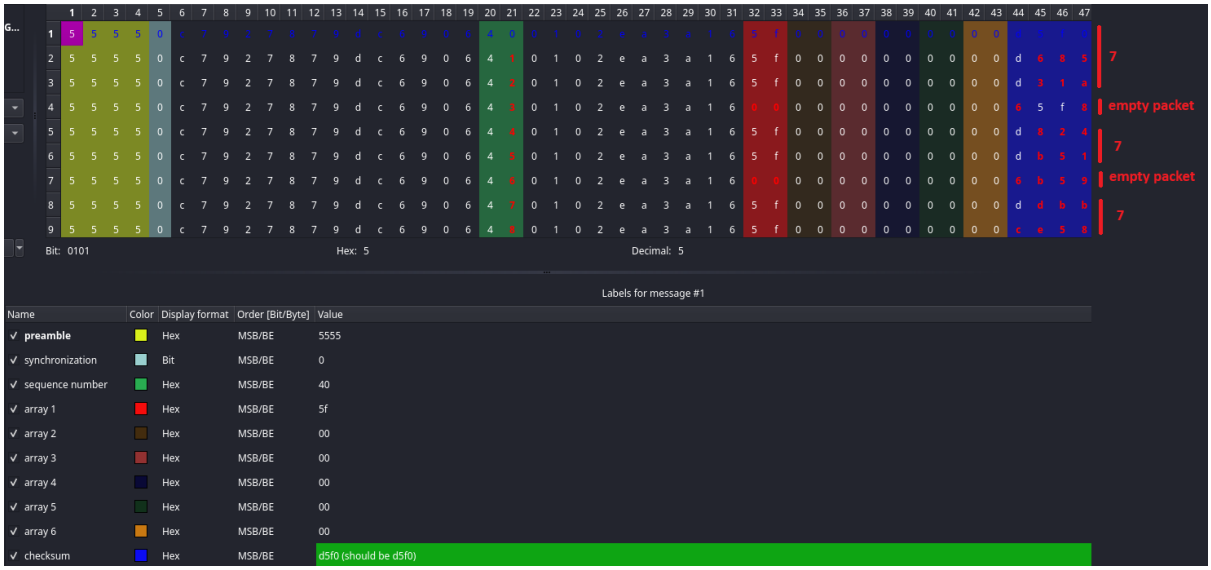


Figure 64. Reverse-engineered protocol for the E1050 keyboard

Rapoo mouse M10

Multiple mouse actions were captured and analyzed. The demodulation variables are similar to the ones for the keyboard. The CRC parameters are identical as well. Figure 65 shows the reverse-engineered protocol for the M10 mouse.

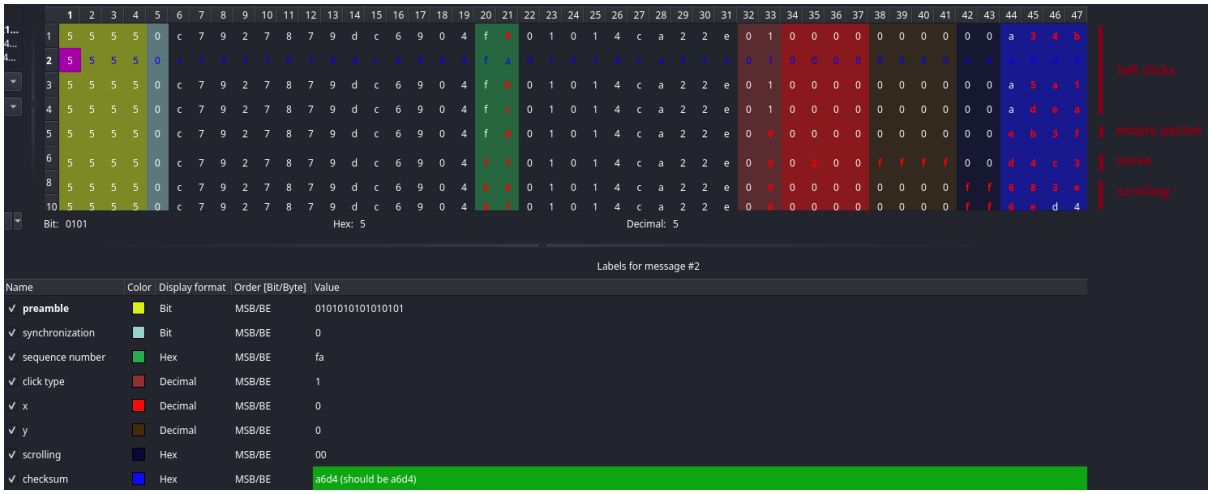


Figure 65. Reverse-engineered protocol for the M10 mouse

By comparing the M10 mouse and the E1050 keyboard, it can be assumed that the address is 7 bytes long (hex 6 to hex 19 in Figure 65). The difference is in hex 19: 0x06 for the keyboard and 0x04 for the mouse. The Crazyradio PA seems to accept an address of a maximum of 5 bytes; therefore, in the script, the first five were considered to be the address and a check was made on index 19 to know if the packet corresponds to a mouse or a keyboard action.

Result

It is possible to sniff and spoof the mouse and the keyboard. The implementation of the exploit is available in the *devices/Rapoo* folder [2].

5.2.4. Edenwood

The Edenwood 963716 CWL01 consists of a keyboard and a mouse, as seen in Figure 66. The results of the USB sniffing performed on the devices' dongle can be found in Table 5.

Figure 66. Edenwood 963716 CWL01 set

USB sniffing

Table 5. Data found with USBPcap for the Edenwood dongle

Vendor ID	**0x25A7** (Beken Corporation)
Product ID (dongle)	**0xFA61** (2.4G Receiver)
String descriptor	2.4G Receiver
Interfaces	Has Mouse and Keyboard interfaces
HID Data	For mouse: 5 buttons, 32 bits for X/Y, 8 bits wheel, 8 bits AC pan For keyboard: 8 bits modifiers, 6-elements array

OSINT

No FCC ID was found. Edenwood, owned by "Electro Depot," lists the devices on its website [39], but no useful information was available. Since nothing was found, the keyboard was opened to check the RF chip, but it was obfuscated as seen in Figure 67.

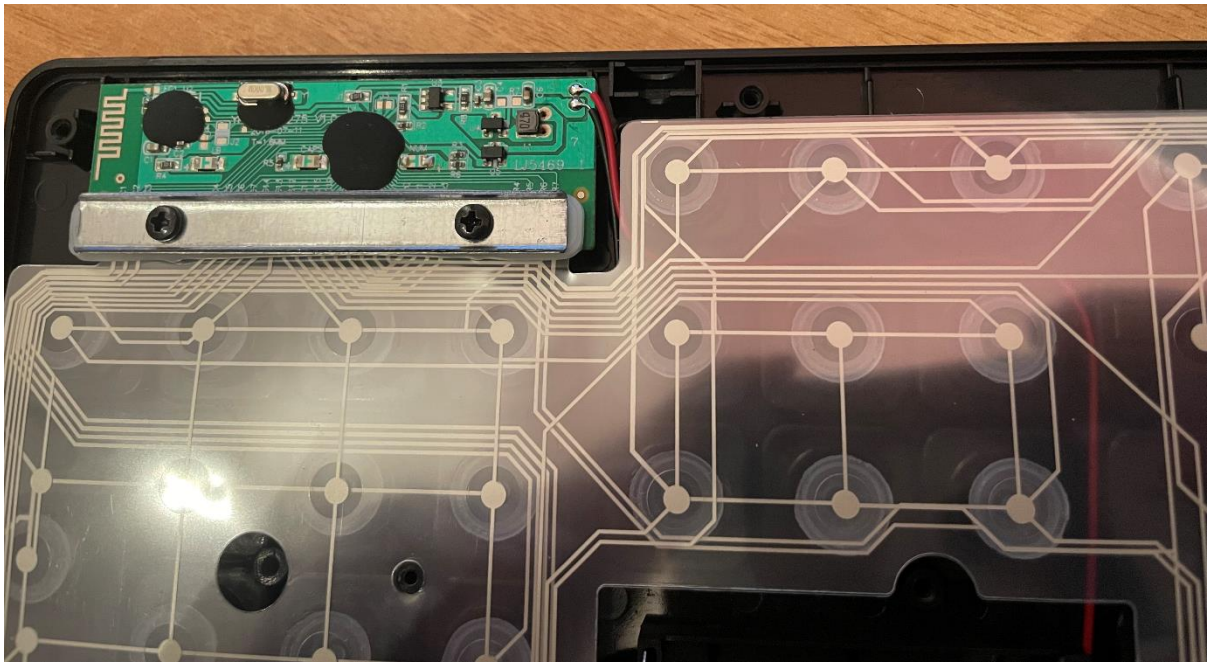


Figure 67. PCB board of the Edenwood keyboard

Reverse-engineer

Edenwood keyboard

A capture of multiple "7" and "L Shift" presses was analyzed. Figure 68 shows that the signals were demodulated using the FSK option, with a data rate of 2 Mbps.

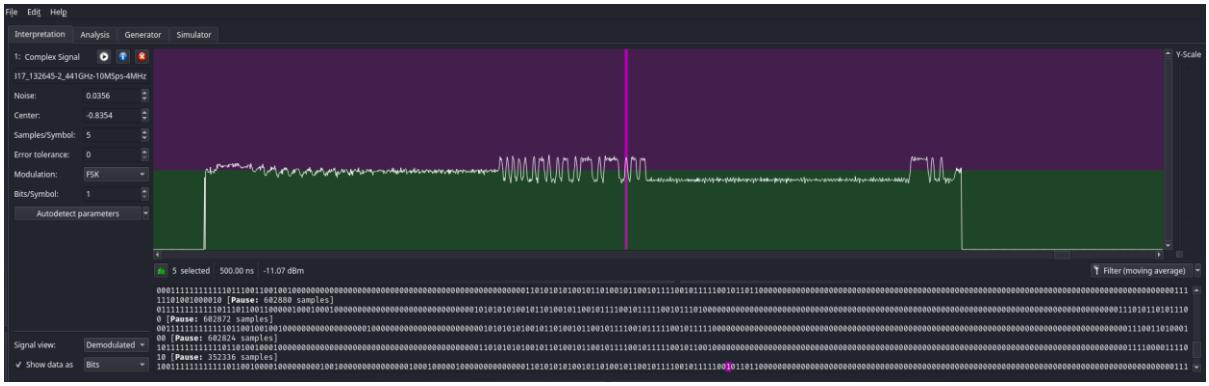


Figure 68. Demodulation parameters for the Edenwood keyboard

The CRC is 16-bit long and has been reverse-engineered using RevEng. Figure 69 shows that the parameters are poly = *0x1021* and init = *0x6818*.

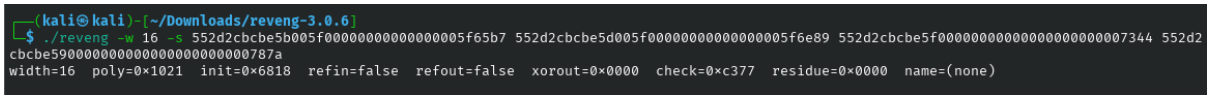


Figure 69. Reverse-engineering of the CRC function of the Edenwood keyboard using RevEng

Figure 70 clearly identifies the sequence number, modifiers, and data array. Additionally, two hexadecimal values in index 34-35 change whenever the data part differs. It seems to be an addition from all the content in the data section.

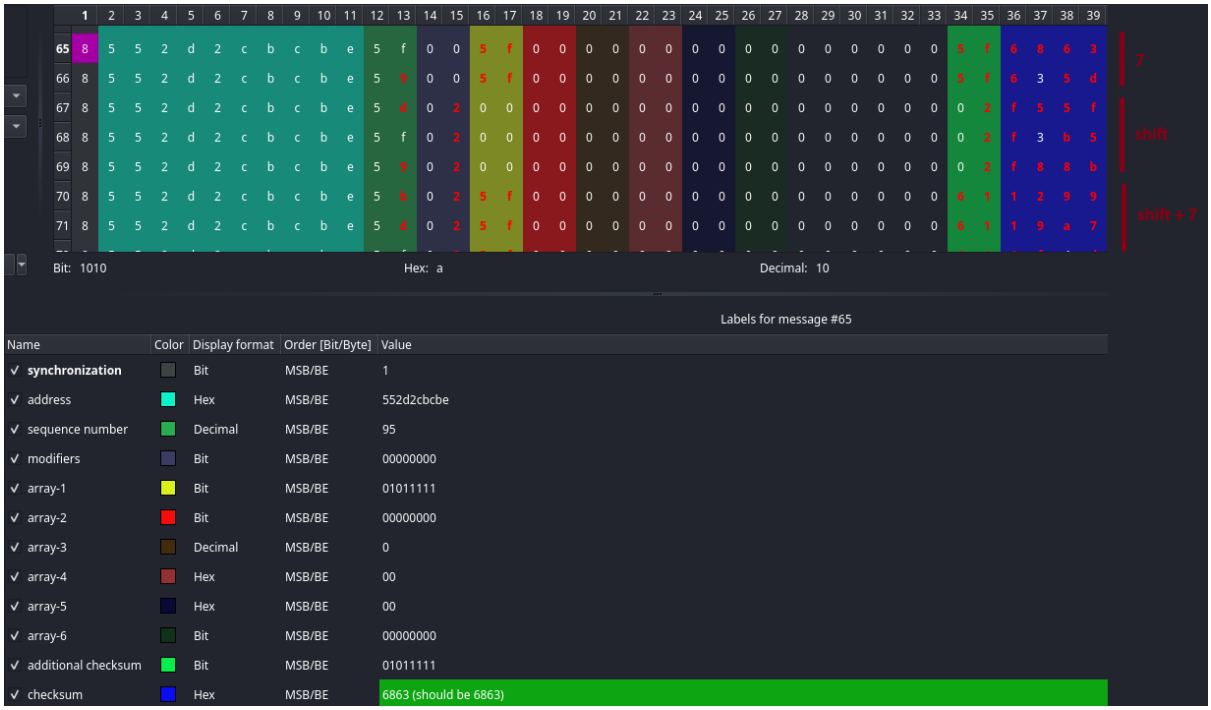


Figure 70. Reverse-engineered protocol for the Edenwood keyboard

By using the *fuzz_channels* function, it was discovered that four channels are used for the keyboard in this set: 20, 40, 54, and 81.

Edenwood mouse

Left clicks, mouse movement, and scrolling were captured and analyzed. The demodulation variables match those of the keyboard, and the CRC parameters are identical. In Figure 71, additional fields were identified, including click type, x/y movement, and scrolling.

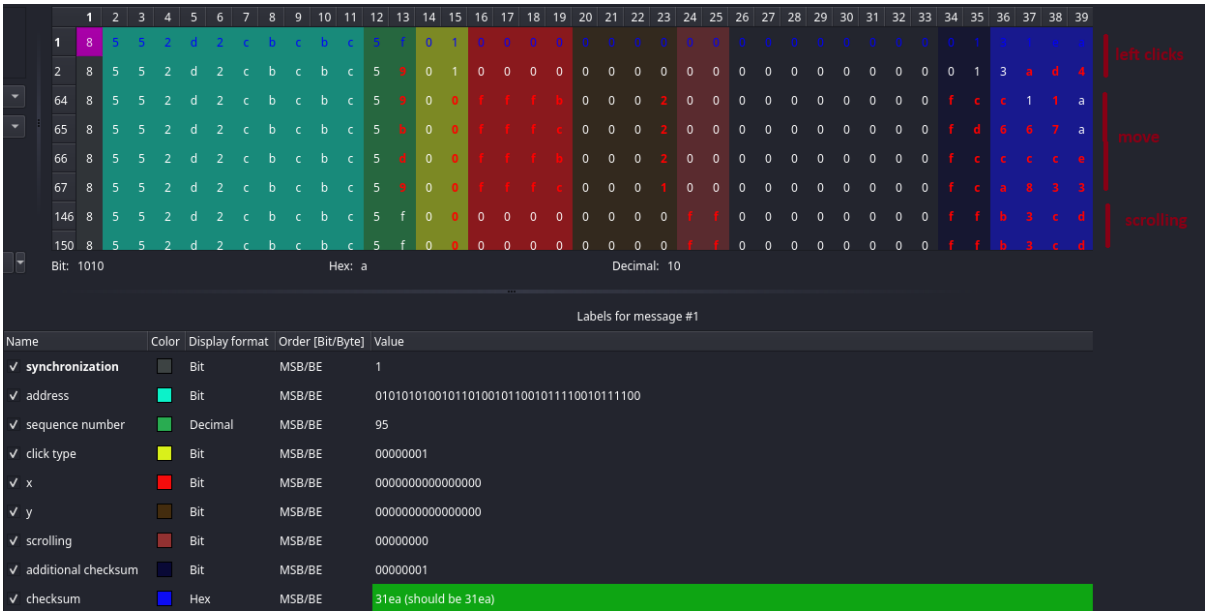


Figure 71. Reverse-engineered protocol for the Edenwood mouse

An additional checksum is present as well. Fuzzing with the Crazyradio PA revealed that when the additional checksum exceeds 0xFF, it resets to 0x00. Additionally, the checksum decreases by one for each part of the data that surpasses 0xFF. Figure 72 shows the code to calculate this checksum.

```python
def calculate_checksum(self, click_type, x, y, scrolling):
    """Calculate the checksum for the Edenwood mouse.

    Args:
        click_type (int): The click value.
        x (int): The x value.
        y (int): The y value.
        scrolling (int): The scrolling value.

    Returns:
        bytes: The checksum for the Edenwood mouse in byte.
    """
    offset = 0
    if scrolling > 255:
        offset += 1
    if x > 255:
        offset += 1
    if y > 255:
        offset += 1
    return  ((click_type + x + y + scrolling - offset) % 256).to_bytes(1, "big")
```

Figure 72. Python method to calculate the additional checksum for Edenwood devices

Result

It is possible to sniff and spoof the mouse and the keyboard. The implementation of the exploit is available in the *devices/Edenwood* folder [2].

