UNIVERSITY OF PIRAEUS

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGIES

DEPARTMENT OF DIGITAL SYSTEMS

Postgraduate Programme: Cybersecurity and AI Technologies

MASTER'S THESIS

Evaluating IoT Device Security Through Penetration Testing with Flipper Zero: Methods, Exploits, and Countermeasures

Αξιολόγηση Ασφάλειας Συσκευών IoT μέσω Δοκιμών Διείσδυσης με το Flipper Zero: Μέθοδοι, Επιθέσεις και Αντίμετρα

Student	Georgios Karanikas
Student ID	Mte24014
Email	gewrgios_karanikas@ssl-unipi.gr
Supervisor	Christos Xenakis, Professor

PIRAEUS

JULY 2026

Approval Page

This Master's Thesis was prepared at the Department of Digital Systems, University of Piraeus, within the Postgraduate Programme "Cybersecurity and AI Technologies".

Thesis title	Evaluating IoT Device Security Through Penetration Testing with Flipper Zero: Methods, Exploits, and Countermeasures
Student	Georgios Karanikas
Student ID	Mte24014
Supervisor	Christos Xenakis, Professor
Examination date	July 2026

Declaration of Originality and Ethics

I hereby declare that this Master's Thesis is the result of my own work, has been prepared in accordance with the principles of academic integrity, and has not been submitted, in whole or in part, for any other academic degree.

All sources used are fully acknowledged in the references, and any direct or indirect use of ideas, results, figures, tables, or code is properly cited.

Student signature	Date
	15/07/2026

Abstract

The proliferation of Internet of Things (IoT) devices has expanded the attack surface of everyday environments, concentrating much of the risk at the perception layer — the sensors, actuators, and short-range radios that connect the digital and physical worlds. Traditionally, assessing the security of these diverse wireless protocols required an array of specialized, expensive instruments, each dedicated to a single technology. This thesis tests a different proposition: that a single, commercially available, sub-$200 multi-tool — the Flipper Zero — can now perform this cross-protocol assessment, and that this accessibility represents a meaningful shift in the threat model itself.

The research follows a structured, four-phase penetration testing methodology (reconnaissance, enumeration, exploitation, and post-attack analysis) applied within a controlled, permission-based laboratory environment. Ten empirical case studies were conducted against representative perception-layer devices, spanning fixed- and rolling-code Sub-GHz systems, RFID and NFC access credentials, Wi-Fi networks, infrared appliances, Bluetooth Low Energy devices, and a workstation targeted through USB HID injection. Where certain advanced attacks could not be safely or legally reproduced, they were examined through open-source intelligence (OSINT).

The results demonstrate that the majority of successful compromises did not require breaking cryptography; they exploited its absence, its misconfiguration, or the implicit trust systems place in unauthenticated input. Fixed-code systems, default credentials, and weak passphrases fell readily, while correctly implemented defenses — protected management frames, a properly configured rolling code — resisted the same attacks. The experimental findings were synthesized into a set of countermeasures organized by root cause rather than by technology.

The principal contribution of this work is empirical evidence for a reframed threat model: cross-protocol, cyber-physical attacks once confined to well-funded specialists are now accessible to a broad population using inexpensive, portable hardware. The central conclusion is that the insecurity of the perception layer is, for the most part, not an absence of solutions but a backlog of unapplied ones.

Subject area	Cybersecurity
Keywords	IoT security·penetration testing·Flipper Zero·perception layer·wireless protocols

Περίληψη

Η ραγδαία εξάπλωση των συσκευών του Διαδικτύου των Πραγμάτων (IoT) έχει διευρύνει την επιφάνεια επίθεσης του καθημερινού περιβάλλοντος, συγκεντρώνοντας μεγάλο μέρος του κινδύνου στο επίπεδο αντίληψης (perception layer) — τους αισθητήρες, τους ενεργοποιητές και τις ασύρματες διεπαφές που συνδέουν τον ψηφιακό με τον φυσικό κόσμο. Παραδοσιακά, η αξιολόγηση της ασφάλειας αυτών των ετερογενών πρωτοκόλλων απαιτούσε πληθώρα εξειδικευμένων και δαπανηρών εργαλείων, καθένα για μία μόνο τεχνολογία. Η παρούσα διπλωματική εξετάζει μια διαφορετική υπόθεση: ότι ένα ενιαίο, εμπορικά διαθέσιμο, φορητό εργαλείο κόστους κάτω των 200 δολαρίων — το Flipper Zero — μπορεί πλέον να πραγματοποιήσει αυτή τη διαπρωτοκολλική αξιολόγηση, και ότι αυτή η προσβασιμότητα συνιστά ουσιαστική μεταβολή στο ίδιο το μοντέλο απειλής.

Η έρευνα ακολουθεί μια δομημένη μεθοδολογία δοκιμών διείσδυσης τεσσάρων φάσεων (αναγνώριση, καταμέτρηση, εκμετάλλευση και ανάλυση μετά την επίθεση), εντός ελεγχόμενου εργαστηριακού περιβάλλοντος. Διεξήχθησαν δέκα εμπειρικές μελέτες περίπτωσης σε αντιπροσωπευτικές συσκευές του επιπέδου αντίληψης, καλύπτοντας συστήματα Sub-GHz σταθερού και κυλιόμενου κώδικα, κάρτες RFID και NFC, δίκτυα Wi-Fi, συσκευές υπερύθρων, συσκευές Bluetooth Low Energy, καθώς και σταθμό εργασίας μέσω έγχυσης εντολών USB HID. Όπου ορισμένες προηγμένες επιθέσεις δεν μπορούσαν να αναπαραχθούν με ασφάλεια ή νομιμότητα, εξετάστηκαν μέσω ανοιχτών πηγών πληροφόρησης (OSINT).

Τα αποτελέσματα καταδεικνύουν ότι οι περισσότερες επιτυχημένες παραβιάσεις δεν απαίτησαν την παραβίαση κρυπτογραφίας· εκμεταλλεύτηκαν την απουσία της, την εσφαλμένη διαμόρφωσή της ή την εμπιστοσύνη των συστημάτων σε μη πιστοποιημένη είσοδο. Τα ευρήματα συντέθηκαν σε ένα σύνολο αντιμέτρων οργανωμένων κατά βαθύτερη αιτία. Η κύρια συνεισφορά της εργασίας είναι η εμπειρική τεκμηρίωση ενός αναπλαισιωμένου μοντέλου απειλής: διαπρωτοκολλικές επιθέσεις που άλλοτε προϋπέθεταν εξειδικευμένο εξοπλισμό είναι πλέον προσβάσιμες μέσω φθηνού, φορητού υλικού. Το βασικό συμπέρασμα είναι ότι η ανασφάλεια του επιπέδου αντίληψης δεν οφείλεται κυρίως σε έλλειψη λύσεων, αλλά σε συσσωρευμένες αδιάθετες λύσεις.

Θεματική περιοχή	Κυβερνοασφάλεια
Λέξεις-κλειδιά	ασφάλεια IoT·δοκιμές διείσδυσης·Flipper Zero·επίπεδο αντίληψης·ασύρματα πρωτόκολλα

Acknowledgements

I would like to sincerely thank my supervisor, Professor Christos Xenakis, for the guidance and valuable feedback provided throughout this thesis, and for the freedom to explore the practical side of this work.

I am also grateful to the Department of Digital Systems at the University of Piraeus, and to everyone who allowed me to test their devices and equipment — the empirical part of this research would not have been possible without their trust.

Finally, I would like to thank my family and friends for their patience and continuous support throughout my studies.

Table of Contents

Lists and Abbreviations

List of Figures

List of Tables

Abbreviations and Acronyms

Abbreviation	Full name / explanation
AES	Advanced Encryption Standard
AM	Amplitude Modulation
AP	Access Point
ASK	Amplitude Shift Keying
BLE	Bluetooth Low Energy
CRYPTO1	Proprietary stream cipher used in MIFARE Classic
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DNS	Domain Name System
DoS	Denial of Service
EAPOL	Extensible Authentication Protocol over LAN
ESP32	Espressif microcontroller with Wi-Fi/Bluetooth
FSK	Frequency Shift Keying
GATT	Generic Attribute Profile (Bluetooth)
GPIO	General-Purpose Input/Output
HF	High Frequency
HID	Human Interface Device

HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HVAC	Heating, Ventilation, and Air Conditioning
IoT	Internet of Things
IP	Internet Protocol
IR	Infrared
ISM	Industrial, Scientific, and Medical (radio band)
LF	Low Frequency
MAC	Message Authentication Code
MFkey32	Attack recovering MIFARE keys from a reader
NFC	Near Field Communication
OOK	On-Off Keying
OS	Operating System
OSINT	Open-Source Intelligence
PBC	Push-Button Configuration (WPS)
PIN	Personal Identification Number
PKES	Passive Keyless Entry and Start
PMF	Protected Management Frames
PMKID	Pairwise Master Key Identifier
PRNG	Pseudo-Random Number Generator
PSK	Pre-Shared Key
RF	Radio Frequency
RFID	Radio Frequency Identification
RKE	Remote Keyless Entry
RTOS	Real-Time Operating System
RTS	Radio Technology Somfy
SAE	Simultaneous Authentication of Equals
SDR	Software-Defined Radio
SSID	Service Set Identifier
SSL	Secure Sockets Layer
Sub-GHz	Sub-Gigahertz (radio below 1 GHz)
TCP	Transmission Control Protocol
TPMS	Tire Pressure Monitoring System
UHF	Ultra High Frequency
UID	Unique Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WPA2 / WPA3	Wi-Fi Protected Access 2 / 3
WPS	Wi-Fi Protected Setup

Glossary

Term	Definition
Attack surface	The set of points where an attacker can enter or extract data from a system.
BadUSB	An attack where a USB device poses as a keyboard and injects keystrokes.
Beacon flooding	Broadcasting fake Wi-Fi beacons to clutter a device's network list.
BLE advertising	The packets a Bluetooth Low Energy device broadcasts to announce itself.
Brute-force attack	Trying all possible keys or passwords until one works.
Captive portal	A login page shown on joining a network; abused here to deliver phishing.
Cloning	Copying a captured RFID/NFC credential onto another device.
Deauthentication attack	Spoofed 802.11 frames that forcibly disconnect a client from an AP.
Defense-in-depth	Layering independent controls so one failure is caught by another.
Dictionary attack	Trying a list of likely keys or passwords until one succeeds.
DuckyScript	The scripting language defining BadUSB keystroke payloads.
Emulation (RFID/NFC)	Reproducing a captured tag or card so a reader accepts it.
Evil Twin	A rogue access point mimicking a trusted network to harvest data.
Firmware	Low-level embedded software; custom firmware extends the Flipper Zero.
Fixed code	A static code sent every time, replayable once captured.
Frequency hopping	Rapidly switching frequency across a band during transmission.
GATTacking	Manipulating the GATT-layer communication of a BLE device.
Handshake (4-way)	The WPA2 key exchange; if captured, attackable offline.
Jamming	Transmitting noise to block legitimate signals on a frequency.
Keystroke injection	Automated command typing by a malicious HID device at machine speed.
Modulation	How data is encoded onto a radio carrier (e.g. ASK, FSK, OOK).
Nonce	A number used once in a cryptographic exchange; weak ones enable attacks.
OSINT	Analysis of publicly available information for security research.
Payload	The data or command carried by a transmission or attack.
Penetration testing	Authorized simulation of attacks to find exploitable vulnerabilities.

Perception Layer	The lowest IoT tier — sensors, actuators, and radios facing the physical world.
Phishing	Deceiving a user into revealing credentials via a fake page or message.
Pixie Dust	An offline WPS attack exploiting weak nonce generation.
Rate limiting	Capping how often an action can occur to contain floods.
Relay attack	Forwarding a car–key exchange over distance to fake proximity.
Replay attack	Retransmitting a captured signal to reproduce its effect.
Rogue Access Point	An unauthorized Wi-Fi AP impersonating a legitimate one.
RollJam	Jamming and capturing rolling codes to retain a valid unused code for later.
Rolling code	A single-use code from an incrementing counter, resisting replay.
Sniffing	Passively intercepting and reading wireless or network traffic.
Social engineering	Manipulating people into granting access, rather than attacking systems.
Spoofing	Forging a transmission or identity to impersonate a legitimate source.
Threat model	A description of the adversaries and attack paths a system must defend against.
Universal remote (dictionary)	An IR attack trying stored code sets until a device responds.

1. Introduction

The proliferation of connected devices has quietly redrawn the boundary between the digital and the physical. Everyday objects now sense, communicate, and act on the world — and in doing so, they inherit the security problems of networked systems while adding new ones of their own. This chapter introduces the subject and scope of the thesis. It sets out the security challenge posed by the IoT perception layer, defines the research problem and the specific objectives the work pursues, states the study's original contribution, describes the ethical methodology that governs the experimental work, and outlines the structure of the chapters that follow.

1.1 Subject of the Thesis

The Internet of Things (IoT) has fundamentally reshaped the modern technological landscape. Physical objects that once operated in isolation now communicate continuously, feeding data into systems that span smart home automation, electronic access control, and consumer connected devices. The scale of this shift is hard to overstate: the number of connected IoT devices reached roughly 18.5 billion in 2024 and is projected to pass 21 billion by the end of 2025, on a trajectory toward 39 billion by 2030 [1]. The convenience this creates is real—but so is the cost. As the number of connected devices has grown, so too has the global attack surface, and the cybersecurity implications are far from trivial. Industry telemetry reflects this directly: IoT-targeted malware rose by 107% in the first half of 2024 alone, with affected devices spending an average of 52.8 hours under attack [2].

A significant share of commercially available IoT devices are built under strict constraints: limited processing power, minimal memory, and tight energy budgets. These limitations frequently push manufacturers toward a pragmatic calculus where time-to-market takes precedence over security architecture. The result is a market saturated with devices that implement weak cryptographic mechanisms, or none at all—many of which receive no firmware updates after deployment and remain exposed indefinitely.

Of particular concern is the "Perception Layer," the foundational tier of IoT architecture responsible for interfacing with the physical world through sensors and actuators. This layer depends heavily on short- and medium-range wireless protocols: Sub-GHz radio frequencies, Wi-Fi (802.11), Bluetooth Low Energy (BLE), Radio

Frequency Identification (RFID) and Near Field Communication (NFC), and Infrared (IR). Wi-Fi alone accounts for roughly a third of all IoT connectivity, and BLE for close to a quarter, making these the dominant channels through which the physical layer reaches the network [1]. Because these protocols transmit over the air, they are inherently exposed to interception, signal manipulation, and physical-proximity attacks. The physical and data link layers, accordingly, have become a primary focus of contemporary security research—and, alongside them, the physical interface of the device itself, where a directly connected peripheral can bypass the network entirely.

1.2 Purpose and Objectives

This thesis undertakes a comprehensive empirical evaluation of the security posture of modern IoT devices, using systematic penetration testing as its primary investigative method. The central instrument of this research is the Flipper Zero—a compact, multi-protocol hardware tool that has gained recognition in the cybersecurity community for its versatility in hardware exploration and ethical hacking.

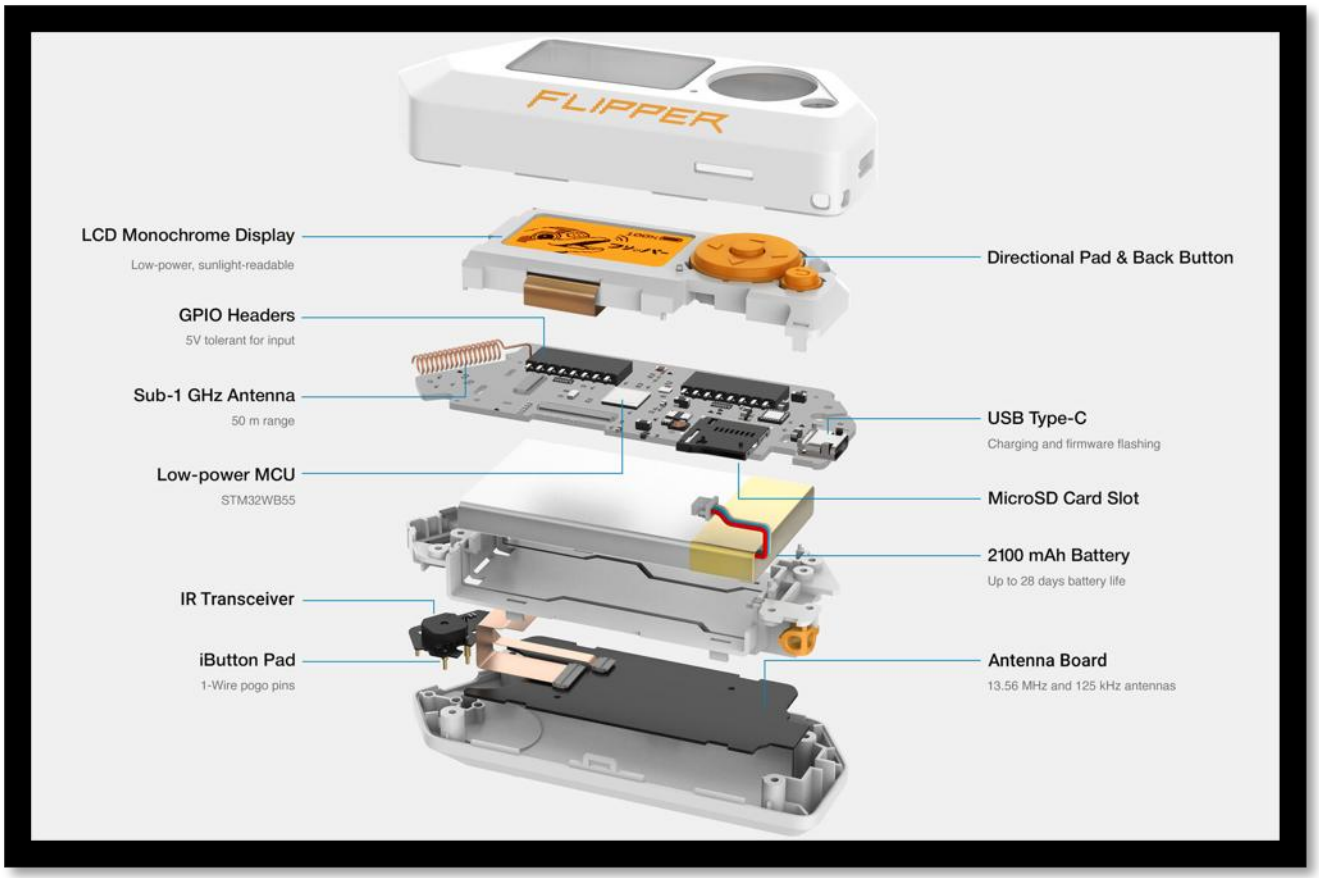

Figure 1.1: The Flipper Zero multi-tool. [3]

Its multi-protocol architecture makes it well-suited for examining the gap between documented, theoretical vulnerabilities and the actual threat conditions present in real-world deployments. The research is organized around four specific objectives:

Identification of Protocol Vulnerabilities. The study critically analyzes security flaws in widely used wireless protocols operating at the perception layer—specifically Sub-GHz communication (fixed and rolling code schemes), RFID and NFC, Wi-Fi, BLE, and legacy IR—categorizing weaknesses by type and exploitability.

Execution of Practical Penetration Tests. A series of controlled, ethically conducted experiments (case studies) are designed and carried out against representative IoT and perception-layer devices. Using the Flipper Zero alongside supplementary hardware such as a Wi-Fi Development Board, these tests execute attacks including signal replay, tag cloning, denial-of-service (DoS), Wi-Fi deauthentication and handshake cracking, rogue access point credential harvesting, and direct USB keystroke injection.

Impact Analysis. Each successful exploit is documented and evaluated in terms of its practical consequences—unauthorized physical access, data exfiltration, or complete loss of device control—to establish a clear picture of what perception-layer compromise means in operational terms.

Formulation of Countermeasures. Based on the experimental findings, the study proposes concrete mitigation strategies and defensive best practices applicable to both existing deployments and future IoT system design.

Original Contribution of the Study. The vulnerabilities of individual IoT protocols are not undocumented. Prior research has examined them in considerable depth. What that body of work has not provided, however, is a unified assessment framework built around a single, commercially available instrument. Earlier studies typically depend on disparate, highly specialized equipment — dedicated Software Defined Radios, Proxmark systems, custom-built microcontroller rigs — each addressing one protocol in isolation. The result is a fragmented picture of the threat landscape that does not reflect how an adversary actually operates in the field. This research takes a different approach. By placing the Flipper Zero at the center of the evaluation methodology, it demonstrates empirically that complex, cross-protocol attacks no longer require a laboratory full of

specialized hardware. They require a device that fits in a jacket pocket and costs less than two hundred dollars. The original contribution of this work, therefore, lies not in the discovery of new vulnerabilities, but in reframing the threat model itself — providing concrete evidence that attacks once confined to well-funded research environments are now accessible to a far broader population. That shift has consequences for how the industry, researchers, and policymakers should define a realistic adversarial threat in everyday IoT deployments.

1.3 Research Methodology

The entire empirical component of this research is grounded in ethical hacking principles and conducted strictly within a permission-based framework. All experiments take place inside a controlled environment, physically isolated from live production networks, to ensure legal compliance and prevent any unintended disruption to external systems. The testbed consists of representative consumer-grade perception-layer components: Sub-GHz gate and shutter automation, RFID and NFC access-control credentials, Wi-Fi access points and range extenders, infrared-controlled appliances, Bluetooth-enabled mobile devices, and a workstation used as the target for physical-interface testing.

The Flipper Zero serves as the primary investigative instrument throughout the practical assessment phase. It is augmented by a Wi-Fi Development Board running specialized firmware, and supported where necessary by a separate workstation running standard security tooling, enabling interception, analysis, and emulation of diverse wireless signals across multiple protocol layers.

The evaluation follows a standardized four-phase penetration testing cycle. Reconnaissance begins the process—identifying active radio frequencies and mapping broadcasting devices in the environment. Enumeration follows, in which captured signal payloads and packet structures are analyzed in detail. The Exploitation phase then executes the actual attack vectors, from basic signal replay to sustained denial-of-service conditions. Finally, Post-Attack Analysis assesses the concrete impact of each successful breach on target device functionality and security state.

Where the laboratory setup cannot directly reproduce certain sophisticated attack scenarios—such as those targeting moving vehicles or large-scale infrastructure—the

research draws on secondary sources through Open-Source Intelligence (OSINT) to examine advanced threat models documented elsewhere in the security literature.

1.4 Structure of the Thesis

The thesis is organized into eight chapters, progressing from foundational theory to experimental results and, ultimately, to defensive recommendations.

Chapter 1 establishes the research context, defines its objectives, and describes the ethical methodology applied throughout.

Chapter 2 provides the theoretical background. It explains the architecture of the IoT Perception Layer and details the operational mechanics of the wireless communication standards under examination: Sub-GHz, RFID/NFC, BLE, and Wi-Fi.

Chapter 3 maps the current wireless threat landscape, categorizing known vulnerabilities across these protocols and surveying the hardware and software tools used in contemporary security research.

Chapter 4 defines the formal penetration testing methodology—its ethical boundaries, the laboratory configuration, and the risk assessment criteria used to evaluate each experiment's outcome.

Chapter 5 is the empirical core of the thesis. It presents detailed case studies of practical attacks against real devices, spanning fixed- and rolling-code Sub-GHz systems, RFID and NFC access credentials, Wi-Fi management-frame and handshake attacks, infrared and BLE exploitation, captive-portal social engineering, and physical USB keystroke injection.

Chapter 6 extends the analytical scope through OSINT-based secondary research, examining advanced cyber-physical attacks—such as vehicle RollJam and RollBack exploits, relay attacks, and automated BLE flooding—that fall beyond the immediate laboratory setup.

Chapter 7 synthesizes the experimental findings into actionable recommendations, organizing defenses around the common root causes the attacks exposed and proposing cryptographic, configuration-level, and system-level mitigation strategies.

Chapter 8 draws the research to a close, summarizing key outcomes, reflecting on the legal and educational implications of accessible penetration testing hardware, and suggesting directions for future investigation.

2. Theoretical Background and Communication Protocols

This chapter establishes the technical foundation on which the rest of the thesis builds. A security evaluation of the perception layer is only as sound as one's understanding of how that layer actually works — what its components are, and how they communicate. The chapter therefore proceeds in two movements. It first sets out the layered architecture of IoT systems and locates the perception layer within it, explaining why this bottom tier concentrates so much of the overall attack surface. It then examines, in turn, the wireless communication protocols that the empirical work of Chapter 5 will target: Sub-GHz radio, RFID and NFC, Bluetooth Low Energy, Wi-Fi, and infrared. For each, the goal is not exhaustive protocol documentation but a focused account of the operating principles that later prove relevant to its security.

2.1 IoT System Architecture and the Perception Layer

Before any meaningful security evaluation can take place, the structural logic of IoT systems needs to be clearly established. Several architectural models exist in the literature, but the most widely accepted divides IoT infrastructure into three distinct tiers: the Application Layer, the Network Layer, and the Perception Layer [4].

The Application Layer sits at the top. It handles user interaction and data processing, typically running on cloud servers or local control panels. Below it, the Network Layer functions as a communication bridge, routing data between physical hardware and the application backend using protocols such as Wi-Fi, Ethernet, or cellular networks. These two layers have received considerable attention in the security literature.

The Perception Layer, however, is the foundation everything else depends on. This bottom tier is composed of the physical devices—sensors, actuators, RFID readers, microcontrollers—that interact directly with the physical environment. Its job is straightforward: collect analog data, convert it into digital signals, and execute physical commands. Simple in description, but deeply consequential in terms of security exposure.

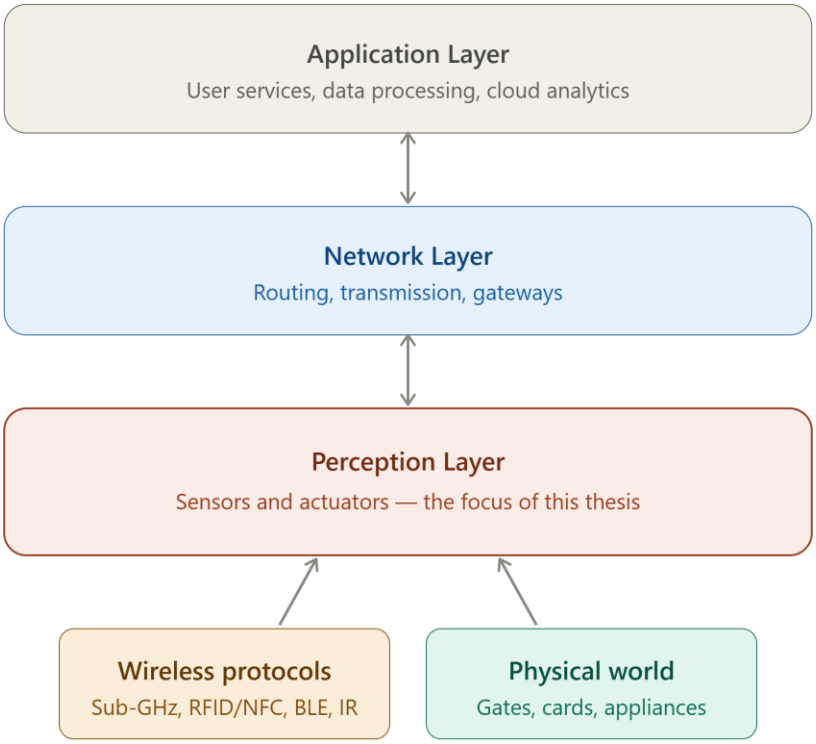

Figure 2.1: The three-layer IoT architecture, highlighting the Perception Layer.

2.1.1 The Role of Microcontrollers and Real-Time Operating Systems

At the heart of the Perception Layer are microcontrollers (MCUs). Chips like the ESP32 and STM32 series are among the most widely deployed, serving as the processing core for edge devices ranging from smart locks and access control panels to environmental monitoring units [4]. Managing multiple concurrent operations—simultaneously tracking a Sub-GHz receiver, maintaining a Wi-Fi connection, and triggering a mechanical relay—demands a level of task coordination that bare-metal programming often cannot reliably provide. For this reason, many of these devices run a Real-Time Operating System (RTOS), with FreeRTOS being the de facto standard in this space [5].

An RTOS enables efficient scheduling and resource management on severely memory-constrained hardware. That is its strength. But it comes with a trade-off that has direct security implications: because processing overhead must be kept to an absolute minimum—both to preserve battery life and to guarantee near-zero-latency responses—developers routinely omit computationally expensive cryptographic routines. The hardware simply cannot afford them, at least not without compromising performance. This constraint is not incidental. It is a structural

feature of the Perception Layer, and it is precisely what makes these devices attractive targets for local proximity attacks.

2.2 Sub-GHz Wireless Communications

A large share of perception-layer devices do not use Wi-Fi or Bluetooth at all. Instead, they rely on wireless communication operating below 1 GHz—what the industry broadly refers to as Sub-GHz technologies. The distinction matters. Where 2.4 GHz bands are congested and power-hungry, Sub-GHz signals carry further, penetrate solid obstacles more effectively, and consume substantially less energy [6]. For battery-powered devices that need to operate reliably through concrete walls or across a parking lot, the physics strongly favor this frequency range.

2.2.1 Frequency Ranges and IoT Applications

The two most commercially dominant Sub-GHz frequencies are 433.92 MHz, prevalent across Europe and Asia, and 315 MHz, which is the North American standard. Both fall within the Industrial, Scientific, and Medical (ISM) band—heavily regulated, but unlicensed for short-range use [6]. This makes them the default choice for a wide array of consumer and commercial applications: Remote Keyless Entry (RKE) systems, automated garage doors, wireless weather stations, basic home alarm systems.

The hardware required to operate on these frequencies is cheap. Very cheap. That cost advantage has driven massive proliferation across both legacy infrastructure and modern smart home products, often with little regard for what the minimal bill-of-materials actually implies for security.

2.2.2 Fixed Code Systems: Mode of Operation and Limitations

The simplest security mechanism found in Sub-GHz devices is the **fixed code** system—sometimes called a static code system. The concept is elementary: a remote transmitter is pre-programmed, either at the factory or by the end user via physical DIP switches, with a specific binary sequence that never changes. When a button is pressed, the microcontroller modulates this static payload onto a radio carrier wave using basic Amplitude Shift Keying (ASK) or On-Off Keying (OOK) and transmits it [6]. The receiver checks whether the incoming binary string matches the stored value. If it does, it executes the associated action—opening a gate, for instance.

The vulnerability here is not subtle. Because the transmitted signal is identical every single time, it carries no cryptographic authentication and no timestamp. There is nothing in the payload that encodes when it was sent or by whom. Any attacker within radio range can record the transmission, store the waveform, and replay it at will. The receiving unit has no mechanism to distinguish a legitimate remote from a replayed copy. Fixed code architectures are, by design, entirely defenseless against capture-and-replay attacks. It is not an edge-case weakness—it is a fundamental architectural flaw.

2.3 Advanced Security Mechanisms: Rolling Codes

Fixed code systems did not survive scrutiny for long. Once their vulnerability to replay attacks became widely understood, manufacturers moved to a different approach: **Rolling Code** technology, also referred to as Code Hopping. The principle is straightforward—the transmitted binary sequence changes with every button press, so any signal captured by an attacker is already stale by the time they attempt to use it.

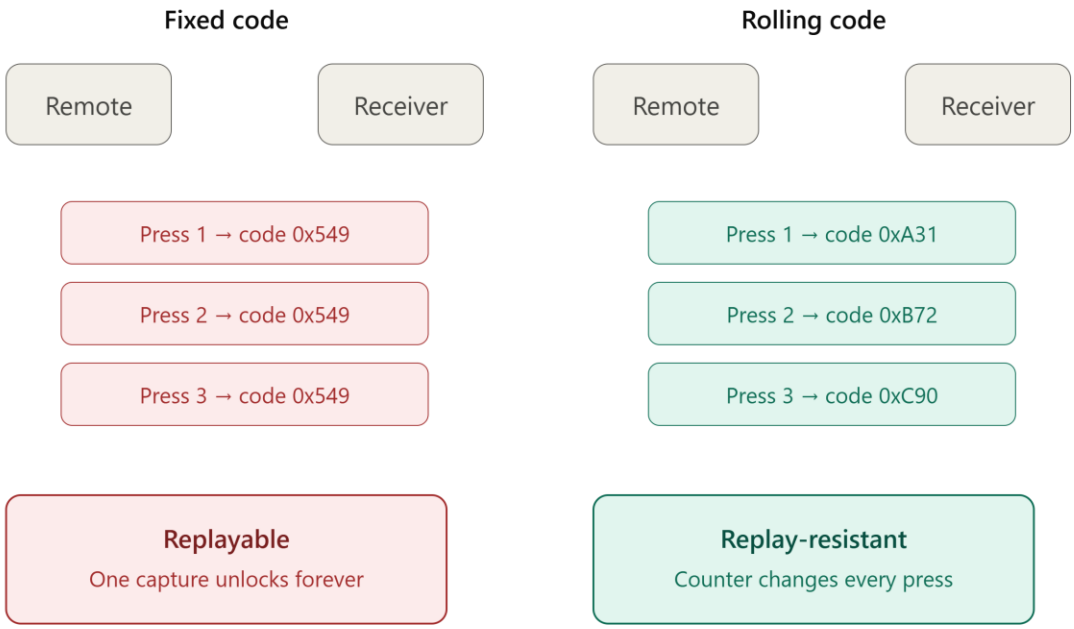

Figure 2.2: Fixed-code versus rolling-code transmission. A fixed code repeats identically and is replayable; a rolling code changes with each press.

2.3.1 Code Hopping Architecture and the KeeLoq Algorithm

The most prominent technical implementation of rolling code in consumer hardware is the **KeeLoq** algorithm, developed by Microchip Technology. KeeLoq is a

proprietary block cipher used extensively in Remote Keyless Entry (RKE) systems and garage door openers, most notably within integrated circuits such as the HCS301 [7]. Each transmitter is provisioned with a unique 64-bit encryption key during manufacture, and this key is shared with the corresponding receiver through an initial pairing—or "learning"—process.

When the user presses a button, the transmitter does not simply broadcast a stored value. Instead, it generates a fresh 32-bit hopping code encrypted via KeeLoq. This payload carries several distinct components: a synchronization counter, a discrimination value, and a set of function bits indicating which button was pressed. The synchronization counter is the mechanism that makes the whole system work—it increments with every transmission, guaranteeing that the resulting ciphertext is never repeated [7].

2.3.2 Encryption Structure and the Synchronization Window

Validation on the receiver side involves two separate checks. First, the receiver decrypts the incoming hopping code and verifies that the embedded serial number matches its internal records. Second, and more critically, it examines the synchronization counter [7].

The problem is a practical one: a user might press their remote while out of range—in a pocket, in a bag, at a distance—producing transmissions the receiver never sees. To accommodate this, receivers implement a **"Synchronization Window,"** typically accepting counter values up to 256 or 512 increments ahead of their current internal count. If the incoming counter falls within this forward window, the receiver updates its own counter and grants access. If the counter lags behind the receiver's current value, or falls too far ahead, the signal is rejected.

This design successfully neutralizes simple replay attacks. But it opens a different exposure. An attacker who can jam the receiver's signal while simultaneously recording valid transmissions can effectively harvest codes the receiver has never processed—codes that remain valid within the synchronization window. This attack class, known broadly as desynchronization or signal harvesting, exploits the very tolerance mechanism that makes rolling codes usable in practice. The window designed for user convenience becomes the margin that an attacker can work within.

This compromise illustrates a tension that runs through almost every IoT design decision: user convenience versus cryptographic integrity. The synchronization window exists because manufacturers needed to prevent legitimate users from being locked out. That is a reasonable operational concern. But the solution — deliberately widening the acceptance range — degraded the security boundary in a measurable, predictable way. The persistence of this vulnerability is not a failure of cryptographic theory. It is the direct consequence of an economic tradeoff, made deliberately, that has left millions of deployed systems structurally exposed [7].

2.4 RFID and NFC Identification Technologies

Radio Frequency Identification (RFID) and its derivative standard, Near-Field Communication (NFC), form the backbone of contactless access control and asset tracking across IoT deployments worldwide. The operating principle differs fundamentally from Sub-GHz communication. Rather than broadcasting signals over distance, RFID and NFC rely on electromagnetic coupling between a reader and a tag—close-range energy transfer that powers the tag and enables data exchange.

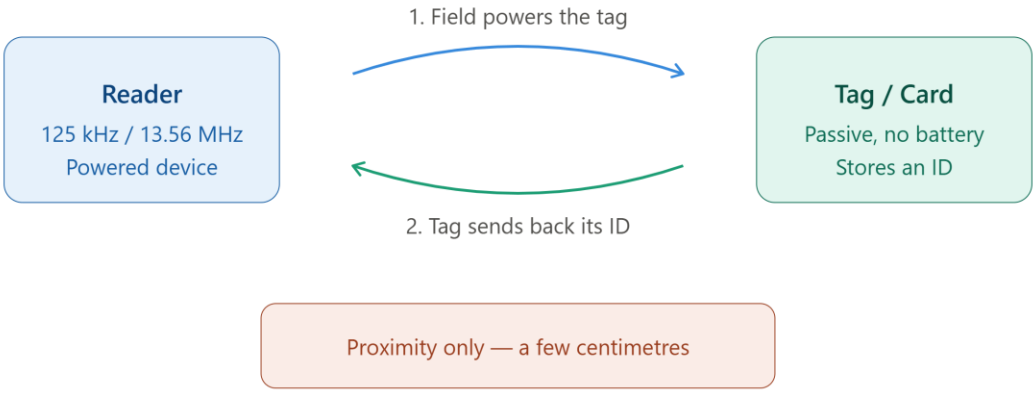

Figure 2.3: RFID/NFC operation. The reader's field powers a passive tag, which transmits its stored identifier back over the same field.

2.4.1 Frequency Separation: 125 kHz vs. 13.56 MHz

Two frequency bands dominate the access control landscape, and they are not interchangeable in either capability or security profile.

Low-Frequency systems (LF – 125 kHz) are characteristic of older proximity badges and key fobs. Read ranges are short, data transfer rates are slow, and—most significantly—the vast majority of these tags broadcast their Unique Identifier (UID) in plaintext the moment they receive power from the reader's magnetic field. No authentication. No encryption. The tag simply announces itself. This makes LF systems trivially easy to sniff and clone with portable hardware, and it is why they remain a persistent target in physical penetration testing.

High-Frequency systems (HF – 13.56 MHz) operate at a different level of sophistication. Used by NFC-enabled devices and modern smart cards, this band supports richer data structures and actual cryptographic protocols. Communication is bidirectional—the card and reader perform a handshake before any meaningful data is exchanged, enabling mutual authentication.

The distinction matters in practice. Frequency band alone does not determine security, but it is a reliable starting indicator of what threat model a given system was designed to address.

2.4.2 Tag Structure and the MIFARE Classic 1K Case

Physically, an RFID tag is minimal: an integrated circuit and an antenna coil. Most tags are passive—they carry no internal power source and rely entirely on energy harvested from the reader's radio field to operate.

The **MIFARE Classic 1K** is one of the most widely deployed HF smart card standards in existence, found in public transit systems and building access control installations across the world. It uses the proprietary **CRYPTO1** encryption algorithm. The scale of its deployment might suggest confidence in its security. That confidence is not warranted.

Researchers have identified significant cryptographic weaknesses in MIFARE Classic [8]. Flaws in the card's random number generator, combined with structural vulnerabilities in the CRYPTO1 cipher itself, allow an attacker to recover secret sector keys through what are known as "Nested" and "Hardnested" attacks. Once the keys are extracted, the complete memory contents of the card become readable—and the card becomes clonable. The MIFARE Classic case is an instructive one: even a system operating in the encrypted HF band is only as secure as the quality of its underlying

cryptographic implementation. A broken cipher, regardless of the frequency it runs on, offers no real protection.

The continued deployment of MIFARE Classic, decades after its cryptographic foundations were publicly dismantled, points to something beyond simple negligence [8]. Replacing millions of wall-mounted readers and issued transit cards is not a software update — it is a capital project of significant scale, and most organizations simply will not absorb that cost. The result is a well-documented, easily exploitable vulnerability that persists not because the industry lacks awareness of it, but because awareness alone does not pay for a hardware replacement cycle.

2.5 Bluetooth and Bluetooth Low Energy (BLE) Protocols

Bluetooth has come a long way from its original purpose as a wireless cable replacement. What began as a convenient way to cut the cord between a phone and a headset has matured into a complex wireless standard embedded in billions of IoT devices. For the perception layer specifically, it is Bluetooth Low Energy (BLE) that now dominates—not its predecessor—owing almost entirely to its power efficiency and minimal resource footprint [9].

2.5.1 Technical Architecture: From Classic to BLE

Classic Bluetooth, operating under the BR/EDR specification, uses Frequency Hopping Spread Spectrum (FHSS) across 79 channels in the 2.4 GHz ISM band. Its network topology is organized as a Piconet, where a Central device—previously called the Master—coordinates communication with one or more Peripherals [9]. This model suits continuous, high-throughput applications like audio streaming well. IoT sensors, however, rarely need that. They need to send a small packet occasionally and then go back to sleep for as long as possible.

BLE is designed around exactly that requirement. It operates across 40 channels, each 2 MHz wide, and allows devices to spend the vast majority of their time in deep-sleep states, waking only to transmit or receive brief data bursts [10]. For battery-operated sensors and wearables expected to run for months or years without intervention, this architecture is not just convenient—it is necessary.

2.5.2 BLE Advertising Channels and the GATT Profile

Two mechanisms define how BLE devices discover each other and exchange data: **Advertising** and the **Generic Attribute Profile (GATT).**

Advertising channels. Before any connection is established, a BLE peripheral broadcasts its presence using three dedicated channels—37, 38, and 39. These Advertising Packets announce the device and its capabilities to any nearby Central that might be listening [10]. The security problem here is immediate: these broadcasts are unencrypted and continuous. Any device within range can receive them. This makes BLE advertising a natural target for resource exhaustion attacks, most notably BLE Advertising Spam, where a flood of spoofed advertising packets can overwhelm nearby devices or degrade their ability to function normally.

GATT Profile. Once a connection is established, data exchange is governed by the GATT hierarchy. Data is organized into Services, which group related data points called Characteristics. Each Characteristic defines what operations are permitted on it—Read, Write, Notify, and so on. When an IoT device is misconfigured with overly permissive GATT access controls, an attacker who connects to it can interact with those characteristics directly: reading out sensitive environmental data, writing commands that alter device state, or both. No further exploitation is required. The device simply complies, because it was never told not to.

This pattern of misconfiguration reflects an assumption that is widespread in IoT development and consistently wrong: that physical proximity implies authorization. A device nearby is not necessarily a device that should be trusted. Developers bypass BLE's built-in cryptographic safeguards to reduce pairing friction and accelerate time-to-market — legitimate commercial pressures, but ones with direct security consequences [9]. The Bluetooth stack, designed with security mechanisms, is reduced to a data conduit. The mechanisms are present. They are simply never enabled.

2.6 Infrared (IR) Technology in the IoT Ecosystem

Not every wireless technology in the modern IoT landscape operates on radio frequencies. Infrared has been declared obsolete many times over—and yet it persists. Its continued presence is easy to explain: it is cheap, simple to implement, and its inability

to pass through walls provides a natural operational boundary that, for many use cases, is sufficient [11].

2.6.1 Modulation Principles and the 38 kHz Standard

IR communication works by pulsing an infrared LED at a carrier frequency—38 kHz is the widely adopted standard—which allows receivers to distinguish the intentional signal from ambient infrared sources such as sunlight or incandescent lighting [11]. The actual data is encoded through specific protocol families, including NEC, Sony, and RC5, each defining its own timing sequences and pulse patterns.

Because IR is a line-of-sight technology, it has historically been treated as inherently safe from remote cyberattack. Physical proximity seemed like a natural security barrier. Portable tools capable of capturing and precisely replaying optical signals have changed that assumption considerably [11].

2.6.2 The Survival of Legacy IR in Hybrid IoT Devices

Many contemporary "smart" devices are, under the surface, hybrid systems. A high-end air conditioning unit or a home cinema receiver will typically connect to the local network over Wi-Fi for mobile app and voice assistant integration, while simultaneously retaining a fully functional IR receiver for compatibility with traditional remote controls. Both interfaces control the same hardware.

This creates a meaningful security gap. An attacker who cannot break the device's Wi-Fi encryption—or who simply has no interest in attempting it—can still achieve full physical control by emulating the device's IR command set [11]. The network-layer security is bypassed entirely. No credentials, no handshake, no authentication of any kind. The legacy interface, retained for the sake of backward compatibility, becomes a side entrance that all the network-level protections leave completely unguarded.

The deeper problem here is architectural. Manufacturers add an encrypted Wi-Fi interface to a device that already has an unauthenticated IR receiver and treat these as separate concerns. They are not. An attacker who identifies the weakest interface does not need to engage with the strongest one. Backward compatibility is a legitimate engineering requirement, but it cannot be pursued in isolation from the threat model. When a legacy interface grants full device control without any authentication, it does not matter how strong the primary network security is.

2.7 The Wi-Fi Protocol (802.11) in IoT Systems

For IoT devices that require reliable high-bandwidth connectivity and cloud integration, Wi-Fi is the standard answer. Operating across the 2.4 GHz and 5 GHz bands under the IEEE 802.11 family of specifications, it provides the communication backbone for everything from smart cameras to industrial gateways [12].

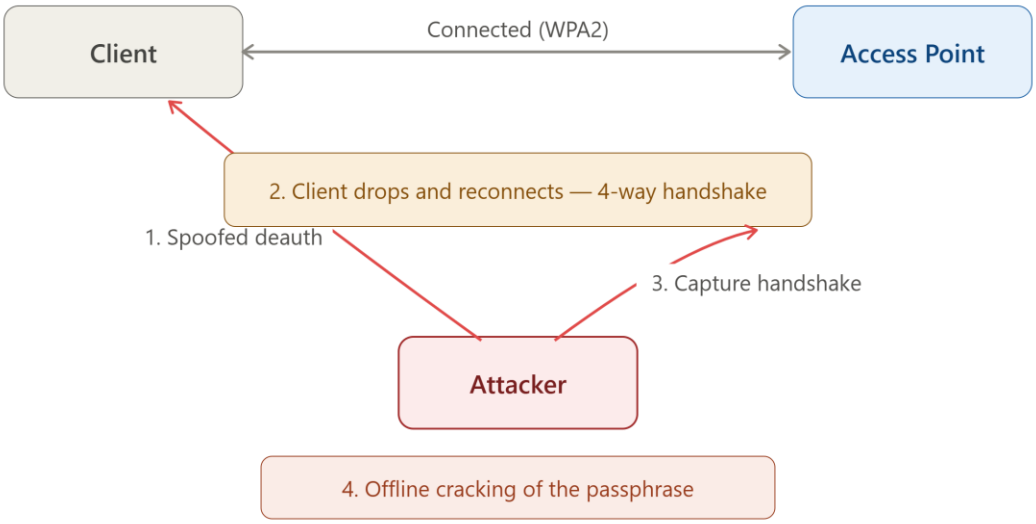

Figure 2.4: Wi-Fi deauthentication attack. A spoofed deauth frame forces the client to reconnect, allowing the attacker to capture the WPA2 handshake for offline cracking.

2.7.1 Basic WLAN Structure and Management Frames

A typical IoT Wi-Fi deployment involves two roles: an Access Point (AP) and a Station—the IoT device itself. Data exchange between them is only part of what the protocol handles. A separate category of traffic, known as Management Frames, coordinates the connection lifecycle. Beacon frames advertise the network's presence. Probe Requests allow stations to discover available networks. Deauthentication frames signal that a connection should be terminated [12].

The problem is that in many older 802.11 implementations, management frames are transmitted without encryption or cryptographic authentication. They are trusted by design. An attacker can exploit this by forging a Deauthentication frame that appears to originate from the legitimate AP, instructing the target device to disconnect—and the device will comply, because it has no way to verify the frame's origin. The result can be a persistent denial-of-service condition, or it can serve as the first step in an Evil Twin

attack, where the disconnected device is coaxed into associating with a rogue access point under the attacker's control. One forged packet. Significant consequences.

Protected Management Frames were introduced in the 802.11w amendment specifically to close this vulnerability [13]. That standard exists. The problem is adoption. Manufacturers of low-cost IoT devices frequently disable PMF to maintain compatibility with older routers and reduce processing overhead — two concerns that are real, but neither of which justifies preserving a known attack surface. The result is an ecosystem where a security standard is available, implemented in many devices, and routinely turned off anyway. Universal connectivity, it appears, consistently outranks network integrity as a design priority.

3. Threats, Vulnerabilities, and Assessment Tools

With the architectural foundations of the IoT Perception Layer established, the focus now shifts to how these protocols behave under adversarial conditions—a concern that has become central as the security challenges of the IoT world have grown in both scale and criticality [14]. Wireless communication has no physical perimeter. There is no door to pick, no cable to tap. An attacker needs only to be within radio range with appropriate hardware—and in many cases, that hardware is inexpensive and readily available. This chapter categorizes the specific threat vectors associated with each major protocol before examining the tools researchers use to exploit them.

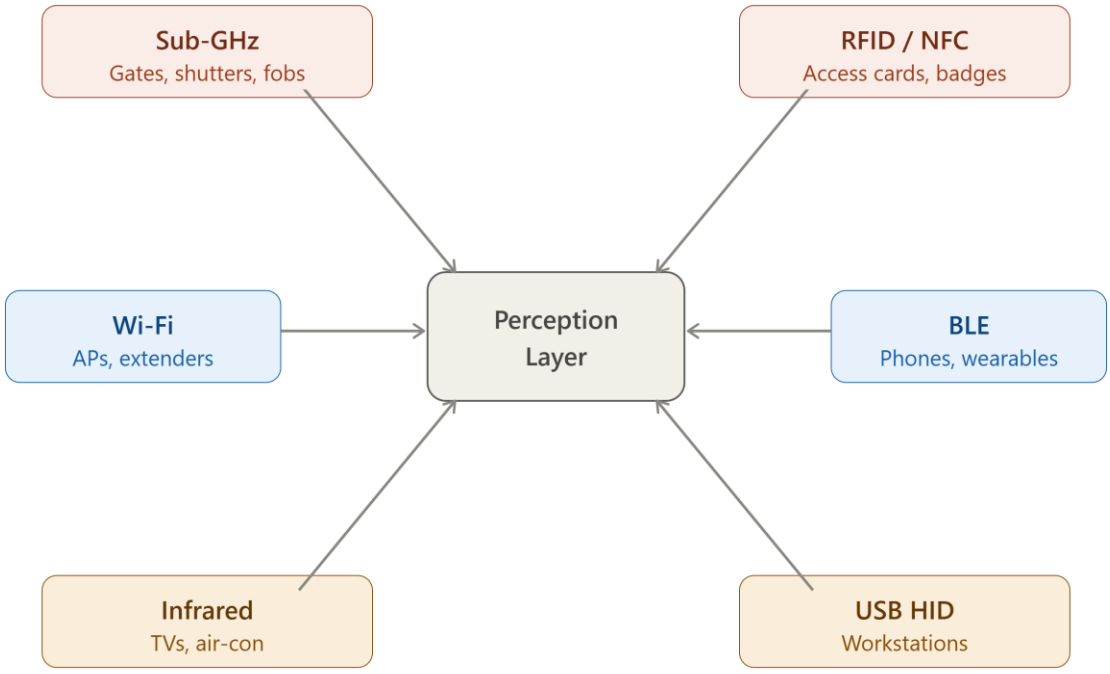

Figure 3.1: The attack surface of the IoT perception layer. Six wireless and physical vectors — Sub-GHz, RFID/NFC, Wi-Fi, BLE, infrared, and USB HID — expose the layer to proximity-based attacks.

3.1 Threats in Radio Frequency Systems (Sub-GHz & RFID)

The vulnerabilities found in radio frequency systems span a wide spectrum, from elementary design oversights to mathematically sophisticated exploits. They fall broadly into three categories, each tied to the technology beneath them.

3.1.1 Attacks on Fixed Codes: Capture and Replay

Fixed code systems have no dynamic authentication. That single fact is the root of everything. The primary attack against them is straightforward: **Capture and Replay.**

An adversary positions a portable transceiver near the target—monitoring the relevant frequency, typically 433.92 MHz—and waits for the legitimate user to press their remote. The raw waveform or decoded binary sequence is recorded. Later, at a time of the attacker's choosing, that same signal is retransmitted. The receiver evaluates only the static payload. It has no awareness of when the signal was sent, or by whom. It accepts the command and grants access. The entire attack requires no cryptographic knowledge and no specialized skills. It requires patience and a sufficiently sensitive receiver [6].

The persistence of fixed-code systems has almost nothing to do with technical merit. Static microcontrollers cost fractions of a cent per unit. For manufacturers operating on thin margins in high-volume consumer markets, that arithmetic is difficult to argue against. Security is not weighed against the vulnerability — it is simply not weighed at all. The replay attack is treated as an acceptable business risk, not an engineering problem that needs solving [6].

3.1.2 Attacks on Rolling Codes: Jamming, RollJam, and RollBack

Rolling codes were designed specifically to defeat replay attacks, so defeating rolling codes requires a more deliberate approach.

The most well-documented technique is the **RollJam attack** [15]. The attacker deploys a device that simultaneously jams the receiver's frequency—preventing any incoming signal from being processed—while capturing transmissions on a slightly narrower band. When the legitimate user presses their remote, the signal is recorded by the attacker but never reaches the receiver. The door does not open. The user, assuming the press failed, presses the button again. A second code is captured. At this point, the attacker immediately retransmits the first captured code to the receiver. The door opens. The user is satisfied. The attacker retains the second code—valid, unused, and redeemable at any future time. One interaction, two codes captured, one entry gained.

A more recently documented variant is the **RollBack attack** [16]. It operates differently. Rather than jamming the receiver in real time, RollBack exploits the receiver's

resynchronization logic. By capturing a sequence of consecutive valid transmissions and replaying them in a specific order, an attacker can force the receiver to revert its internal counter to an earlier state—effectively tricking it into accepting codes it has already processed and should have expired. No active jamming required.

What these attacks reveal is a structural limitation that no firmware update can fully address. Battery-constrained key fobs cannot receive asynchronous status updates from the receiver — the hardware does not support it. Manufacturers compensate by widening the synchronization window, which keeps legitimate users from being locked out but simultaneously gives attackers room to operate. The KeeLoq cipher is not the weak point. The weak point is the design decision to prioritize usability over strict cryptographic state enforcement, and that decision is baked into the hardware [7].

3.1.3 Threats in RFID / NFC: Sniffing, Cloning, and Relay

Physical access control systems rely heavily on RFID, which makes them a consistent target for proximity-based [17] attacks.

Against **Low-Frequency (125 kHz)** systems, the dominant attack is **Cloning**—and it is almost embarrassingly simple. Because LF tags broadcast their Unique Identifier in plaintext with no authentication, an attacker with a concealed reader needs only to pass within a few centimetres of the victim's badge or key fob. The UID is captured. It is then written to a blank, rewritable tag—a T5577 chip is a common choice—producing a functional duplicate that the access control system cannot distinguish from the original [18].

High-Frequency (13.56 MHz) systems present a harder target cryptographically, so attackers often sidestep the cryptography entirely. **Relay attacks** accomplish this through cooperation [19]. Two attackers operate at separate locations simultaneously. One positions themselves near the locked door and uses a device to initiate a challenge from the reader. That challenge is transmitted—over the internet or another communication channel—to a second attacker stationed near the victim, who may be sitting in a café, entirely unaware. The second attacker's device relays the challenge to the victim's card, which computes the correct cryptographic response. That response travels back through the chain to the door reader. The system unlocks. No key has been extracted. No cipher has been broken. The protocol was executed correctly from end to

end—just with the legitimate cardholder's participation happening at a different location without their knowledge.

These attacks persist for a reason that has nothing to do with cryptography [20]. Upgrading a facility from unencrypted low-frequency readers to a mutually authenticated high-frequency standard is not a configuration change — it is a capital project. New readers, new cards, new administrative workflows. Most facility managers look at that cost and decide that the theoretical risk of badge cloning is the cheaper problem to live with. The result is critical physical security infrastructure running on technology that the research community declared obsolete years ago [18].

3.2 Threats in Bluetooth / BLE Networks

BLE's design priorities—minimal power consumption, fast connection setup, lightweight packet structures—create attack surfaces that are distinct from those of traditional radio frequency systems. Threats against BLE generally target three areas: its advertising mechanism, its data exchange architecture, or the host operating system of the connected device.

3.2.1 Resource Exhaustion: BLE Spamming and Flooding

BLE devices advertise their presence continuously using unencrypted packets broadcast across dedicated channels. This is by design—it is how devices are discovered. Attackers exploit it by generating enormous volumes of spoofed advertising packets, mimicking legitimate devices such as Apple AirPods, Samsung Galaxy Buds, or generic smart televisions, at rates of thousands of packets per second [21].

When a smartphone or IoT gateway receives this volume of fabricated pairing requests, its Bluetooth stack cannot process them all. The consequences vary by device and implementation: the user interface may freeze under a cascade of pop-up notifications, the Bluetooth subsystem may become unresponsive, or the device may crash and reboot. In each case, the result is a denial-of-service condition achieved without ever establishing a legitimate connection or exploiting a single authentication mechanism.

The underlying problem is that the BLE specification demands this behavior. Seamless pairing requires devices to listen aggressively for incoming advertising packets, and ecosystem providers — Apple, Google — engineered their operating systems to process those packets as efficiently as possible. Frictionless connectivity was the goal.

Attackers have simply recognized that the same mechanism which makes pairing effortless also makes flooding trivially easy. There is no clean patch for this. The vulnerability is the feature [21].

3.2.2 GATT Services Violation (GATTacking)

Once an attacker establishes a connection to a BLE peripheral, the attack surface shifts inward to the Generic Attribute Profile. In a significant number of consumer IoT devices—smart padlocks and medical monitors among the most commonly documented—GATT Characteristics are exposed with open Read/Write permissions and no pairing requirements enforced [9].

A **GATTacking** exploit follows a predictable sequence. The attacker scans for the target device, connects directly without triggering any authentication challenge, identifies the specific Characteristic responsible for controlling the device's physical output—a mechanical relay, for instance—and writes a new hexadecimal value to it. The padlock opens. No alarm is raised. No log entry distinguishes this interaction from a legitimate one. The device behaved exactly as programmed; the programming was simply wrong.

This is not a sophisticated attack exploiting an obscure edge case. It is the consequence of developers skipping steps. Proper Bluetooth bonding, authenticated pairing, and encrypted characteristic access require additional development time and memory overhead — neither of which is abundant in a race-to-market IoT project. The implicit assumption is that attackers will not bother reverse-engineering undocumented hexadecimal control commands. That assumption is wrong, and the open GATT characteristic is the evidence [9].

3.2.3 Human Interface Device (HID) Attacks and Code Injection

Among the more severe Bluetooth attack classes is the exploitation of the Human Interface Device profile—the same protocol that allows a wireless keyboard or mouse to communicate with a computer. The attack surface here is not a peripheral device. It is the host machine.

Vulnerabilities such as CVE-2023-45866 have demonstrated that certain operating systems—including older releases of Android, macOS, and Linux—can be manipulated into accepting a Bluetooth HID connection from a rogue device masquerading as a keyboard, bypassing the standard user confirmation prompt entirely [22]. Tools like

BlueDucky automate this process end-to-end. Once the unauthorized connection is accepted, the attacker's device injects a pre-programmed sequence of keystrokes—a DuckyScript—at speeds no human could replicate. Within seconds, that script can open a terminal, retrieve a malicious payload from a remote server, and execute it. Full remote access, achieved through a Bluetooth pairing that the victim never knowingly approved.

This vulnerability exists because of a historical assumption that has never been formally revisited: that a keyboard should work immediately upon connection, without cryptographic verification of any kind. Users expect it. Developers built for it. The result is that HID profiles sit outside the security model that governs most other connection types. Attackers do not exploit a flaw in that model — they exploit the absence of one, using the system's own plug-and-play infrastructure to inject arbitrary input with no authentication required [22].

3.3 Threats in Wireless Local Area Networks (Wi-Fi 802.11)

Many edge sensors operate on low-power protocols and never touch Wi-Fi at all. IoT gateways and bandwidth-intensive devices, however, depend on it almost entirely. The 802.11 standard provides meaningful encryption through WPA2 and WPA3, but the mechanics governing how devices connect and disconnect contain structural weaknesses that have been exploited consistently for years.

3.3.1 IoT Device Disconnection (Deauthentication Attacks)

The most common threat to IoT Wi-Fi stability is also one of the simplest to execute: the Deauthentication attack. In networks that do not enforce 802.11w Protected Management Frames (PMF), connection control packets are transmitted in plaintext with no authentication. An attacker who knows the MAC address of the legitimate Access Point can spoof it—trivially—and broadcast a continuous stream of forged deauthentication frames directly at a target device, such as a security camera or a smart thermostat. The device trusts the source address. It drops its connection immediately and unconditionally. The result is a precisely targeted, localized denial-of-service condition, achieved without the attacker ever needing to know the network password [12].

The reason this attack still works is straightforward: enforcing PMF universally would break connectivity for a large share of deployed IoT devices that lack the processing capacity to support it. Router manufacturers know this. Rather than fragment

the market, they ship with PMF configured as optional. The security standard exists, is well understood, and is routinely left disabled by default to accommodate hardware that should, by any reasonable assessment, have been retired years ago [13].

3.3.2 Rogue Access Points and the Evil Portal

Deauthentication rarely serves as the final objective. More often, it is the first step [23]. Once a device—or a human user—is forcibly disconnected, it automatically attempts to reconnect to a known network. An attacker can position a Rogue Access Point, commonly called an Evil Twin, that broadcasts the identical SSID as the legitimate network but at a higher signal strength [24]. If the target associates with this malicious AP, the attacker gains a man-in-the-middle position over all of its traffic. From there, an Evil Portal can be deployed: all web requests are intercepted and redirected to a counterfeit captive page—a fake router login screen, a simulated firmware update prompt—designed to extract administrative credentials from whoever is on the other end.

The effectiveness of Evil Twin attacks reflects a broader problem with how IoT devices handle network reconnection [25]. Most embedded systems are programmed to associate automatically with the strongest available signal matching a known SSID, without any certificate validation or server authentication. This behavior, designed for seamless connectivity, makes the device an entirely passive participant in its own compromise.