5.2.5. Qware

The Qware QW PCB-238BL consists of a keyboard and a mouse, as seen in Figure 73. The results of the USB sniffing performed on the devices' dongle can be found in Table 6.

Figure 73. Qware QW PCB-238BL set

USB sniffing

Table 6. Data found with USBPcap for the Qware dongle

Vendor ID	**0x1EA7** (SHARKOON Technologies GmbH)
Product ID (dongle)	**0x0066** (Mediatrack Edge Mini Keyboard)
String descriptor	2.4G Mouse
Interfaces	Has Mouse and Keyboard interfaces
HID Data	For mouse: 8 buttons, 24 bits for X/Y, 8 bits wheel, 8 bits AC pan For keyboard: 8 bits modifiers, 6-elements array

OSINT

The vendor ID and product ID descriptions do not match the Qware devices but likely share the same RF chip. No FCC ID was found, and while Qware has a page about the set [40], it provides no useful information. Similarly, no details were available for the Sharkoon

Mediatrack device. Since no documentation was found, the keyboard was opened to inspect the RF chip, but it was obfuscated, as shown in Figure 74.

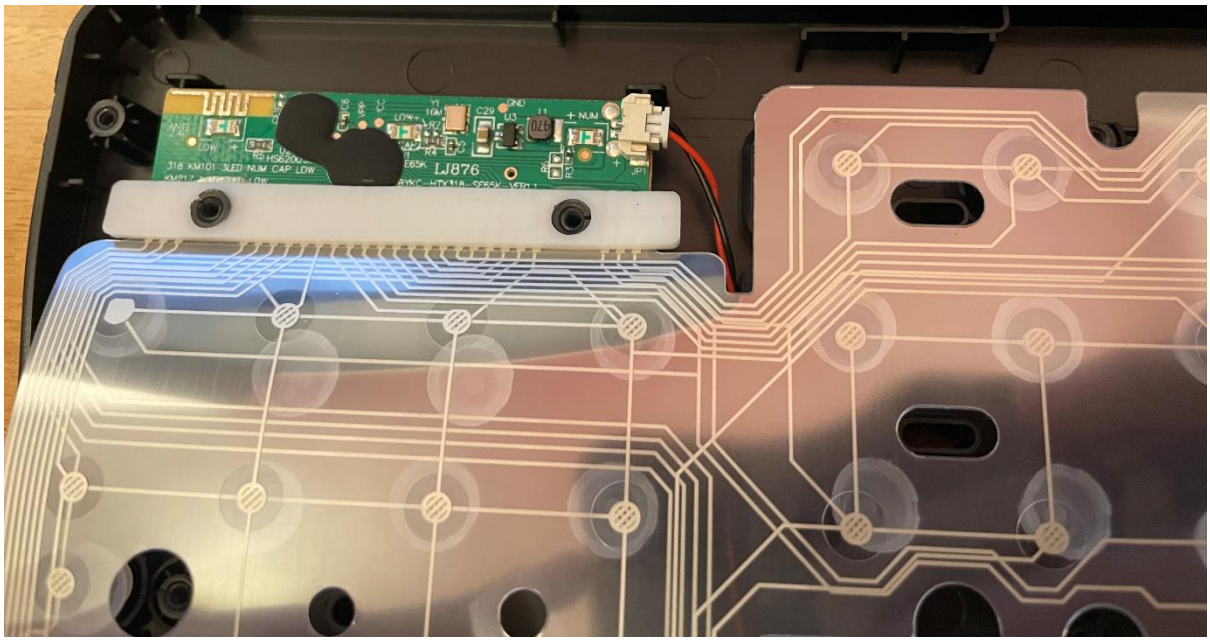


Figure 74. PCB board of the Qware keyboard

Reverse-engineer

Qware keyboard

A capture of multiple "7" presses was analyzed. Figure 75 shows that the signals were demodulated using the FSK option, with a data rate of 2 Mbps.

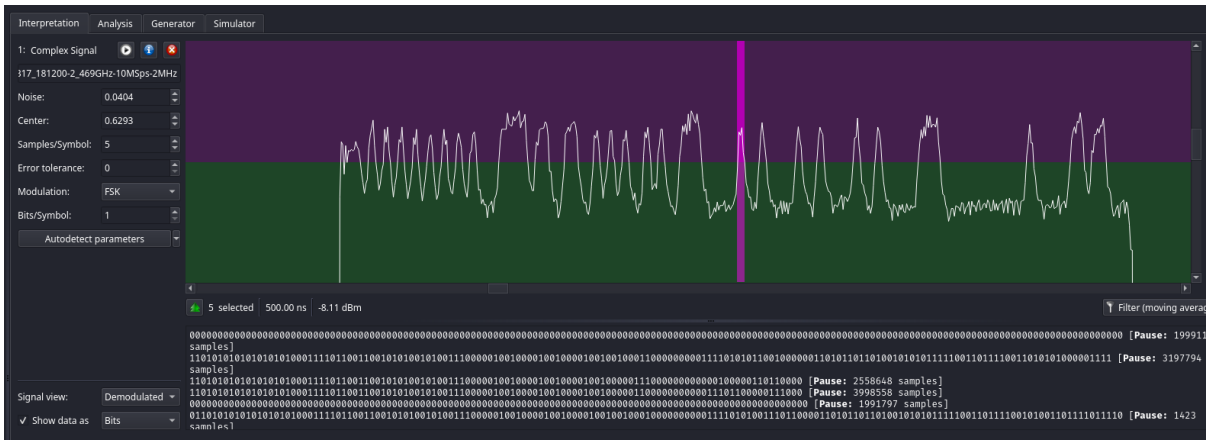


Figure 75. Demodulation of a “7” packet sent by the Qware keyboard

Two types of packets were identified based on their length. In Figure 76, the CRC parameters were reverse-engineered using RevEng, with poly = *0x1021* and init = *0xC5C5*. These parameters apply to both long and short packets.

```
┌──(kali㉿kali)-[~/Downloads/reveng-3.0.6]
└─$ ./reveng -w 16 -s 3d99529c1212124000f5640d6d2af9bc8f8f 3d99529c1212124600f5640d6d2af9bcd507
 3d99529c1212124400f53b0d6d2af9bca6f7 3d99529c1212124000f53b0d6d2af9bc3a18
width=16  poly=0×1021  init=0×c5c5  refin=false  refout=false  xorout=0×0000  check=0×e58b  res
idue=0×0000  name=(none)
```

Figure 76. Reverse-engineering of the CRC function of the Qware keyboard using RevEng

Figure 77 presents the reverse-engineered protocol. The short packets do not contain useful information, they may be used for some sort of acknowledgement. In the long packets, pressing "7" causes changes in index 26-27, suggesting that this is the first element of the array. Hex 26 to 37, just before the CRC, spans 48 bits, which aligns with the typical size of an HID array.

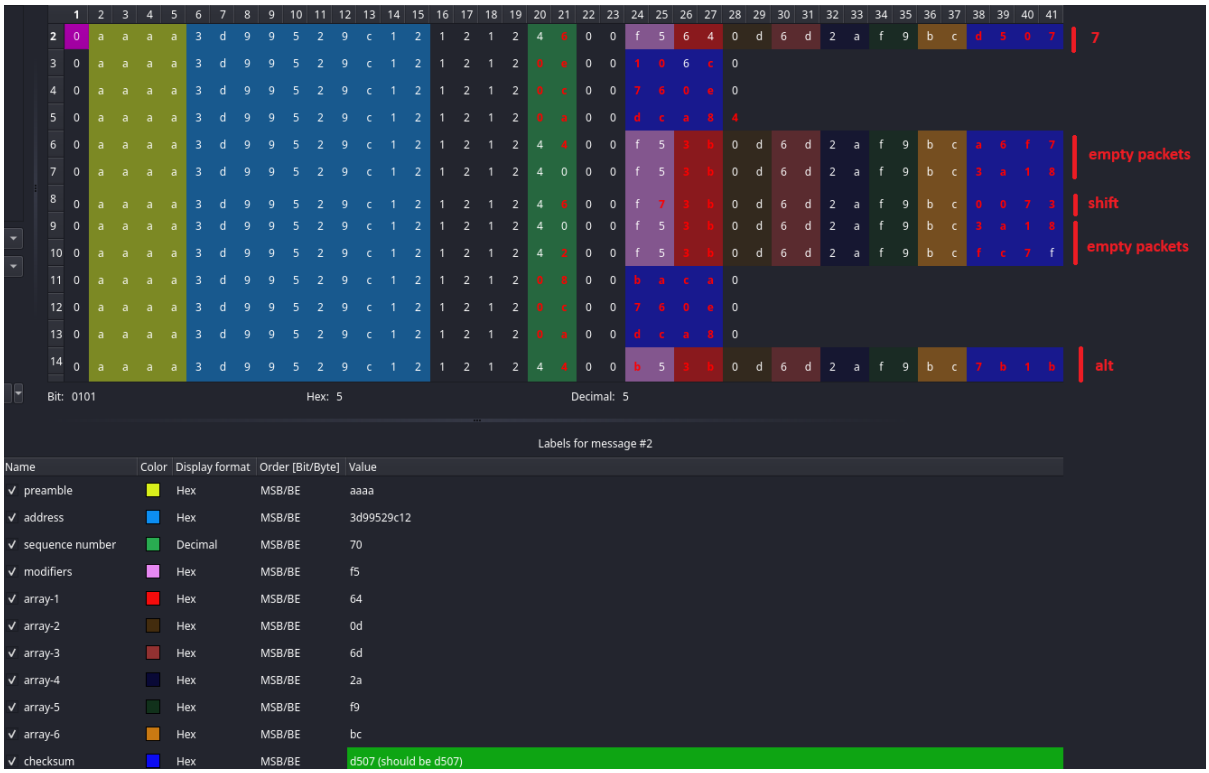


Figure 77. Reverse-engineered protocol for the Qware keyboard

It was observed that pressing only modifier keys resulted in non-zero values in the data array, which is unexpected as these fields are typically empty (filled with zeros) when no regular keys are pressed. A XOR operation was performed between the array when pressing the "7" key and the “empty” array when pressing only a modifier. For instance, 0x64 XOR 0x3B equals 0x5F, corresponding to the scancode for the "7" key. Similarly, XORing the modifier values with the

default value 0xF5 gives the standard values for the modifiers. This means that the data part of the protocol is XORed with a mask.

Qware mouse

The mouse signals were complicated to capture because there were many channels. The Crazyradio PA was used to fuzz all the channels. The keyboard's address, without the last byte, was used to capture the mouse signals.

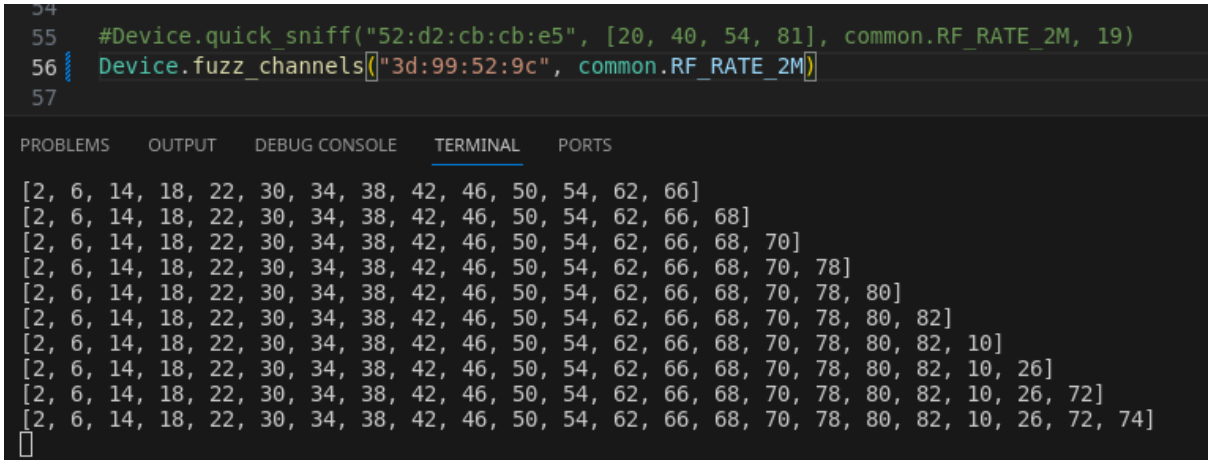


Figure 78. Fuzzing of the Qware's mouse channels using the Crazyradio

Then, the *quick_sniff* method was used to capture the mouse signals. Many signals, as seen in Figure 79, with the last byte being 0x04, 0xA4, and 0x0C, were observed without interaction with the mouse, suggesting they are likely keep-alive packets. Additionally, other addresses ending in 0x10 and 0x11 were discovered.

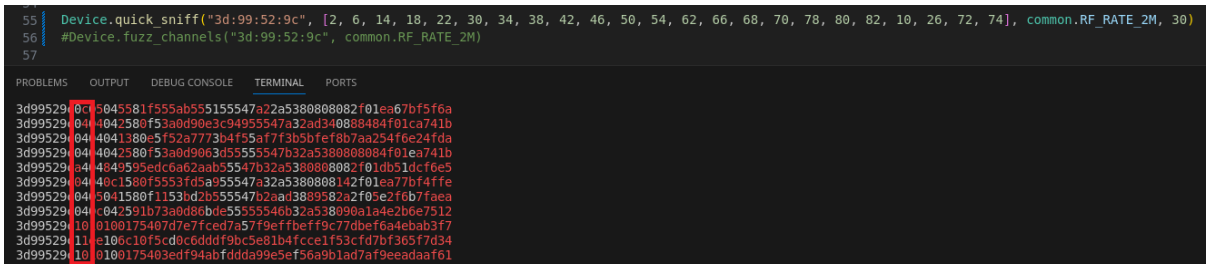


Figure 79. Capture of Qware's mouse signals using the Crazyradio

The 0x10 address appears to correspond to dongle packets, while 0x11 seems to represent mouse packets. The last four packets from Figure 80 were taken and inputted into RevEng. Each time RevEng failed to find the CRC parameters, the last hexadecimal value of the packets was removed. Eventually, the correct parameters were identified as shown in Figure 81.

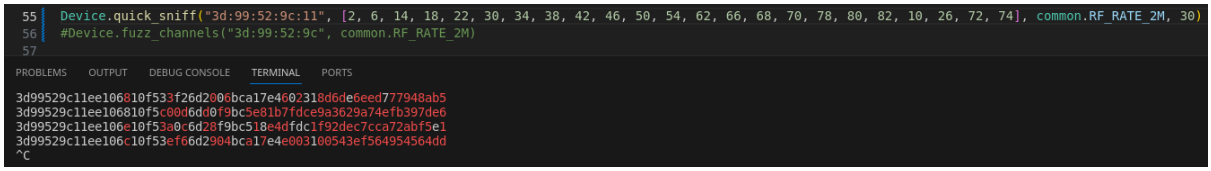


Figure 80. Capture of Qware's mouse packets

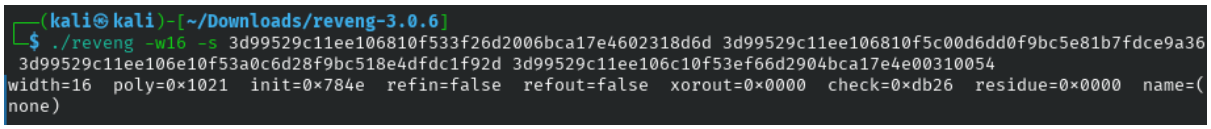


Figure 81. Reverse-engineering of the CRC function of the Qware mouse using RevEng

The packet length is 23 bytes. Different actions were captured using the Crazyradio PA, and the hex data was then copied and pasted into URH for further analysis. Figure 82 illustrates the reverse-engineered protocol. This protocol also utilizes masks.

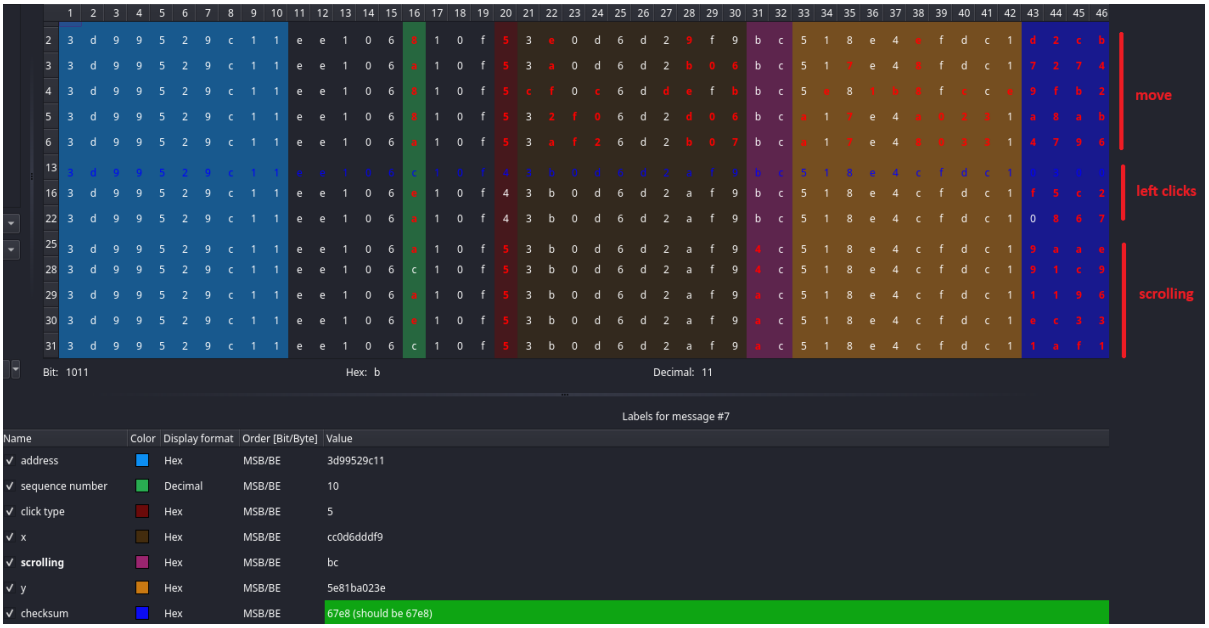


Figure 82. Reverse-engineered protocol for the Qware mouse

Result

It is possible to sniff and spoof the mouse and the keyboard. The implementation of the exploit is available in the *devices/Qware* folder [2].