3.3.3 WPA/WPA2 Handshake Capture

For an attacker whose goal is to permanently compromise a wireless network rather than just disrupt it, the target is the 4-way cryptographic handshake that occurs during device authentication. The process begins with the same deauthentication technique described above—forcing a target IoT device to disconnect and then capturing the encrypted handshake packets as it automatically reconnects. The attacker does not need to crack the handshake in the field. The captured hash is stored and taken offline, where it can be subjected to dictionary or brute-force attacks using GPU-accelerated hardware at speeds far beyond what would be feasible over the air. The physical presence near the network is brief. The computational work happens elsewhere, at leisure [24].

The continued vulnerability of WPA2 to offline handshake attacks highlights the fundamental tension between password-based authentication and computational

advances in GPU processing. A network secured with a weak or common passphrase is effectively insecure regardless of the encryption standard applied to it — the cryptographic wrapper is only as strong as the secret it protects.

3.3.4 Exploitation of Wi-Fi Protected Setup (WPS)

WPA2 secures the data flowing across a network. It does not necessarily secure every mechanism used to join that network in the first place.

Wi-Fi Protected Setup (WPS) was introduced specifically to simplify the onboarding of headless IoT devices—smart plugs, wireless printers, IP cameras—that have no screen or keyboard through which a user could type a passphrase. The solution was an 8-digit PIN. Press a button on the router, enter the PIN on the device, and the connection is established without ever handling the full WPA2 credential.

The vulnerability was identified early and documented thoroughly [26] [27]. The authentication protocol verifies the PIN in two separate halves rather than as a single value. This partitioning collapses the effective keyspace from one hundred million possible combinations to roughly eleven thousand—a figure that can be exhausted through brute-force within hours using tools such as Reaver or Bully running on a standard laptop. A more efficient variant, the Pixie Dust attack, targets implementation flaws in the router's random number generator. Where brute-force takes hours, Pixie Dust can recover the PIN offline in seconds—without sending a single additional packet to the target after the initial exchange.

The persistence of WPS in deployed hardware is consistent with a pattern observed throughout this chapter. A usability feature introduced to make smart home device pairing marginally more convenient has never been fully retired, despite a well-documented and easily exploitable structural flaw. An attacker who cannot crack a strong WPA2 passphrase does not need to—if the router is simultaneously advertising an 8-digit backdoor.

3.4 Vulnerabilities in Hybrid Systems & USB Interfaces

Network-level defenses only protect the network interface. When devices expose additional physical interfaces—and many do—those interfaces become independent attack surfaces that network security controls cannot address.

3.4.1 BadUSB Attacks and Keystroke Injection

A significant number of IoT hubs, smart televisions, and connected workstations include exposed USB ports—sometimes by design, sometimes simply because no one removed them. A **BadUSB** attack begins with a malicious microcontroller being inserted into one of these ports [28]. It does not present itself as a storage device. Instead, it identifies itself to the host operating system as a Human Interface Device—a keyboard. Operating systems trust keyboards implicitly; they accept input from them without security prompts, without pairing confirmations, without hesitation of any kind. The malicious device then executes a pre-programmed keystroke sequence at speeds no human could replicate. Within seconds, it can open a terminal, modify system configuration, and retrieve external malware. The system is compromised from the inside, through an interface that was never considered part of the threat model.

The root cause is architectural. The USB specification has no universal mechanism for cryptographic hardware attestation — the host machine cannot verify whether a connected device is a genuine keyboard or a microcontroller programmed to impersonate one [29]. Physical connection is treated as sufficient authorization. It is not. This is not a vulnerability that emerged from a coding error or a misconfiguration. It is a structural gap in the standard itself, and it has existed since USB became the dominant peripheral interface [28], and defending against it requires mechanisms that inspect or constrain device behaviour rather than trusting the connection itself [30].

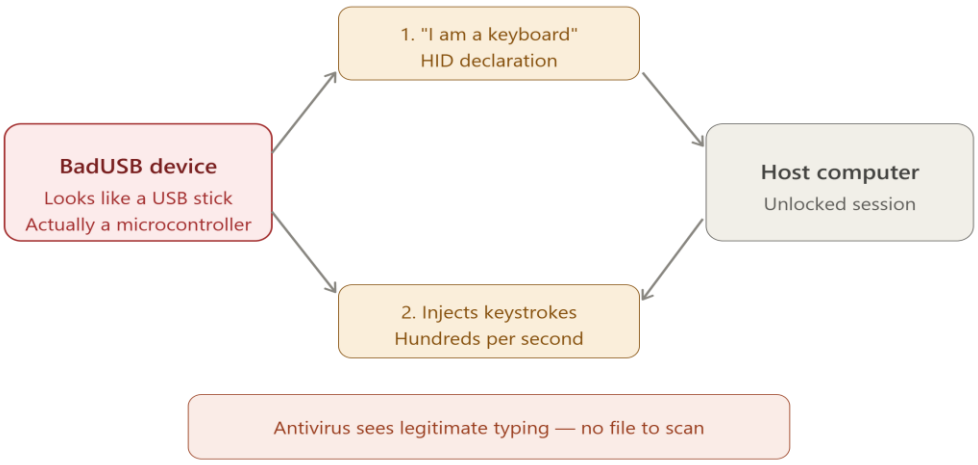

Figure 3.2: BadUSB device impersonates a keyboard. The host trusts the HID declaration and accepts injected keystrokes, which antivirus cannot flag as malicious.

3.4.2 Exploitation of Legacy Protocols

The architectural reality of hybrid smart appliances—discussed in the previous chapter—has a direct parallel in the threat landscape. An IR transceiver in the hands of an attacker becomes an unauthenticated command interface to any device that exposes an IR receiver. By systematically transmitting known device control codes, an attacker can issue commands to an appliance—changing its settings, disabling it, or cycling its power state—without interacting with the network at any point. The Wi-Fi encryption is irrelevant. The primary security infrastructure is bypassed entirely, because IR was never designed with adversarial conditions in mind [11].

The decision to retain an IR receiver on a network-connected appliance is rarely made by a security team. It is made by a product team trying to avoid customer complaints about compatibility.

That is understandable. What is less understandable is the absence of any compensating control — no authentication, no command filtering, no logging. The network interface is hardened; the IR interface is open. An attacker does not need to choose between them. They will simply use whichever one offers less resistance.

3.5 Analysis of Penetration Testing Tools (Hardware & Software)

The gap between a documented vulnerability and a practical, field-deployable exploit has narrowed significantly over the past decade. Hardware that once required substantial financial investment and specialized expertise is now portable, affordable, and often open-source. This democratization has accelerated security research considerably—and it has lowered the barrier to misuse by the same measure.

3.5.1 Hardware Instruments

The Flipper Zero — A Multi-Protocol Security Auditing Platform. The Flipper Zero is a compact, open-source, multi-protocol hardware tool designed for interacting with digital systems, wireless protocols, and physical access control mechanisms. Originally conceived as an educational device for security researchers and penetration testers, it has since established itself as a standard instrument within the cybersecurity community — combining capabilities that previously required multiple separate tools into a single, pocket-sized platform [31].

At its hardware core, the Flipper Zero integrates a CC1101 sub-1 GHz transceiver operating across the 300–348 MHz, 387–464 MHz, and 779–928 MHz bands, with a maximum transmission range of approximately 50 meters. Beyond Sub-GHz, the device incorporates a dedicated 125 kHz RFID module driven by the STM32WB55 microcontroller, a 13.56 MHz NFC module based on the ST25R3916 chip, a built-in infrared transceiver capable of both learning and replaying IR signals, Bluetooth Low Energy connectivity, a 1-Wire iButton interface, and an exposed GPIO header enabling connection to external hardware modules [3]. All captured signals, payloads, and keys are stored on a microSD card, enabling persistent logging across sessions.

The device ships with an official stock firmware. For the purposes of this research, the stock firmware was replaced with Momentum — a community-developed, open-source firmware fork that removes artificial geographical frequency restrictions, expands protocol support, and provides access to advanced security research tooling not present in the official build. This modification is a prerequisite for conducting the full range of experiments described in this study.

For 802.11 Wi-Fi auditing, the Flipper Zero is paired with the official Wi-Fi Development Board — an ESP32-S2 based module that connects directly to the device's GPIO header. The board runs Marauder v1.9.0, a firmware suite developed by justcallmekoko [32] that provides a structured interface for passive Wi-Fi reconnaissance, targeted deauthentication, handshake capture, and Evil Portal deployment. Together, the Flipper Zero and the Marauder-equipped Dev Board constitute a self-contained, battery-powered auditing platform capable of executing cross-protocol attacks across Sub-GHz, RFID, NFC, IR, BLE, and Wi-Fi — all from a device that fits in a jacket pocket.

The selection of the Flipper Zero as the primary auditing instrument over alternatives such as the Proxmark3 or standalone SDR platforms was deliberate. Its unified multi-protocol architecture allows a single operator to transition seamlessly between Sub-GHz, RFID, NFC, IR, and BLE assessments without reconfiguring hardware — a capability that more accurately reflects the operational reality of a covert proximity attacker. For situations requiring broader frequency spectrum coverage beyond the CC1101's operating range, Software Defined Radios such as the HackRF One serve as the comparative alternative, spanning 1 MHz to 6 GHz. The trade-off is portability and

operational simplicity — the Flipper Zero is purpose-built for field use, while an SDR setup is better suited to a controlled laboratory environment.

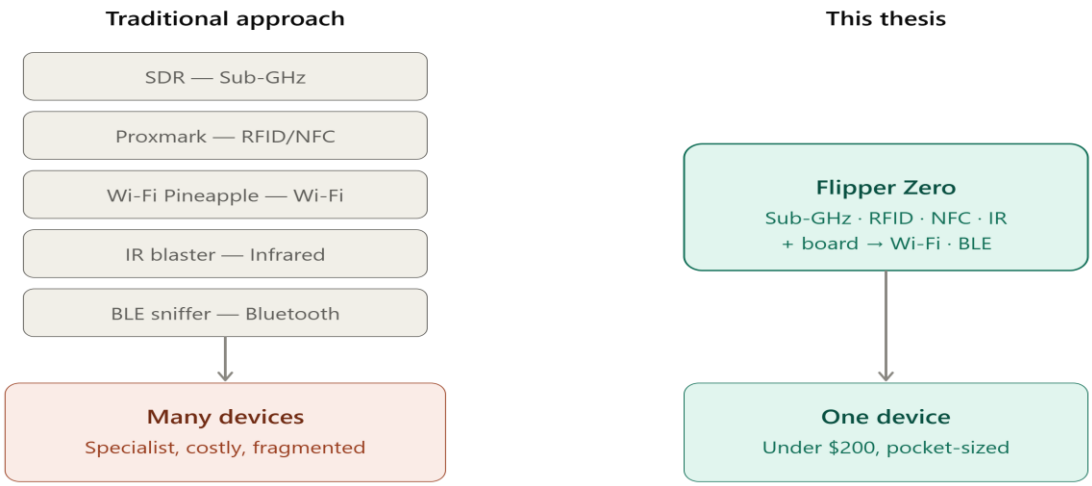

Figure 3.3: Consolidation of specialist tools into a single multi-tool. Where each protocol traditionally required a dedicated instrument — an SDR for Sub-GHz, a Proxmark for RFID/NFC — the Flipper Zero covers them in one sub-$200 device.

3.5.2 Software and Firmware Ecosystems

Hardware is only as useful as the software running on it. The penetration testing community has invested heavily in custom firmware and analytical tooling to support the attack workflows described in this chapter.

The Marauder firmware suite, designed for ESP32-based boards, provides an organized interface for passive Wi-Fi reconnaissance, active deauthentication, and Wi-Fi packet capture [32]. It is what transforms a generic development board into a focused auditing instrument. On the analytical side, Wireshark remains the standard tool for deep packet inspection of 802.11 traffic—its protocol dissectors and filtering capabilities making it well-suited for examining captured management and data frames. For Sub-GHz signal analysis, Universal Radio Hacker (URH) allows researchers to record, demodulate, and interpret unknown radio transmissions, often revealing encoding schemes and payload structures that would otherwise require manual bit-level analysis. Finally, for offline cryptographic work, Hashcat applies GPU acceleration to captured WPA handshake hashes, operating at speeds that render many real-world passwords recoverable within a practical timeframe.

4. Penetration Testing Methodology

Theoretical vulnerability analysis has limits. It can establish that a flaw exists, but it cannot reliably establish whether that flaw is exploitable in practice, under real physical conditions, against actual hardware. Empirical evaluation fills that gap. This chapter defines the procedural framework used to assess the devices in this study—its ethical boundaries, its structure, and the reasoning behind each phase.

4.1 The Framework of Ethical Hacking in IoT

Security research occupies an uncomfortable position. The tools and techniques used to identify weaknesses in a system are, functionally, the same ones used to exploit it. What separates a penetration test from an attack is not the method. It is authorization and intent.

4.1.1 Definition and Necessity of Penetration Testing

In the context of IoT, penetration testing is not a matter of running an automated scanner against an IP address. It requires physical proximity, direct hardware interaction, and in some cases active signal interference. That requirement is not incidental—it reflects the nature of the Perception Layer itself.

Software emulators cannot replicate the precise timing constraints of a rolling code receiver. They cannot reproduce the radio frequency noise floor of a specific physical environment, or the exact signal propagation characteristics of a concrete wall between a transmitter and a receiver. To evaluate these systems with any accuracy, a researcher must work with the actual hardware, using the same methods an adversary would use in the field. Only then does it become possible to determine whether a documented theoretical flaw translates into a tangible, reproducible threat—or whether it remains a laboratory abstraction [33].

4.1.2 Ethical and Legal Constraints

All experiments in this research were conducted in strict accordance with the principles of ethical hacking. The most fundamental of these is permission. Every device tested was either owned by the researcher or subject to explicit, documented authorization from the equipment owner. No exceptions [34].

Beyond authorization, the methodology is governed by a principle of minimal disruption. Active jamming and deauthentication attacks, by their nature, degrade radio

performance and interrupt device connectivity. Applied carelessly, they can affect systems well beyond the intended target—neighboring devices, shared infrastructure, third-party networks. To prevent any such collateral impact, all Sub-GHz, RFID, and Wi-Fi assessments were conducted within a controlled, private indoor environment. Rather than relying on industrial electromagnetic shielding—which falls outside the typical scope of an academic testbed—containment was achieved through strict operational constraints. All Wi-Fi and Bluetooth assessments were explicitly configured to target only the specific MAC addresses and SSIDs of the owned test devices, preventing broadcast interference with third-party networks. The temporal scope of active exploits, such as deauthentication or jamming, was restricted to brief intervals—typically under ten seconds—sufficient to verify the vulnerability without causing prolonged denial-of-service conditions. Finally, experiments leveraged the natural signal attenuation provided by structural walls to ensure that residual RF emissions did not extend beyond the immediate physical perimeter. This localized approach ensures compliance with the legal frameworks governing unauthorized access to computer systems and telecommunications interference, while preserving the privacy and operational continuity of any infrastructure outside the research boundary.

4.2 Vulnerability Assessment Methodology (The 4 Phases)

An unstructured approach to security testing produces unreliable results. Without repeatability, observations cannot be verified. Without a consistent sequence, it becomes impossible to determine whether an outcome was caused by the attack or by some confounding variable in the environment. To ensure that every experiment in this study is reproducible, measurable, and scientifically defensible, the research adopts a formalized four-phase assessment model derived from industry-standard penetration testing guidelines [33]. Every practical case study in the following chapter adheres to this sequence without exception.

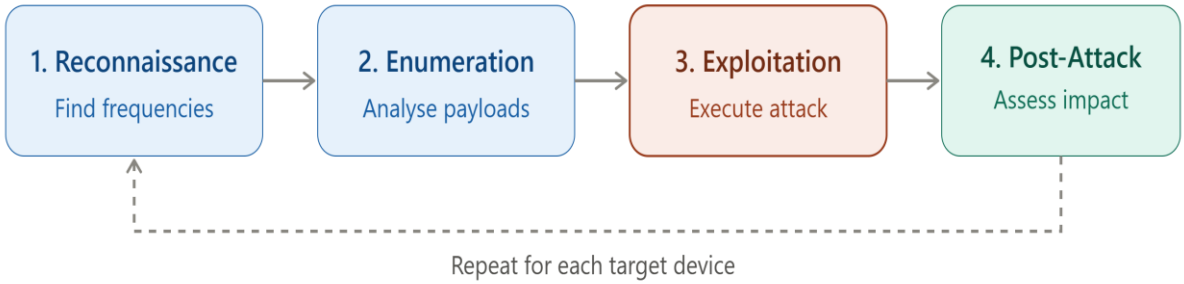

Figure 4.1: The four-phase penetration testing cycle applied to each target device.

4.2.1 Phase 1: Reconnaissance and Scanning

The evaluation begins without transmitting a single signal. Before any active interaction takes place, the environment must be mapped—passively, carefully, and completely.

For Sub-GHz systems, this means monitoring common carrier frequencies such as 433.92 MHz using a spectrum analyzer to detect active transmissions without alerting the target. For BLE and Wi-Fi, it means listening on advertising channels and observing beacon frames to identify device MAC addresses, broadcasted SSIDs, and the encryption standards in use. The Flipper Zero serves as the primary instrument throughout this phase, operating in a receive-only mode. The objective is straightforward: confirm the target is present, determine its communication medium, and establish the parameters of the signal environment before any further action is taken.

4.2.2 Phase 2: Enumeration and Analysis

Detection alone is not enough. Once a signal is identified, it must be understood.

Enumeration is the process of taking an intercepted transmission and extracting everything useful from it. For a captured radio signal, this means analyzing the waveform to determine the modulation scheme—whether the device is using Amplitude Shift Keying, Frequency Shift Keying, or another encoding method. For a Wi-Fi management frame, it means inspecting packet headers to determine whether Protected Management Frames are enforced, or whether the device is vulnerable to spoofed disconnection commands. For an RFID tag, it means reading publicly accessible memory sectors to

identify the chipset—MIFARE Classic versus DESFire, for instance—which determines what cryptographic attacks, if any, are applicable.

This phase is analytical rather than active. Its output is a specific, informed attack strategy, not a generic one.

4.2.3 Phase 3: Exploitation

This is the active phase. Based on the intelligence gathered during enumeration, the selected attack vector is executed.

What that looks like depends entirely on the target. Against a fixed-code system, it means retransmitting a captured waveform and observing whether the receiver accepts it. Against a smart card, it means initiating a cryptographic attack sequence to extract sector keys from a MIFARE Classic chip. Against a Wi-Fi-connected camera, it means transmitting forged deauthentication frames and confirming that the device disconnects. Each exploitation attempt is targeted and deliberate—executing the minimum sequence of actions necessary to test the identified vulnerability, nothing more.

4.2.4 Phase 4: Post-Attack Analysis

An exploit that is not evaluated for impact is only half a finding. The final phase asks the questions that matter operationally: what actually happened, and how bad is it?

Did the garage door open? Did the IoT sensor lose its network connection permanently, or did it reconnect automatically within seconds? Did the access control system log anything that would alert an administrator? The answers determine the practical severity of the vulnerability—distinguishing a minor, recoverable disruption from a complete failure of the security boundary. These severity assessments feed directly into the countermeasures proposed later in the study, ensuring that defensive recommendations are proportional to the actual risk each vulnerability represents.

4.3 Testbed Setup Design

A methodology is only as reliable as the environment in which it runs. Poorly defined experimental conditions introduce variables that cannot be controlled, and results that cannot be replicated carry little scientific weight. This section details the specific target devices selected for evaluation and the precise configuration of the offensive hardware used against them.

4.3.1 Selection of Target Devices

The hardware selected for this testbed was chosen on two criteria: commercial availability and accurate protocol representation. The objective is not to audit exotic or end-of-life equipment—it is to evaluate the systems that ordinary consumers and small businesses are actively deploying today. Accordingly, the testbed covers four distinct categories of IoT infrastructure.

Sub-GHz Infrastructure. A commercial fixed-code gate/garage automation remote operating in the 433 MHz band, and a rolling-code smart-home shutter system (Somfy RTS) implementing a code-hopping algorithm. These two devices represent opposite ends of the Sub-GHz security spectrum—one entirely static, one employing rolling-code encryption.

Access Control Systems. A standard 125 kHz low-frequency proximity badge reader and a 13.56 MHz high-frequency MIFARE Classic smart card terminal. Together, they allow for a direct comparison between an unencrypted LF deployment and an HF system with cryptographic protections.

Wi-Fi and BLE Endpoints. Consumer Wi-Fi access points and a range extender representing typical small-network infrastructure, together with standard consumer smartphones and a workstation serving as Bluetooth and BLE-facing targets.

Hybrid Smart Appliances (IR). A standard network-connected consumer television retaining a legacy Infrared (IR) interface. This device is included specifically to evaluate whether network-level security can be bypassed entirely through an unauthenticated physical channel—a scenario the theoretical analysis in Chapter 2 identifies as a systemic architectural risk.

All equipment was used in its unmodified, factory-default state. This is a deliberate choice. Testing hardened or custom-configured hardware would produce results with limited generalizability; testing devices as they are shipped reflects the conditions an attacker actually encounters [31].

4.3.2 Attacker Setup Configuration

The offensive toolchain used during the active proximity phase of this research deliberately avoids the laptop-centric, high-visibility setup typical of conventional

penetration testing engagements. The goal is to simulate a realistic covert proximity attack—the kind an adversary could execute in public without drawing attention.

The primary instrument is the Flipper Zero [31]. For this research, the device runs a community-developed custom firmware—either Momentum or Unleashed—rather than the manufacturer's stock build. This is not optional. The stock firmware enforces artificial geographical frequency restrictions and omits several protocol capabilities required for the experiments described in this study. The custom firmware removes those constraints and enables the advanced signal analysis and transmission logic the methodology demands.

For 802.11 auditing, the Flipper Zero is paired with a compatible Wi-Fi Development Board via its GPIO pin header. The board, based on an ESP32-S2 module, runs the Marauder firmware suite [32]. This combination—a Flipper Zero handling Sub-GHz, RFID, NFC, and IR, and a Marauder-equipped board handling Wi-Fi packet operations—constitutes a self-contained, battery-powered auditing platform that fits in a jacket pocket.

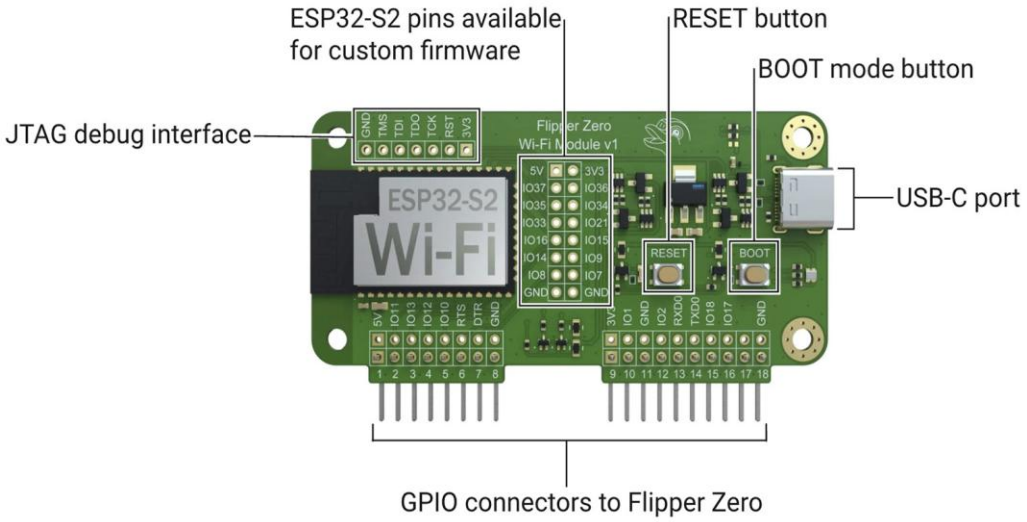

Figure 4.2: The Flipper Zero Wi-Fi Development Board (ESP32-S2), showing the GPIO connectors, ESP32-S2 module, and programming interfaces. [3]

Post-capture cryptographic operations are handled separately. Captured WPA2 handshake files are transferred to a laboratory workstation running Kali Linux, where GPU-accelerated tools such as Hashcat perform the offline password recovery phase. This separation is not a workaround—it is an accurate reflection of how modern adversaries operate. The field phase prioritizes concealment and speed. The cracking

phase prioritizes computational power. Combining both into a single, portable toolchain would compromise the realism of each.

4.4 Risk Assessment Framework

Not every successful exploit carries the same practical weight. A vulnerability may be real and demonstrable in controlled conditions yet largely irrelevant as an operational threat—because it takes too long, requires too much specialist knowledge, or demands equipment that is prohibitively expensive. Conversely, a flaw with modest theoretical impact can represent a critical real-world risk if it can be exploited in two seconds by anyone with a twenty-dollar tool.

To account for this distinction, each attack demonstrated in the following chapter is evaluated against a tailored risk assessment matrix rather than a purely theoretical severity scale.

4.4.1 Evaluation Criteria

The framework assesses threat level across three practical axes.

Probability of Success. How reliably can the attack be executed under realistic field conditions—accounting for environmental radio interference, physical proximity constraints, and the likelihood of triggering any monitoring or alarm system?

Required Time. Is this an attack that completes in a few seconds of passive proximity, or does it require sustained physical presence followed by hours of offline computation? Time directly determines operational exposure: the longer an attacker must remain near a target, the higher the risk of detection.

Cost and Skill Barrier. Can the exploit be automated using commercially available hardware and require no cryptographic background to execute? Or does it demand expensive Software Defined Radio equipment and deep mathematical expertise? This axis reflects the realistic population of potential adversaries—a low barrier means a broader threat.

Traditional vulnerability scoring frameworks, such as the Common Vulnerability Scoring System (CVSS) [35], frequently struggle to quantify the physical reality of IoT threats. CVSS was fundamentally designed for traditional IT networks, meaning it often underestimates vulnerabilities that require adjacent physical access, while failing to

accurately capture the safety and operational consequences of a compromised physical actuator. Applying this tailored framework produces classifications that diverge meaningfully from purely impact-based scoring. A basic RFID cloning attack, completable in under two seconds with a Flipper Zero and no technical knowledge, rates as a critical immediate risk regardless of its theoretical sophistication—or lack thereof. A WPA2 handshake capture, despite its potentially severe consequences, rates lower on the immediate operational scale because recovering the password requires significant offline GPU processing time and is not guaranteed to succeed. This distinction matters when deciding where to direct defensive effort first, and it shapes the prioritization of the countermeasures proposed at the conclusion of the study.

5. Empirical Experiments and Practical Vulnerability Assessment

Theoretical frameworks define the structural limits of IoT and wireless protocols. They cannot, on their own, show how a latent weakness behaves once it is embedded in a product someone actually bought and installed. That is the gap this chapter addresses. What follows is a sequence of controlled laboratory and field experiments, each one targeting a different communication layer — Sub-GHz, RFID, NFC, Wi-Fi, and Bluetooth Low Energy.

The devices examined here were chosen deliberately. Some are modern, network-connected endpoints. Others are simple radio-frequency systems with no networking stack of their own. The second category deserves a brief justification. A garage gate remote is not, in the strictest sense, an IoT device — it holds no IP address and joins no network on its own. It does, however, operate on the same Sub-GHz perception-layer protocols that underpin a large share of commercial IoT peripherals, and it fails in the same ways. Several of these systems are now sold with optional connectivity modules that pull them squarely into the IoT category, a point examined in detail below. Treating them as part of the perception layer, rather than excluding them on a technicality, gives a more honest picture of the threat surface a real attacker would meet.

Each case study records the same operational variables: operating frequency, modulation, captured payload, execution time, distance, and the final outcome. Where an attack failed, that failure is reported as a finding in its own right.

5.1 Case Study 1: Fixed-Code Exploitation in Sub-GHz Frequencies (Gate Automation)

Automated gate systems sit at the boundary between physical and digital security. They are everywhere — residential driveways, apartment complexes, commercial yards — and most of them rely on a small handheld remote transmitting in the Sub-GHz band. This case study examines one such system to determine whether it implements any form of origin authentication, or whether possession of a single captured signal is enough to defeat it.

5.1.1 Radio Frequency Reconnaissance

The first task was to find the exact frequency and modulation the gate's remote used. Nothing about this stage is invasive. Using the frequency analyzer on the Flipper

Zero, the local Sub-GHz spectrum was monitored while the legitimate remote was pressed during an authorized opening.

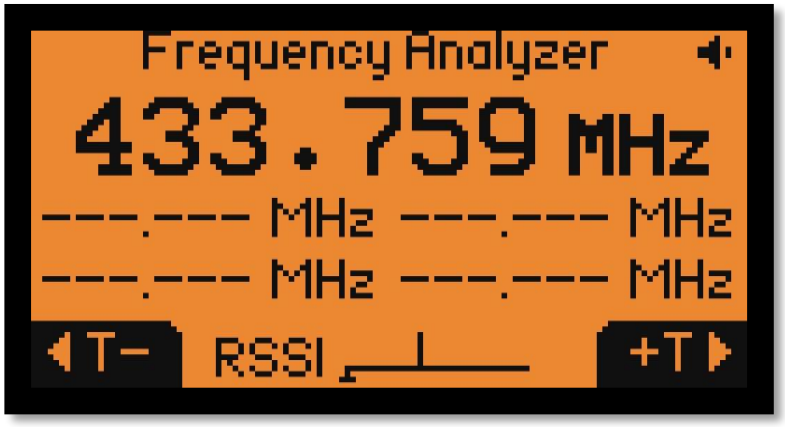

Figure 5.1: Passive frequency analysis locating the target signal in the 433–434 MHz band.

A clear, repeatable peak appeared in the 433–434 MHz region. The frequency analyzer is a coarse-resolution instrument — repeated sweeps of the same transmission returned values drifting across roughly 433.76 MHz to 434.39 MHz — so it serves to locate the signal's neighborhood rather than to pin it exactly. The precise operating frequency, 433.88 MHz, was established at the next stage, when the signal was decoded and the tool locked onto it directly. The shape of the envelope pointed to amplitude modulation, specifically On-Off Keying (OOK / ASK).

5.1.2 Signal Capture and Protocol Decoding

With the physical-layer parameters known, the platform was switched to passive read mode, tuned to 433.88 MHz with AM. The capture window was busy. Alongside the target, the receiver picked up unrelated chatter from nearby peripherals, including Holtek-based sensors broadcasting in the same band. The target's payload was isolated from that background without much difficulty.

Figure 5.2: Demodulated Sub-GHz packet log isolating the target CAME 12-bit sequence from Holtek background noise.

The decoder identified the transmission as a CAME 12-bit protocol sequence — one of the oldest and most common fixed-code schemes still in service. To test whether the code varied between presses, the legitimate remote was triggered several times in succession and each transmission was captured.

Figure 5.3: Bit-level breakdown of the decoded CAME 12-bit payload, showing a static identifier value.

The result was unambiguous. The payload did not change. Every press produced the same fixed value: 0x00000549, with a bit-reversed validation string of 0x0000092A. There is no rolling code here, no nonce, no timestamp, no handshake of any kind. The receiver decides whether to open based on one thing only — whether it sees that 12-bit string. Possession of the code is identity. That is the entire authentication model, and it is the vulnerability.

5.1.3 Replay Attack and Physical-Layer Denial of Service

To confirm the system was genuinely exploitable, the legitimate remote was removed from the area entirely, ruling out any chance of an authorized signal being present. The Flipper Zero was then instructed to retransmit the captured string, 0x00000549. The gate's relay clicked and the gate opened. No authorized transmitter was involved. Any attacker who can capture a single press — a few seconds of proximity — gains repeatable physical access for as long as the code stays unchanged.

A second, more disruptive test followed. Rather than decode and replay a clean token, the platform captured the transmission as raw waveform data — every pulse and RSSI transition across a window of 4,705 samples — bypassing the CAME decoder entirely.

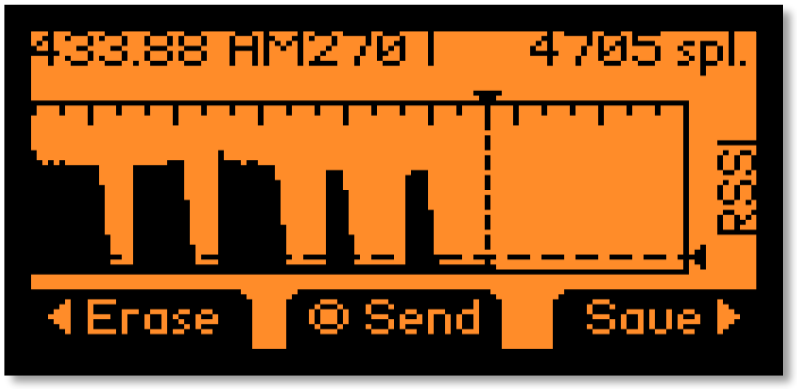

Figure 5.4: RAW Sub-GHz capture recording physical-layer pulse transitions across 4,705 samples.

Replaying that raw data in a continuous, unthrottled loop turns the device into a localized jammer on 433.88 MHz. The receiver's front end is flooded with energy.

Figure 5.5: Continuous RAW transmission loop (434.77 MHz, 3765 samples) producing a localized physical-layer denial of service.

While the loop ran, the owner's legitimate remote stopped working. Every command it sent was lost in the noise. The receiver could no longer pick out the preamble or sync words of a valid frame, and the gate stayed unresponsive for as long as the transmission continued — a clean physical-layer denial of service that required no knowledge of the protocol at all.

5.1.4 Findings

Case Study 1 shows that a legacy fixed-code system like CAME 12-bit offers essentially no resistance to a low-cost multi-tool. A single captured press grants persistent, non-destructive entry through a straightforward replay. The same hardware, used differently, locks legitimate users out through continuous-carrier jamming. Two distinct attacks — unauthorized access and denial of service — both trace back to the same root cause: the protocol carries no origin authentication and no cryptographic freshness whatsoever. [36]

The IoT dimension is worth drawing out, because it is easy to dismiss a gate remote as too simple to matter. Systems of this class are increasingly sold with optional Wi-Fi modules that add smartphone control, cloud access, and smart-home integration. Once such a module is fitted, the gate becomes a fully networked IoT device in every meaningful sense. The connectivity does nothing to fix the problem underneath it. The legacy 433 MHz channel stays active and unauthenticated beneath the new smart interface, which means an attacker can ignore the entire networked control plane — and whatever encryption guards it — and go straight for the radio layer, exactly as shown here. Making the device smart does not make it secure. It stacks a new attack surface on top of an old one that was never resolved.

That distinction matters for how this risk is rated. The attack needs only seconds of proximity, costs almost nothing in equipment, and requires no cryptographic skill — placing it firmly in the **critical category** under the framework set out in Chapter 4. It points directly toward the rolling-code and mutual-authentication schemes examined in the next case study, in a system that does at least attempt them [37].

5.2 Case Study 2: Rolling Code Desynchronization and Denial of Service in Smart Home Actuators

Case Study 1 dealt with a protocol that made no attempt at security. This one is different. Modern high-value systems rarely transmit a static code anymore — they use rolling codes, where every press produces a fresh, single-use payload that should be worthless the moment it is transmitted. The question here is not whether rolling codes work in principle. They do. The question is whether a specific, widely deployed implementation closes the door completely, or merely narrows it. The target is a set of motorized shutters running the Somfy Radio Technology (RTS) protocol — a genuine smart-home actuator, compatible with Somfy's TaHoma and myLink hubs, which places it firmly in the hybrid IoT category.

5.2.1 Frequency Reconnaissance

As before, the first step was passive. The local spectrum was monitored while the legitimate Somfy Telis remote was operated.

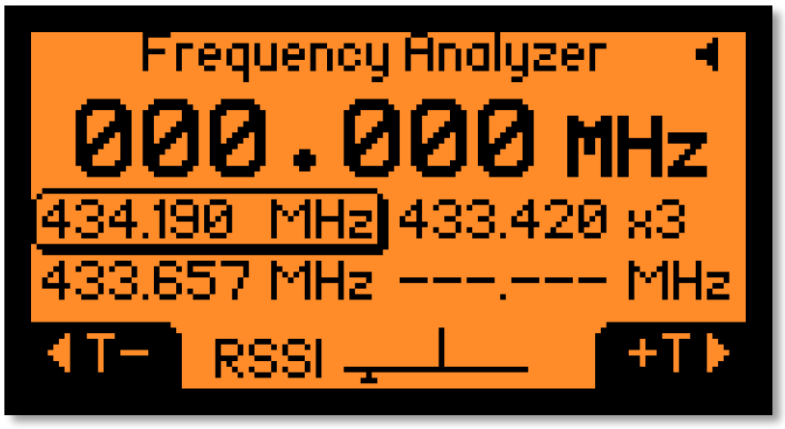

Figure 5.6: Frequency analyzer identifying the Somfy RTS transmission footprint, centered at 433.42 MHz.

The transmission sat at 433.42 MHz, using amplitude modulation (AM/OOK). The frequency itself is informative. Where the gate in Case Study 1 used the generic 433.92 MHz, Somfy operates a few hundred kilohertz lower, at 433.42 MHz — a choice tied to its own ecosystem and an early hint of the proprietary protocol underneath.

5.2.2 Protocol Identification and Counter Analysis

Several consecutive presses were captured from the legitimate remote. The difference from CS1 was immediate: no two payloads matched.

Figure 5.7: Passive interception of multiple Somfy Telis transmissions, each carrying a different payload.

Figure 5.8: Further captured sequences confirming the changing payload between presses.

The Flipper Zero identified the protocol as Somfy RTS (reported on-screen against the Telis remote model). Examining a single frame in detail exposed its structure.

Figure 5.9: Breakdown of a Somfy RTS frame, showing the encryption key, the sender Serial Number (Sn: 0x8BC82A), and the rolling Counter (Cnt: 2131).

The frame carries three things that matter: a fixed Serial Number identifying the remote (here, 0x8BC82A), a button code (Btn: My), and a rolling counter (Cnt: 2131). The counter is the security mechanism. Published reverse-engineering of the RTS protocol describes this counter as only 16 bits wide, obfuscated rather than strongly encrypted, with the frame structure itself documented in detail [38]. Each press increments it; the receiver tracks the last value it accepted and is supposed to reject anything it has already seen.

That promise held up under direct test. Re-transmitting the captured frame verbatim — counter still at 2131 — did nothing. The receiver had already advanced past that value and discarded the stale frame. A naive replay, the attack that defeated CS1 outright, fails here. So far, the rolling code does its job.

5.2.3 Exploiting the Synchronization Window

The weakness is not in the counter. It is in how forgiving the receiver is about which counter values it will accept [16]. Rolling-code receivers face a practical problem: if the owner presses the remote a few times out of range, the transmitter's counter races ahead of the receiver's. To avoid locking the legitimate user out, the receiver accepts not just the next expected value but any value within a forward window ahead of it. This behavior is not a Somfy quirk — it is described in the foundational rolling-code patents, which specify a receiver that recognizes "a plurality of rolling codes within a certain window" rather than a single value [39]. Reverse-engineering work on RTS specifically notes that with a window of roughly 100, a 16-bit counter collapses to an effective 9–10 bits of real security [38].

That window is the attack surface. Knowing the captured counter (2131), the next step was simply to transmit a frame for the same Serial Number with the counter pushed slightly forward.

Figure 5.10: A forged transmission with the counter advanced to 2137, accepted by the receiver.

A frame with the counter set to 2137 — six steps ahead — was transmitted. The receiver accepted it as legitimate and the shutters moved. The significance is worth stating plainly: no cryptographic key was extracted, nothing was brute-forced over hours. A single sniffed frame yields the Serial Number and a counter reference, and from there any value inside the forward window is a working command. The usability feature is the vulnerability.

5.2.4 Physical-Layer Denial of Service

The final test on the primary system set aside the protocol logic entirely and went after the radio itself. The Somfy waveform was captured as raw samples (1,967 of them), with no decoding.

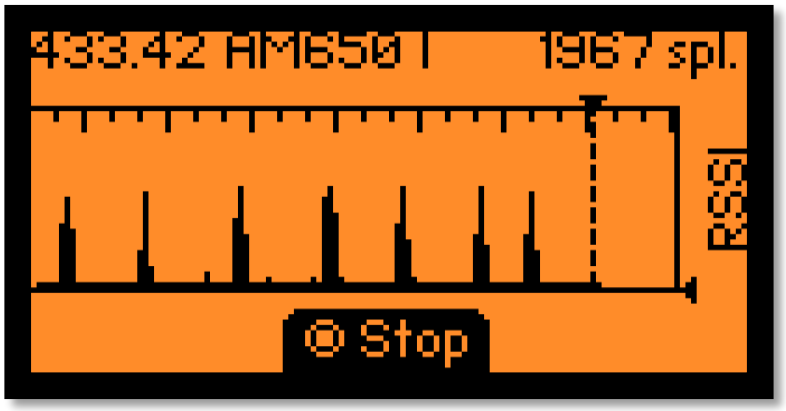

Figure 5.11: RAW capture of the Somfy RTS AM650 waveform (1,967 samples).

That raw capture was then replayed in a continuous loop.

Figure 5.12: Continuous RAW playback producing sustained RF jamming on 433.42 MHz.

With the loop running, the legitimate remote went dead. Nothing it sent reached the shutters. The receiver's front end was saturated by the constant signal and could no longer pick a valid frame's preamble out of the noise. The shutters could not be commanded for as long as the jamming continued.

5.2.5 High-Frequency Evaluation and Attacker Hardware Limitations

A complete picture of the target environment required looking beyond the primary 433.42 MHz system. A second Somfy actuator in the same space operated on a different band entirely, and testing it exposed something the first device could not: the gap between a vulnerability that exists in theory and one an attacker can actually reach with portable hardware.

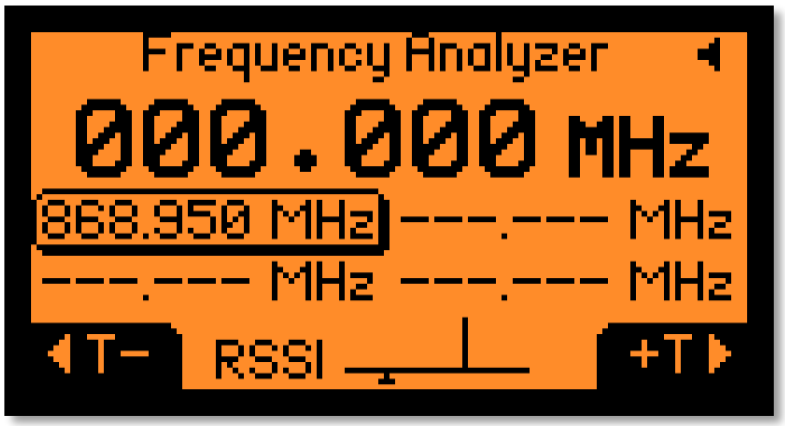

Figure 5.13: Spectrum analysis identifying a secondary Somfy device in the 868 MHz band, at 868.95 MHz.

Passive reconnaissance placed the second device at 868.95 MHz. The reading stage, however, behaved nothing like it had at 433 MHz. The platform could not parse the transmission at all — it stayed locked in a scanning state and never resolved a Serial Number or counter.

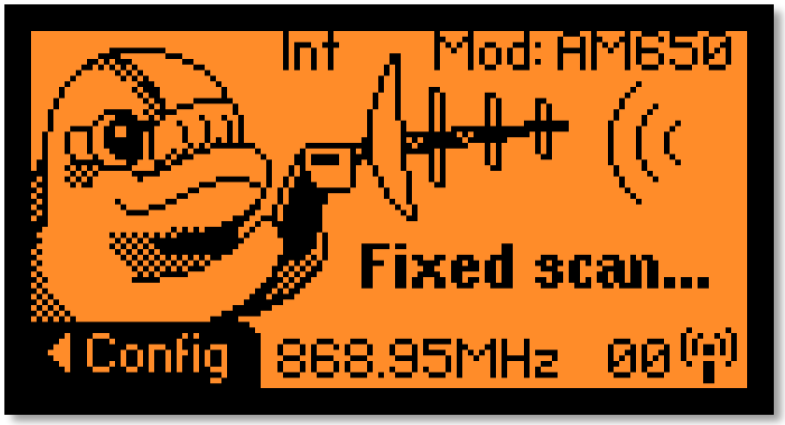

Figure 5.14: The platform failing to decode the 868.95 MHz transmission, remaining in a "Fixed scan…" state.

This is a notable result on its own. At 433.42 MHz the RTS frame decoded cleanly; at 868.95 MHz the same firmware extracted nothing. Somfy's higher-frequency product lines are known to use more modern signaling than the legacy RTS scheme — including two-way, FSK-based protocols with stronger framing — and any of several factors could explain the failure here, from a protocol the firmware does not natively demodulate to reduced receive sensitivity in this band. The honest position is that the protocol was not identified; the platform simply did not decode it, and the reason was not conclusively established.

With decoding off the table, a raw capture was taken to record the waveform directly, sidestepping the parser.

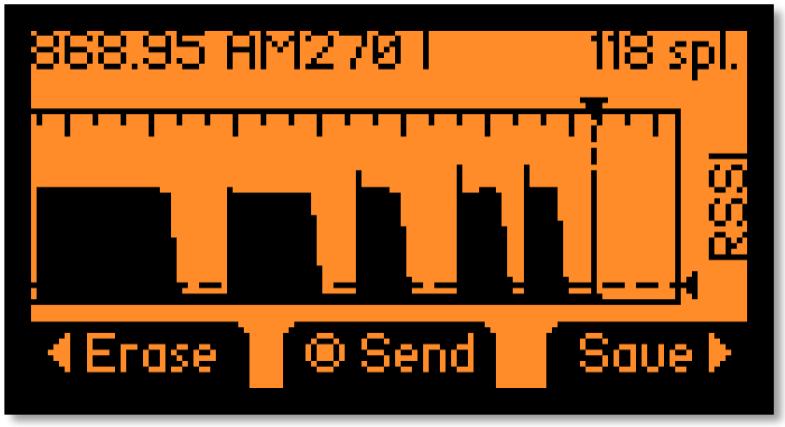

Figure 5.15: RAW analog capture of the 868.95 MHz transmission (118 samples).

The jamming attack that had worked at 433 MHz was then repeated with this raw capture, replayed in a continuous loop.

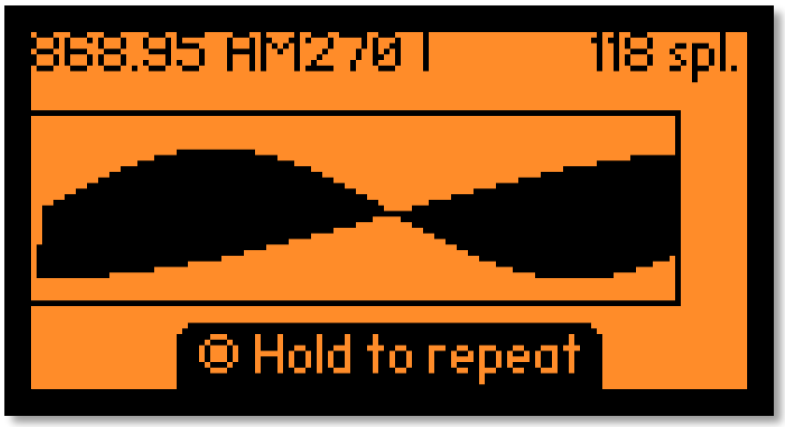

Figure 5.16: Continuous RAW playback at 868.95 MHz, attempting a denial of service.

It failed. The target kept receiving its legitimate commands and operated normally throughout the interference. The important point is what this failure does and does not mean. It is not evidence that the 868 MHz system is immune to jamming. It is evidence of a hardware ceiling in the attacking device. The Flipper Zero's CC1101 transceiver can be programmed for comparable output power across bands, but its integrated antenna and matching network are optimized for the 433 MHz region [40]. At 868 MHz the antenna efficiency falls, and with it the effective isotropic radiated power (EIRP), even when the transceiver itself is driven at a similar output level. The limiting factor is not the chip's rated power but how efficiently that power actually radiates at this frequency. The device simply could not put enough energy into the band to saturate the receiver's front end.

That distinction matters for the threat model. Physical-layer jamming is not a universal capability of a tool like this. It is bounded by where the hardware radiates efficiently, and a target operating outside that sweet spot may be incidentally protected — not by design, but by the attacker's equipment limits.

5.2.6 Findings

The Somfy RTS system tells a more nuanced story than the gate in CS1. Its rolling code is not decorative — it genuinely defeats the straightforward replay attack that opened the gate. But the protection is shallower than it looks. The permissive forward synchronization window lets an attacker who captures a single frame forge a valid future command without touching the cryptography, dropping the effective security well below what the 16-bit counter would suggest [38] **(Risk: High).** The denial-of-service picture is split by frequency: trivial at 433.42 MHz, where the legitimate remote was fully disabled, but unreachable at 868.95 MHz — not because that system is hardened, but because the attacking hardware could not radiate enough power there [40].

As a hybrid IoT actuator, the Somfy system carries the same lesson as the gate before it. A cloud hub and a smartphone app sit on top of this radio link, but they do not protect it. The RTS channel is independently exploitable beneath whatever network security the TaHoma or myLink layer provides — the convenience layer and the vulnerable layer coexist, and hardening one does nothing for the other.

5.3 Case Study 3: Low-Frequency (LF) RFID Access Control Emulation

The earlier case studies stayed in the Sub-GHz spectrum. This one drops lower, to 125 kHz — the Low-Frequency band that a huge share of physical access control still runs on. Office buildings, parking garages, hotels: if entry depends on tapping a badge against a wall reader, there is a good chance it is LF RFID underneath. This experiment takes one such badge and asks a simple question. How hard is it to capture its identity and then become it?

5.3.1 Configuration and Methodology

The target was a generic HID Proximity badge operating at 125 kHz — a non-cryptographic legacy credential that remains common precisely because it is cheap, durable, and "good enough" in the eyes of many facility operators. The goal was a non-destructive identity capture: read the card's identifier, then emulate it toward a reader, without ever altering the original badge or breaking any cipher. There is no cipher to break, which is rather the point.

5.3.2 Non-Contact Identity Harvest

The RFID application was launched on the Flipper Zero, putting it into a passive listening mode tuned to couple with the magnetic field a 125 kHz reader produces.

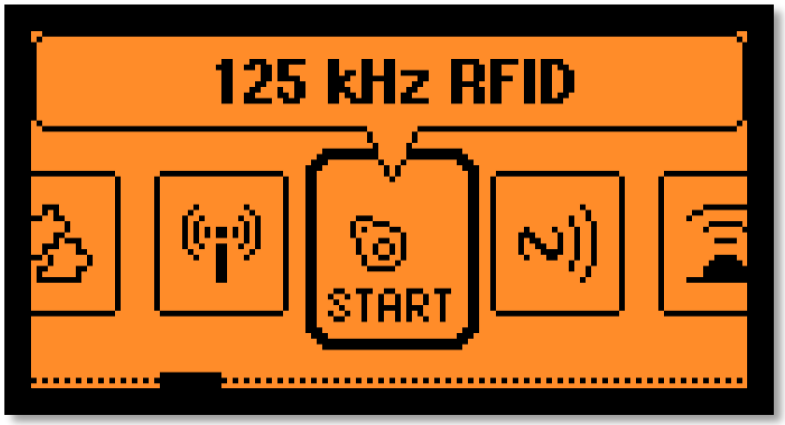

Figure 5.17: Selecting the RFID application from the main menu.

Figure 5.18: Initiating the Read command within the RFID application

Selecting Read energizes the device's internal LF coil and starts it hunting for a card in range. The badge was held near the antenna.

Figure 5.19: The device actively scanning for a Low-Frequency RFID tag.

5.3.3 Protocol Identification and Data Extraction

Coupling was almost instant. Within a fraction of a second the platform had demodulated the badge's signal and resolved its protocol.

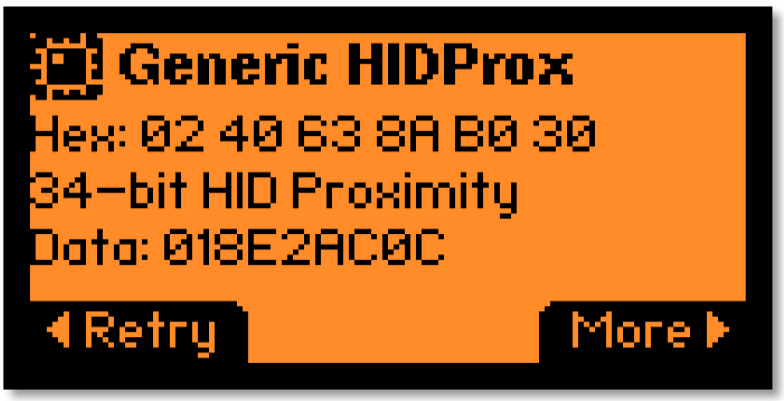

Figure 5.20: Successful capture of the target badge, identified as a Generic HID Proximity (34-bit) credential.

The Flipper Zero identified the badge as a Generic HIDProx card using the 34-bit HID Proximity format. It read out the full hex identifier — 02 40 63 8A B0 30 — along with the decoded data string 018E2AC0C. This is the entire credential. Nothing else is required to reproduce it.

That fact is the finding. The HID Proximity protocol, like most 125 kHz LF schemes, carries no encryption and no authentication of any kind [17]. The exchange is plaintext. When the badge enters the reader's field, it is powered by that field and simply broadcasts its identifier — unconditionally, every time, to anything listening on the right frequency. There is no challenge, no rolling element, no secret. Proximity is the only "credential" the system actually checks, and proximity is trivial to obtain.

5.3.4 Active Identity Assumption (Emulation)

With the identifier captured, it was saved to the device.

Figure 5.21: The captured HIDProx credential stored locally, with the Emulate option available.

The original badge was then set aside entirely, to rule out any chance it was assisting. Holding only the saved credential, the Flipper Zero was presented to the access reader in place of the card.

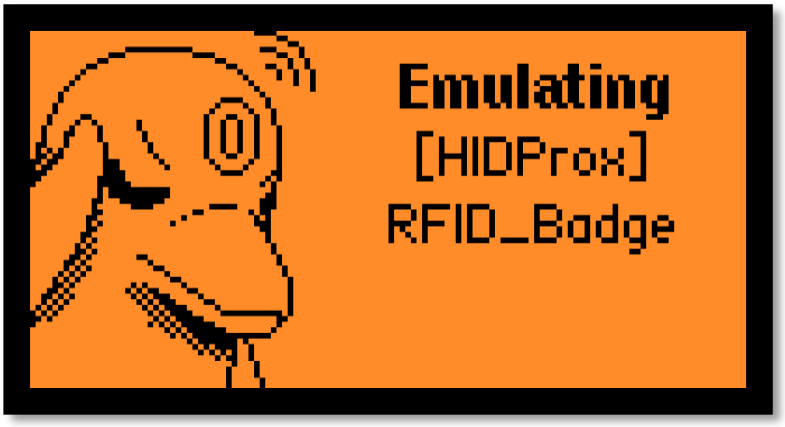

Figure 5.22: The device emulating the captured credential and gaining access — the Flipper Zero now acts as the badge itself.

The reader accepted it without hesitation. The device was not presenting a copy of the badge — for the purposes of the reader, it was the badge, transmitting the same 34-bit identity. Worth underlining: no blank card was written, no token was physically cloned. The Flipper Zero held the identity in memory and replayed it directly. The whole sequence, from read to successful emulation, took under a minute of non-contact interaction.

5.3.5 Findings

A 125 kHz HID Proximity badge offers no meaningful resistance to capture or emulation [20]. The credential is read in under a second from a few centimetres away, stored, and replayed back to a live reader that cannot tell the difference. The absence of cryptographic authentication means the usual security properties simply do not exist here — confidentiality and integrity both collapse, because the identifier is broadcast in the clear and nothing verifies who is replaying it **(Risk: Critical).**

The attacker's shopping list is short. No original badge in hand, no key extraction, no physical tampering, no forced entry. A few seconds of proximity to a legitimate badge — in a lift, a queue, a coat pocket brushed in passing — is enough to harvest a credential that then works indefinitely, until someone reissues the badge. For a technology still guarding doors in commercial buildings worldwide, that is a critical-severity finding, and it is one of the clearest illustrations in this thesis of how "security through obscurity plus low cost" fails completely once a cheap, capable tool enters the picture.

5.4 Case Study 4: High-Frequency (NFC) Exploitation and Cryptographic Bypass

The previous badge had no cipher at all. This one does — and that makes it the more interesting target. Most modern access control has moved up to the High-Frequency band, 13.56 MHz, where NFC and the MIFARE family dominate. Unlike the plaintext LF cards, MIFARE Classic actually encrypts its data and authenticates against the reader, using a proprietary stream cipher called CRYPTO1. On paper, that is a real improvement. This case study looks at whether the improvement survives contact with the real world, where two things tend to go wrong: how the system is configured, and how strong the cipher really is.

5.4.1 Protocol Identification and Dictionary Attack

The Flipper Zero's NFC application was launched and set to poll the 13.56 MHz band.

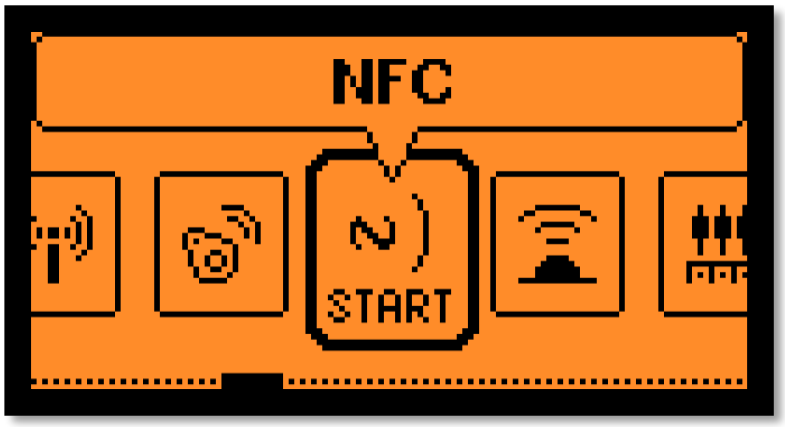

Figure 5.23: Initializing the NFC application for high-frequency analysis.

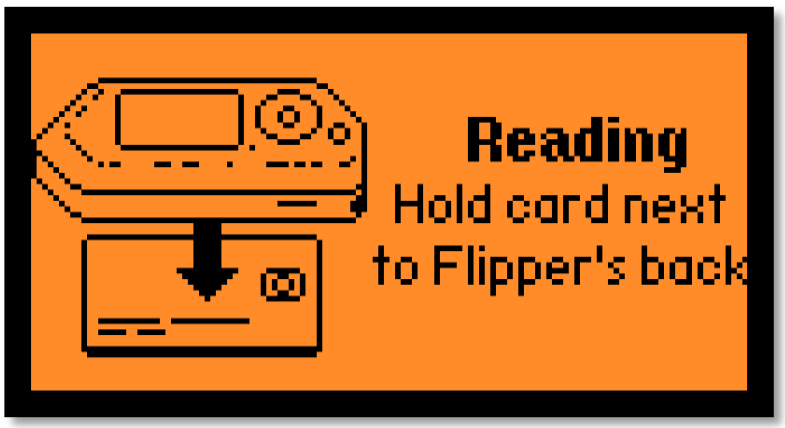

Figure 5.24: The platform polling for 13.56 MHz ISO 14443A targets.

On coupling with the badge, the device identified it as a MIFARE Classic 1K. The card's memory is divided into 16 sectors, and each sector is locked by two separate keys — Key A and Key B — so a full readout requires recovering 32 keys in total. That sounds like a lot of cryptography to defeat. In practice, the first thing worth trying is not cryptography at all.

Before reaching for any mathematical attack, the tool ran a dictionary attack: it simply tried a built-in list of factory-default and commonly leaked keys against every sector.

Figure 5.25: The dictionary attack recovering all 32 sector keys and reading all 16 sectors.

It worked, and it worked in seconds. Every key fell — 32 out of 32 — and all 16 sectors were read in full. The card's UID (20 B7 50 9F) was captured along with the complete contents.

5.4.2 The Configuration Failure

Why did a dictionary attack flatten an encrypted card so quickly? The memory dump, viewed through a companion mobile application, answers that immediately.

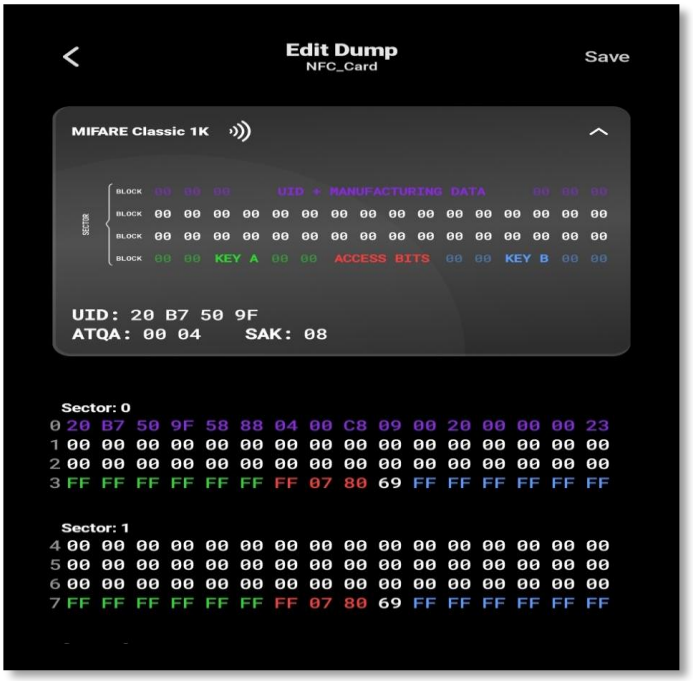

Figure 5.26: Hexadecimal dump of the MIFARE Classic 1K card (Sectors 0 and 1).

The dump shows the UID (20 B7 50 9F), the ATQA (00 04) and SAK (08) values, and — in the sector trailers, where the keys live — the keys themselves, sitting in plain view as FF FF FF FF FF FF.

Those are the manufacturer's default transport keys. They were never changed when the access system was installed. This is the part worth being precise about: CRYPTO1 was not broken here. It was never engaged. A 48-bit stream cipher, whatever its mathematical merits, protects nothing when the key guarding it is the one printed in the datasheet. The card's entire security collapsed not through cryptanalysis but through a provisioning oversight — and that oversight is extraordinarily common in deployed systems.