5.2.6. Think Xtra

The Think Xtra (TX) ms6-TXn-wh pictured in Figure 83 is a mouse bought in a secondhand shop. The results of the USB sniffing performed on the device's dongle can be found in Table 7.

Figure 83. Think Xtra ms6-TXn-wh mouse

USB sniffing

Table 7. Data found with USBPcap for the TX dongle

Vendor ID	**0x25A7** (Areson Technology Corp or Beken Corporation)
Product ID (dongle)	**0x0701** (Unknown)
String descriptor	Smart Wireless Device
Interfaces	Has a mouse and 2 keyboard interfaces
HID Data	5 buttons, 32 bits for X/Y, 8 bits wheel, 8 bits AC pan

OSINT

The vendor ID corresponds either to Areson Technology or Beken Corporation, similar to the Rapoo mouse. No further information was found about the vendor ID and product ID. The packaging from Amazon has a website on the back [41], but it no longer works. With no additional information available, both the mouse and the dongle were opened. Both devices have epoxy resin obfuscating the RF chip, as shown in Figure 84 and Figure 85.

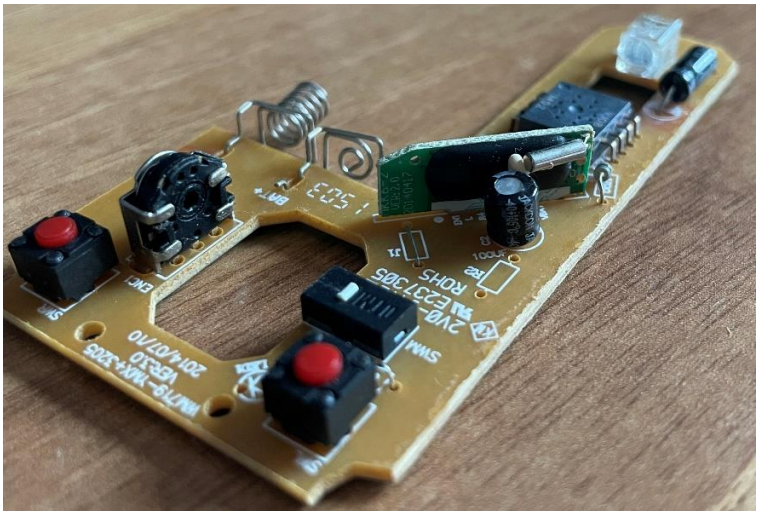


Figure 84. PCB of the TX mouse

Figure 85. PCB of the TX dongle

Reverse-engineer

A capture of multiple left click presses was analyzed. The signals were demodulated using the FSK option with a data rate of 1 Mbps, as seen in Figure 86.

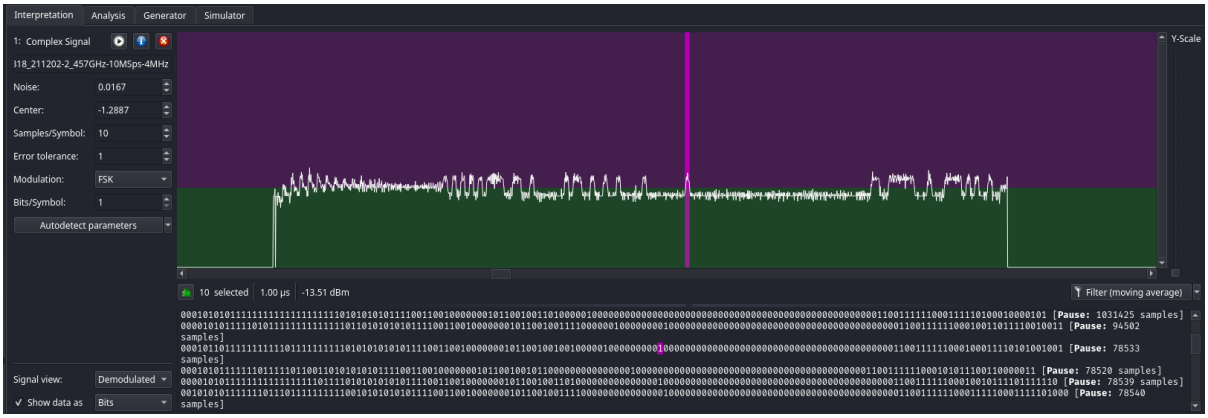


Figure 86. Demodulation of a left click packet sent by the TX mouse

The CRC is 16 bits long, and RevEng was used to determine its parameters. The results from RevEng, shown in Figure 87, indicate the parameters: poly = *0x1021* and init = *0x6818*.

```
┌──(kali㉿kali)-[~/Downloads/reveng-3.0.6]
└─$ ./reveng -w16 -s 557990164d02010000000000033f1e919 557990164f04010000000000033f16d1b
557990164900010000000000033f1824b 557990164b0001000000000033f15cc1
width=16  poly=0×1021  init=0×6818  refin=false  refout=false  xorout=0×0000  check=0×c
377  residue=0×0000  name=(none)
```

Figure 87. Reverse-engineering of the CRC function of the TX mouse using RevEng

Figure 88 shows the reverse-engineered protocol. The sequence number and packet type each take up 2 bits but are displayed as a single hexadecimal value in the figure.

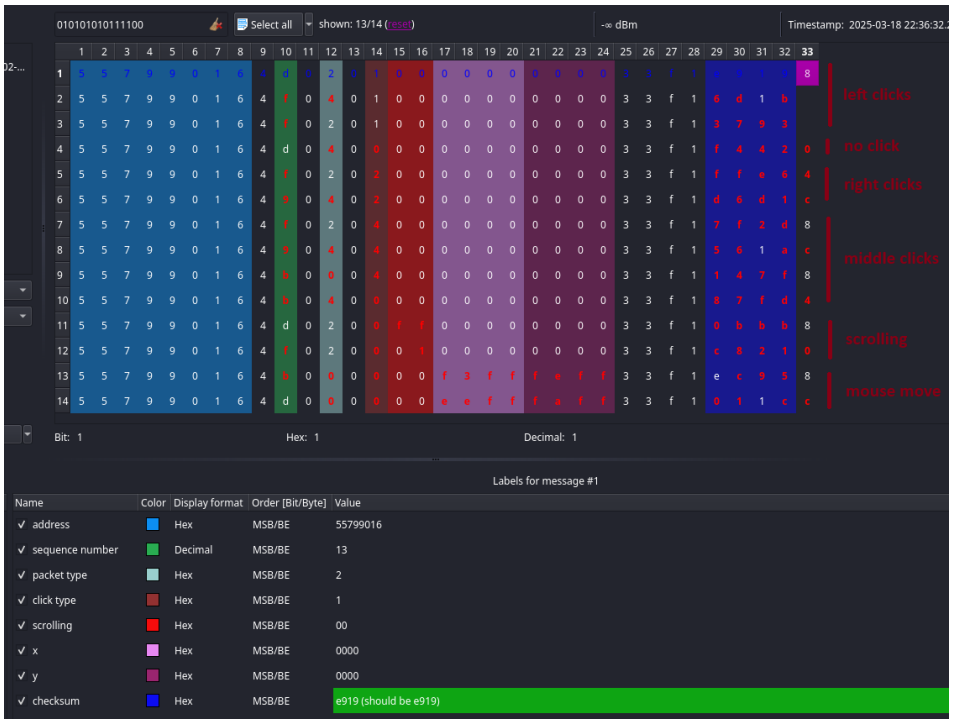


Figure 88. Reverse-engineered protocol for the TX mouse

Figure 89 shows the packet type field with the bits view. The packet type field is 0b*01* at the beginning of a packet, *10* for the second and last packet, and *00* for a normal packet.

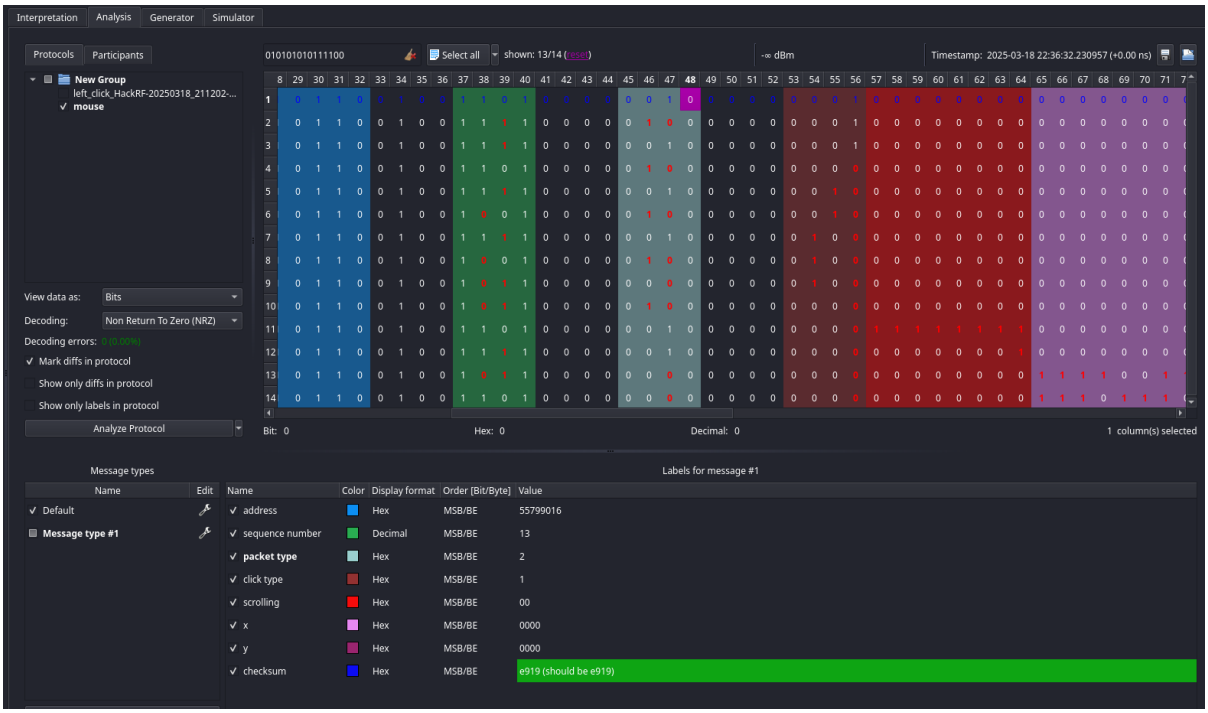


Figure 89. Packet type field of the TX mouse

Sniffing and spoofing were implemented using the Crazyradio PA. By mistake, a packet with a packet type field value of 0b*11* was sent, which resulted in a random keyboard input. Manually crafted packets with different values were sent to determine the keyboard's protocol. Based on the findings from other keyboards, it was assumed that the protocol should include a 1-byte modifier and a 6-byte array. The final protocol was determined through fuzzing. Figure 90, shows valid packets that trigger keyboard actions.

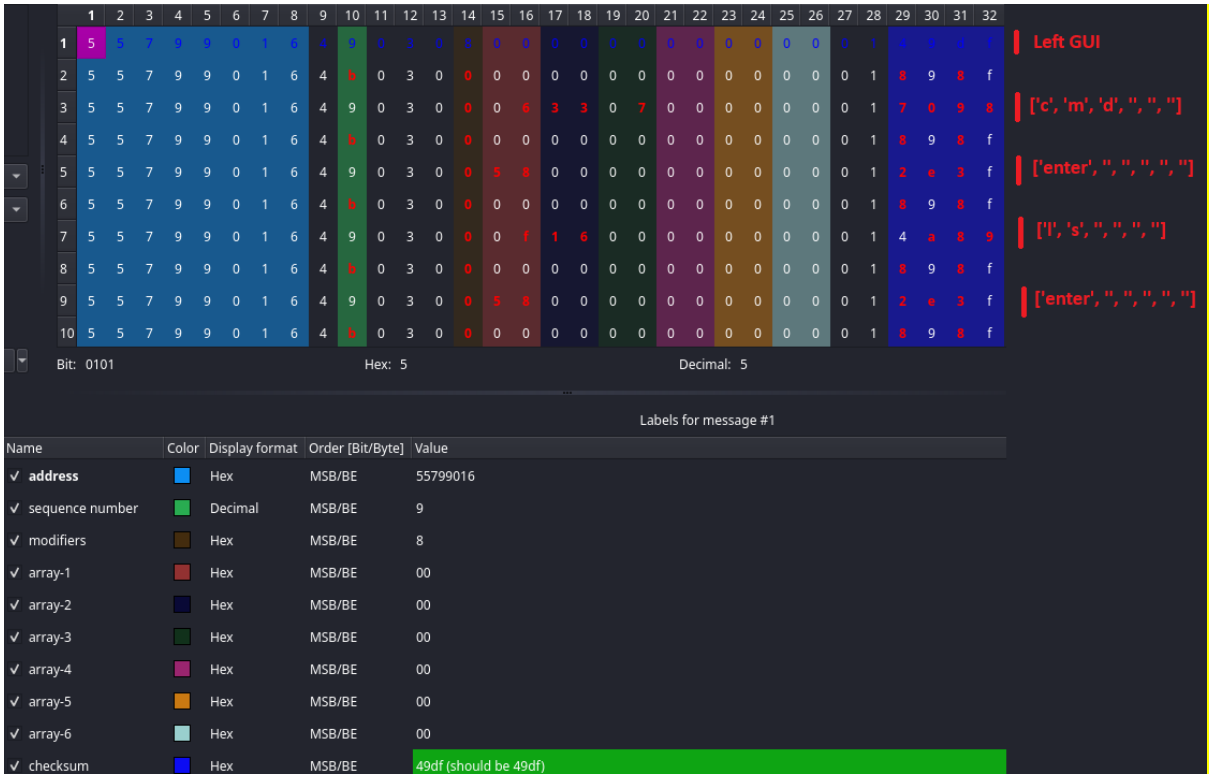


Figure 90. Reverse-engineered protocol for TX keyboards

Result

The sniffing and spoofing of the mouse were successfully implemented. It is also possible to send keyboard packets. While the keyboard sniffing was implemented, it could not be tested. The implementation of the exploit is available in the *devices/TX* folder [2].

5.2.7. Hama

The Hama AKMW-100 set consists of the F2182637 keyboard and the F2182637 mouse, as seen in Figure 91. The results of the USB sniffing performed on the devices' dongle can be found in Table 8.

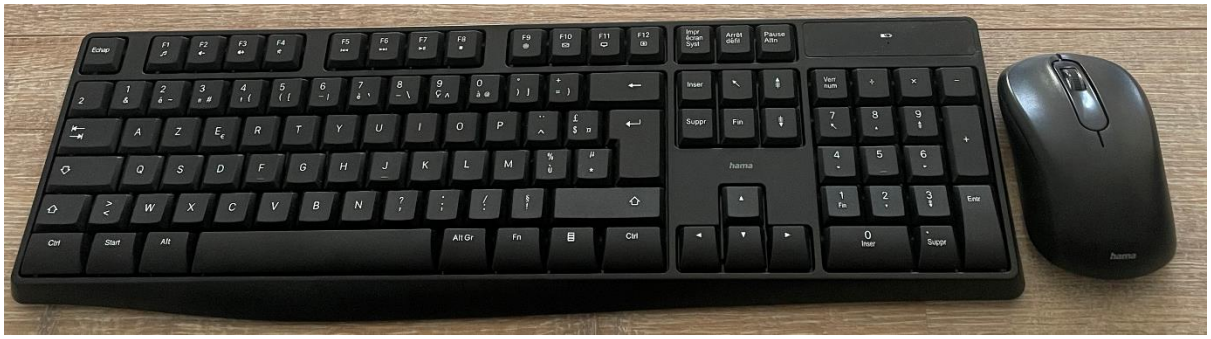


Figure 91. Hama AKMW-100 set

USB sniffing

Table 8. Data found with USBPcap for the Hama dongle

Vendor ID	**0x3151** (Unknown)
Product ID (dongle)	**0x3021** (Unknown)
String descriptor	Hama AKMW-100
Interfaces	Has Mouse and Keyboard interfaces
HID Data	For mouse: 5 buttons, 32 bits for X/Y, 8 bits wheel, 8 bits AC Pan For keyboard: 8 bits modifiers, 6-elements array

OSINT

Hama has a website, but the AKMW-100 set is not listed. No FCC ID was found. By checking the USB ID database [26], it was discovered that the vendor ID corresponds to Yichip Microelectronics. In Figure 92, the mouse was opened to identify the RF chip, the YC1011-T [42]. This chip supports Bluetooth and 2.4GHz proprietary communication, but no datasheet or protocol details were found.