5.4.3 Full Credential Emulation

With every sector decrypted and the UID in hand, the device held a complete one-to-one digital copy of the badge.

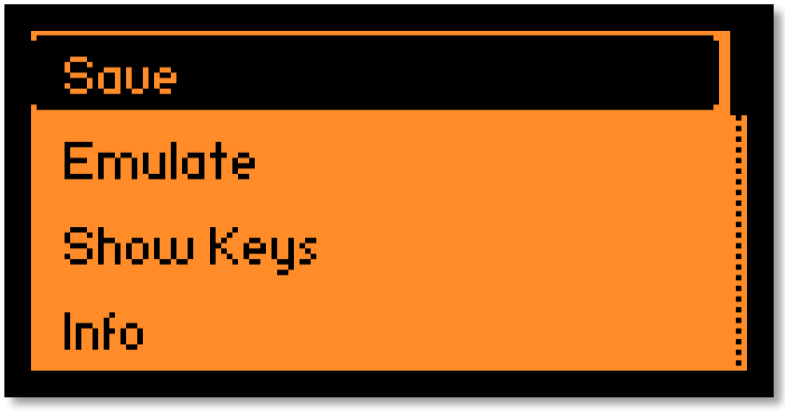

Figure 5.27: Saving the fully decrypted MIFARE payload to local storage.

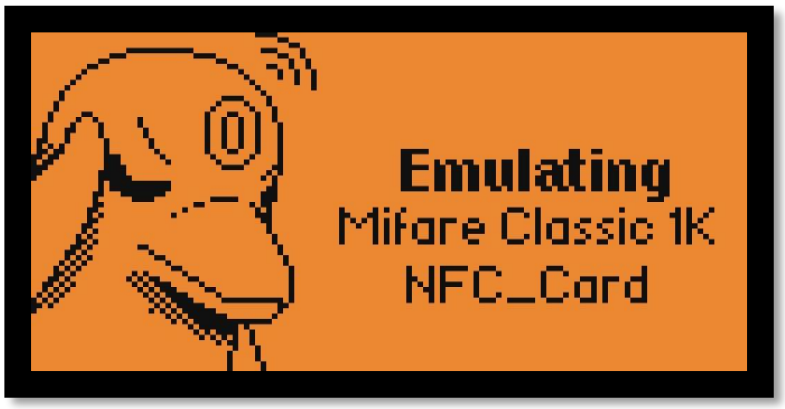

Figure 5.28: The platform emulating the MIFARE Classic 1K identity and clearing the reader's authentication challenge.

The saved profile was emulated against the physical reader. Because the device now held the correct keys, it completed the three-pass mutual authentication that MIFARE mandates — the very step meant to prove a card is genuine — and the reader opened. A total physical compromise (Risk: Critical).

5.4.4 Beyond Default Keys: The Reader Attack (MFkey32)

Relying on default keys is the easy path, and it happens to have worked here. But a thesis should address the harder case too: what if the installer had done their job and changed the keys? The attack does not end there. It pivots — away from the card, toward the reader.

The Flipper Zero supports a reader-side technique built on the MFkey32 method. It is worth stating clearly at the outset what the next two figures show and what they do not. The target card in this experiment used default keys, so the dictionary attack already granted full access and there was no need to attack the reader. The MFkey32 module is

presented here to document the method and the platform's capability, not the results of a live attack — the reader-extraction path was not executed, as the available card did not require it and no separately configured hardened reader was on hand to attack legitimately. Had the card carried changed, non-default keys, this is the route the assessment would have taken.

Figure 5.29: Selecting the key-extraction module — the entry point for a reader-side attack when card-side dictionary attacks fail.

The idea is to turn the problem around. Rather than attack the card, the attacker presents the tool to the wall-mounted reader, emulating a card the reader will then try to authenticate.

Figure 5.30: The key-extraction interface, which in a live attack would harvest authentication nonces from the physical reader.

The mechanism is well understood. When a reader tries to authenticate a card, it sends cryptographic challenges — nonces. The weakness that makes this fatal lies in MIFARE's own pseudo-random number generator: it is cryptographically weak, and this has been documented in the academic literature for over a decade [8]. A couple of

captured nonces are enough to compute the sector keys offline with a tool such as mfkey32. With the keys recovered mathematically, an attacker returns to the genuine card, reads it with the derived keys, and proceeds to emulation exactly as in the previous section.

The implication is the harder-hitting point, and it holds regardless of whether the attack is run in this particular experiment. Even a correctly configured MIFARE Classic system — keys changed, defaults removed — is broken at the protocol level, because the flaw lives in the cipher and its PRNG rather than in any single installation [8], [41]. Subsequent research extended these breaks even to the "hardened" MIFARE Classic variants that NXP introduced specifically to resist them [42]. The default-key failure is a configuration problem that can be fixed; the CRYPTO1 weakness cannot be, short of migrating to a different card technology entirely.

5.4.5 Findings

This case study produced two compromise paths against the same card — one executed, one analyzed. The executed attack relied on a configuration failure, the untouched default keys, and granted instant full access **(Risk: Critical).** The analyzed path, the reader-based MFkey32 attack, needs no such luck: it defeats the protocol itself through the documented weakness in CRYPTO1 and its random number generator, and the literature shows it succeeding even against hardened deployments [8], [41], [42]. It was not run here only because the target card did not call for it and attacking an unrelated reader would have fallen outside the ethical scope set in Chapter 4.

Read together, the two paths bracket the real-world risk neatly. A well-configured system escapes the first but not the second. A poorly configured one — and given how often defaults survive into production, that describes a great many live installations — falls to either. MIFARE Classic was a genuine step up from plaintext LF cards, but it has been comprehensively dismantled in the open literature since 2008, and these results confirm that on commodity hardware the theoretical breaks are now effectively point-and-click. Systems still trusting it for physical security are running on borrowed time.

5.5 Case Study 5: 802.11 WPA2 Exploitation (Deauthentication and Handshake Capture)

The case studies so far stayed close to physical access — gates, badges, cards. This one moves to the layer that actually carries most IoT traffic: the wireless LAN. Nearly every smart device in a home or small office reaches the internet through 802.11 Wi-Fi, and the overwhelming majority of those networks are secured with WPA2 and a pre-shared key. WPA2-PSK is the lock on the front door of the modern connected home. This case study examines two weaknesses in it — one in how the standard handles management traffic, and one in how a captured handshake can be taken offline and cracked at leisure.

5.5.1 Hardware Configuration and Network Reconnaissance

The Flipper Zero has no Wi-Fi radio of its own. To work in the 802.11 band it was paired with a dedicated Wi-Fi Development Board built around an ESP32-S2 microcontroller, flashed with the Marauder penetration-testing firmware.

Figure 5.31: Launching the ESP32 Wi-Fi Marauder module from the platform.

The first step was to map the local airspace. The ESP32 module scanned the 2.4 GHz band to enumerate nearby access points and the clients attached to them.

Figure 5.32: Initiating a scan for local access points.

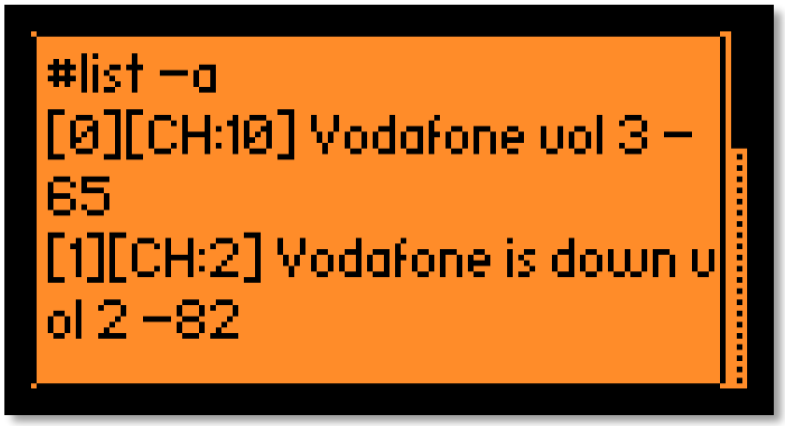

Figure 5.33: The resulting AP list. The target — "Vodafone vol 3" — appears as index 0 on Channel 10 with a strong −65 dBm signal.

The scan returned a short list. The chosen target, a network broadcasting "Vodafone vol 3" on Channel 10, came back at −65 dBm — a strong, close signal, identifying it as a Wi-Fi range extender on the local network. A second extender ("Vodafone is down vol 2") appeared on Channel 2 at a much weaker −82 dBm. The tool was locked onto target index 0, focusing every subsequent radio action on that one access point and its clients.

Figure 5.34: Locking onto the target (select -a 0) to isolate the attack to a single access point.

5.5.2 Active Denial of Service: The Deauthentication Attack

WPA2 carries a structural weakness written into the original 802.11 standard. Data frames are encrypted, but management frames — the control messages that set up and tear down connections — are sent in the clear and are not authenticated. Nothing stops a third radio from impersonating an access point and broadcasting "deauthentication" frames to its clients, ordering them off the network. The clients have no way to tell the forged command from a genuine one. This is not a newly discovered flaw; it was characterized in detail as far back as 2003, when Bellardo and Savage demonstrated that the unauthenticated management plane made practical, low-cost denial-of-service attacks feasible against any 802.11 network [43].

Marauder exposes this through its PMKID sniffing module, set to an active mode that forces the issue.

Figure 5.35: Selecting the Sniff → PMKID module.

Figure 5.36: Choosing "Active (Force Deauth)" rather than passive listening — the mode that actively evicts clients to trigger a reconnection.

On execution, the ESP32 began transmitting spoofed deauthentication frames on Channel 10, using a placeholder source address (the firmware's characteristic DE:AD:BE:EF:FE:ED) and targeting the link between the access point and its connected client.

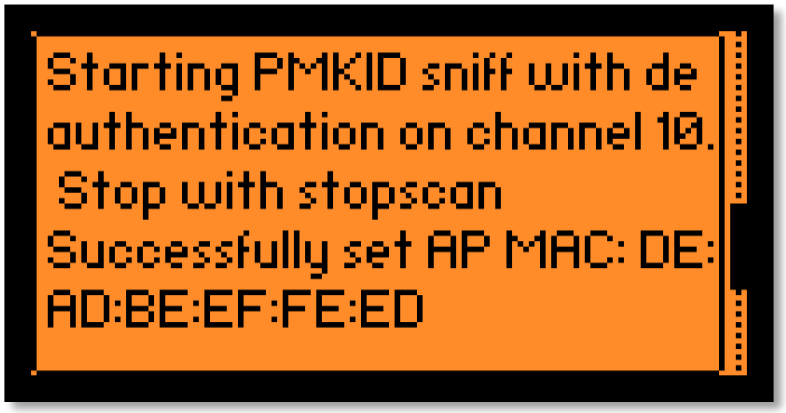

Figure 5.37: The active PMKID sniff with deauthentication running on Channel 10, with the spoofed AP MAC set.

The effect on the victim was immediate. The connected smartphone was thrown off the network and could not get back on.

Figure 5.38: The victim handset during the attack — unable to connect to "Vodafone vol 3," a clear denial of service.

For as long as the spoofed frames kept flowing, the client stayed locked out (Risk: High for availability). This half of the attack is, on its own, a complete denial-of-service primitive — no cryptography involved, just the protocol's own trust in unauthenticated control messages turned against it.

5.5.3 EAPOL Handshake Capture

Denial of service is rarely the real goal of a deauth, though. The reason an attacker forces a disconnect is what happens next. A WPA2 client, once dropped, tries immediately and automatically to reconnect — and reconnection means renegotiating a session key through the WPA2 four-way handshake, carried in EAPOL frames. Those frames contain values cryptographically derived from the network password. By forcing the disconnect, the tool guaranteed the handshake it wanted would be retransmitted within seconds, right when it was listening.

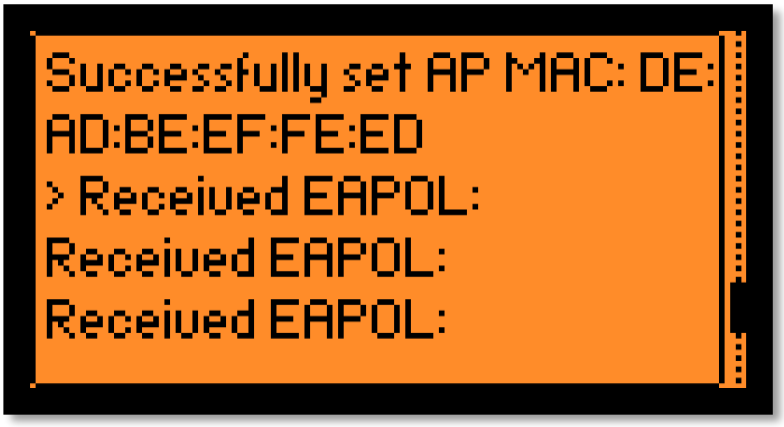

Figure 5.39: Multiple EAPOL frames captured as the evicted client reconnects — the four-way handshake, intercepted

The module logged several "Received EAPOL" events as the client came back online, and the handshake was written to a packet-capture (.pcap) file on the device. That single file changes the nature of the attack. The active, noisy, location-bound radio phase is over. What remains is a cracking problem the attacker can carry away and solve offline, at any distance and any pace — which is the subject of the second half of this case study.

5.5.4 Offline Cryptographic Cracking

The capture files were pulled from the platform's SD card for processing on a separate machine.

Figure 5.40: The captured PMKID/EAPOL files in the Marauder pcaps directory; sniffpmkid_1.pcap was selected for analysis.

Before spending any effort on cracking, the capture was checked. Opening sniffpmkid_1.pcap in Wireshark and filtering for EAPOL confirmed what was needed: a

complete four-way handshake, messages 1 through 4, exchanged between the target extender (BSSID 30:16:9D:56:34:64) and a connected client. A handshake missing any of the four messages is useless for cracking, so this verification is not a formality — it is the difference between a workable capture and wasted hours.

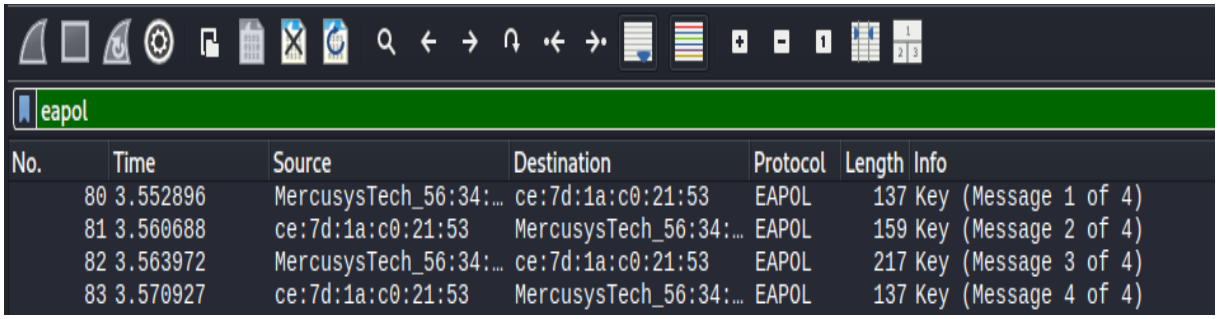

Figure 5.41: Wireshark confirming all four EAPOL messages (M1–M4) of the WPA2 handshake.

With a valid handshake confirmed, the file moved to a Kali Linux workstation. There, hcxpcapngtool read the capture and converted its cryptographic material into the hc22000 hash format that modern crackers require.

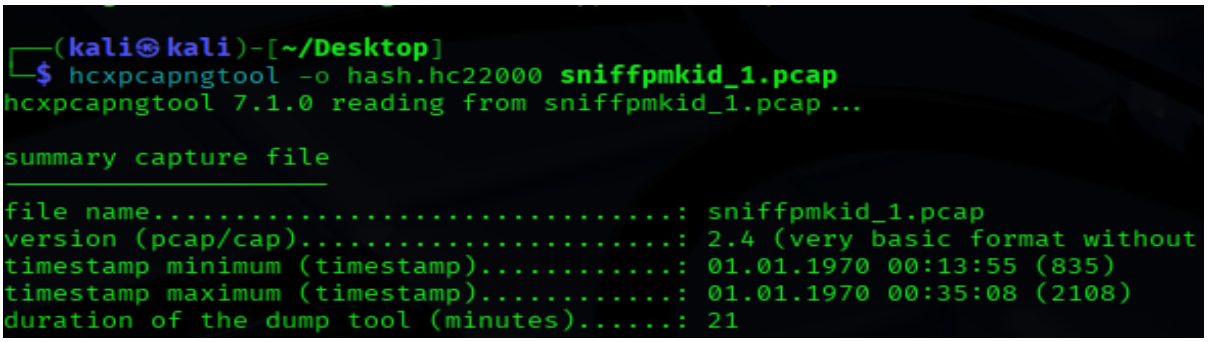

Figure 5.42: Converting the capture to hc22000 format with hcxpcapngtool on Kali Linux.

The hash was then run through Hashcat against rockyou.txt — a dictionary of millions of passwords leaked in past breaches and the standard first pass for any real-world WPA2 audit. Attacks of this kind, using nothing more than commodity hardware and freely available wordlists, are well documented in the literature [24].

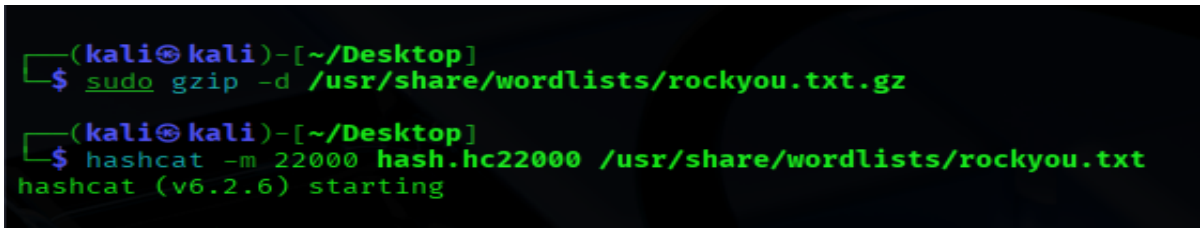

Figure 5.43: Launching the Hashcat dictionary attack (mode 22000) against the converted hash.

The result came back in thirteen seconds. Hashcat matched the handshake against a dictionary entry and recovered the network's plaintext WPA2 key outright.

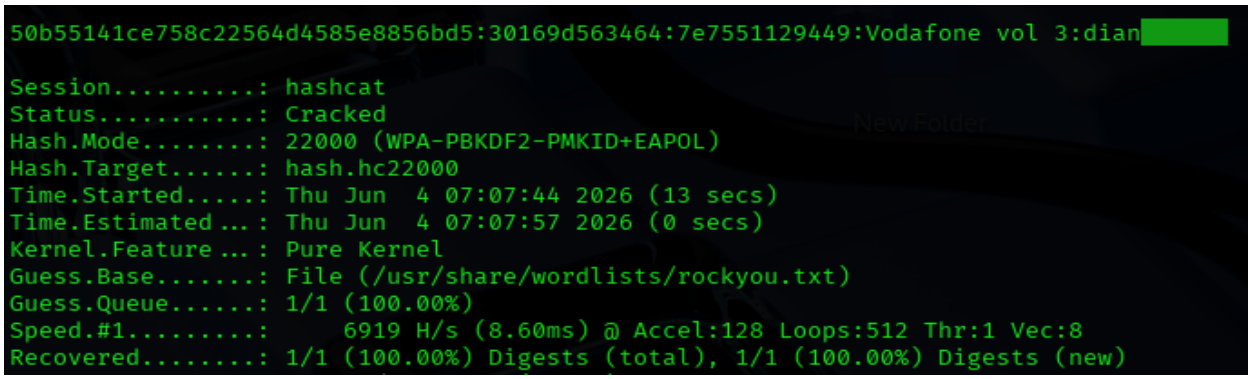

Figure 5.44: Hashcat reporting the hash as cracked and revealing the recovered pre-shared key.

It is worth being precise about what just happened, because it is easy to misread. WPA2 was not broken. The AES encryption held, the protocol did exactly what it was designed to do, and the handshake itself gave away nothing on its own. What fell was the password behind it — a short key built from lowercase letters and digits, common enough to sit inside rockyou.txt. The deauthentication attack guarantees an attacker can obtain the handshake; from that point the network's entire security rests on whether the pre-shared key can survive an offline dictionary or brute-force search. Here it lasted thirteen seconds. A long, random passphrase would have survived indefinitely against the same wordlist. The cipher is strong; the weak link is human password choice (Risk: Critical, contingent on password strength).

5.5.5 The Defensive Case: Protected Management Frames (802.11w)

An assessment that only reports where an attack succeeds tells half the story. The identical method was turned against a different device in the same environment — the main ISP-supplied router, VODAFONE_7004, on Channel 9.

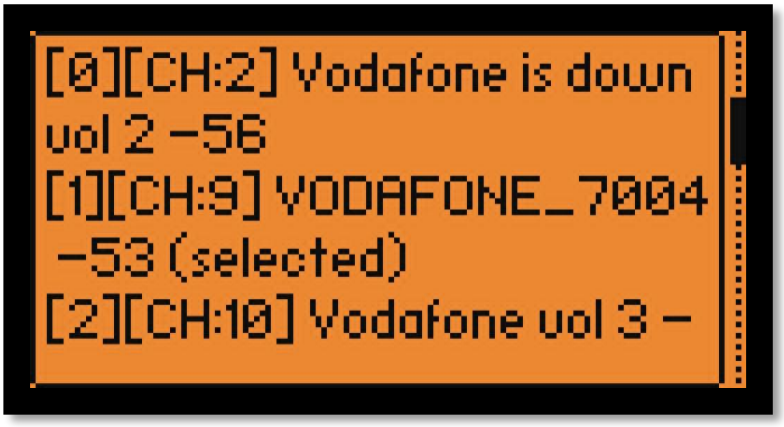

Figure 5.45: Re-targeting the tool toward the main router (VODAFONE_7004, Channel 9), selected from the AP list.

The same active forced-deauthentication routine was started. This time it went nowhere.

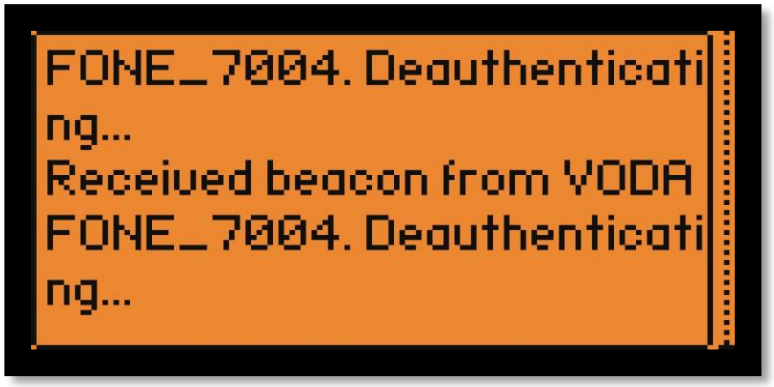

Figure 5.46: The attack against VODAFONE_7004 looping on "received beacon… deauthenticating…" — the client is never disconnected.

The tool reported beacons and dutifully announced it was deauthenticating, but the connected client never dropped. No disconnection, no reconnection, no handshake to capture — the whole chain stalled at the first step. That failure is one of the more useful results in this case study, and it points squarely at Protected Management Frames, the protection added in the IEEE 802.11w amendment [13] and now standard on current ISP routers.

PMF closes the exact gap the attack relies on. With it active, the access point and its clients cryptographically authenticate their management frames, deauthentication and disassociation included. The attacking tool holds no session key, so it cannot forge a frame the client will accept. The spoofed deauth packets still reach the client — but the

client inspects them, finds them unauthenticated, and drops them on the floor. No denial of service, and with no forced disconnect, no handshake to take offline. A single amendment breaks the entire attack chain at its root.

5.5.6 Findings

End to end, this case study captures both faces of wireless security in one environment. Against the legacy extender, with no management-frame protection, the attack was quick and complete: a forced deauthentication delivered an instant denial of service and handed over the handshake, and a weak password did the rest, falling in thirteen seconds. Against the PMF-enabled router in the same room, running the very same attack, nothing happened at all.

That contrast is the finding worth carrying forward. The deauthentication weakness is no museum piece — it stays critical for the enormous installed base of older access points, range extenders, and IoT hubs that ship without 802.11w. Yet the mitigation is real, it works, and it is already deployed in mainstream equipment. Both statements are true at once, and which one governs a given network depends entirely on whether its weakest device speaks 802.11w. In a typical home, the shiny main router may be protected while a cheap extender beside it is wide open — and an attacker only needs the weakest link. That, more than any single cracked password, is the practical lesson of this case study.

5.6 Case Study 6: Beacon Flooding and the WPS Reconnaissance Discrepancy

Case Study 5 was about getting in. This one is about two different things the 802.11 standard gets wrong — neither of which involves cracking a handshake. The first is a denial-of-service trick that clutters a victim's screen with phantom networks. The second is more subtle and, for a thesis, more interesting: a case where the portable tool and a professional workstation flatly disagreed about whether a network was vulnerable, and working out which one to believe.

5.6.1 Interface Disruption via Beacon Flooding

Access points announce themselves with beacon frames — small broadcasts carrying the network name and capabilities, sent constantly so that nearby devices know the network exists. Like the management frames in the previous case study, legacy beacon frames are unauthenticated. Nothing verifies that a beacon comes from a real access point, which means anything with a transmitter can manufacture as many fake ones as it likes.

The ESP32 Marauder module was set to do exactly that.

Figure 5.47: Selecting the Beacon Spam module on the platform.

For a clear demonstration, the SSID list was loaded with a sequence of joke strings — the well-known "Rickroll" payload, whose fake network names spell out song lyrics.

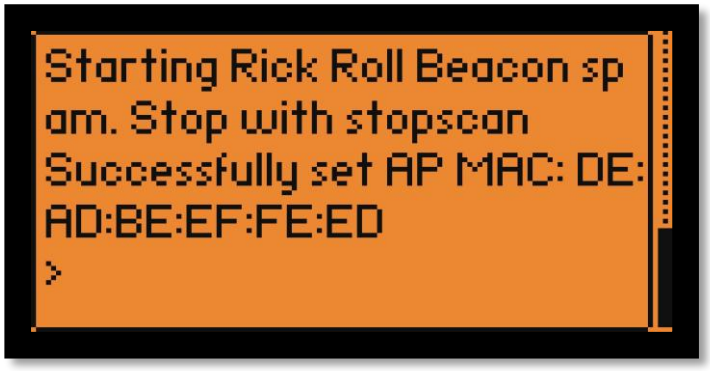

Figure 5.48: Starting the Beacon Spam attack, broadcasting dozens of spoofed access points.

The module began pumping these SSIDs into the air, and because every operating system listens passively for beacons and lists what it hears, the fake networks appeared on nearby devices almost at once.

Figure 5.49: A Windows client's Wi-Fi list flooded with the spoofed "Rickroll" SSIDs.

On the target Windows machine the effect was obvious and disruptive. The Wi-Fi list filled with bogus entries, burying the real networks somewhere in the noise. It is not a sophisticated attack — no encryption is touched and no device is compromised — but in

a crowded setting it makes the network list genuinely hard to use, hides legitimate access points among the decoys, and can slow the interface as the OS struggles to sort a flood of incoming SSIDs. A perception-layer nuisance more than a breach, but a real one (Risk: Medium for usability).

5.6.2 The WPS False Negative

The second part of the case study set out to check the target networks for a Wi-Fi Protected Setup (WPS) weakness. WPS is a legacy convenience feature, and its PIN method has been known to be badly broken since 2011, when Viehböck showed that the eight-digit PIN could be brute-forced in hours because of a flaw in how the PIN is validated in two halves [26]. A later refinement, the offline "Pixie Dust" attack described by Bongard, cut that time to seconds against routers with weak nonce generation [44]. So the presence or absence of WPS matters, and the first job was simply to detect it [27].

The Flipper Zero was asked to report the WPS status of the target network.

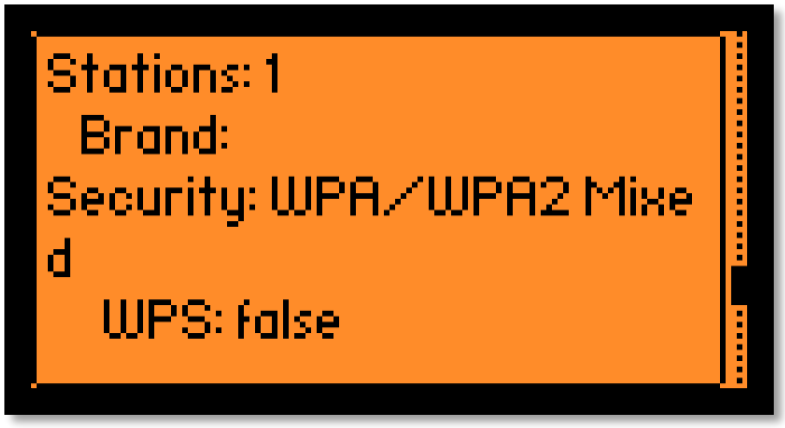

Figure 5.50: The Flipper Zero's Marauder scan reporting WPS as "false" for the target.

It returned WPS: false. Taken at face value, that is reassuring — it suggests the network does not expose WPS at all, and an auditor trusting the portable tool alone would tick the box and move on. That conclusion would be wrong.

To check it, the assessment moved to a Kali Linux workstation with an external monitor-mode adapter. The wash utility, which exists specifically to read WPS information elements out of beacon frames, was pointed at the same airspace.

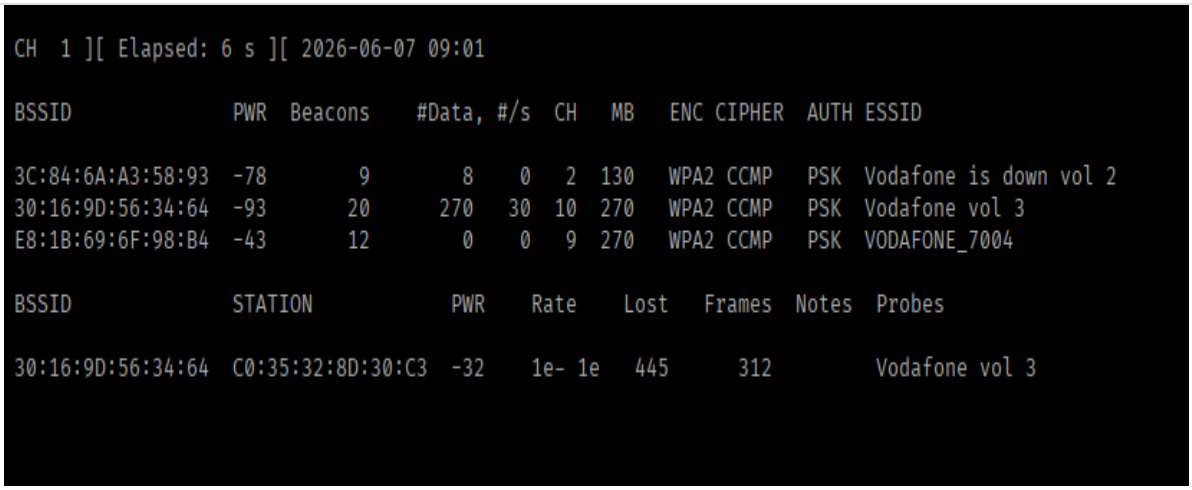

Figure 5.51: Network reconnaissance from the Kali Linux workstation with a monitor-mode adapter.