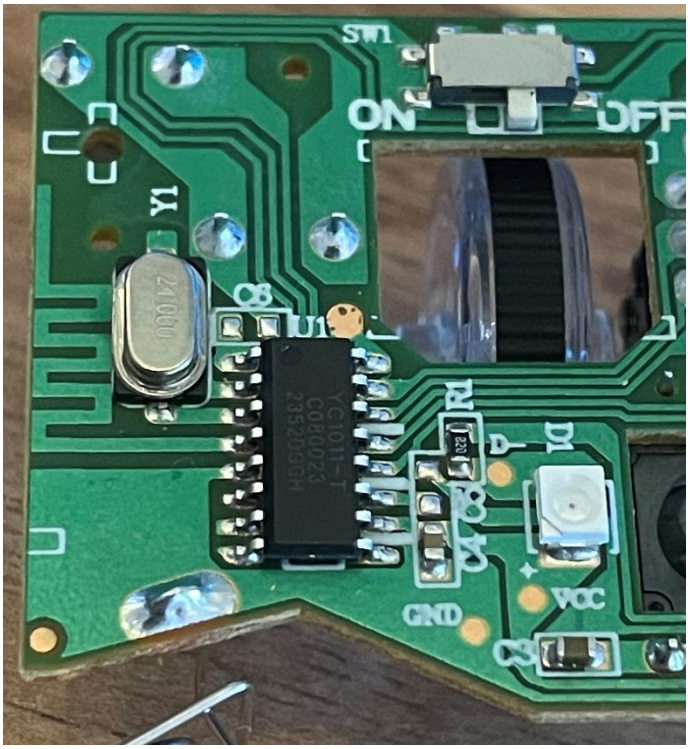


Figure 92. PCB of the Hama mouse

Reverse-engineer

Hama keyboard

Multiple "7" key presses were recorded. The signal was demodulated using FSK, and it was observed that the samples per symbol varied between 10 (Figure 93) at the beginning and at the end, and 30 (Figure 94) in the middle.

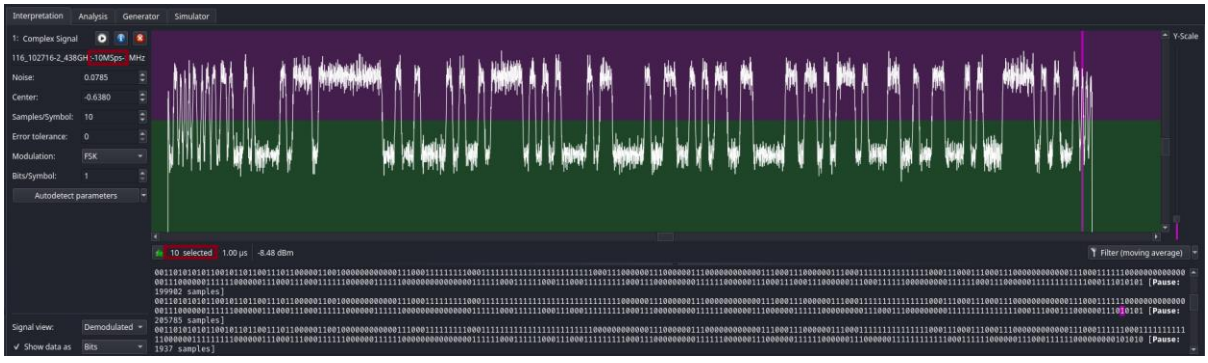


Figure 93. Demodulated view of a “7” signal sent by the Hama keyboard with 10 samples per symbol at the end of the signal

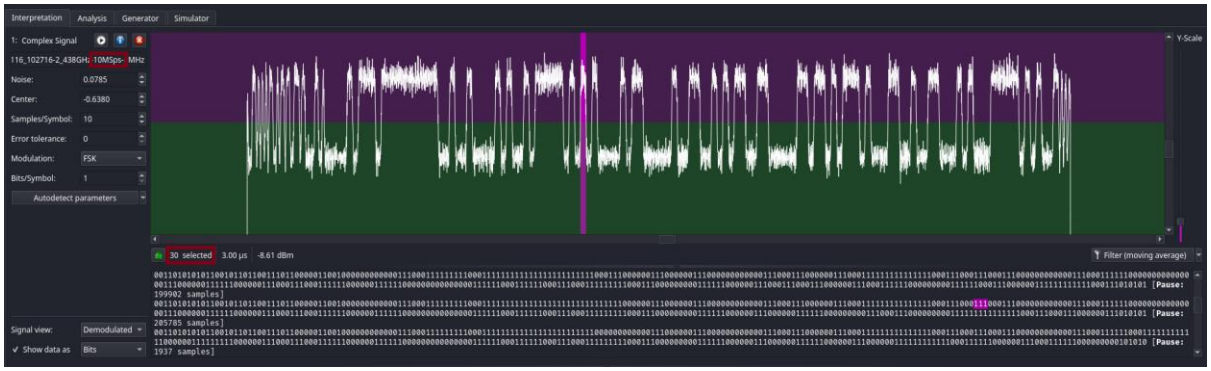


Figure 94. Demodulated view of a “7” signal sent by the Hama keyboard with 30 samples per symbol in the middle of the signal

The protocol was first analyzed using 10 samples per symbol, but the CRC could not be found with RevEng. In Figure 95, part of the suspected 72-bit CRC shows a pattern of three consecutive identical bits. To work around this, the analysis was redone using 30 samples per symbol, where bit sequences like “000 111” in 10 samples per symbol condensed into “0 1” in 30 samples per symbol.

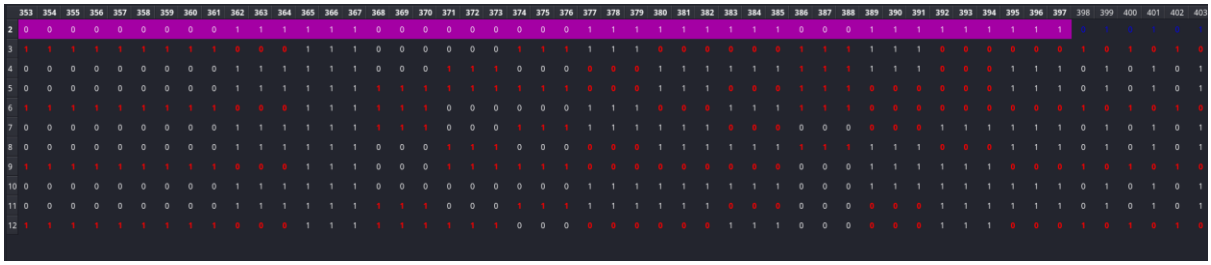


Figure 95. Identification of the CRC for the Hama keyboard with 10 samples per symbol

A new decoding was created to remove the first bits, aligning the packets. In Figure 96, a sequence number is immediately identified in index 20 and 21, which was promising.

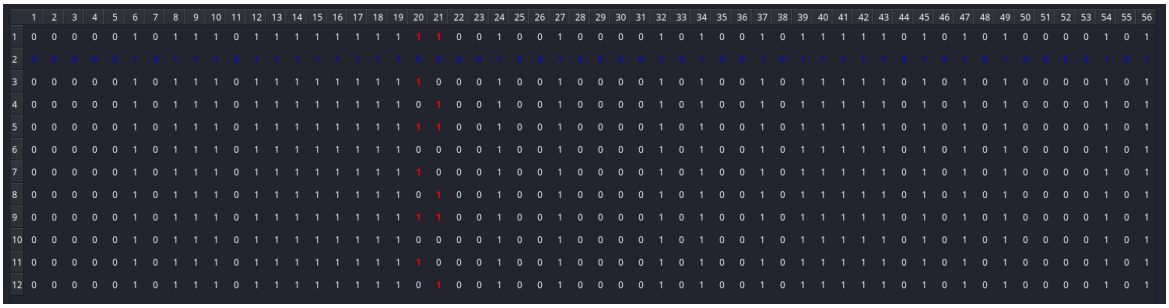


Figure 96. Hama keyboard’s packets aligned

In Figure 97, four packets were given to RevEng to find the CRC parameters.

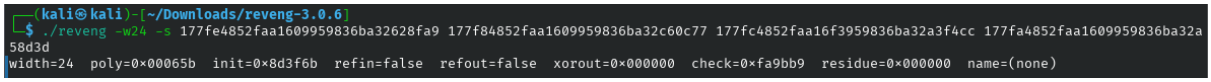


Figure 97. Identification of the CRC parameters of the Hama keyboard using RevEng

Figure 98 shows the reverse-engineered protocol for the Hama keyboard. When the key “7” is pressed, the indexes 22-23 in Figure 98 change. This byte represents the first element of the array. The rest of the array is the following bytes, which end conveniently just before the CRC. It can also be seen that the array is not empty, suggesting the utilization of a mask.

The modifier labels are only 1 bit long, but since the view is in hexadecimal, they appear as a full hexadecimal value, making the packet look longer.

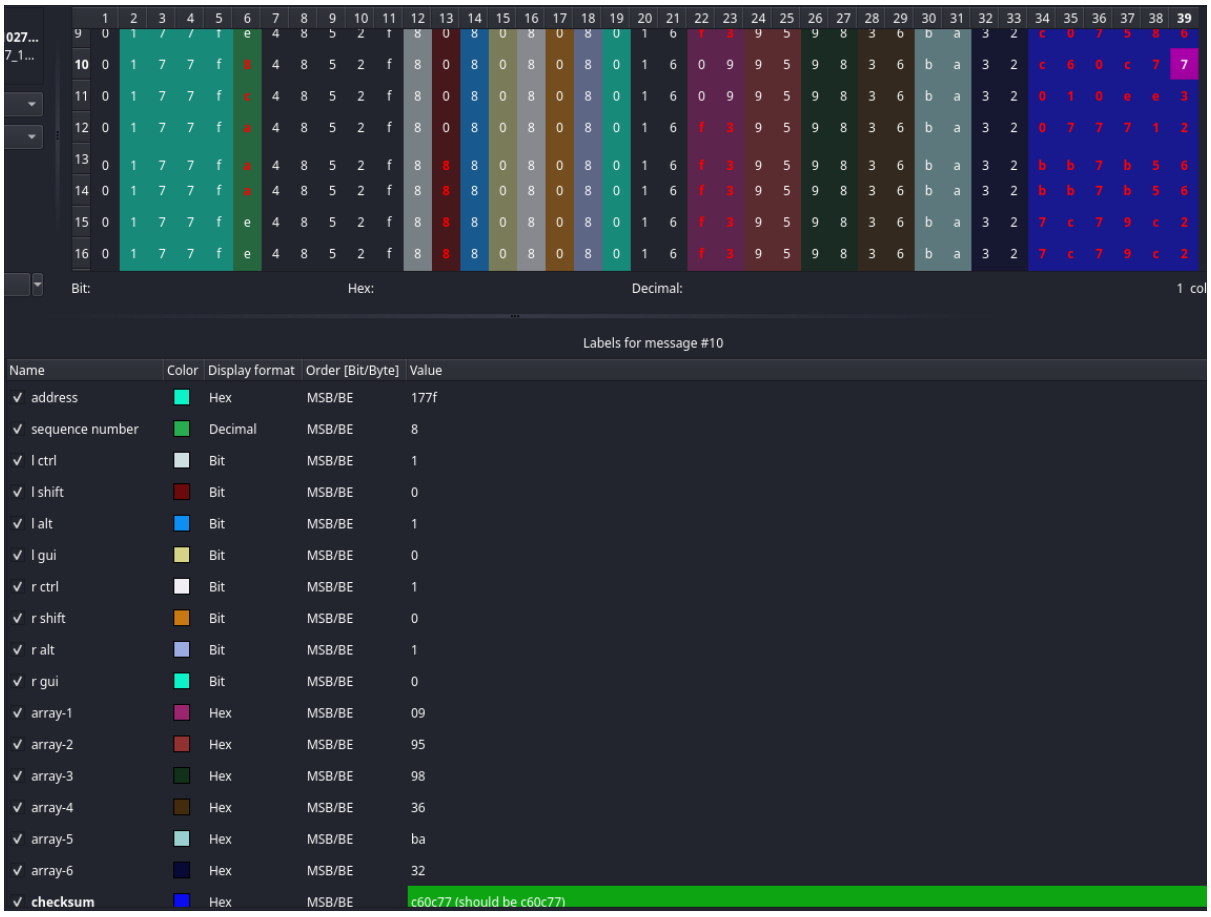


Figure 98. Reverse-engineered protocol of the Hama keyboard

The modifiers and the array values are not zero by default, suggesting the usage of a mask. In Figure 98, the whole modifiers' value when no key is pressed is 0x*AA* (0b*10101010*). By XORing the modifier value with the default value of 0x*AA*, the actual value is obtained. The

same logic applies to the array: the default value for the first element is 0x*F3*. XORing the value for the “7” key press (0x*09*) with the default (0x*F3*) gives 0x*FA*.

Additionally, when capturing signals on different frequencies, the mask changes. Figure 99 shows the “7” key press at a different frequency, where the mask applies to the entire packet, resulting in a different address and CRC parameters. After XORing the first element of the array (0x*D7*) with its default value (0x*2D*), the value 0x*FA* is again obtained. Across all the identified frequencies, each with a different mask, applying the XOR operation consistently results in 0x*FA* when the "7" key is pressed.

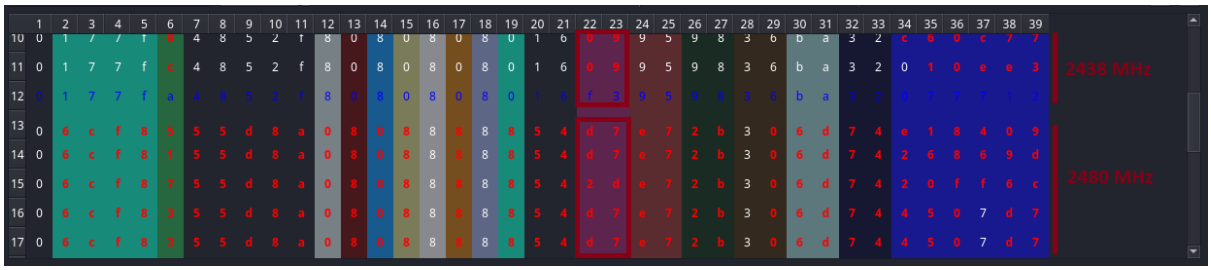


Figure 99. Comparison between “7” key press and “key release” Hama keyboard’s packets in two different frequencies

The signal data rate, with 30 samples per symbol and a capture of 10M samples, is 1/3 Mbps. The Crazyradio PA's nRF chip supports data rates of 250 Kbps, 1 Mbps, or 2 Mbps. When attempting to transmit the signal at 1 Mbps, the nRF's maximum packet size of 32 bytes posed a limitation. Given that the packet size at 30 samples per symbol is 16 bytes, using the 1 Mbps rate required tripling the packet size, resulting in 48-byte packets. As the Crazyradio PA cannot handle such packet sizes, even though the protocol was reverse-engineered and shown to be vulnerable, sniffing and spoofing could not be executed. However, a basic replay attack using recorded packets with the HackRF successfully confirmed the device's vulnerability.

Hama mouse

Multiple mouse movements were captured. Figure 100 shows that the demodulation variables are the same as the Hama keyboard. The data rate can be considered as either 1Mbps or 1/3Mbps.

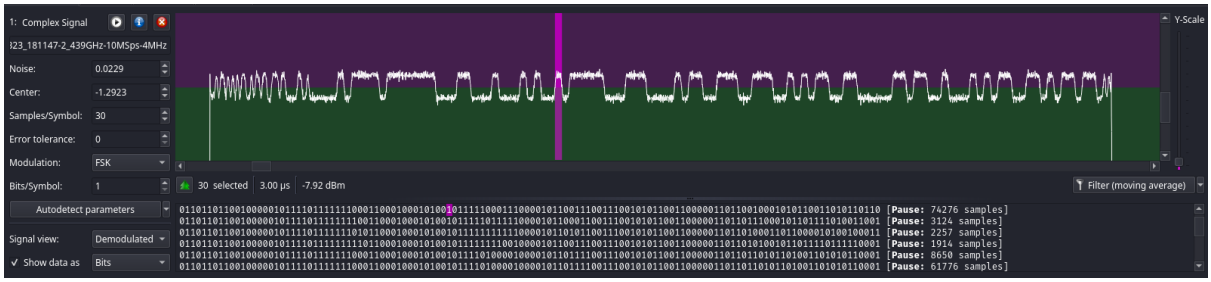


Figure 100. Demodulated Hama mouse's packet

A value of 30 samples per symbol was used for the reverse-engineering. The CRC is 24 bits long, like the Hama keyboard. Figure 101 shows that the CRC parameters were found using RevEng. The poly is *0x00065b*, and the init is *0x0b00e0*.

```
┌──(kali㉿kali)-[~/Downloads/reveng-3.0.6]
└─$ ./reveng -w24 -s 6417bfac452ffa16f39598366f34f8 6417bfec452f5a16f3959836888d79 6417bf8c452fe216f39598363ecfd4
6417bfac452fba160c6a9836c6f0f1
width=24  poly=0×00065b  init=0×0b00e0  refin=false  refout=false  xorout=0×000000  check=0×fcee49  residue=0×0000
00  name=(none)
```

Figure 101. Identification of the CRC parameters of the Hama mouse using RevEng

Multiple actions were recorded to reverse-engineer the protocol. Figure 102 shows the complete reverse-engineered protocol. With USBCAP, it was seen that there are 8 bits of AC pan, which is horizontal scrolling. However, the Hama mouse does not have this feature, so it could not be tested. It can be assumed that the AC scroll is in index 26-27.

It can be seen that the protocol also possesses a mask; when there is no click or no scroll, those values are still filled by default, respectively, with 0x2 and 0x98. As for the Hama keyboard, capturing signals on other frequencies gives another set of masks.

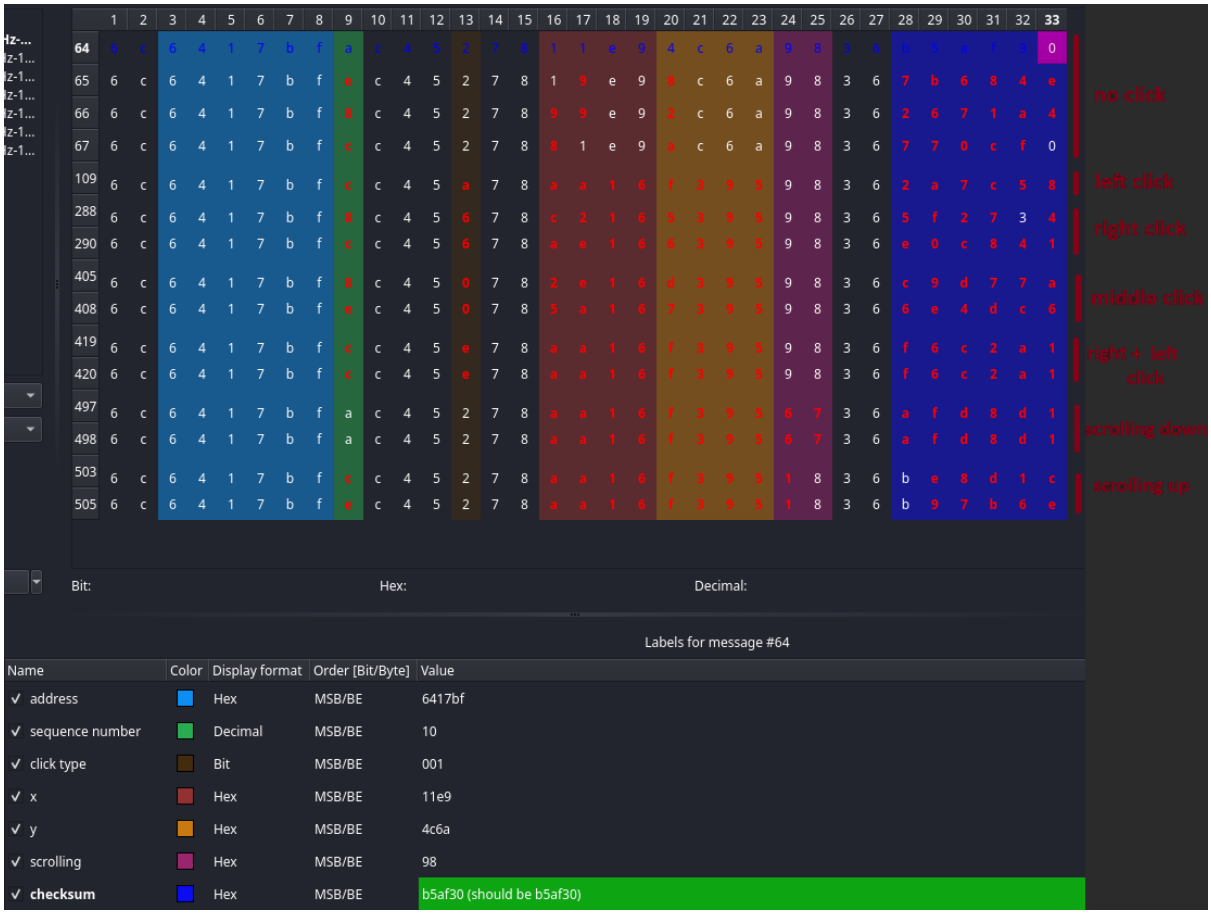


Figure 102. Reverse-engineered protocol of the Hama mouse

Because of the masks, it can be hard to understand the fields. Figure 103 shows the view of the click type in bits. When no click is pressed, the default value is 0b*001*; this is the XOR mask for the click type field. Therefore, after XORing, pressing the left click gives 0b*100*, the right click gives 0b*010*, the middle click gives 0b*001*, and a combination of clicks is the sum of the clicks. Normally, five buttons are allowed in the protocol, but as for the AC pan, there is no way to test the two remaining buttons because the mouse has only three buttons. The two remaining buttons can be considered to be at bit 50 and 51 in Figure 103.

Figure 104 shows the scrolling field in bits. The mask for this frequency is 0b*1001100*. When there is a scroll down, the value is 0b*11111111*, and when there is a scroll up, it is 0b*10000000*.

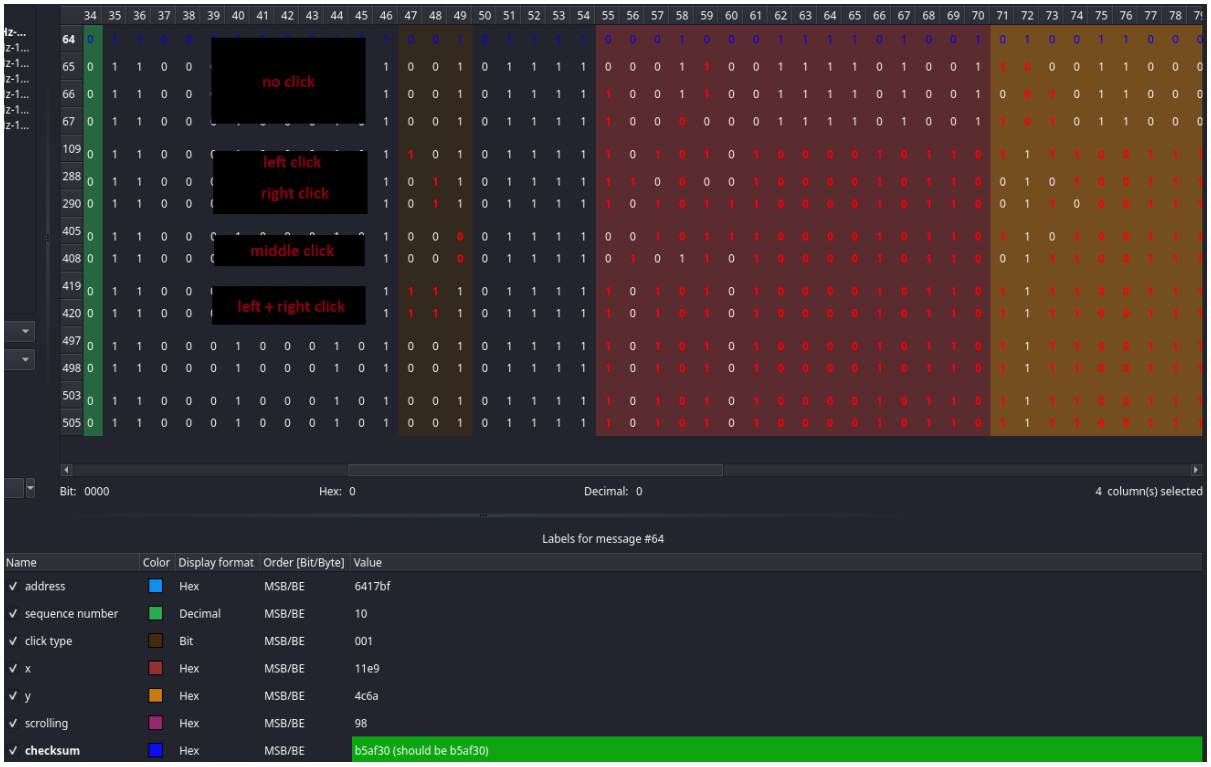


Figure 103. Visualization of the Hama's click type field in bits

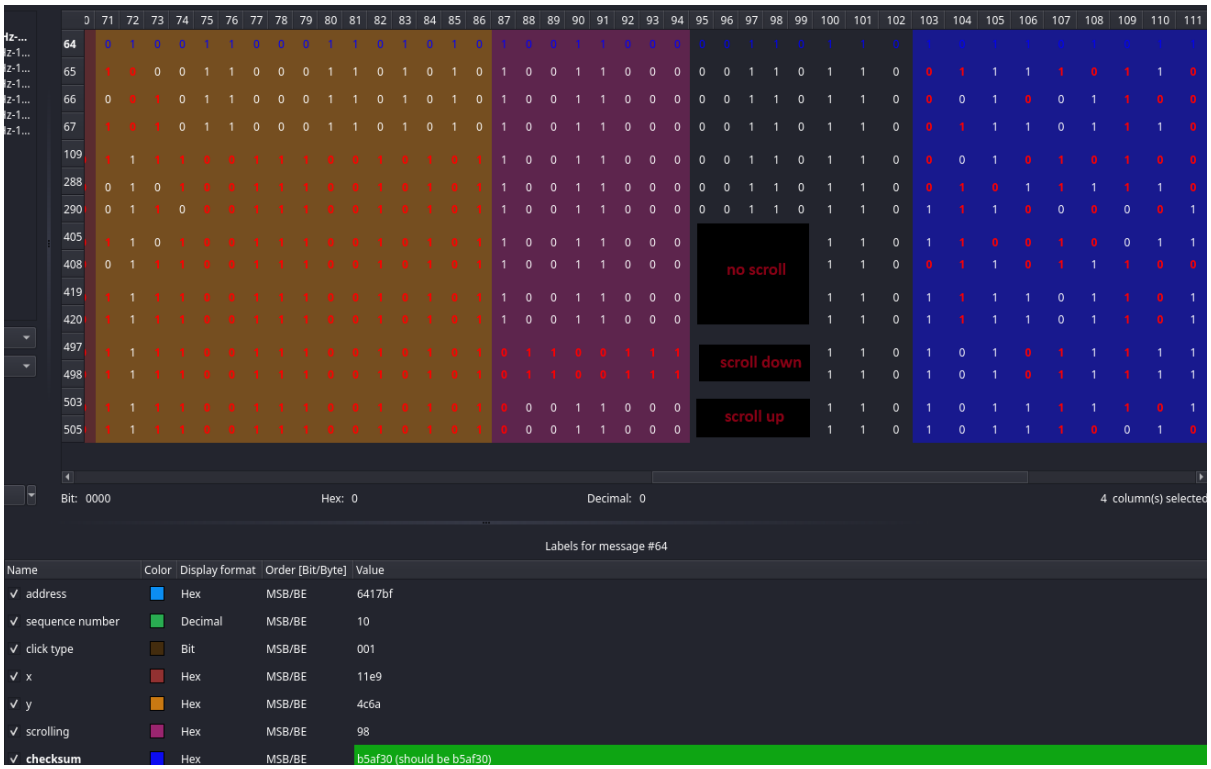


Figure 104. Visualization of the Hama's scrolling field in bits

The Hama mouse could not be exploited with the Crazyradio PA for the same reason as the keyboard: the packet length exceeds the Crazyradio PA's maximum supported size. However, simple replay attacks with the HackRF works.

Result

Neither the Hama mouse nor the keyboard could be exploited for the Crazyradio PA. However, the protocols of those devices had been fully reverse-engineered. The simple replay attack with the HackRF shows that the peripherals are vulnerable. Another piece of hardware must be used to sniff and spoof those devices. For example, the HackRF could be used with a Python script and pyhackrf [43] to craft and send packets. The HackRF can handle packets with significant sizes and different data rates. However, the attacks would not be very efficient with the HackRF as switching channels takes a long time. Another solution would be to program a Yichip yc1011-t RF chip, which is used in the Hama mouse and keyboard, to send crafted packets.

5.2.8. Omega

The Omega OM08WBL pictured in Figure 105 is a mouse bought in a secondhand shop. The results of the USB sniffing performed on the device's dongle can be found in Table 9.

Figure 105. Omega OM08WBL

USB sniffing

Table 9. Data found with USBPcap for the Omega dongle

Vendor ID	**0x3151** (Unknown)
Product ID (dongle)	**0x3020** (Unknown)
String descriptor	Wireless Device
Interfaces	Has Mouse and Keyboard interfaces
HID Data	5 buttons, 32 bits for X/Y, 8 bits wheel, 8 bits AC Pan

OSINT

Omega has a website, but it is no longer functional. No datasheet for this specific mouse was found, though some stores still sell it. The vendor ID corresponds to Yichip Microelectronics,

the same manufacturer of the RF chip in the Hama devices. After opening the mouse, Figure 106 confirms that it uses the same YC1011-T RF chip [42] as the Hama keyboard, suggesting that both devices likely follow the same protocol.

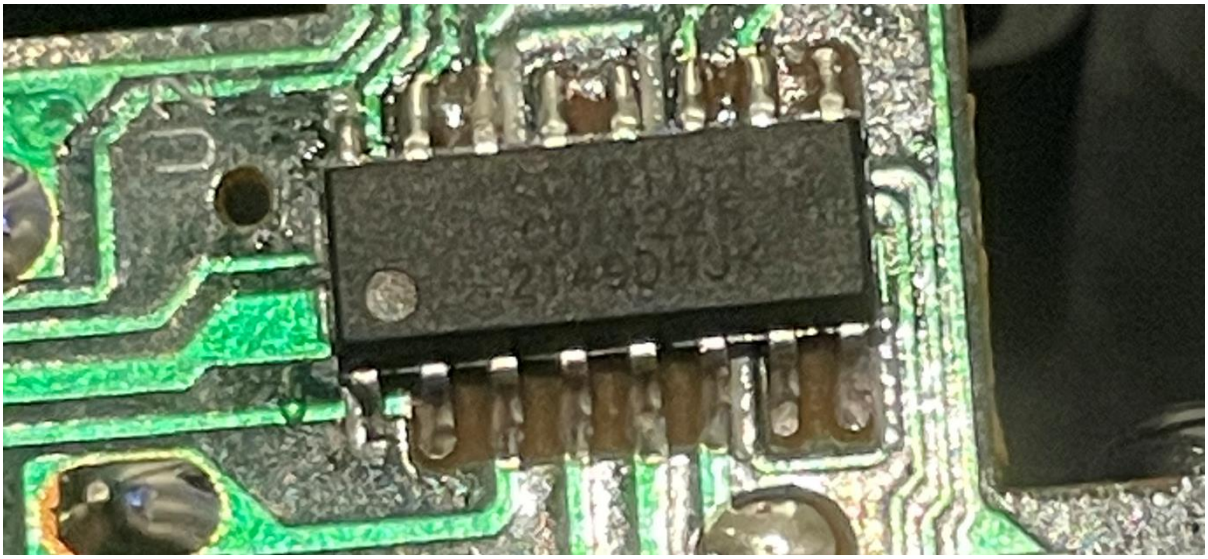


Figure 106. PCB of the Omega mouse

Reverse-engineer

Signals were recorded while simultaneously pressing the left click and moving the mouse. As observed with the Hama keyboard, the data uses 10 and 30 samples per symbol. Figure 107 presents a demodulated packet with 30 samples per symbol.

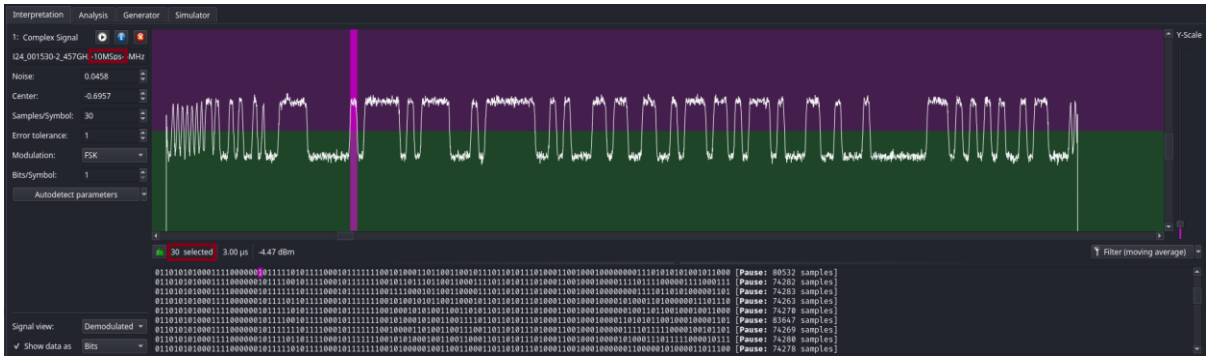


Figure 107. Demodulation of an Omega mouse packet with 30 samples per symbol

The CRC is 24 bits long. The packets were processed through RevEng, which identifies in Figure 108 the CRC parameters as poly = *0x00065B* and init = *0xC099F0*.

```
┌──(kali㉿kali)-[~/Downloads/reveng-3.0.6]
└─$ ./reveng -w24 -s 7817d78bf92933f4a2191022913c 7817978bf9613304a2191040bd9d 7817f78bf96d3344a2191067efd3 7817b7
8bf9353364a21910d3f7bd
width=24  poly=0×00065b  init=0×c099f0  refin=false  refout=false  xorout=0×000000  check=0×38260e  residue=0×0000
00  name=(none)
```

Figure 108. Identification of the CRC parameters of the Omega mouse using RevEng

To confirm the similarity of the protocols, different mouse actions were captured. Figure 109 presents the fully reverse-engineered protocol. While the mask is different, all fields remain identical. The Omega mouse has five buttons, validating the assumption made about the Hama mouse: the two bits following the identified click type correspond to buttons 4 and 5. Figure 110 illustrates the click type as a 5-bit value.