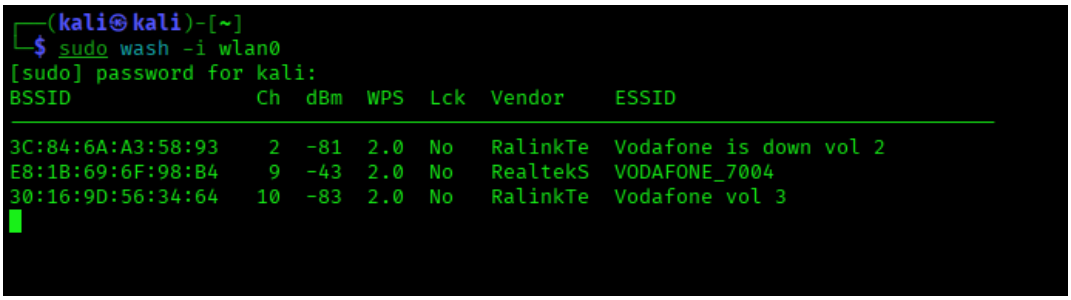

Figure 5.52: The wash utility reporting WPS 2.0 present and unlocked (Lck: No) on the networks the Flipper marked as having no WPS.

The two tools disagreed outright. Where the Flipper saw nothing, wash reported WPS version 2.0, active and not rate-locked, on three nearby networks. The portable tool had produced a false negative. This was not an isolated misread of one awkward router, either: the Flipper returned WPS: false uniformly across every network in range, including the three that wash independently confirmed were running WPS 2.0. A one-off parsing failure on a single vendor's beacon format would be understandable; a consistent false negative across every target points to a systematic limitation in the firmware's WPS detection, not an edge case.

That gap is the real finding of this case study, and it cuts against the tool being evaluated throughout this thesis. The Flipper's Marauder firmware leans on a simplified passive parser, and across every network tested it failed to surface the WPS information element from the beacon frames. Whatever the precise cause — a limitation in how the firmware parses WPS tags, or how it reports them — the behavior was consistent enough to treat as a systematic blind spot rather than a vendor-specific quirk. The practical lesson

is uncomfortable but important: a portable multi-tool is a reconnaissance aid, not a source of ground truth. An auditor who trusted the Flipper's "WPS: false" would have walked past a live attack surface that a thirty-second wash scan exposed immediately. Validation against a proper software stack is not optional.

5.6.3 The Limits of the WPS Attack on This Target

Detecting WPS is one thing; exploiting it is another, and here the two part ways. The logical next step after wash confirms an active, unlocked WPS service would be a PIN attack with a tool such as Reaver, or the offline Pixie Dust variant — recovering the WPS PIN and, through it, the WPA2 password regardless of how strong that password is [26], [44].

That path was not available against this particular router, and the reason is worth recording. The target operates WPS in Push-Button Configuration (PBC) mode only — the variant where pairing is authorized by physically pressing a button for a two-minute window — rather than the static-PIN mode. The PIN-based attacks depend on there being a fixed PIN to guess. PBC offers no such PIN to attack: there is nothing for Reaver or Pixie Dust to brute-force, because authorization is gated on a physical button press, not a guessable number. So while wash correctly reports WPS as present, the specific attack surface those tools target is absent on this device.

This is itself a useful result. PBC-only configuration sidesteps the entire class of WPS PIN attacks that have plagued the protocol since 2011 — not by patching the PIN flaw, but by not exposing the PIN method at all. It is a reminder that "WPS is enabled" and "WPS is exploitable" are not the same statement, and that an honest assessment has to distinguish detection from exploitability rather than assuming the first implies the second.

5.6.4 Findings

Two separate weaknesses, two different lessons. The beacon flood is trivial to run and trivially disruptive — a low-severity but real attack on the usability of the wireless environment, made possible once again by unauthenticated 802.11 frames **(Risk: Medium).** The WPS phase produced something more valuable than an exploit: a documented disagreement between the portable tool and a professional one, in which the portable tool was wrong in the dangerous direction, reporting a vulnerability as absent.

That the WPS service turned out not to be exploitable on this specific router — because it runs in PBC mode — does not soften the reconnaissance lesson. It sharpens it. Because the false negative was systematic rather than incidental, the risk it poses is not hypothetical: the tool that under-reported WPS on every network in range could just as easily under-report it on a router that was exploitable. Portable hardware earns its place in a kit, but not the last word in an audit.

5.7 Case Study 7: Perception-Layer Exploitation Beyond Radio — Infrared

The case studies up to now lived on radio frequencies. Infrared is a different medium with the same underlying weakness. It is light, not radio — invisible pulses from an LED, read by a photodiode — and it still controls a great deal of expensive hardware: televisions, media boxes, and the climate-control units that sit in nearly every modern building. None of it authenticates anything. This case study looks at two ways into an IR device: guessing its commands from a built-in library, and simply recording and replaying the light pulses it responds to.

5.7.1 Configuration and Method

The Flipper Zero carries its own IR transceiver — a photodiode to receive, an LED to transmit — working at the 38 kHz carrier frequency that almost all consumer remotes use. IR differs from the Sub-GHz work in one important way. Radio passes through walls and travels in every direction; IR needs line of sight. That is a real constraint on an attacker, but it cuts both ways: an IR attack is also harder to notice, since it leaves no radio footprint and works only when aimed at the target.

5.7.2 The Dictionary Attack: Universal Remote

The first approach assumes the attacker knows nothing about the target's specific protocol. The Flipper's Universal Remote mode carries a pre-built library of standard IR codes, organized by appliance type and brand, and simply transmits them in sequence until something responds.

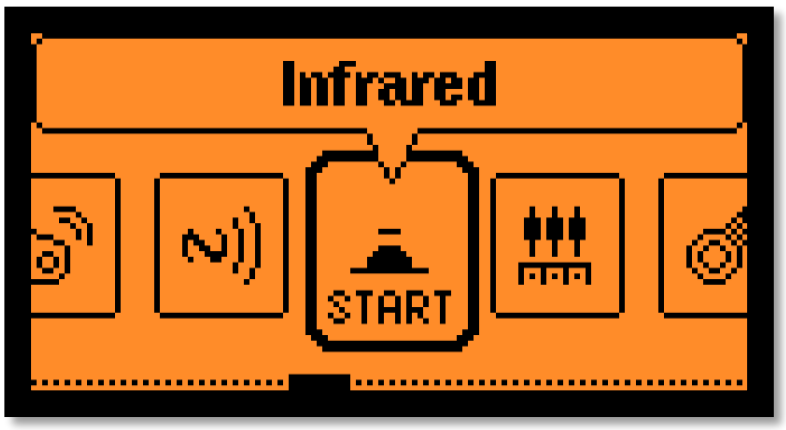

Figure 5.53: The Infrared application menu.

Figure 5.54: Selecting Universal Remote — a dictionary of standardized IR codes.

The air-conditioning unit. The tool was pointed at an HVAC unit and told to work through its AC library.

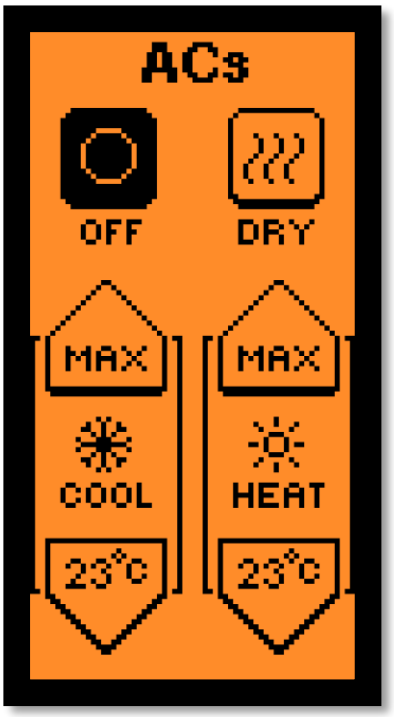

Figure 5.55: Starting the dictionary run against the air-conditioning category.

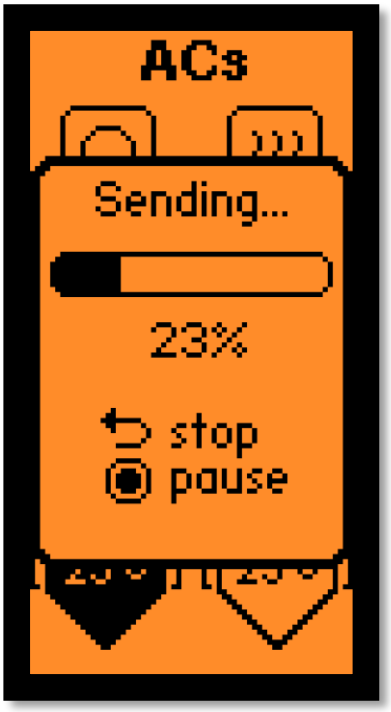

Figure 5.56: The attack cycling through manufacturer code sets, transmitting standard commands such as power and temperature.

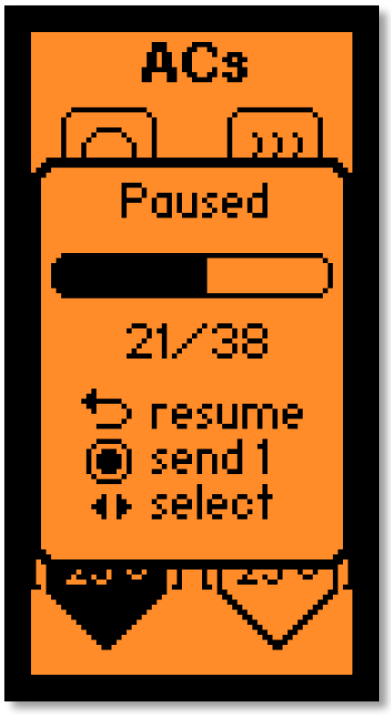

Figure 5.57: The unit responds — a matching code from the library has actuated the HVAC system.

The television. The same method was turned on a TV.

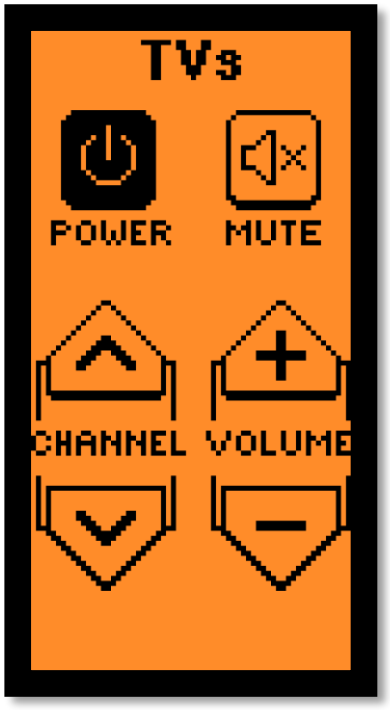

Figure 5.58: Starting the Universal Remote run against the television category.

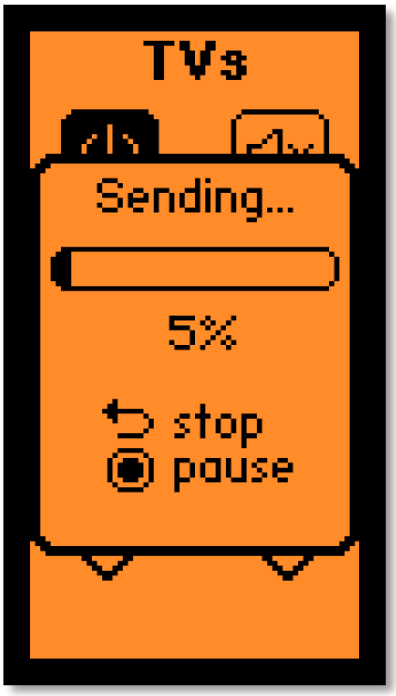

Figure 5.59: The TV responding to a transmitted power command from the library.

Both devices fell to the same simple idea. Without ever having seen the original remote, the attacker cycles through a dictionary of known codes and waits for the appliance to react. For widely used brands the library is small enough that a match comes

in seconds. No capture, no analysis — just a list of standard codes and the patience to play them.

5.7.3 Capture and Replay — and an Instructive Difference

The second approach is quieter: record a real command and play it back. The Flipper's "Learn New Remote" function was set to listen.

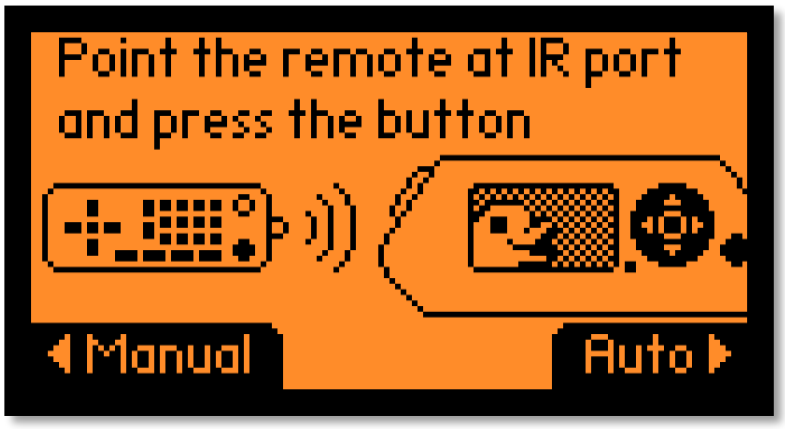

Figure 5.60: The tool in learning mode, watching the optical spectrum for an incoming IR command.

Pointing the legitimate remotes at the receiver produced two different outcomes, and the difference between them is the most interesting technical point in this case study.

The television — decoded cleanly. The TV's commands were recognized and parsed straight away.

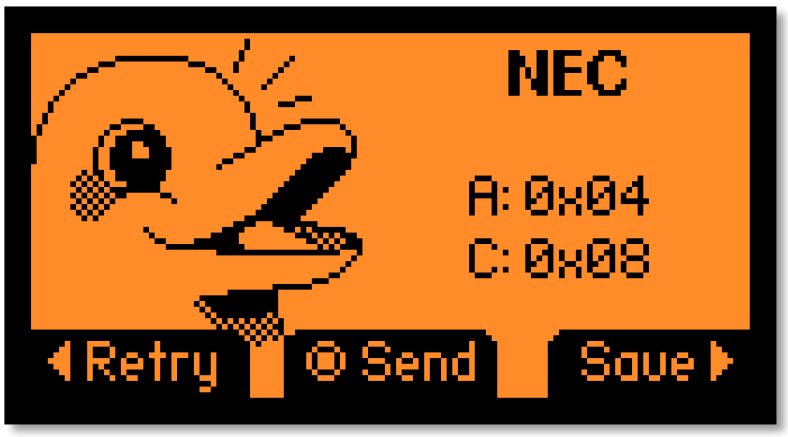

Figure 5.61: A captured TV "power" command, decoded as the NEC protocol (Address 0x04, Command 0x08).

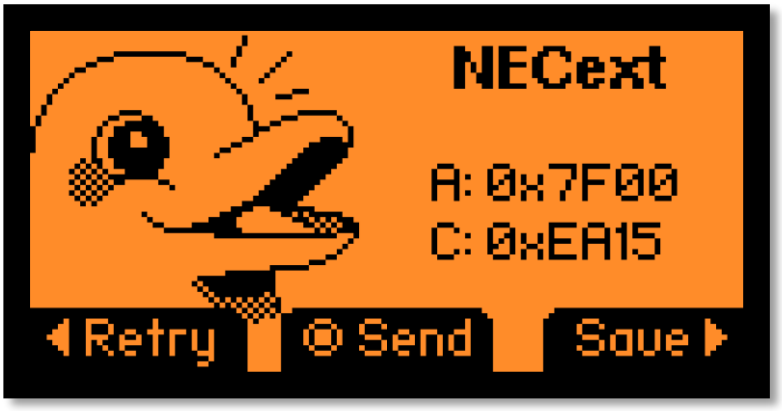

Figure 5.62: A captured TV "mute" command, decoded as the NEC-extended (NECext) protocol (Address 0x7F00, Command 0xEA15).

NEC is the most common consumer IR protocol, and the Flipper has a parser for it [45]. A TV command is short and self-contained — a brief code that means, in effect, "toggle power" — so it decodes into a tidy address-and-command pair.

The air conditioner — unknown, but still cloned. The AC remote behaved completely differently. The Flipper could not identify the protocol and fell back to a raw capture, recording 455 individual pulse-timing samples.

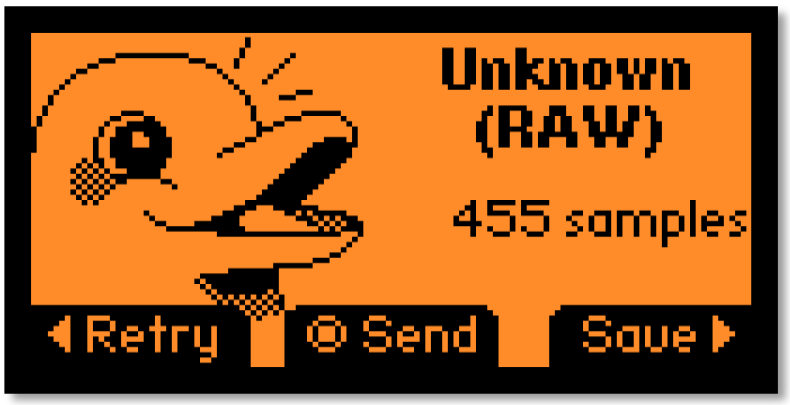

Figure 5.63: The HVAC signal captured as "Unknown (RAW)" — 455 discrete pulse-length samples, with no protocol decoded.

The reason is architectural, and it matters. A TV remote sends discrete, single-purpose codes. An air-conditioner remote does something else: each press transmits the unit's entire state in one long frame — mode, target temperature, fan speed, vane position, timer, all of it at once. That is why pressing "temperature up" on an AC remote sends a far larger payload than pressing "power" on a TV. The Flipper had no parser for

this particular AC state-frame, so instead of decoding it, it recorded the raw analog timings of the light pulses — the 455 samples.

Here is the part that matters for security. The signal was labelled "Unknown," undecoded, its structure never interpreted — and replaying the raw pulse sequence actuated the air conditioner anyway. The attacker did not need to understand the protocol, parse the state-frame, or know what any field meant. Recording the shape of the light and playing it back was enough. Decoding is a convenience for the attacker, not a requirement.

5.7.4 Findings

Infrared security comes down almost entirely to physical barriers — a wall, a closed door, the need to point at the target. There is no authentication in the signal itself. The dictionary attack showed that an attacker carrying no prior knowledge can take control of common appliances by replaying standard codes until one works. The capture-and-replay test showed something more pointed: even when the protocol is complex enough that the tool cannot decode it, raw analog replay still works. The 455-sample AC frame looked like it might offer some protection through sheer complexity, and it offered none. Complexity is not secrecy, and secrecy is not security (**Risk: Medium**, bounded only by the line-of-sight constraint).

This is not a niche concern. The same IR interfaces sit on devices increasingly marketed as smart — air conditioners with companion apps, TVs with network stacks — and the unauthenticated IR channel persists underneath the connected features, a documented risk in IoT devices that retain IR control [46]. As with the gate and the shutters in earlier case studies, adding a network layer on top does nothing to close the unauthenticated control path beneath it.

5.8 Case Study 8: Personal Area Networks — BLE Reconnaissance and Interface Spoofing

The earlier wireless work targeted infrastructure: routers, access points, gates. This case study turns the other way, toward the devices in people's pockets and on their desks. Bluetooth Low Energy (BLE) is everywhere in that space, and it carries a design assumption worth examining. To make pairing effortless—so a phone notices your earbuds the instant you open the case—iOS, Android, and Windows all listen continuously for BLE advertising packets and act on them without any authentication. That eagerness to be helpful is the attack surface. This study shows how it is turned into a denial of service.

5.8.1 Passive Reconnaissance and a Hardware Limitation

Before transmitting anything, the plan was to survey the ambient BLE environment. This is where the first practical finding appeared, and it concerns the testing platform itself. The Flipper Zero's Wi-Fi Development Board is built on an ESP32-S2—a micro-controller with no Bluetooth radio at all. The Marauder firmware that handled the Wi-Fi work in earlier case studies simply cannot scan for BLE, because the hardware underneath it lacks BLE capability. Reconnaissance therefore had to be conducted utilizing a dedicated BLE scanner application running on a separate smartphone.

That is worth recording rather than hiding. It marks a real boundary of this particular hardware combination: the Dev Board is a Wi-Fi tool, not a Bluetooth one. Any BLE exploitation executed by the evaluation platform relies entirely on the Flipper Zero's own internal radio module.

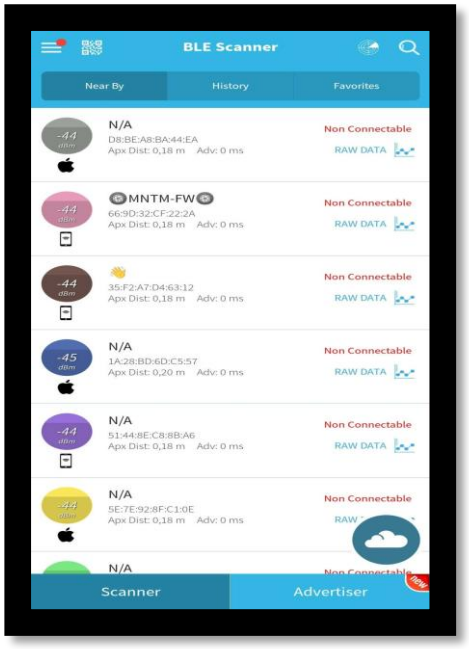

Figure 5.64: Passive BLE reconnaissance (smartphone scanner) showing nearby Apple devices broadcasting non-connectable advertising packets at close range.

The scan immediately picked up a cluster of Apple devices very close by—roughly 0.18 m, at a strong −44 dBm—broadcasting non-connectable advertising packets in the clear. The detail that matters is not the brand but the behavior: these devices announce their presence and state continuously, unprompted, to anyone listening.

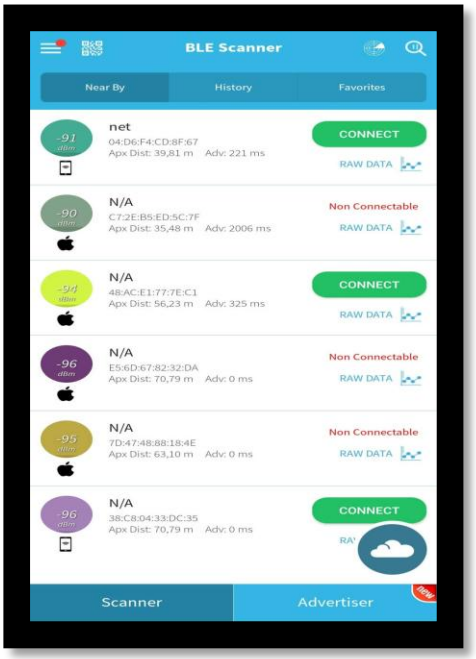

Figure 5.65: Expanded reconnaissance capturing connectable and non-connectable beacons from several vendors at varying distances.

Widening the scan brought in other vendors—Microsoft/Windows tags among them—and devices in different states, some explicitly advertising themselves as open to a connection. The takeaway from the passive phase is simple and sets up everything that follows: every nearby operating system is constantly listening for and parsing BLE beacons, and none of it requires authentication first.

5.8.2 Active Exploitation: BLE Spam

To abuse that listening state, the Flipper Zero—using its own built-in Bluetooth radio—was set to broadcast spoofed advertising packets at aggressive intervals.

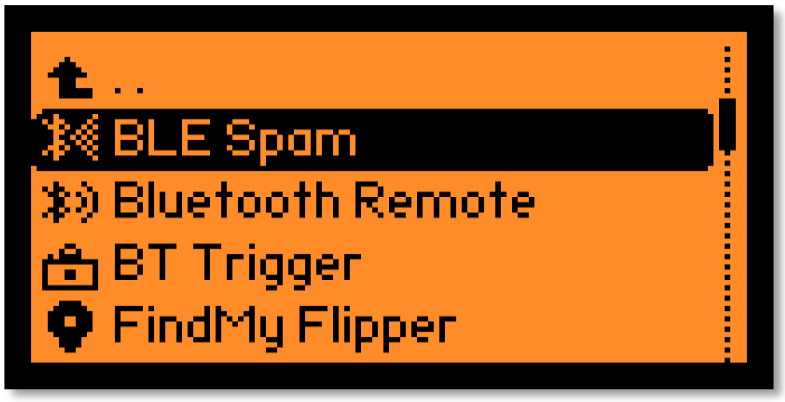

Figure 5.66: Launching the BLE Spam module on the Flipper Zero.

Two variants were run, aimed at two different OS behaviors.

Generalized Flooding ("Kitchen Sink"): The first mode cycles rapidly through dozens of fake device profiles at once—phantom earbuds, trackers, and the like—broadcasting them every 20 milliseconds.

Figure 5.67: The multi-profile BLE flood running at an aggressive 20 ms broadcast interval.

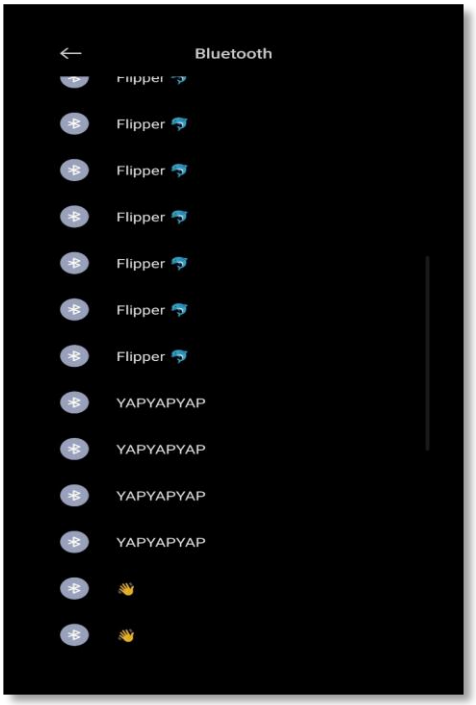

Figure 5.68: An Android Bluetooth menu filling with spoofed, non-existent devices.

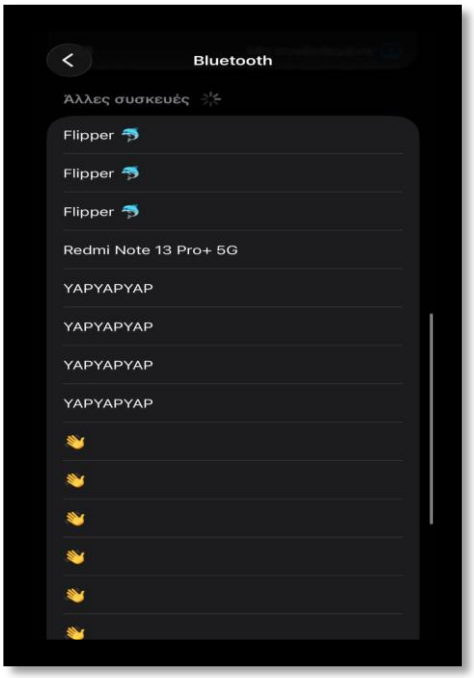

Figure 5.69: An iOS device exhibiting identical interface pollution from the BLE flood.

Both phones reacted the same way. Each parsed every spoofed beacon and dutifully listed the phantom hardware—devices named "YAPYAPYAP," emoji entries, and multiple "Flipper" handsets that do not exist. In that state, a user genuinely cannot find or

pair their real device; the list is buried in noise. It constitutes a localized denial of service against the pairing interface.

Windows SwiftPair: The second variant targeted a foreground behavior rather than a background list. Microsoft's SwiftPair protocol pushes an active pop-up when a compatible peripheral seems nearby. The Flipper was set to broadcast nothing but spoofed SwiftPair beacons with no cooldown.

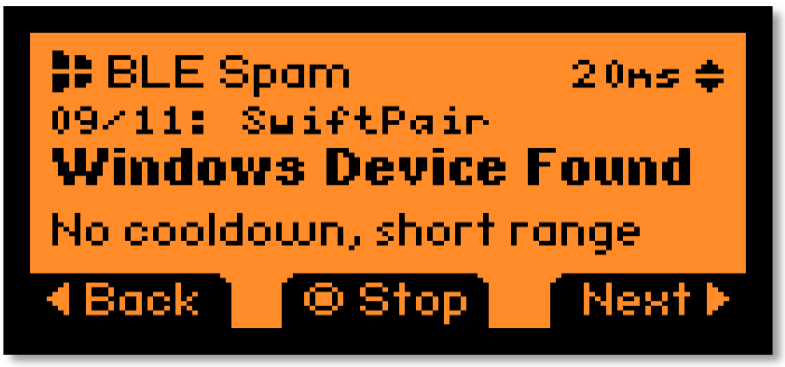

Figure 5.70: Configuring a continuous SwiftPair-only broadcast targeting Windows environments.

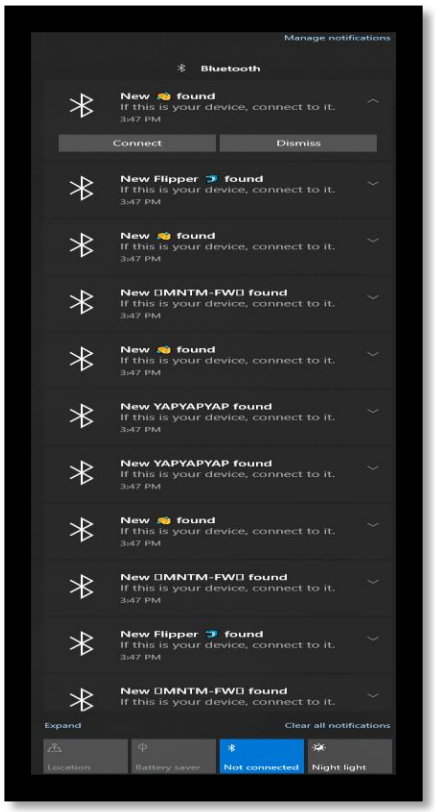

Figure 5.71: The Windows Action Center saturated with a non-stop stream of forced SwiftPair pairing prompts.

Windows trusted every beacon and generated a fresh foreground pop-up for each one, faster than they could be dismissed. The Action Center filled and stayed full, dragging the user's attention away and making the notification surface effectively unusable for as long as the broadcast ran.

5.8.3 The Rate-Limiting Caveat — An Honest Boundary

One point must be stated carefully, as it connects directly to the mitigations discussed in Chapter 6. Earlier iterations of this attack did more than merely clutter a screen. On iOS 17 before the 17.2 update, sustained BLE spam could freeze and forcibly reboot iPhones outright — a flaw severe enough to be assigned CVE-2023-42941, leading to documented incidents of commuters' phones rebooting en masse in public transit spaces [47]. Apple's subsequent patch throttled the notifications rather than removing the pairing feature entirely, which blunted the fatal crash without eliminating the underlying nuisance [47].