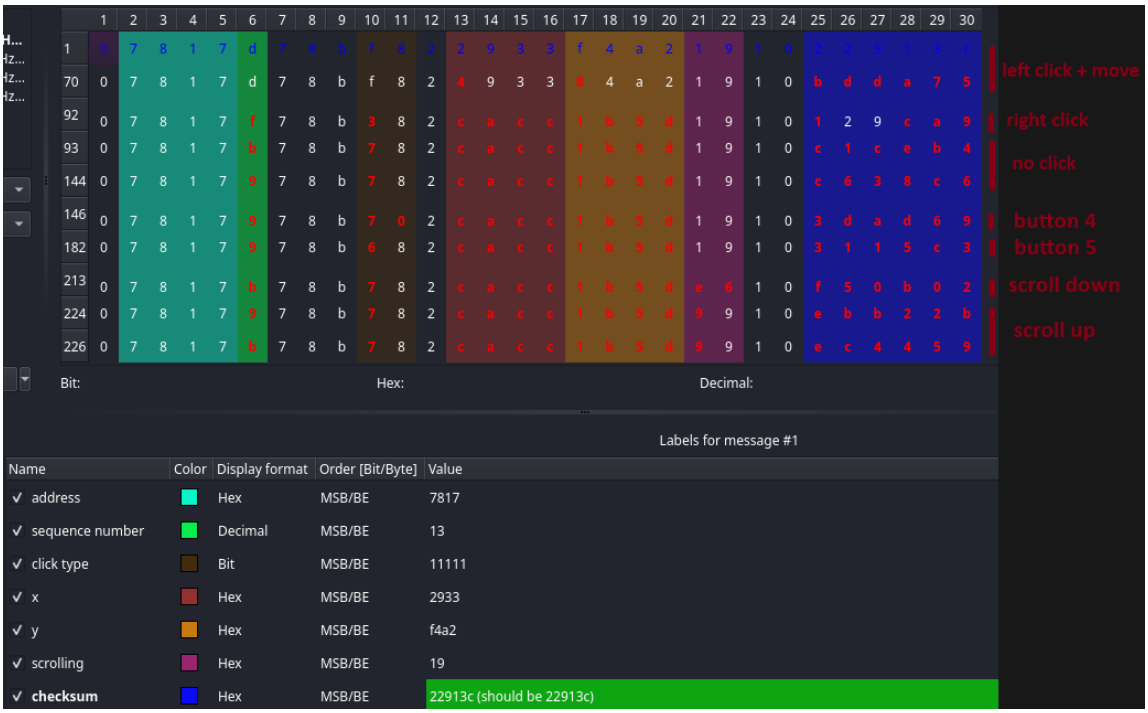


Figure 109. Reverse-engineered protocol of the Omega mouse

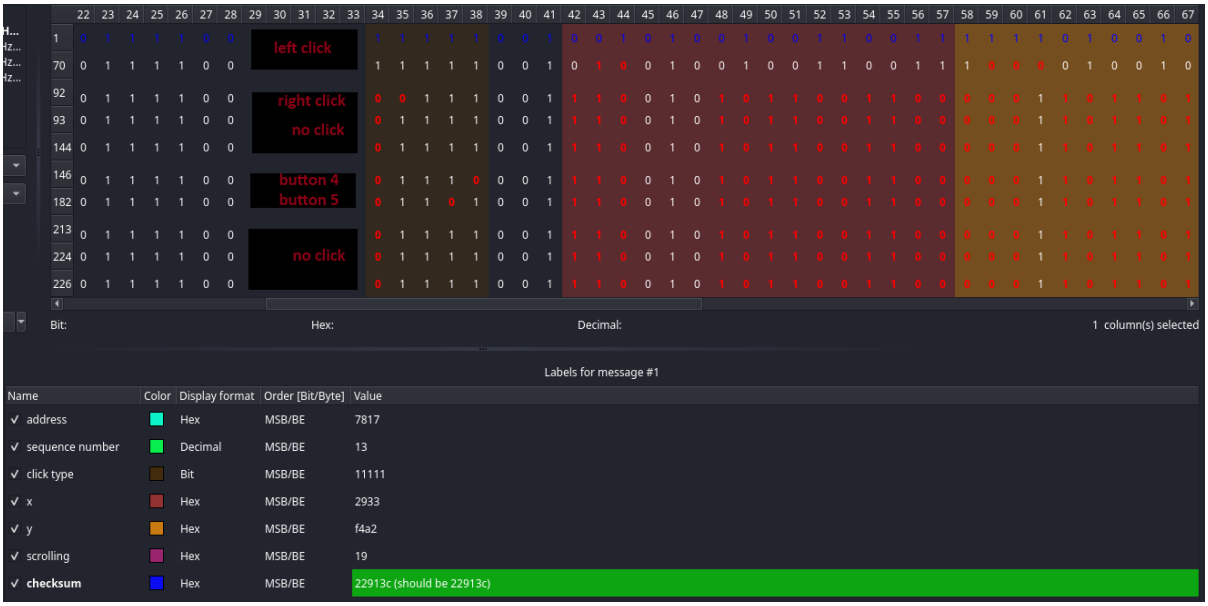


Figure 110. Visualization of the Omega's click type field in bits

Due to the same packet size limitations as the Hama mouse, exploiting the Omega mouse with a Crazyradio PA is not possible. However, a basic replay attack using the HackRF successfully demonstrates the vulnerability.

Result

Sniffing and spoofing cannot be implemented with the Crazyradio PA due to packet size limitations, but the protocol remains vulnerable. Since the Omega mouse uses the same protocol as the Hama devices, it should be possible to transmit keyboard packets using the protocol identified from the Hama keyboard.

5.2.9. HP

The HP 230 set pictured in Figure 111 consists of the HSA-A011M keyboard and the HSA-A005K mouse. The results of the USB sniffing performed on the devices' dongle can be found in Table 10.

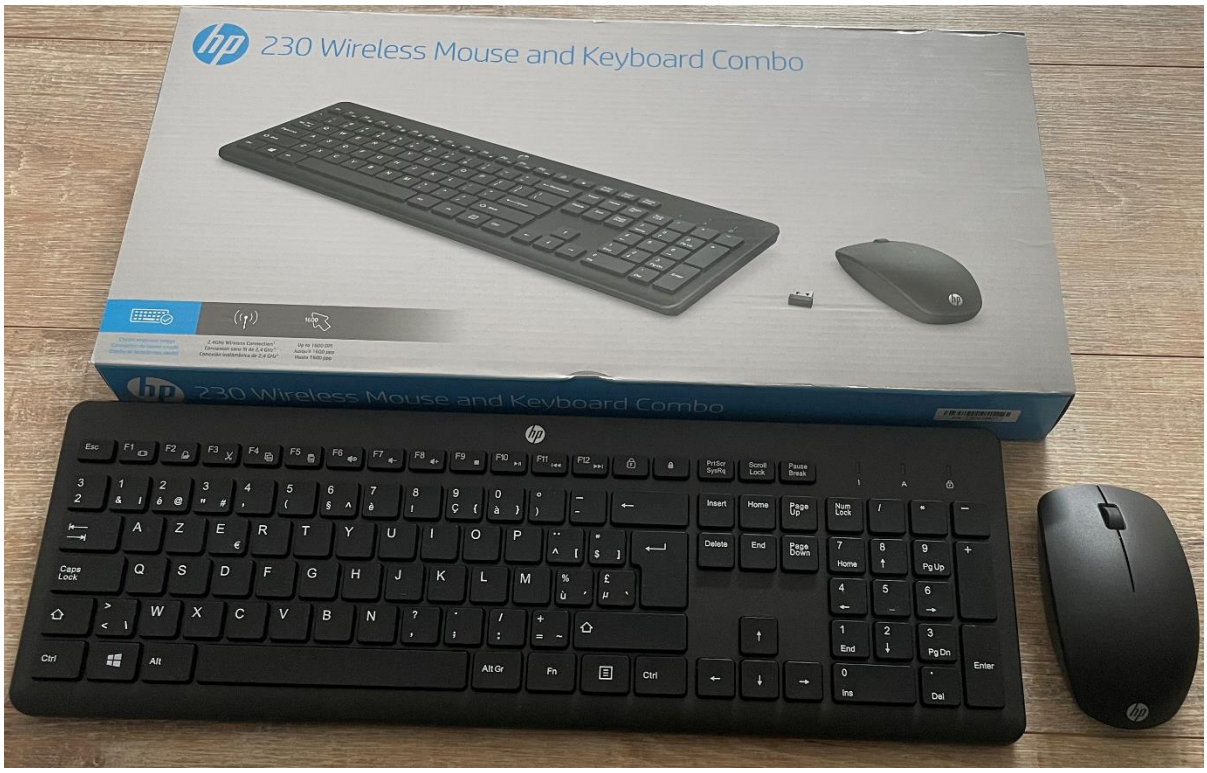


Figure 111. HP 230 set

USB sniffing

Table 10. Data found with USBPcap for the HP dongle

Vendor ID	**0x03F0** (HP, Inc)
Product ID (dongle)	**0x5341** (Unknown)
String descriptor	HP Wireless Keyboard and Mouse
Interfaces	Has Mouse and Keyboard interfaces
HID Data	For mouse: 8 buttons, 24 bits for X/Y, 8 bits wheel For keyboard: 8 bits modifiers, 6-elements array

OSINT

HP provides a product page for these devices, which are still available for purchase [44]. Each device has an FCC ID. The mouse, has the FCC ID PRDMU87, uses GFSK modulation across 12 channels, but the RF chip is concealed [45]. The keyboard, with FCC ID PRDKB42, also operates on 12 channels with GFSK modulation, and its RF chip is similarly obfuscated [46]. The dongle, identified as model HAS-A011D with FCC ID PRDRX0U, shares the exact channel count and modulation method, though its RF chip is also hidden [47].

Reverse-engineer

HP keyboard

Multiple "a" key presses were recorded. The signal was demodulated using FSK, revealing a symbol rate of 10 samples per symbol. Figure 112 highlights the preamble, characterized by a long stabilization phase followed by eight troughs.

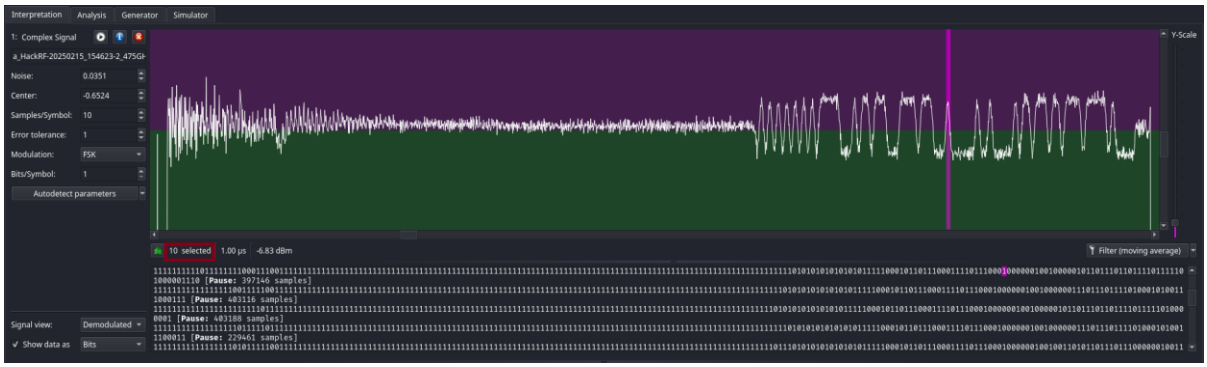


Figure 112. Demodulation of an HP keyboard packet with 10 samples per symbol

Figure 113 reveals three distinct packet types based on their length.

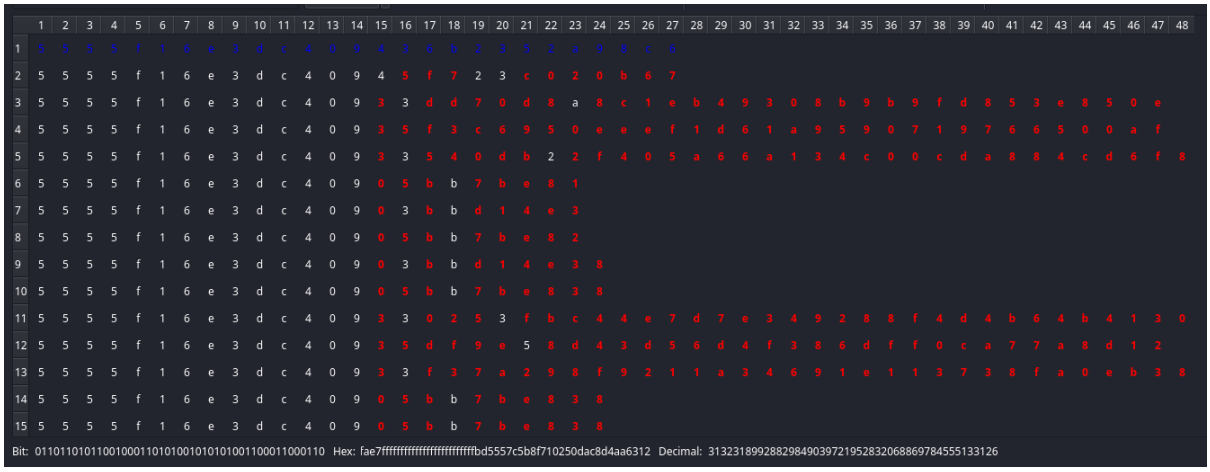


Figure 113. Visualization of three types of HP keyboard packets based on their length

Identifying the CRC was difficult for the short packets because only two unique packets were repeated multiple times, providing insufficient data for RevEng. In Figure 114, two potential CRC parameters were found, with poly = 0x1021 assumed to be the correct one.

```
┌──(kali㉿kali)-[~/Downloads/reveng-3.0.6]
└─$ ./reveng -w16 -s f16e3dc40903bbd14e f16e3dc40905bb7be8
./reveng: warning: you have only given 2 samples
./reveng: warning: to reduce false positives, give 4 or more samples
width=16  poly=0x1021  init=0x93ff  refin=false  refout=false  xorout=0x0000  check=0x6484  residue=0x0000  name=(
none)
width=16  poly=0xa031  init=0xbd60  refin=true  refout=true  xorout=0x0000  check=0x6833  residue=0x0000  name=(no
ne)
```

Figure 114. Identification of the CRC parameters of the short packets of the HP mouse using RevEng

Labels were added in Figure 115, highlighting a field that alternates between 01 and 10. The short packets appear to function as acknowledgments, containing no identifiable data.

Figure 115. Visualization of the HP mouse's short packets in hexadecimal

The longer packets in Figure 113 seem to carry actual data. CRC checksums were identified using RevEng, as shown in Figure 116, and labels were assigned in Figure 117. One field is suspected to indicate packet length, but no further conclusions could be drawn.

```
┌──(kali㉿kali)-[~/Downloads/reveng-3.0.6]
└─$ ./reveng -w16 -s f16e3dc40933dd70d8a8c1eb49308b9b9fd853e850 f16e3dc40935f3c6950eeef1d61a9590719766500a f16e3dc
40933540db22f405a66a134c00cda884cd6 f16e3dc409330253fbc44e7d7e349288f4d4b64b41
width=16  poly=0×1021  init=0×63da  refin=false  refout=false  xorout=0×0000  check=0×433b  residue=0×0000  name=(
none)
```

Figure 116. Identification of the CRC parameters of the long packets of the HP mouse using RevEng

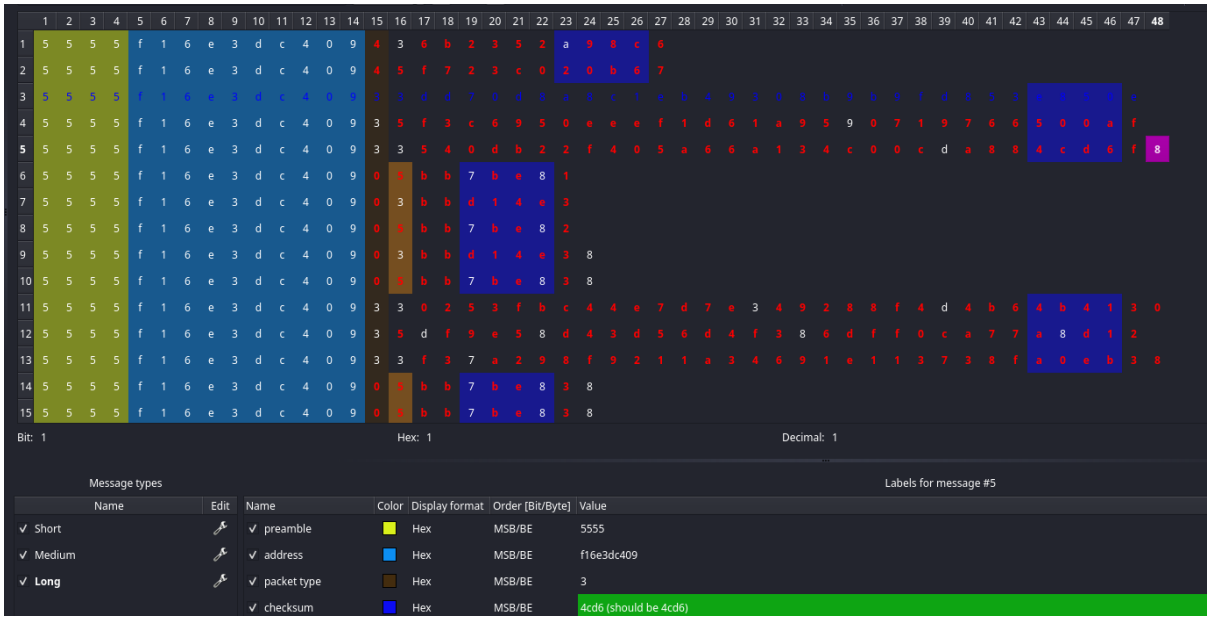


Figure 117. Visualization of the three types of packets sent by the HP keyboard

The data parts of the long packets are all different. This strongly suggests that encryption is used to protect the packets. Attempts to execute replay attacks with the HackRF were made, but they were unsuccessful.

HP mouse

Since the keyboard employs encryption, likely, the mouse does as well. Signals from left-click actions were captured and demodulated, as shown in Figure 118. Packet lengths vary, and all packets are unique, confirming the presence of encryption. Replay attacks with the HackRF were unsuccessful.

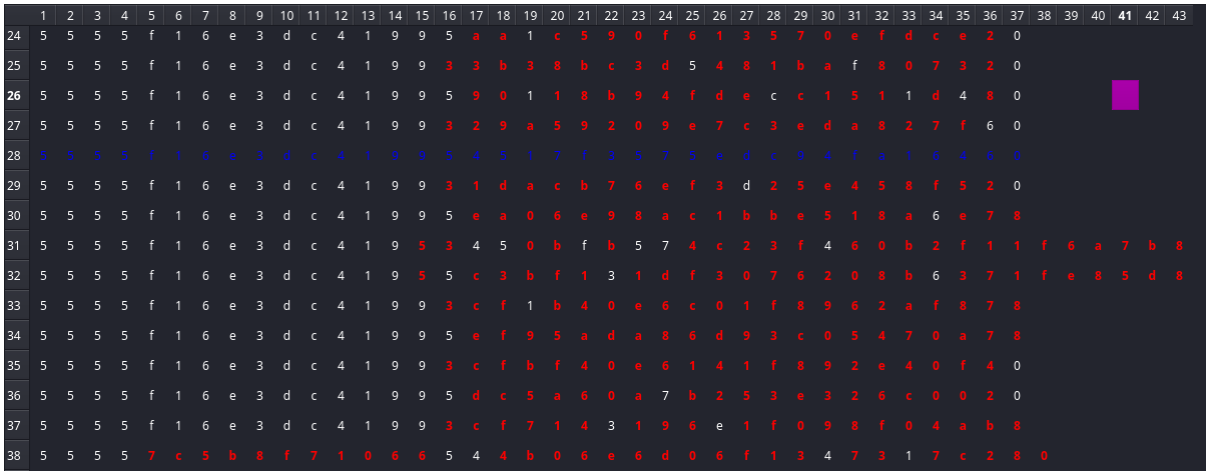


Figure 118. Visualization of mouse packets from the HP mouse

Result

The HP devices use encryption, which falls outside the scope of this thesis. Future research could focus on extracting encryption keys using techniques such as firmware extraction. Additionally, the HackRF replay attacks' failure suggests that the HP devices likely implemented an unknown authentication mechanism beyond encryption.