That history shapes how the results here should be interpreted. On the current, updated devices evaluated in this study, the attack did not crash or reboot anything. The operating systems absorbed the flood and kept running. What persisted was the interface-level Denial of Service — the polluted lists and the flooded Action Center — not a full system collapse. This represents the honest, current threat landscape: contemporary operating-system rate-limiting has contained the catastrophic execution of the attack, while leaving the usability disruption substantially intact. Broader analyses of BLE security confirm that this advertising-trust weakness is structural to the protocol's discovery model rather than specific to any one vendor [48].

5.8.4 Findings

This vulnerability lives at the intersection of wireless protocol architecture and User Experience (UX) design. To make discovery seamless, operating systems trust the very first, unauthenticated BLE advertising frame they receive—and that trust is inherently abusable. The economics of the attack are lopsided in the adversary's favor: broadcasting a tiny spoofed beacon costs almost nothing, while the victim's OS expends real computational effort parsing, rendering, and alerting the user to each phantom device. A second hardware point reinforces a core theme of this thesis: the attack runs off the Flipper's modest internal radio, emphasizing how little raw capability an attacker actually requires.

No data is compromised, and no encryption is broken. The damage is strictly to availability and usability, and on modern devices, it is bounded. Rate-limiting mitigations have removed systemic crashes from the equation, leaving a strong but non-catastrophic interface disruption (**Risk: Medium** for usability/availability on patched devices; **historically Critical** on unpatched iOS 17.0–17.1). The broader lesson carries forward to the mitigations chapter: convenience features that act on unauthenticated input are a recurring source of perception-layer denial of service, and the pragmatic fix is throttling and validation, not removing the convenience.

5.9 Case Study 9: Layer 8 Exploitation — Rogue Access Point and Captive Portal

Every case study so far attacked a machine — a cipher, a protocol, a parser. This one attacks the person using it [23]. The most durable weakness in any network is not its encryption but its operator, and no amount of WPA2 or WPA3 helps if the user simply hands over the password. This study builds a rogue access point — an "evil twin" — paired with a fake login page, and shows how the victim is led to surrender their credentials voluntarily. The cryptography is never touched. It does not need to be. Although evil-twin attacks predate the modern smart-home era, they remain acutely relevant to IoT environments, where the majority of devices connect through 802.11 Wi-Fi and where a rogue access point can serve as the entry point for a wider compromise of the connected ecosystem [49].

5.9.1 Configuration and Payload

The attack runs on the ESP32-S2 Dev Board, which here plays two roles at once: a small web server hosting the phishing page, and a DNS sinkhole that answers every lookup with its own address.

Figure 5.72: Selecting the Evil Portal module from the ESP32 toolset.

The whole attack lives or dies on plausibility — the victim has to believe the network and the page in front of them are real.

Figure 5.73: The captive-portal configuration menu.

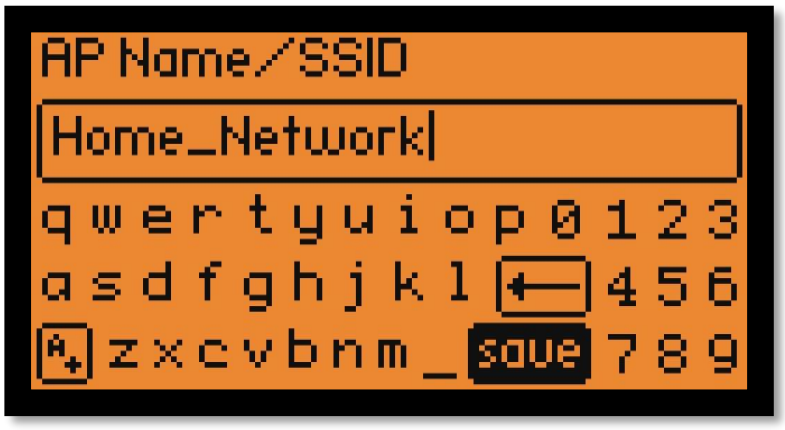

Figure 5.74: Setting the rogue AP's name to "Home_Network" — generic, familiar, and unremarkable.

The SSID was set to "Home_Network," a name ordinary enough to attract no suspicion. The HTML payload was then selected from the device's SD card. Rather than any exploit code, the payload is simply a convincing imitation of a router's administration page.

Figure 5.75: Selecting the phishing page (index.html) from the SD card.

A note on that page, since the implementation detail turned out to matter. The form was deliberately built to submit its data via an HTTP GET request rather than POST, and its input fields were named so the firmware's /get endpoint would capture them. This is not a cosmetic choice: in testing, a POST form left the captured fields blank on the Flipper's screen, while the GET form surfaced the typed credentials directly. The HTML itself is unremarkable — a logo, two input boxes, a "session expired" message — and is omitted here, since the security interest is in the delivery mechanism, not the markup.

5.9.2 Execution and the Captive-Portal Trigger

With everything set, the ESP32 was told to start broadcasting.

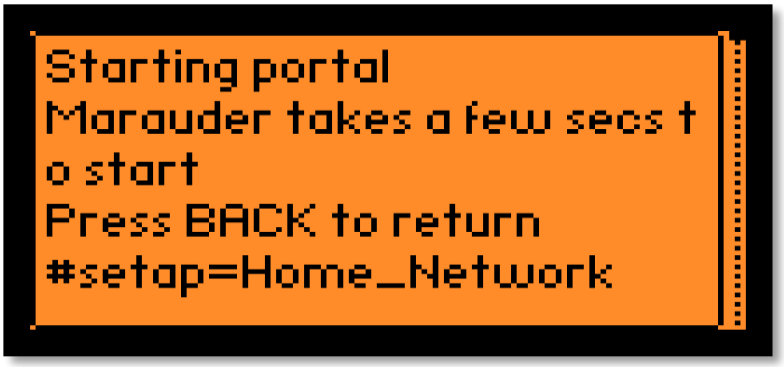

Figure 5.76: The tool broadcasting the open "Home_Network" AP and starting its internal DNS and web servers.

The network was broadcast open — no password — so it appeared on nearby devices without friction. Both a phone (Android) and a desktop (Windows) were connected, to check the behavior on each.

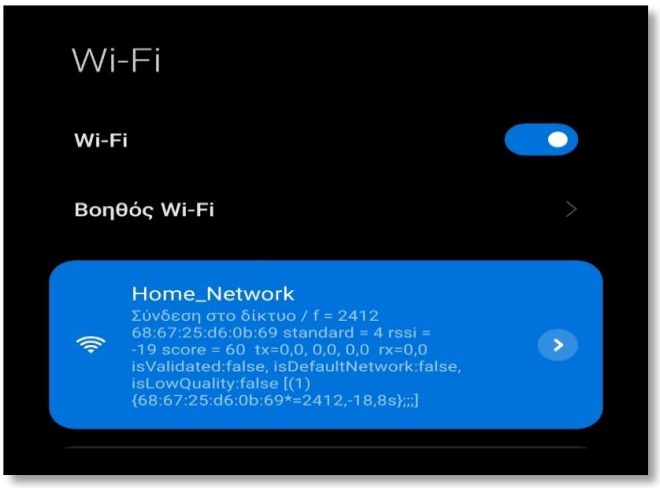

Figure 5.77: A mobile device connecting to the open rogue AP.

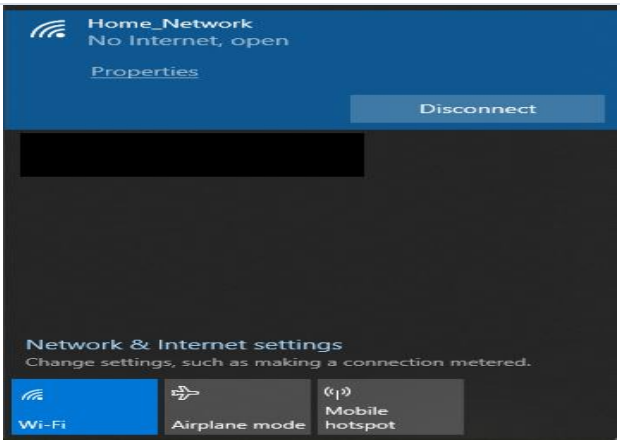

Figure 5.78: A Windows workstation connecting to the same open network.

What happens next is the clever part, and it is the operating system that does the work, not the attacker. Modern systems run Captive Portal Detection: the moment they join a network, they quietly request a known vendor URL — connectivitycheck.gstatic.com on Android, msftconnecttest.com on Windows — to check whether the internet is actually reachable. The ESP32, acting as the network's DNS server, intercepts that request and answers it with its own IP. The check fails in a very specific way, and the OS concludes the network has a login page it must show — exactly as a hotel or airport portal would. It then pops that page up on its own, unprompted.

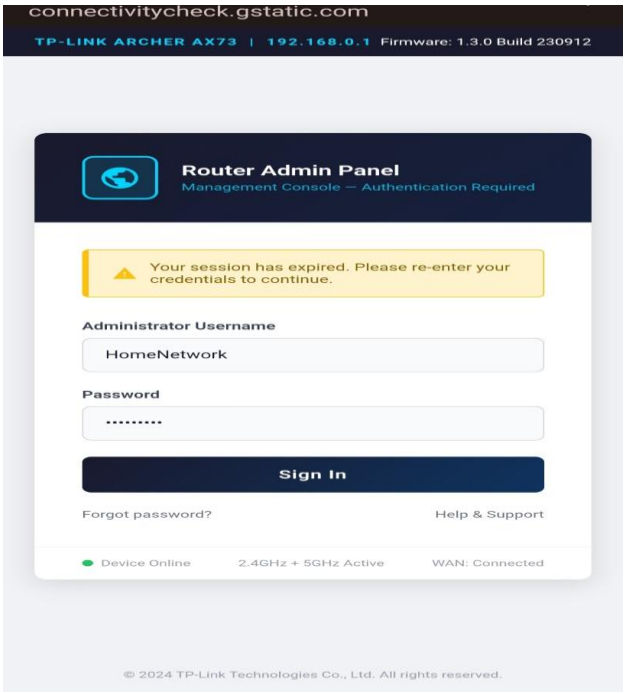

Figure 5.79: The captive portal triggering automatically. The intercepted Google connectivity-check URL is visible in the address bar, redirected to the fake TP-Link administration page.

That detail in the address bar — the Google connectivity-check URL resolving to the fake router page — is the whole mechanism made visible. The victim did not type a URL or open a browser. The phone's own connectivity check delivered the phishing page to the screen automatically.

5.9.3 Credential Harvesting

Believing it was their own router asking them to re-authenticate after a "session expired," the user entered a username and password. The GET form passed those values straight back to the ESP32.

Figure 5.80: The Flipper Zero logging and displaying the captured plaintext credentials on its own screen (u: HomeNetwork, p: admin1234).

The credentials appeared on the Flipper's display in plaintext, in real time. At that point the attack is over and it succeeded completely — and note what was never required to get here. No handshake was captured, no PIN brute-forced, no cipher attacked. Whatever WPA2 or WPA3 protects the real network is now irrelevant, because the password was typed willingly into the attacker's page.

5.9.4 Findings

The evil portal turns a convenience feature into a delivery system [25]. Captive Portal Detection exists to make joining public Wi-Fi painless — and it is precisely that automation that hands the attacker a reliable way to put a phishing page in front of a victim without the victim doing anything at all. The hardware costs a few dollars. It ran a DNS sinkhole, served a web page, and harvested credentials, and at no point did either operating system raise a security warning, because from the OS's perspective nothing abnormal happened: an open network, a failed connectivity check, a login page. All expected.

The lesson sits alongside the WPA2 case study rather than replacing it. Strong cryptography is necessary but not sufficient. As long as users are trained to type their password whenever a familiar-looking page asks them to fix their connection, an open, unauthenticated rogue AP remains a serious threat — one that sidesteps the entire cryptographic stack by going around it (**Risk: High**, dependent on user interaction). It is the cleanest demonstration in this thesis of a simple truth: you cannot patch the user, so the defenses have to assume the user will be fooled.

5.10 Case Study 10: Physical-Interface Exploitation — BadUSB (HID Injection)

Every attack so far has been wireless — radio, light, Wi-Fi, Bluetooth. This one needs a cable. BadUSB drops the radio entirely and exploits something more basic [28]: the unconditional trust an operating system places in anything that claims to be a keyboard. Plug in a USB keyboard and the machine does not ask who is typing or whether the keystrokes are sensible. It just obeys them. A device that impersonates a keyboard can therefore type its own commands — faster than any human, and with the full authority of whoever is logged in. That is the entire vulnerability, and it has no software patch.

5.10.1 Payload Design

The Flipper Zero runs keystroke-injection payloads through its "Bad KB" module, written in DuckyScript — the de facto scripting language for this class of attack.

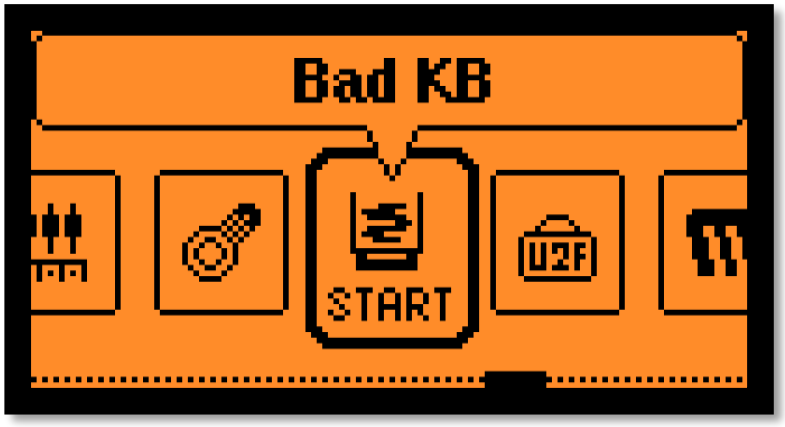

Figure 5.81: The Bad KB (BadUSB) module on the Flipper Zero.

The payload was kept deliberately benign, within the ethical scope set in Chapter 4. Rather than download or alter anything, it was written to prove the attack path and nothing more.

Figure 5.82: Selecting the custom demo_payload DuckyScript from device storage.

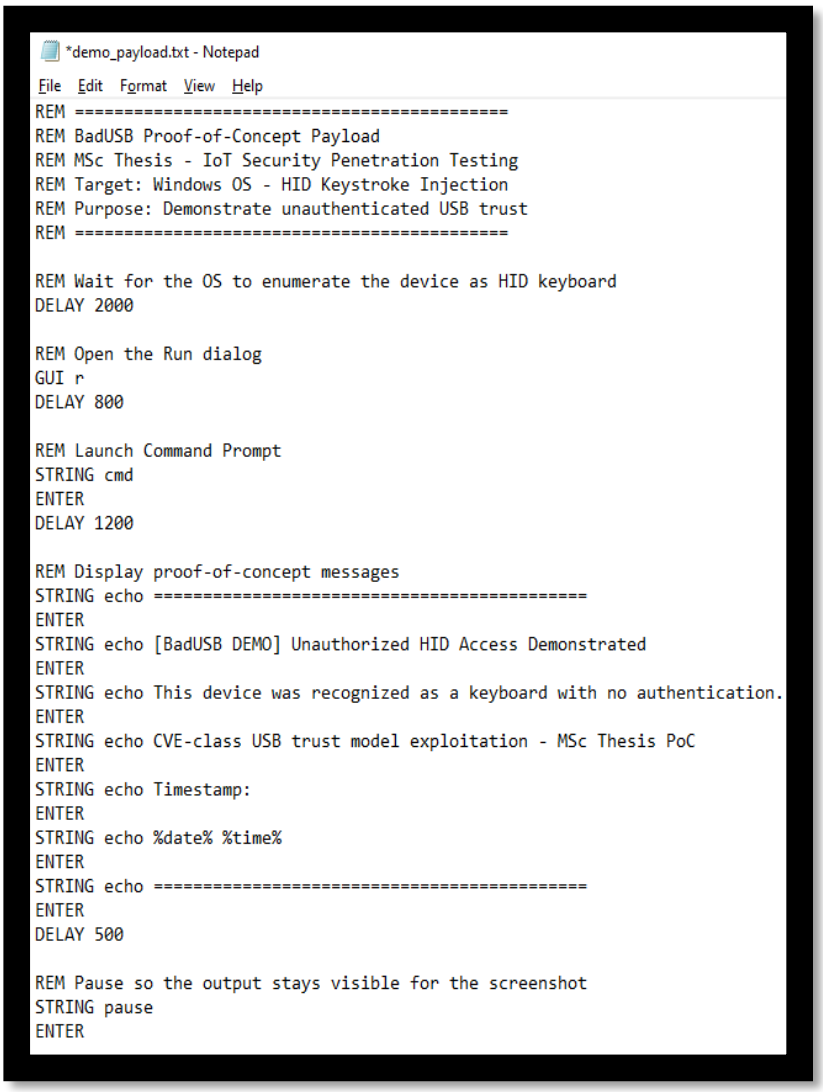

Figure 5.83: The DuckyScript source. It opens the Run dialog (GUI r), launches the command prompt, and echoes a series of proof-of-concept messages ending in a system timestamp.

The choice to make the payload harmless is itself a point worth stating. The same technique that types these harmless echo commands could just as easily type commands that download and run a remote executable, disable defenses, or open a reverse shell. The mechanism is identical; only the text being typed differs. The PoC demonstrates the delivery without crossing into anything destructive.

5.10.2 Execution

The attack requires brief physical access to an unlocked machine. The moment the Flipper was connected to the Windows workstation by USB, the OS enumerated it as a standard keyboard — no driver prompt, no confirmation, no warning.

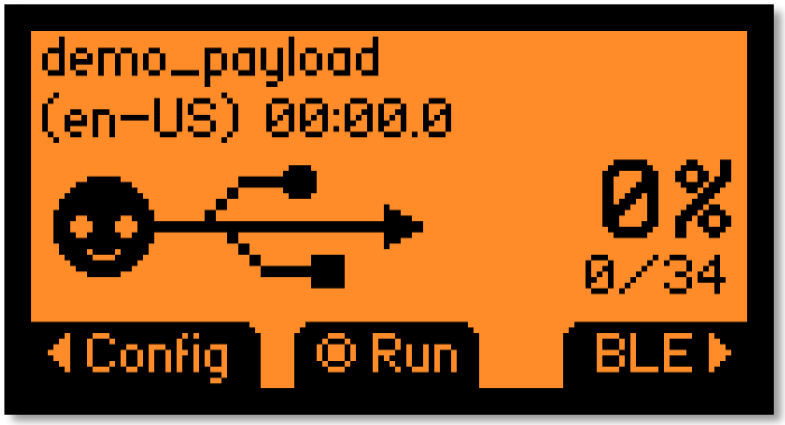

Figure 5.84: The payload loaded and awaiting the run command.

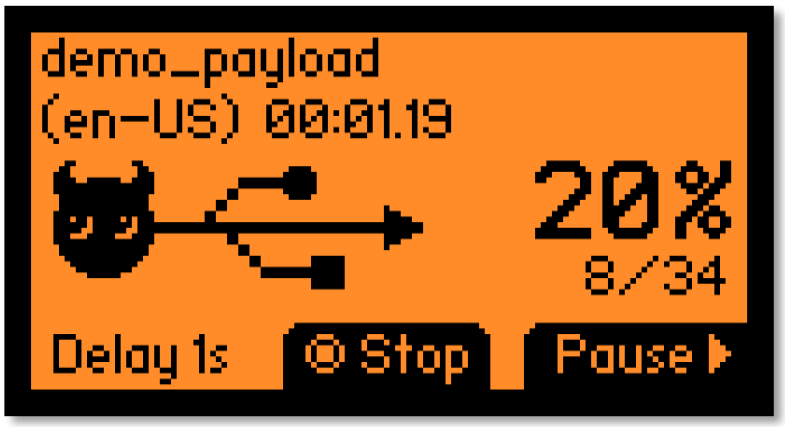

Figure 5.85: Injection underway — the script is emulating keystrokes far faster than a person could type.

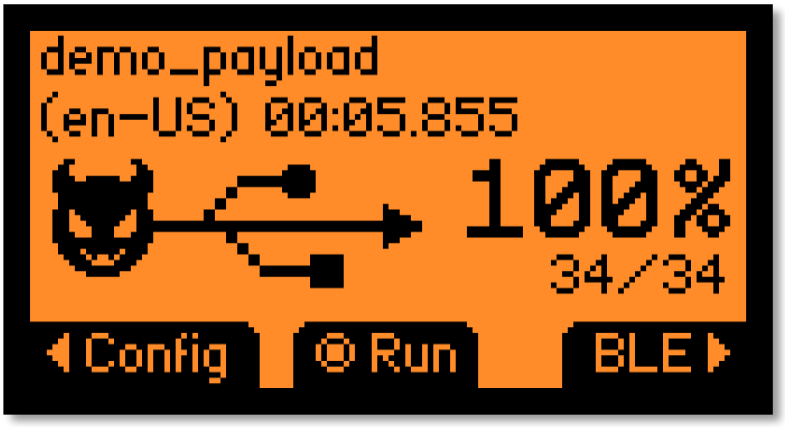

Figure 5.86: Execution complete — 34 command lines injected in under six seconds (00:05.855).

The timing is the point. In 5.855 seconds, the device opened the Run dialog, launched a terminal, and typed 34 lines of commands. A human watching the screen would barely register what was happening before it finished. There is no window in which to intervene.

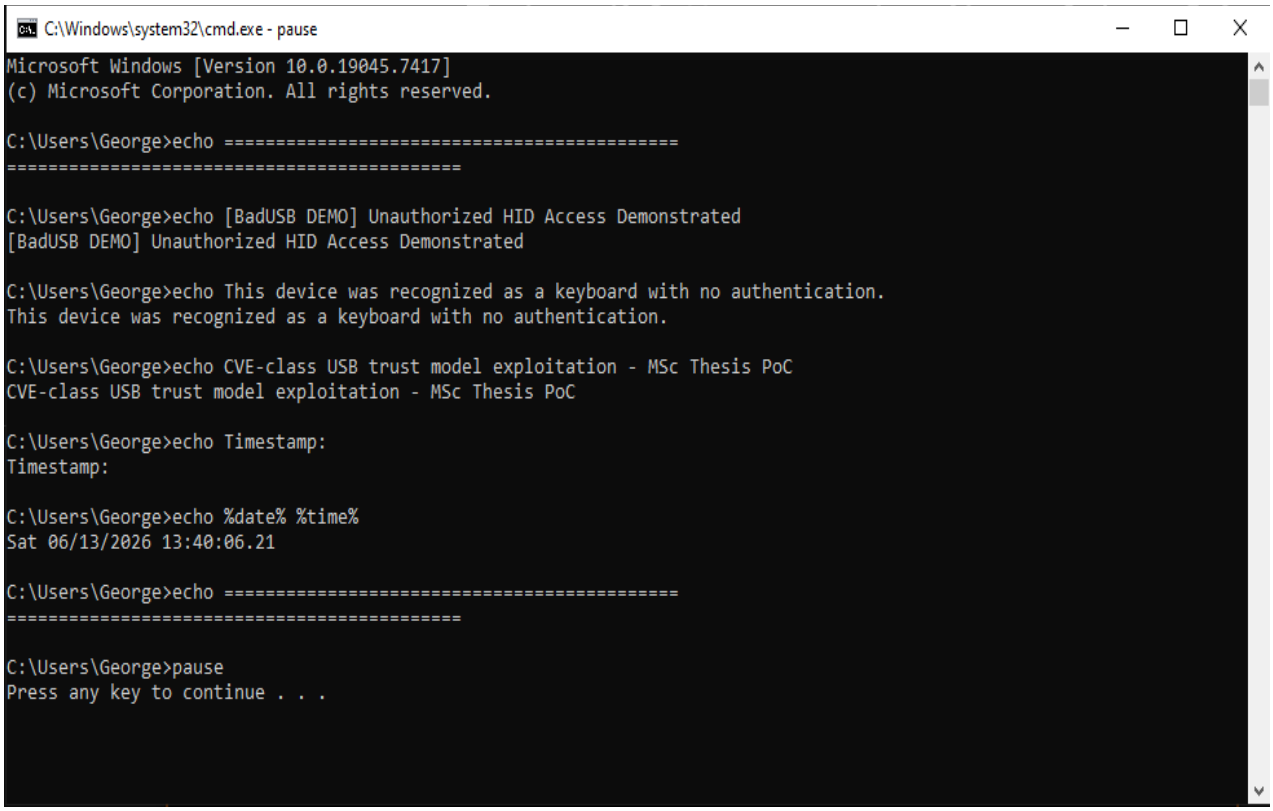

Figure 5.87: The result on the workstation — the command prompt showing the echoed proof-of-concept text and the recorded timestamp, with no security prompt at any stage.

5.10.3 Findings

BadUSB exposes a blind spot that sits below the level any antivirus or firewall operates at. Those tools inspect files, processes, and network traffic [30]. They do not — and by design cannot — question whether the keyboard is a real keyboard. From the operating system's point of view, no exploit took place at all: a keyboard was attached and some keys were pressed. Every keystroke was legitimate input. That is exactly why it slips past software defenses untouched — there is nothing for them to flag.

The consequence is blunt. On an unlocked machine, brief physical access equals full control at the privilege level of the logged-in user (**Risk: Critical**, contingent on physical access). And because the flaw is in the trust model of USB HID itself, not in any one piece of software, it cannot be patched [29] away. The defenses are different in kind: locking the screen when away, device-control policies that whitelist approved USB hardware, and endpoint protection that watches for the behavioral signature of injection — a keyboard that suddenly "types" hundreds of characters per second. This is the one case study in the set where the answer is not a firmware update or a stronger key, but physical and policy discipline.

5.11 Comparative Summary

Taken together, the ten case studies map the practical security posture of the perception layer across every major protocol it depends on. Table 5.1 consolidates these empirical results, juxtaposing each target with the protocol attacked, the technique applied, the outcome observed, and the resulting risk level.

Table 5.1: Summary of Empirical Findings (Chapter 5)

#	Target / Asset	Protocol & Band	Attack Technique	Result	Risk
CS1	Gate automation (CAME)	Fixed code, 433.88 MHz Sub-GHz	Capture & replay + RAW jamming	Gate opened; localized DoS achieved	Critical
CS2	Smart-home shutters (Somfy RTS)	Rolling code, 433.42 / 868.95 MHz	Sync-window replay (cnt 2131→2137) + jamming	433 MHz unit defeated & jammed; 868 MHz jamming failed (HW limit)	High
CS3	RFID access badge	LF 125 kHz (HID Proximity, 34-bit)	Read & emulate (5–7 cm)	Credential cloned; emulation granted access	Critical
CS4	NFC access card (MIFARE Classic 1K)	HF 13.56 MHz (CRYPTO1)	Dictionary attack (default keys) + emulation	All 32 keys / 16 sectors recovered; full clone	Critical
CS5	Wi-Fi WPA2 (extender + router)	802.11, 2.4 GHz	Deauth → EAPOL capture → offline Hashcat	Extender PSK cracked in 13 s; router resisted (PMF/802.11w)	Critical*
CS6	Wi-Fi environment / WPS	802.11, 2.4 GHz	Beacon flooding + WPS recon (Flipper vs Kali)	UI DoS achieved; systematic WPS false-negative documented	Medium
CS7	HVAC + television (IR)	Optical IR, 38 kHz carrier	Universal-remote dictionary + RAW capture/replay	Both devices controlled; AC "Unknown (RAW)" replayed successfully	Medium
CS8	Smartphones + workstation (BLE)	Bluetooth LE, 2.4 GHz	BLE advertising spam (Kitchen Sink / SwiftPair)	Persistent UI DoS on iOS/Android/Windows	Medium**
CS9	End user / credentials (Rogue AP)	802.11 open AP + captive portal	Evil twin + DNS sinkhole phishing	Plaintext credentials harvested; no OS warning	High
CS10	Windows workstation (BadUSB)	USB HID (wired)	DuckyScript keystroke injection	34 commands injected in 5.855 s; bypassed AV/firewall	Critical

** Critical, contingent on PSK strength. ** Medium on patched devices; historically Critical on unpatched iOS 17.0–17.1.*

Two patterns stand out across the table. First, the attacks that achieved the highest impact rarely involved breaking cryptography; they exploited its absence (CS1, CS3, CS7), its misconfiguration (CS4, CS5), or the trust a system placed in unauthenticated input (CS9, CS10). Second, the defenses that held — the PMF-protected router in CS5, the correctly implemented rolling code in CS2 — did so not by chance but because a sound, available protection had actually been deployed. These observations are examined in depth in Chapter 7. First, however, Chapter 6 extends the analysis beyond the laboratory, to the advanced attacks that could not be reproduced here but which define the broader threat landscape.

6. Secondary Research (OSINT) & Advanced Attacks

Laboratory experiments offer empirical certainty, but they operate within strict boundaries. Testing a deauthentication attack against commercial aviation systems, or executing a signal-jamming exploit against a moving vehicle on a public road, falls well outside the scope of what academic research can ethically or safely reproduce. To bridge the gap between controlled laboratory conditions and the broader reality of deployed threats, this chapter draws on Open-Source Intelligence (OSINT) to examine advanced cyber-physical attacks that have been documented, demonstrated, and in some cases actively exploited within the security community.

Where Chapter 3 established the technical mechanics of each attack class, this chapter examines how those attacks have manifested in documented real-world incidents, drawing on publicly available disclosures, CVE databases, and conference presentations.

6.1 The Importance of Secondary Research (OSINT) in IoT Security

Open-Source Intelligence refers to the systematic collection and analysis of publicly available information to produce actionable findings [50]. In IoT cybersecurity specifically, OSINT is not a supplement to primary research—it is often the only practical mechanism for tracking how vulnerabilities evolve between their initial discovery and eventual remediation.

6.1.1 Documenting Real-World Breaches

Formal academic publishing moves slowly. Security research does not. Conference presentations at DEF CON and Black Hat, technical blogs maintained by independent researchers, and dedicated vulnerability disclosure forums frequently serve as the first public record of critical hardware flaws—often months or years before a manufacturer issues a patch, if one is issued at all. Reviewing these sources systematically allows researchers to analyze an attack's mechanics at the point of disclosure, rather than waiting for it to appear in peer-reviewed literature. It also reveals how threat actors adapt laboratory-grade concepts into tools that can be carried into the field and used without specialist knowledge.

6.1.2 The Role of Open-Source Communities

The accessibility of modern penetration testing owes a great deal to open-source collaboration. GitHub repositories host specialized firmware forks, custom attack scripts, community-maintained signal databases, and fully documented protocol reversals. A researcher who would have previously spent weeks reverse-engineering a proprietary encoding scheme can now find a published demodulation analysis with a search. That is genuinely useful for defensive research. It is equally useful for anyone with less constructive intentions. The same sharing that accelerates patch development also converts theoretical vulnerabilities into ready-to-run exploits, available to anyone willing to look for them.

6.2 Specialized Attacks on Vehicle RKE Systems (Vehicle Security)

Automotive security depends heavily on Sub-GHz radio communications—specifically for Remote Keyless Entry (RKE) and Passive Keyless Entry and Start (PKES) systems. Vehicles are high-value targets, and the attacks documented against these systems represent some of the most technically sophisticated applications of radio frequency exploitation currently known.