5.2.10. Cherry

The Cherry DW5100 set pictured in Figure 119 consists of the DW5100 keyboard and the MW 3000 mouse. The results of the USB sniffing performed on the devices' dongle can be found in Table 11.

Figure 119. Cherry DW5100 set

USB sniffing

Table 11. Data found with USBPcap for the Cherry dongle

Vendor ID	**0x046A** (CHERRY)
Product ID (dongle)	**0xC092** (Unknown)
String descriptor	CHERRY Wireless Device
Interfaces	Has Mouse and Keyboard interfaces
HID Data	For mouse: 5 buttons, 24 bits for X/Y, 8 bits wheel, 8 bits AC Pan For keyboard: 8 bits modifiers, 6-elements array

OSINT

Cherry's website lists these devices as still available for purchase [48]. Datasheets confirm that neither the mouse nor the keyboard uses key encryption, but no additional technical details were found.

Both devices have FCC IDs. The mouse, with FCC ID GDDJF-T01REV, has an obfuscated RF chip, operates on 34 channels, and employs GFSK modulation. The keyboard, with FCC ID GDDJD-05REV, has an obfuscated RF chip, operates on 34 channels, and uses GFSK modulation.

Reverse-engineer

Cherry keyboard

Multiple “7” keypresses are captured and analyzed. Figure 120 shows that the first signal, when pressed, has a bigger amplitude; the following packets are suspected to be dongle packets. FSK can be used to demodulate the signal; there are 10 samples per symbol, as shown in Figure 121, and the data rate is 1Mbps.

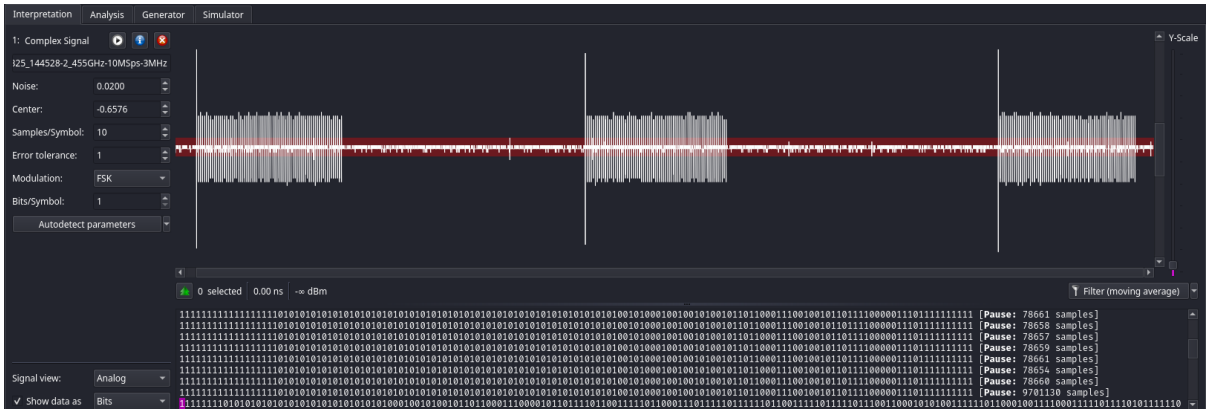


Figure 120. Visualization of signals captured after a “7” Cherry keyboard press

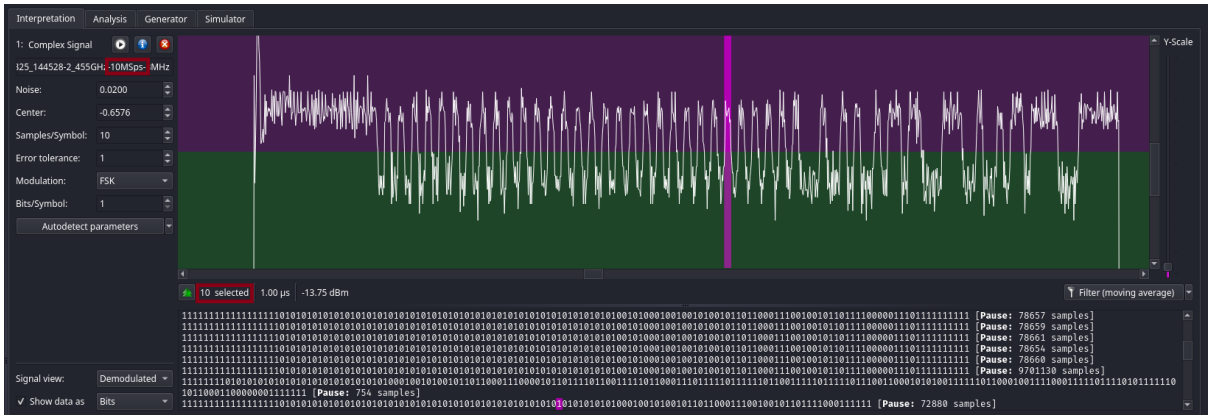
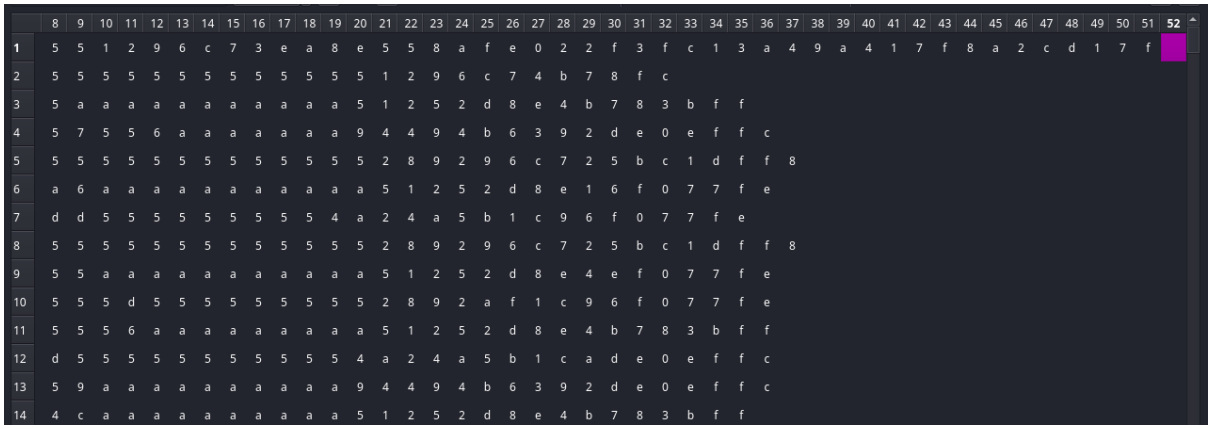


Figure 121. Demodulation of a Cherry keyboard signal with a data rate of 1Mbps

Figure 122 shows three types of packets based on their lengths. When pressing a key, the first signal is longer, the second is smaller, and the remaining is longer than the second. The second and third are identical, indicating no data inside them. Those signals were muted to work only with the long signals.



	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
1	5	5	1	2	9	6	c	7	3	e	a	8	e	5	5	8	a	f	e	0	2	2	f	3	f	c	1	3	a	4	9	a	4	1	7	f	8	a	2	c	d	1	7	f	
2	5	5	5	5	5	5	5	5	5	5	5	5	5	1	2	9	6	c	7	4	b	7	8	f	c																					
3	5	a	a	a	a	a	a	a	a	a	a	5	1	2	5	2	d	8	e	4	b	7	8	3	b	f	f																			
4	5	7	5	5	6	a	a	a	a	a	a	9	4	4	9	4	b	6	3	9	2	d	e	0	e	f	f	c																	
5	5	5	5	5	5	5	5	5	5	5	5	5	5	2	8	9	2	9	6	c	7	2	5	b	c	1	d	f	f	8														
6	a	6	a	a	a	a	a	a	a	a	a	a	a	5	1	2	5	2	d	8	e	1	6	f	0	7	7	f	e																
7	d	d	5	5	5	5	5	5	5	5	4	a	2	4	a	5	b	1	c	9	6	f	0	7	7	f	e																	
8	5	5	5	5	5	5	5	5	5	5	5	5	5	2	8	9	2	9	6	c	7	2	5	b	c	1	d	f	f	8														
9	5	5	a	a	a	a	a	a	a	a	a	a	a	5	1	2	5	2	d	8	e	4	e	f	0	7	7	f	e																
10	5	5	5	d	5	5	5	5	5	5	5	5	5	2	8	9	2	a	f	1	c	9	6	f	0	7	7	f	e																
11	5	5	5	6	a	a	a	a	a	a	a	a	a	5	1	2	5	2	d	8	e	4	b	7	8	3	b	f	f																
12	d	5	5	5	5	5	5	5	5	5	5	5	4	a	2	4	a	5	b	1	c	a	d	e	0	e	f	f	c																
13	5	9	a	a	a	a	a	a	a	a	a	a	9	4	4	9	4	b	6	3	9	2	d	e	0	e	f	f	c																
14	4	c	a	a	a	a	a	a	a	a	a	5	1	2	5	2	d	8	e	4	b	7	8	3	b	f	f																		

Figure 122. Visualization of three types of Cherry keyboard packets based on their length

Figure 123 shows a preamble (0xAAA) and an address (0x252D8E); the data parts are all different, indicating encryption.

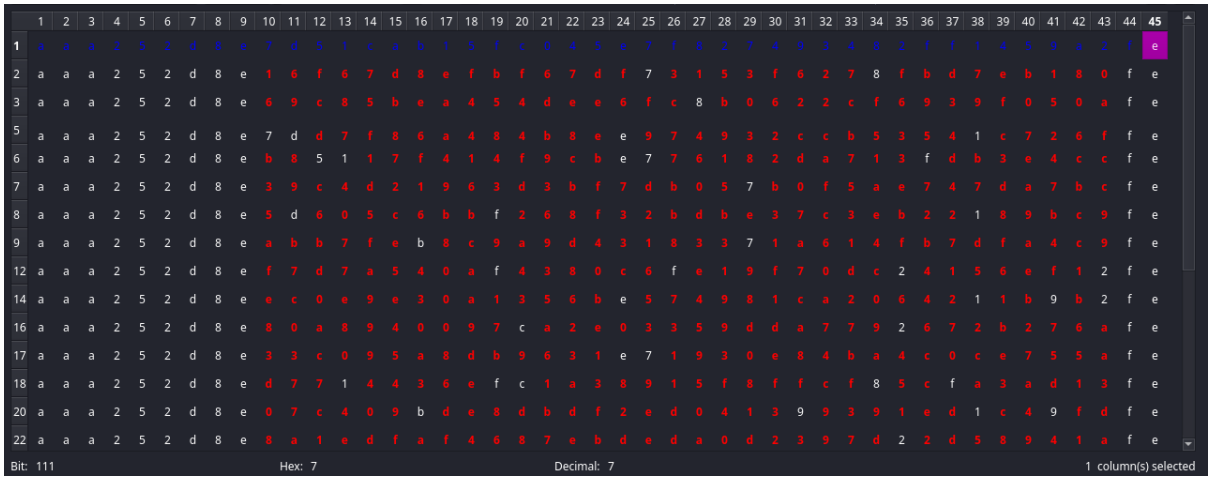


Figure 123. Visualization of Cherry keyboard packets in hexadecimal

Multiple replay attacks with the HackRF were attempted and were successful. Repeating the same signals 10 times can make the dongle accept the signal once or twice. There is probably a form of sequence number. The Crazyradio PA was used to attempt to weaponize the replay attack. First, the channels were found using the *fuzz_channels* method. Figure 124 shows that a lot of channels were found.

```
56  Device.fuzz_channels("25:2d:8e", common.RF_RATE_1M)
57

PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

└─$ ./main.py
[22]
[22, 28]
[22, 28, 44]
[22, 28, 44, 54]
[22, 28, 44, 54, 60]
[22, 28, 44, 54, 60, 93]
[22, 28, 44, 54, 60, 93, 7]
[22, 28, 44, 54, 60, 93, 7, 67]
[22, 28, 44, 54, 60, 93, 7, 67, 76]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38, 0]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38, 0, 29]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38, 0, 29, 33]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38, 0, 29, 33, 2]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38, 0, 29, 33, 2, 45]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38, 0, 29, 33, 2, 45, 94]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38, 0, 29, 33, 2, 45, 94, 95]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38, 0, 29, 33, 2, 45, 94, 95, 8]
[22, 28, 44, 54, 60, 93, 7, 67, 76, 68, 78, 13, 61, 73, 69, 38, 0, 29, 33, 2, 45, 94, 95, 8, 20]
^C
```

Figure 124. Identification of the Cherry keyboard channels using the *fuzz_channels* method

The *quick_sniff* method was then used to see the different data captured. As shown in Figure 125, many dongle packets are captured, which is not interesting. Those dongle packets were

identified as having the 0x783bff sequence. A new class was created in the *devices/cherry* folder to handle the sniffing and spoofing of Cherry keyboard packets.

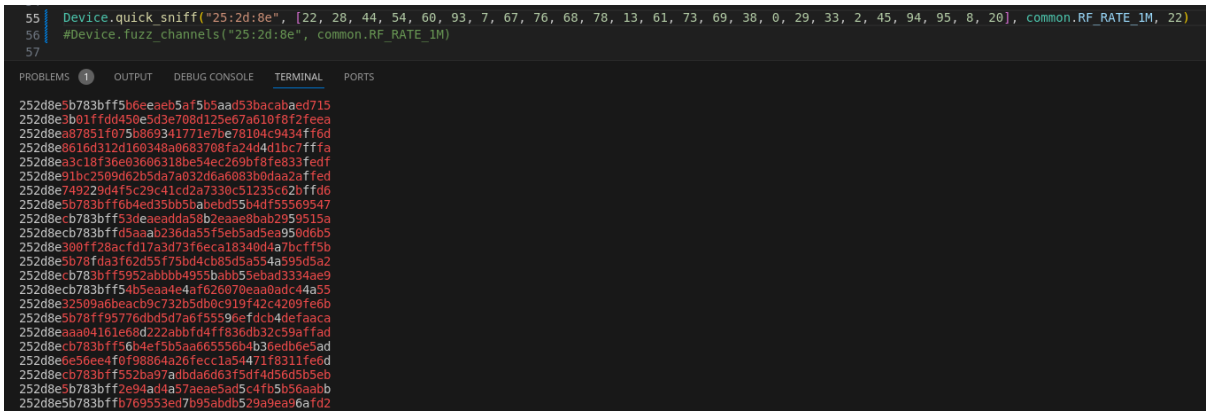


Figure 125. Sniffing of the Cherry keyboard using the quick_sniff method

The packet sent by the keyboard for a specific action can be sniffed. Figure 126 shows basic sniffing for an “A” packet. The entire packet can be copied and used with the *spoof* method. The goal is to record what the Cherry dongle expects for a particular keypress, hardcode the packet, and send it back. This attack is technically successful. However, only a few packets are accepted by the dongle. Sending three keypresses results in the dongle accepting only two of them.

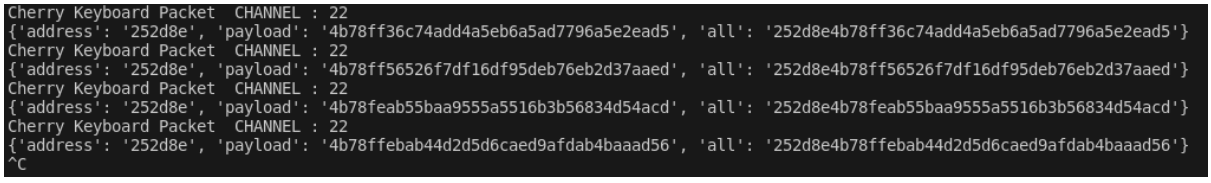


Figure 126. Sniffing of Cherry keyboard packets

Cherry mouse

Multiple right clicks were captured. Figure 127 shows that the signals resemble those of the Cherry keyboard, with a high-amplitude signal followed by what appears to be multiple dongle packets. The demodulation parameters are the same as those used for the Cherry keyboard.

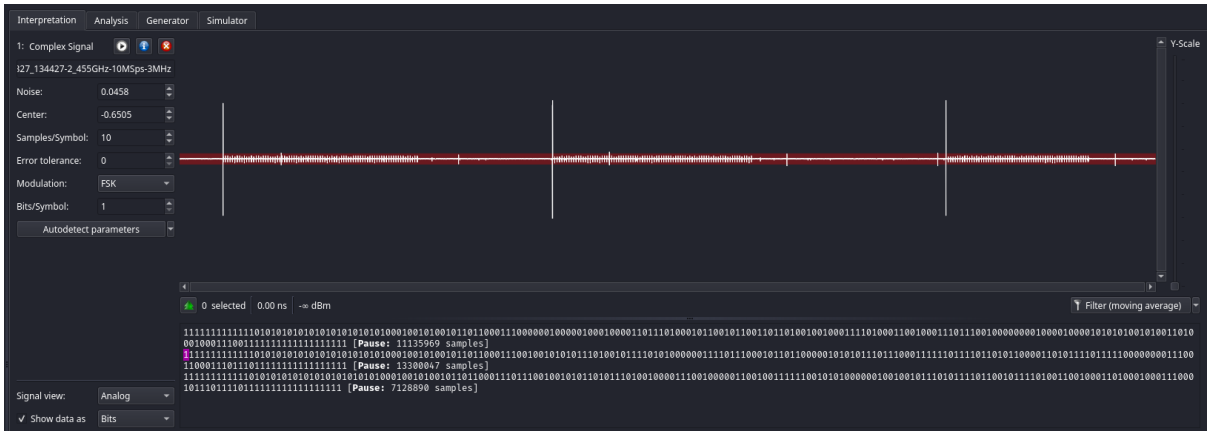


Figure 127. Visualization of signals captured after a “Right click” Cherry keyboard press

In Figure 128, the packet format appears identical, featuring the same preamble and address. The Crazyradio PA was used to sniff and spoof the mouse. Figure 129 shows the packet sent to the *spoof* method, successfully executing a right-click.

Figure 128. Visualization of Cherry mouse packets in hexadecimal

```
cherry = Cherry("25:2d:8e")
#cherry.sniff()

#cherry.spoof([b"\x25\x2d\x8e\xfe\xe0\x39\x8f\xad\x85\xac\xf0\xf7\x23\xb0\xe8\x90\xbc\x6c\xc8\x82\xfe"]) # L GUI
#cherry.spoof([b"\x25\x2d\x8e\xea\xd4\x0f\xe4\xa4\x86\x2f\x10\x5a\x23\xda\x81\x16\xd4\x32\x49\x09\xfe"]) # L
#cherry.spoof([b"\x25\x2d\x8e\xab\x72\x8a\x15\x3c\x7e\x8b\xde\xba\xd6\x8c\x42\x4f\xe1\xb9\xee\x62\xfe"]) # S
#cherry.spoof([b"\x25\x2d\x8e\x97\xe4\x83\xf2\xa7\x0f\x62\xfc\xa7\xc4\x46\x0e\x0a\xa5\xd6\x50\x6c\xfe"]) # A
#cherry.spoof([b"\x25\x2d\x8e\x14\x84\xa1\x0c\x4f\x6e\x49\xcd\xcc\x9c\xb4\x00\x74\xfb\x79\x6a\xba\xfe"]) # Z
cherry.spoof([b"\x25\x2d\x8e\x4a\xba\x5e\xa0\x7b\x8b\x60\xab\xb8\xfd\xed\x61\xaf\x7c\x03\x98\xee"]) # Right click
```

Figure 129. Usage of the *Cherry* class to send a right click to the Cherry dongle

Result

The Cherry mouse and keyboard implement encryption to prevent eavesdropping but remain vulnerable to basic packet injection through replay attacks. Further research could focus on extracting the encryption key through firmware analysis or improving the Python script to increase the number of packets the Cherry dongle accepts.

5.3. Summary of testing

Out of the ten different brands of devices tested, only one, HP, was not successfully attacked. Furthermore, encryption was only implemented on the HP and Cherry devices, making them harder to reverse-engineer. Despite this, the Cherry devices were still vulnerable to spoofing attacks.

Also, the TX mouse shows that capturing USB communication can help determine if the USB dongle can support other types of packets. Although not provided with a keyboard, this mouse was found to be vulnerable to keyboard spoofing.

The vulnerabilities found for each peripheral are summarized in Table 12.

Table 12. Summary of finding

Brand	Device	Type	Sniffing	Spoofing
Trust	ODY-II	Keyboard	Yes	Yes
	Yvi+	Mouse	Yes	No
Poss	PSKEY530BK	Keyboard	Yes	Yes
Rapoo	E1050	Keyboard	Yes	Yes
	M10	Mouse	Yes	Yes
Edenwood	963716 CWL01 keyboard	Keyboard	Yes	Yes
	963716 CWL01 mouse	Mouse	Yes	Yes
Qware	QW PCB-238BL keyboard	Keyboard	Yes	Yes
	QW PCB-238BL mouse	Mouse	Yes	Yes
Think Xtra	Ms6-TXn-wh	Mouse	Yes	Yes
Hama	AKMW-100 keyboard	Keyboard	Yes	Yes
	AKMW-100 mouse	Mouse	Yes	Yes
Omega	OM08WBL	Mouse	Yes	Yes
HP	HSA-A011M	Keyboard	No	No
	HAS-A005K	Mouse	No	No
Cherry	DW5100	Keyboard	No	Yes
	MW3000	Mouse	No	Yes

6. Conclusion

This chapter summarizes the findings of this thesis. Chapter 6.1 discusses the security risks identified in wireless peripherals and their implications. Chapter 6.2 answers the research questions based on the results obtained. Chapter 6.3 reviews the methodology used for reverse-engineering. Finally, Chapter 6.4 explores potential directions for future research.

6.1. Discussion

This thesis aimed to reverse-engineer previously untested wireless devices utilizing proprietary protocols and assess their potential security vulnerabilities. To achieve this, a dedicated methodology was developed in Chapter 4. Seventeen devices from ten different brands were analyzed, all purchased in Belgium between 2024 and 2025 without targeting any specific manufacturer or model. The objective was to determine whether commercially available wireless devices remain vulnerable to attacks.

In 2016, Bastille's MouseJack set of vulnerabilities highlighted the security risks associated with wireless peripherals [12]. However, awareness of these risks remains limited, and many users continue to rely on proprietary protocol wireless devices without considering potential security implications. This research investigated both sniffing and spoofing attacks on various devices. Sniffing enables an attacker to intercept wireless signals, potentially capturing sensitive information such as keystrokes, which could include passwords. Spoofing involves crafting and transmitting signals that deceive the USB dongle into accepting them as legitimate, potentially allowing an attacker to execute commands on the victim's computer, such as opening a terminal, downloading malware, and executing malicious scripts.

The findings demonstrate that widely available wireless mice and keyboards pose a security risk. Most tested devices were relatively easy to reverse-engineer using the methodology created in Chapter 4. Only two of the ten brands analyzed implemented encryption to prevent sniffing. However, encryption alone was insufficient to guarantee security, as demonstrated by the Cherry devices, which remained vulnerable to replay attacks due to a lack of authentication mechanism.

To attack a host device connected to a wireless peripheral, the communication protocol must first be understood. This can be done by capturing and analyzing signals from the peripheral, or by purchasing the same model and analyzing it directly. Once the protocol is understood, an

attacker can interpret the victim's actions and craft packets to trigger specific behaviors on the victim's computer.

6.2. Answering the research questions

The research questions investigated in this thesis were:

- [RQ1] Are wireless devices using proprietary protocols, still available on the market, vulnerable to security attacks?
    - [RQ1.1] Sniffing: Can the signals transmitted by these devices be intercepted and analyzed to reveal sensitive information, such as keystrokes?
    - [RQ1.2] Spoofing: Is injecting malicious keystrokes into these devices possible?
- [RQ2]: Can a methodology be defined to aid in the process of testing the security of wireless mice and keyboards?

Regarding **[RQ1]**, the results indicate that many wireless devices currently on the market remain vulnerable to sniffing ([RQ1.1]) and spoofing ([RQ1.2]) attacks. The findings align with those of MouseJack in 2016, demonstrating that security has not significantly improved. While certain manufacturers, such as HP, have implemented protective measures, this does not necessarily imply that their devices are entirely secure, only that no vulnerabilities were identified within the scope of this thesis.

For **[RQ2]**, a methodology was created in Chapter 4. The approach was designed to be accessible even to individuals with limited experience. Chapter 5 applied this methodology to real case scenarios, demonstrating its effectiveness; almost all the tested devices had been shown to be vulnerable. While some situations required extra steps, the methodology still provided a solid foundation for testing the security of wireless mice and keyboards. Additional critical thoughts about the methodology are discussed in Chapter 6.3.

6.3. Methodology review

The methodology developed in Chapter 4 proved effective in testing the security of wireless devices. Open-source intelligence (OSINT) was initially employed to identify similarities between devices, such as shared RF chips, which could indicate the usage of similar protocols. However, this process was time-consuming, and by the conclusion of the research, it was determined that OSINT provided limited practical benefits. The primary goal of OSINT was

to find the RF chip specifications and potential documentation on proprietary protocols. However, none of the seventeen tested devices had publicly available documentation regarding their communication protocols.

Other parameters, such as channel frequencies, modulation schemes, and data rates, were occasionally accessible via FCC ID records. Nevertheless, these details were also easily obtainable through software-defined radio analysis using Universal Radio Hacker (URH). Additionally, FCC-reported channel frequencies were often inaccurate.

USB sniffing was, however, useful. Analyzing USB HID data allowed for a faster understanding of packet formats, as the bit-length of HID reports often correlated with packet structures. A particularly significant finding involved the TX mouse dongle, which was found to possess keyboard interfaces despite being designed for mouse input. This discovery enabled the injection of keystroke packets, demonstrating that even wireless mice can introduce severe security risks.

6.4. Future research

This research primarily focused on radio signal analysis, involving the interception and reverse-engineering of wireless transmissions to identify vulnerabilities and develop corresponding exploits. Future research could explore additional techniques, such as hardware analysis, to extract firmware and encryption keys from devices. This approach has been successfully employed in previous work, such as that of Matthias Deeg and Gerhard Klostermeier on wireless presenters [18]. Also, rather than using a generic transceiver like the Crazyradio PA with its nRF24, using the device's RF chip could potentially bypass obfuscation mechanisms.

This study examined seventeen devices, fifteen of which contained security vulnerabilities. While this indicates that such risks are common, the sample size is relatively small. Testing a larger number of devices would provide a clearer picture of security trends across different manufacturers.

The Python script developed for this research currently uses a basic approach to handle the frequency hopping mechanism of devices, which limits the accuracy of sniffing. It also only supports the Crazyradio PA dongle, which is no longer in production. Adapting the code to support alternative dongles would increase its usability and relevance for future research. Additionally, implementing live sniffing capabilities similar to those in JackIT—where all

channels are scanned continuously to detect vulnerable devices—would be a valuable extension to attack devices in real-world scenarios.

References

[1] R. Metz. How hackable is your wireless keyboard and mouse?. 2016. https://www.technologyreview.com/2016/02/23/161905/how-hackable-is-your-wireless-keyboard-and-mouse/ (18.04.2025)

[2] L. Agro. RFForge. 2025. https://github.com/AgroLucas/RFForge (10.05.2025)

[3] K. Vo, M. Hernandez, and N. Patel. Electromagnetic radiation. *Introduction to Organic Spectroscopy*, 2020. https://chem.libretexts.org/Courses/University_of_Illinois_Springfield/Introduction_to_Organic_Spectroscopy/1%3A_Introduction_to_Organic_Spectroscopy/1.3%3A_Electromagnetic_Radiation (18.04.2025)

[4] Popular Electronics. Radio frequency Modulation Made Easy. 2018. https://popularelectronics.technicacuriosa.com/2017/03/08/radio-frequency-modulation-made-easy/ (18.04.2025)

[5] EverythingRF. What is GFSK Modulation?. 2022. https://www.everythingrf.com/community/what-is-gfsk-modulation (18.04.2025)

[6] Integrated Publishing. Digital transmission. s.a. https://www.tpub.com/neets/tm/112-2.htm (18.04.2025)

[7] Great Scott Gadgets. HackRF One. s.a. https://greatscottgadgets.com/hackrf/one/ (18.04.2025)

[8] Great Scott Gadgets. ANT700. s.a. https://greatscottgadgets.com/ant700/ (18.04.2025)

[9] Bitcraze. CrazyRadio PA. s.a. https://www.bitcraze.io/products/crazyradio-pa/ (18.04.2025)

[10] Jopohl. Universal Radio Hacker: Investigate Wireless Protocols Like A Boss. 2024. https://github.com/jopohl/urh (18.04.2025)

[11] Bastille. Defending your wireless airspace, beyond the network perimeter. 2025. https://bastille.net/about/ (18.04.2025)

[12] Bastille. MouseJack Technical Details. 2024. https://bastille.net/research/vulnerabilities-mousejack-technical-details/ (13.03.2025)

[13] M. Newlin. MouseJack, KeySniffer and Beyond: Keystroke sniffing and injection vulnerabilities in 2.4GHz wireless mice and keyboards. 2016. https://github.com/BastilleResearch/mousejack/blob/master/doc/pdf/DEFCON-24-Marc-Newlin-MouseJack-Injecting-Keystrokes-Into-Wireless-Mice.whitepaper.pdf (18.04.2025)

[14] Bastille. KeySniffer. 2024. https://bastille.net/research/vulnerabilities-keysniffer-technical-details/ (13.03.2025)

[15] Bastille. KeyJack. 2024. https://bastille.net/research/vulnerabilities-keyjack-keyjack-intro/ (13.03.2025)

[16] SySS. SySS: Your contact in all matters of IT security. 2025. https://www.syss.de/en/about-us/syss-gmbh (18.04.2025)

[17] M. Deeg and G. Klostermeier. New tales of wireless input devices. 2019. https://www.syss.de/fileadmin/user_upload/2019_06_04_New_Tales_of_Wireless_Input_Devices_-_CONFidence_2019.pdf (18.04.2025)

[18] M. Deeg and G. Klostermeier. Keystroke injection tool collection for 2.4 GHz wireless input devices. 2023. https://github.com/SySS-Research/keyjector (18.04.2025)

[19] N. Tomsic. Penetration testing wireless keyboards : Are your devices vulnerable?. *DIVA*, 2022. https://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1701492&dswid=-8832 (18.04.2025)

[20] T. Schroeder and M. Moser. Practical exploitation of modern wireless devices. http://www.remote-exploit.org/content/keykeriki_v2_cansec_v1.1.pdf (18.04.2025)

[21] T. Goodspeed. Promiscuity is the nRF24L01+'s Duty. 2011. https://travisgoodspeed.blogspot.com/2011/02/promiscuity-is-nrf24l01s-duty.html (18.04.2025)

[22] S. Kamkar. KeySweeper. s.a. https://samy.pl/keysweeper/ (18.04.2025)

[23] insecurityofthings. JackIt - Exploit Code for Mousejack. 2020. https://github.com/insecurityofthings/jackit (18.04.2025)

[24] USBPCap. s.a. https://desowin.org/usbpcap/index.html (18.04.2025)

[25] Wireshark. USB Capture setup. 2020. https://wiki.wireshark.org/CaptureSetup/USB (18.04.2025)

[26] USB ID Database. s.a. https://the-sz.com/products/usbid (18.04.2025)

[27] FCC Federal Communications Commission. Equipment Authorization – Importation. s.a. https://www.fcc.gov/oet/ea/importation (15.05.2025)

[28] FCC Federal Communications Commission. FCC ID Search. s.a. https://www.fcc.gov/oet/ea/fccid (18.04.2025)

[29] Searchable FCC ID Database. s.a. https://fccid.io/ (18.04.2025)

[30] E. Por, M. Van Kooten, and V. Sarkovic. Nyquist–Shannon sampling theorem. 2019. https://home.strw.leidenuniv.nl/~por/AOT2019/docs/AOT_2019_Ex13_NyquistTheorem.pdf (18.04.2025)

[31] G. Cook. CRC RevEng: arbitrary-precision CRC calculator and algorithm finder. 2024. https://reveng.sourceforge.io/ (18.04.2025)

[32] B. Ben Atar. Bad USB Case Study. 2020. https://sepiocyber.com/resources/case-studies/badusb-attack/ (15.05.2025)

[33] Trust. Ody II Silent Wireless Keyboard & Mouse set. s.a. https://www.trust.com/en/product/25018-ody-ii-silent-wireless-keyboard-mouse-set-black-us (18.04.2025)

[34] Telink Semiconductor. TLSR8516. 2017. https://wiki.telink-semi.cn/doc/ds/DS_TLSR8516-C_Datasheet%20for%20Telink%20TLSR8516.pdf (18.04.2025)

[35] Telink Semiconductor. Chips for a Smarter IoT. 2025. https://www.telink-semi.com/ (18.04.2025)

[36] FCC ID PP2M10. 2013. https://fccid.io/PP2M10 (18.04.2025)

[37] FCC ID PP2E1050. 2012. https://fccid.io/PP2E1050 (18.04.2025)

[38] Rapoo. X1800S. s.a. https://www.rapoo-eu.com/product/x1800s/ (18.04.2025)

[39] Clavier + Souris EDENWOOD CWL01 BE. s.a. https://www.electrodepot.be/fr/clavier-souris-ss-fil-edenwood-cwl01-be.html (18.04.2025)

[40] Qware. Nottingham Draadloze Bundel - Zwart. s.a. https://qware.nl/products/product_24 (18.04.2025)

[41] TX MS6TXN-BK Souris optique sans fil 1000 dpi Noir. s.a. https://www.amazon.fr/MS6TXN-BK-Souris-optique-sans-1000/dp/B00QGDPJKQ (18.04.2025)

[42] YiChip. YC10XX series (YC1021D/S/...). s.a. http://www.yichip.com/yc10XX (18.04.2025)

[43] pyhackrf. s.a. https://pypi.org/project/pyhackrf/ (15.05.2025)

[44] HP Inc. HP 230 Wireless Mouse and Keyboard Combo. s.a. https://www.hp.com/us-en/shop/pdp/hp-230-wireless-mouse-and-keyboard-combo (18.04.2025)

[45] FCC ID PRDMU87. 2020. https://fccid.io/PRDMU87 (18.04.2025)

[46] FCC ID PRDKB42. 2020. https://fccid.io/PRDKB42 (18.04.2025)

[47] FCC ID PRDRX0U. 2020. https://fccid.io/PRDRX0U (18.04.2025)

[48] Cherry. CHERRY DW 5100. s.a. https://www.cherry.de/en-us/product/dw-5100 (18.04.2025)

Appendix

I. License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, **Lucas Salvatore G Agro**,

1. grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

   Radio Based Analysis of Wireless Mice and Keyboards using Proprietary Protocols,

   supervised by Danielle Melissa Morgan;

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Lucas Salvatore G Agro

15/05/2025