6.2.1 The RollJam Attack

RollJam was first publicly demonstrated by security researcher Samy Kamkar at the DEF CON 23 hacking conference [15], and it remains one of the most elegant attacks on rolling code systems. The attack requires a device capable of transmitting and receiving simultaneously—a Software Defined Radio or a suitably modified multi-tool. Setup is simple: the attacker places the device near the target vehicle and waits. When the owner presses their key fob, the attack device transmits a burst of noise on the car's exact receiving frequency, effectively blinding the receiver. At the same time, it listens on a slightly wider frequency band and records the valid hopping code. The car does not unlock. The owner, assuming a momentary fault, presses the button again. The device jams and records this second code as well—then immediately stops jamming and replays the first captured code to the car. The car unlocks. The owner drives away, none the wiser. The attacker is left holding the second code. It is valid. It has never been processed by the receiver. It can be used at any point in the future to unlock the vehicle. One brief roadside encounter. Indefinite future access.

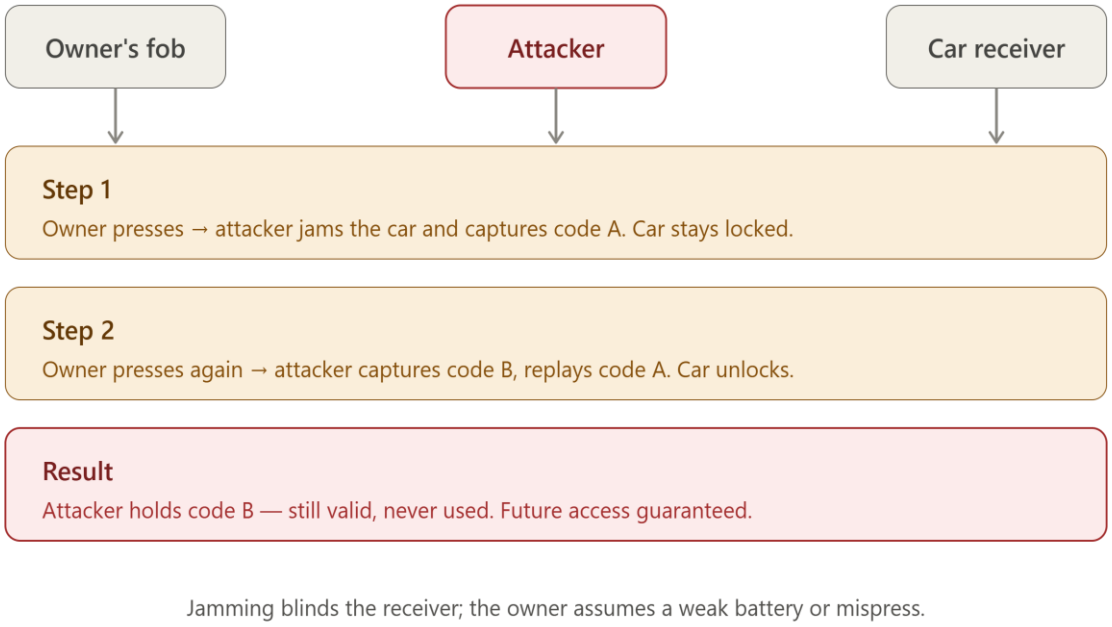

Figure 6.1: The RollJam attack. By jamming and capturing two consecutive codes while replaying the first, the attacker retains an unused valid code for future access.

6.2.2 The RollBack Attack

Where RollJam requires active interference, **RollBack** requires none. It is a purely passive attack—at least in the capture phase—and it targets a specific logical flaw in how certain rolling code receivers handle resynchronization, as documented in the research presented at Black Hat 2022 [16]. A related but distinct automotive weakness stems from systems that omit rolling codes altogether: the Honda RKE flaw CVE-2022-27254, where a static, unencrypted signal is transmitted for every command, leaving it trivially vulnerable to simple replay [51]. The attacker records a sequence of consecutive unlock signals over a period of time, simply by being within range when the owner uses their key. No jamming, no interaction, no detection risk. These captured codes are then replayed in a rapid, precisely ordered sequence. Certain receiver implementations interpret this pattern as a legitimate resynchronization event—a known recovery mechanism designed to handle cases where the transmitter and receiver have drifted apart in counter value. The receiver rolls its internal counter back to an earlier state, and codes that should have expired become valid again.

The resynchronization mechanism, built into the system as a usability feature, becomes the entry point. This is not merely a theoretical concern observed in automotive disclosures. The empirical Somfy RTS analysis in Chapter 5 (Case Study 2) demonstrated the same underlying weakness first-hand: a captured code (counter 2131) was rejected on replay, yet a later code (counter 2137) was accepted within the receiver's permissive forward-synchronization window. The principle is identical across both cases—the tolerance deliberately built into rolling-code systems to keep them usable, accommodating the counter drift that occurs when a fob is pressed out of range, is precisely the property that undermines them. RollBack weaponizes that tolerance against automotive receivers; the Somfy result confirms the same class of weakness on commodity hardware in a building-automation context.

6.2.3 Relay and Reflection Attacks

RollJam targets an active button press. **Relay attacks** go further—they target systems that do not require any user action at all. Passive Keyless Entry and Start (PKES), standard on a large share of modern vehicles, unlocks and starts the car simply by detecting the physical proximity of the key fob [19]. No button press. No deliberate authentication. Just presence. The attack exploits this by eliminating the distance constraint. The victim parks their car, locks it, and walks inside, leaving the key fob somewhere near the front door. Attacker A stands beside the car with a transceiver. Attacker B positions themselves outside the house, close to the door, with a second device. The car continuously broadcasts a low-frequency challenge signal searching for the key. Attacker A captures this challenge and relays it over a secondary radio link to Attacker B. Attacker B rebroadcasts the challenge toward the house. The key fob receives it, calculates the correct cryptographic response—because as far as it can determine, the car is right there—and transmits that response back. It travels the relay chain in reverse: Attacker B to Attacker A to the car. The doors unlock. The engine starts. No key was duplicated. No cipher was broken. The authentication protocol executed exactly as designed. The only thing the attackers changed was the apparent physical distance between the key and the car.

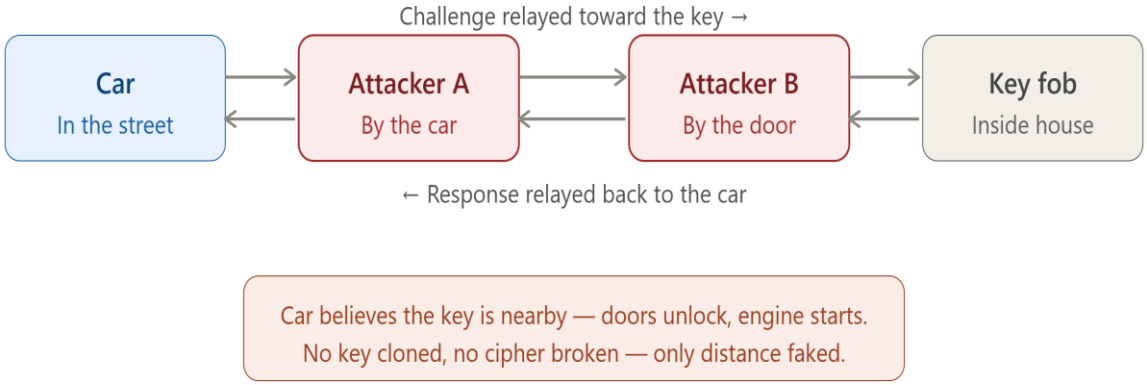

Figure 6.2: A relay attack on passive keyless entry. Two attackers relay the challenge-response exchange between the car and a distant key fob, faking proximity to unlock the vehicle.

6.2.4 Tire Pressure Monitoring Systems (TPMS) Telemetry

A separate Sub-GHz channel on the same vehicles deserves mention: the **Tire Pressure Monitoring System**. Mandated on new cars across many jurisdictions, a direct **TPMS** places a battery-powered sensor in each wheel that broadcasts pressure, temperature, and a unique identifier to the vehicle—on the 433.92 MHz band in Europe. The foundational analysis by Rouf et al. at USENIX Security 2010 remains definitive [52]. It established that these transmissions carry no authentication and that the vehicle trusts whatever it receives, with no real input validation. Two consequences follow: the static, clear-text sensor IDs turn each wheel into a persistent tracking beacon, readable at up to 40 metres from a moving car; and spoofed messages can be injected to raise false low-pressure warnings, a trivial route to driver distraction [52].

What tempers the threat is the sensors' own design rather than any security control. To preserve battery life they sleep when the vehicle is stationary and transmit only short, low-power bursts once wheel rotation passes roughly 20–40 km/h—which is why a parked car radiates almost nothing, and the dashboard merely shows values cached from the previous drive. Passive capture with commodity hardware therefore demands either a moving vehicle or a deliberate trigger such as an abrupt pressure change, not a simple roadside sweep. The vulnerability is real and unpatched in design, but its exploitation is rate-limited by power-saving engineering—the same incidental-

protection pattern seen with BLE in Section 6.3.1, where safety emerges as a by-product of implementation rather than intent.

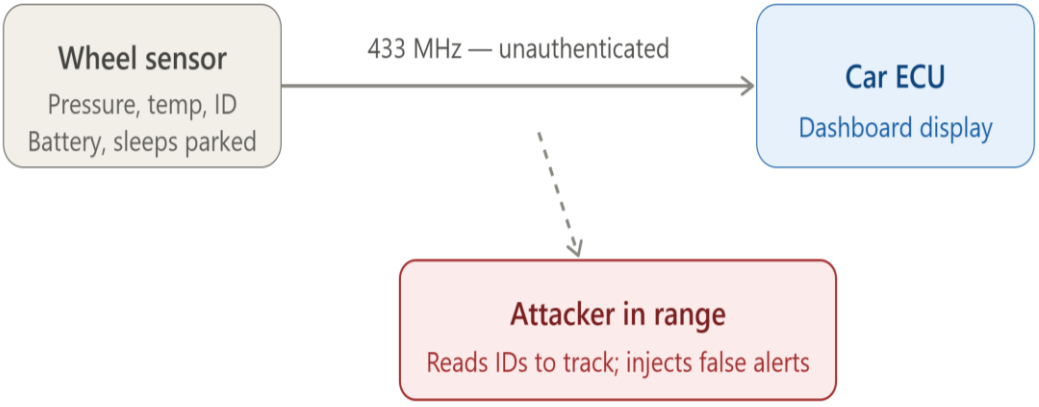

Figure 6.3: TPMS telemetry and its interception. Wheel sensors broadcast unauthenticated pressure, temperature, and ID data over 433 MHz, which an attacker in range can read to track the vehicle or inject false warnings.

6.3 Vulnerabilities and Threats in Proximity Protocols (Bluetooth / BLE)

Automotive RKE attacks target a specific, relatively contained ecosystem. BLE vulnerabilities operate at an entirely different scale. Personal wearables, smartphones, medical sensors, industrial monitors, smart home devices—the list of BLE-dependent hardware is effectively endless, and the protocol underpinning all of it is built on an implicit assumption: that proximity implies trust. Secondary research consistently shows that breaking this trust does not require cryptographic sophistication. Attackers routinely exploit the protocol's own normal behaviors rather than its encryption.

6.3.1 BLE Spamming and Flooding Attacks

To enable seamless device discovery, major mobile ecosystems—Apple's iOS and Google's Android among them—continuously monitor the radio environment for specific BLE advertising packets. This is a fundamental feature of how the protocol works. It is also an attack surface. Documented cases analyzed through OSINT, including open-source proof-of-concept repositories like "AppleJuice" [53], show that attackers use portable microcontrollers to broadcast thousands of spoofed advertising packets per

second. This simulates the simultaneous presence of large numbers of nearby devices—AirTags, AirPods, smart televisions, generic sensors. The receiving device cannot distinguish these from legitimate advertisements and begins processing each one. It cannot keep up. Battery depletes rapidly. The user interface locks under a cascade of pairing notifications.

In the most severe documented cases—affecting unpatched iOS 17.0–17.1 devices, the flaw later catalogued as **CVE-2023-42941**—the host operating system crashed entirely and rebooted. It should be noted, consistent with the empirical results in Chapter 5 (Case Study 8), that this catastrophic behaviour is no longer representative of current devices. Subsequent operating-system updates introduced notification rate-limiting that has contained the full crash-and-reboot outcome; on the patched hardware tested in this work, the attack produced only the persistent interface-level denial of service—polluted device lists and a flooded notification centre—rather than a system collapse. The distinction matters for accurate threat modelling: the attack remains a reliable availability and usability threat, but the historical "phones reboot en masse" claim now applies only to unpatched legacy devices.

What makes this particularly notable from a threat modeling perspective is the asymmetry. An attacker with a microcontroller that costs well under fifty dollars can render high-end consumer electronics effectively unusable—without ever completing a connection, without breaking a cipher, without touching the target device at all.

6.3.2 Connection Interception and GATTacking

When an attacker opts to establish an actual connection rather than flood the environment, the threat shifts inward to the application layer. Security audits published by independent researchers and academic groups have repeatedly identified the same failure pattern across commercially available BLE devices: GATT Characteristics controlling physical functions are left exposed without requiring cryptographic pairing or authentication credentials of any kind. Prior to establishing a connection, passive BLE sniffing tools can capture unencrypted advertising data and, in some implementations, intercept unprotected characteristic traffic, providing the attacker with device identifiers and behavioral patterns without any active interaction. The exploit sequence is straightforward. The attacker scans for the target device, connects to it directly—no authentication challenge is presented—identifies the Characteristic responsible for a

physical output such as a door lock relay, and writes a value to it. Sending a single byte to the correct memory address is sufficient to unlock the device. The Bluetooth link itself may be encrypted. It does not matter. The encryption secures the communication channel; it does nothing to restrict what an authenticated-by-default connected device is permitted to do. The application layer never checks whether the connected party is authorized. It simply executes the command.

6.3.3 Automated Code Injection (HID Attacks)

The most operationally severe Bluetooth vulnerability documented in recent years is not an attack on the encryption at all. It targets the operating system's handling of the Human Interface Device protocol—the same mechanism that makes wireless keyboards and mice function. **CVE-2023-45866**, a critical vulnerability discovered and disclosed by security researcher Marc Newlin in late 2023 [22], exposed a structural flaw affecting multiple platforms, including specific versions of Android, macOS, and Linux. The vulnerability allows an attacker to force a target device to accept a Bluetooth HID connection from a rogue peripheral masquerading as a keyboard, bypassing the standard user confirmation prompt entirely. The user does not approve the pairing. The device accepts it anyway. From that point, the attack is functionally identical to a physical BadUSB insertion. Tools like BlueDucky [54] automate the keystroke injection sequence—opening terminals, downloading remote access trojans, establishing persistent backdoors—within seconds of the connection being accepted. A proximity-based vulnerability with no physical access required produces an outcome indistinguishable from direct system compromise. The empirical BadUSB assessment in Chapter 5 (Case Study 10) demonstrates the wired equivalent of precisely this attack; the only difference is the transport, since the underlying exploitation of the HID trust model is the same.

6.4 Comparative Threat Analysis

Collecting vulnerability disclosures through OSINT is only useful if the findings are interpreted in context. A documented attack is not automatically an active threat. The gap between theoretical possibility and practical exploitation varies enormously depending on the resources, knowledge, and equipment the attack actually requires.

6.4.1 Theoretical Risk vs. Practical Exploitation

Consider the contrast between two well-documented attack classes. Recovering a KeeLoq encryption key via differential power analysis is theoretically possible and has been demonstrated in controlled research settings. It also requires physical access to the transmitter hardware, laboratory-grade oscilloscopes, and a level of mathematical expertise that essentially restricts this attack to well-resourced specialists. High potential impact, very low probability of execution in the field. BLE Spamming requires none of that. Neither does RFID cloning. Both can be executed with hardware available for under fifty dollars, with no cryptographic background, in under a minute. The practical risk landscape is not shaped by the most mathematically complex attacks—it is shaped by the most accessible ones. Complexity is a poor proxy for danger when accessibility determines who is actually capable of executing an attack.

6.4.2 The Impact of Portable Penetration Tools

The shift toward accessible, low-skill attacks is not coincidental. It reflects a structural change in how penetration testing hardware has evolved over the past decade. Executing a RollJam attack once required custom-built SDR platforms and non-trivial programming work. Conducting a BLE flood required writing custom firmware from scratch. That barrier has largely disappeared. Devices like the Flipper Zero consolidate these capabilities into a single portable unit with an interface accessible to non-specialists, while open-source communities continuously translate newly published vulnerabilities into ready-to-flash firmware modules. For security researchers and penetration testers, this is straightforwardly valuable—it accelerates evaluation cycles and lowers the cost of thorough assessments. The same development also means that attack categories previously associated with state-sponsored actors or highly specialized criminal groups are now within reach of a far broader population. Threat models that treated certain attacks as theoretical edge cases need to be revisited. For everyday IoT deployments, they are not edge cases anymore.

6.4.3 Regulatory Response and the Threat-Inflation Problem

Accessibility cuts in an unexpected direction too. The same visibility that makes these tools easy to obtain also makes them easy to misunderstand, and the regulatory reaction to portable multi-tools has at times outpaced the technical reality. The clearest example is the Flipper Zero itself. In February 2024, following a national summit on auto

theft, the Canadian Minister of Innovation, Science and Industry announced an intention to ban the importation, sale, and use of "consumer hacking devices, like flippers," naming the Flipper Zero specifically as a tool used to steal cars [55]. The device had already been removed from sale by Amazon in 2023, classified as a card-skimming device [56].

The technical premise of the ban did not survive scrutiny. The stated justification was that the device could defeat keyless entry and start systems, yet—as security researchers and the device's manufacturer pointed out—a simple capture-and-replay tool cannot defeat the rolling-code systems fitted to essentially every car built in the last three decades [56]. This thesis provides direct empirical confirmation of that limitation. The fixed-code system in Case Study 1 (a CAME 12-bit gate at 433 MHz) was trivially replayed, but the rolling-code Somfy system in Case Study 2 resisted naïve replay exactly as the protocol intends; defeating it required exploiting the synchronization window rather than a simple capture. Independent peer-reviewed work reaches the same conclusion, confirming openings against fixed-code Sub-GHz systems while reporting that rolling-code implementations blocked replay and brute-force attempts on the same hardware [31].

The episode is instructive beyond the specific device. It illustrates a recurring pattern in which the publicity surrounding an accessible tool inflates its perceived capability well past what it can actually do, prompting policy responses aimed at the instrument rather than the underlying insecurity—in this case, automotive immobilizer and key-fob designs that remain the real point of failure. For the purposes of this thesis, the lesson reinforces the comparative framing of Section 6.4.1: an accurate threat model must rest on demonstrated technical capability, not on media narrative or the symbolic appeal of banning a recognizable object. Banning the tool leaves the vulnerable systems exactly as vulnerable as before.

To provide a structured overview of these advanced threats, Table 6.1 synthesizes the operational characteristics of the attacks discussed in this chapter, highlighting the disparity between technical complexity and the resources required for execution.

Table 6.1: Comparative Analysis of Advanced IoT Attack Vectors

Attack Type	Target Protocol	Technical Complexity	Hardware Cost	Requires Active Jamming	Physical Proximity Required
RollJam	Sub-GHz (Rolling Code)	Moderate	Low (<$50)	Yes	Yes
RollBack	Sub-GHz (Rolling Code)	Low	Low (<$50)	No	Yes
TPMS Sniffing / Spoofing	Sub-GHz (433.92 MHz)	Low	Low (<$50)	No	Yes (vehicle in motion)
Relay Attack	LF / UHF (PKES)	Moderate	Moderate (~$100)	No	Yes (Two-person)
BLE Spamming	Bluetooth LE	Low	Very Low (<$20)	No	Yes
GATTacking	Bluetooth LE (GATT)	Low	Low (<$50)	No	Yes
HID Injection	Bluetooth HID	Low (Automated)	Low (<$50)	No	Yes

6.5 Synthesis

The attacks surveyed in this chapter share a defining characteristic visible throughout Table 6.1: a persistent disparity between their technical sophistication and the resources they demand. Attacks capable of stealing a car or hijacking a Bluetooth session require, in most cases, under a hundred dollars of hardware and little specialist expertise — the barrier is knowledge, not equipment. This is the same pattern established empirically in Chapter 5, now confirmed across the wider threat landscape that the laboratory could not directly reproduce.

Two themes carry forward. First, accessibility has decoupled capability from cost: the tools of attack are no longer confined to well-funded actors, and any realistic threat model must account for a broad population of low-budget adversaries. Second, and running through both this chapter and the last, the vulnerabilities being exploited are overwhelmingly ones with known remedies — rolling codes implemented correctly, authenticated protocols, validated input, hardened configuration. The problem is rarely

the absence of a solution. Having established both what these tools can do in practice (Chapter 5) and how the threat extends beyond the laboratory (Chapter 6), the analysis now turns to that gap directly: Chapter 7 examines the countermeasures that address these recurring weaknesses at their root.

7. Countermeasures and Mitigation Strategies

The previous two chapters were concerned with breaking things. Chapter 5 broke them in the lab, with hardware in hand; Chapter 6 documented how others have broken them in the field. This chapter turns the question around. Given everything that failed, what actually works — and why do some defenses hold while others collapse on contact?

7.1 From Attack to Defense

A natural way to write a countermeasures chapter would be to walk back through each technology in turn: here is how to secure Sub-GHz, here is how to secure RFID, here is how to secure Wi-Fi, and so on. That structure is tempting because it mirrors the case studies, but it is the wrong one. It would force the same advice to be repeated in section after section — change the default keys, authenticate the sender, do not trust unvalidated input — because the same few weaknesses appear again and again across protocols that otherwise have nothing in common.

The more useful observation, and the one this chapter is built around, is that the ten attacks documented in this thesis do not represent ten different problems. They represent a handful of recurring failures, dressed in different protocols. A garage gate, a hotel-style Wi-Fi portal, and a USB keyboard injection look unrelated until you ask what each one actually exploited — and the answers rhyme. This chapter therefore organizes its defenses by principle rather than by technology. Each countermeasure is presented as the answer to a root cause, and each root cause is one that the earlier experiments demonstrated directly. The aim is not a catalogue of fixes but an argument: that securing the IoT perception layer is less about chasing individual vulnerabilities than about addressing the small number of design habits that produce them.

7.2 The Root Causes: A Synthesis

Before prescribing defenses, it is worth naming the diseases. Reviewing the empirical results of Chapter 5 alongside the documented attacks of Chapter 6, five root causes account for almost everything that went wrong.

Absence of origin authentication. The single most common failure. Nothing in the signal lets the receiver confirm who sent it. The CAME gate (Case Study 1) accepted any transmission carrying the right fixed code, regardless of source. The Wi-Fi

deauthentication attack (Case Study 5) worked because 802.11 management frames are unauthenticated by default. Infrared appliances (Case Study 7) actuate on any correctly timed pulse train. In every case the receiver answered the wrong question — "is this message well-formed?" — instead of the right one: "is this message from someone I trust?"

Static, replayable signals. Where the secret never changes, capturing it once is equivalent to holding it forever. The fixed-code gate, the 125 kHz RFID badge (Case Study 3), and the MIFARE card's cloned identity (Case Study 4) all fell to this: a single capture, replayed, reproduced full authority. Even rolling codes, designed precisely to defeat replay, proved vulnerable where the resynchronization window was too forgiving — as the Somfy result (Case Study 2) and the documented RollBack attack both showed.

Implicit trust in input. Several systems failed not because their cryptography was weak but because they never questioned what they were given. The operating system accepts a USB device's claim to be a keyboard and executes whatever it types (Case Study 10). Mobile platforms parse and display every BLE advertising packet they hear (Case Study 8). A phone's own connectivity check hands control of the screen to whatever captive portal answers it (Case Study 9). The trust is granted before any verification takes place, and the attack simply supplies input the system was always going to believe.

Weak or default cryptographic configuration. Sometimes the protocol was sound and the deployment undid it. The MIFARE Classic card (Case Study 4) used CRYPTO1, but its keys were the manufacturer's defaults, never changed at installation — so the cipher was irrelevant. The cracked WPA2 network (Case Study 5) used strong AES, defeated by a weak, dictionary-present passphrase. The strength on paper meant nothing once the configuration gave it away.

The human factor. The last cause cannot be patched in firmware. The evil portal (Case Study 9) did not defeat a machine; it persuaded a person to type their password into a convincing fake. No cryptographic measure addresses a user who voluntarily surrenders the credential. This root cause is singled out here because the defenses against it, discussed in Section 7.6, are categorically different from the rest.

The sections that follow take these causes in a deliberate order — from the defenses that are purely technical (7.3), through those that are matters of configuration

and discipline (7.4, 7.5), to those that are physical and human (7.6) — before drawing them together into a layered model (7.7).

7.3 Cryptographic and Protocol-Level Defenses

The first two root causes — missing authentication and static, replayable signals — are failures of protocol design. They are also the ones with the cleanest fixes, because the cryptographic tools to address them already exist and are, in several cases, already standardized. The problem is rarely that a solution is unknown; it is that the vulnerable system predates it, omits it, or ships with it disabled.

7.3.1 Authenticating the Sender

The recurring failure throughout Chapter 5 was that receivers verified the form of a message but never its origin. The defense is to make authentication intrinsic to the protocol, so that a well-formed message from an untrusted source is rejected as readily as a malformed one.

The clearest illustration in this thesis is one the experiments produced directly. In the Wi-Fi case study (Case Study 5), the deauthentication attack succeeded against the legacy extender but failed completely against the PMF-enabled router. Protected Management Frames, introduced in the IEEE 802.11w amendment [13], cryptographically authenticate management frames — including the deauthentication and disassociation frames the attack relied on — so that a spoofed frame lacking the correct Message Integrity Code is simply rejected. That was not a theoretical defense; it was observed working in the lab. The lesson generalizes: the same deauthentication primitive that has existed since the beginning of Wi-Fi is neutralized not by patching each attack but by authenticating the frame class the attack abuses.

WPA3 makes this the default rather than an option. Where WPA2 left Protected Management Frames optional — and they were widely left disabled — WPA3 makes them mandatory [57], removing an entire class of denial-of-service attack that had persisted for the lifetime of the standard. For the perception-layer RF systems examined in Chapter 5 — gates, shutters, and the like — the equivalent principle is a rolling code or challenge-response scheme implemented correctly, so that no captured transmission can be replayed for authority it should no longer hold.

7.3.2 Defeating Replay: Rolling Codes and Their Correct Implementation

Static secrets are indefensible once captured, as the fixed-code gate (Case Study 1) and the cloned RFID badge (Case Study 3) demonstrated. The standard answer is to make every valid transmission single-use, so that a captured code is worthless the moment it has been spent. Rolling codes do exactly this, and the contrast with the fixed-code gate is the strongest argument in the thesis for their adoption: the same replay that opened the CAME gate instantly failed against the Somfy system (Case Study 2), because the captured counter value had already been consumed.

But Case Study 2 also exposed the caveat that makes this section necessary. A rolling code is only as strong as its resynchronization logic. The Somfy receiver accepted a forged code with an advanced counter because its forward-synchronization window — the tolerance that lets a fob pressed out of range stay usable — was too permissive. The documented RollBack attack (Chapter 6) exploits the same logic against vehicles. The countermeasure is therefore not merely "use rolling codes" but "implement the window conservatively": keep the acceptance window as narrow as usability allows, and pair it with a resynchronization procedure that cannot be driven backwards by a replayed sequence. A rolling code with a careless window offers the appearance of replay protection without the substance.

7.3.3 Closing the Offline-Cracking Path

A subtler protocol weakness surfaced in the WPA2 crack (Case Study 5): the handshake could be captured passively and attacked offline, at GPU speed, with no rate limit, so the network's entire security reduced to the strength of one passphrase. WPA3 addresses this at the protocol level [57] through Simultaneous Authentication of Equals (SAE), a Dragonfly-based key exchange in which the pairwise key is no longer derived solely from the passphrase [58] — meaning a captured handshake can no longer be subjected to the offline dictionary attack that recovered the password in thirteen seconds. SAE additionally provides forward secrecy: each session uses ephemeral keys, so compromising one passphrase does not retroactively expose past captured traffic.

The general principle behind all three subsections is the same. A protocol should never place the burden of its security entirely on a secret that can be captured and attacked at the adversary's leisure — whether that secret is a static code, a replayable

counter, or a passphrase exposed to offline guessing. Authentication, single-use credentials, and key-exchange schemes that resist offline attack are the protocol-level expression of a single idea: capturing the conversation should not be the same as winning it.

7.4 Configuration and Operational Hardening

The defenses in the previous section assume the protocol itself is the weak point. Often it is not. Several of the most complete compromises in Chapter 5 succeeded against cryptographically sound systems that were simply deployed badly. This is an uncomfortable but important finding, because it shifts responsibility from the protocol designer to the installer and the operator — and the fixes cost nothing but discipline.

7.4.1 The Default-Credential Problem

The MIFARE Classic case (Case Study 4) is the cleanest example in the thesis of a configuration failure masquerading as a cryptographic one. The card's CRYPTO1 cipher was never broken. The dictionary attack succeeded because every one of the card's sector keys was still set to the manufacturer's default transport value, never changed when the access system was installed. A correctly keyed card would not have fallen to that attack at all. The cipher was sound; the deployment threw it away.

This is not an isolated quirk. Weak, guessable, or hardcoded default credentials sit at the very top of the OWASP IoT Top 10 list of vulnerabilities [59], and for good reason: IoT devices frequently ship with identical factory credentials that are publicly documented in the product manual, and many users — or installers — never change them [59]. The same pattern enabled the WPA2 crack (Case Study 5), where the network's AES encryption was intact but the passphrase was weak enough to sit inside a public wordlist, and it is the mechanism by which large-scale IoT botnets such as Mirai have historically spread, simply by trying lists of well-known default username-and-password pairs. The countermeasure is procedural, not technical: every device must be provisioned with unique, non-default credentials at installation, and systems should where possible refuse to operate until factory defaults have been changed [59].

7.4.2 Disabling Unnecessary Attack Surface

A second configuration lesson came from the WPS evaluation (Case Study 6). Wi-Fi Protected Setup is a convenience feature, and on the target router it was active — yet

the specific PIN-based attack did not apply, because the device was configured for push-button (PBC) mode only, with no static PIN exposed. That is configuration working for security: a feature whose dangerous mode had been disabled could not be attacked through that mode. The general principle is the inverse of the default-credential one. Where defaults should be changed, unnecessary features and services should be switched off entirely. Every enabled-but-unused capability — a legacy pairing mode, an open management port, a debugging interface — is attack surface that earns its keep only if something actually needs it. Reducing that surface is among the cheapest and most effective hardening steps available [59].

7.4.3 Configuration as a First-Class Security Control

Taken together, these cases reframe what "secure" means in practice. A system is not secure because it can be configured securely; it is secure only when it actually has been. The MIFARE card could have been keyed properly. The WPA2 network could have used a strong passphrase. The protocols supported all of this; the deployments did not use it. For an installer or administrator, the implication is that configuration is not a setup detail to be rushed through but a first-class security control, equal in weight to the choice of protocol. The most sophisticated cipher in the world protects nothing if it is left guarding a door whose key was never changed from the one printed in the manual.

7.5 Rate-Limiting and Input Validation

The third root cause — implicit trust in input — produced some of the most striking results in Chapter 5, and it is the one whose defenses are the most distinctively modern. The attacks in this category did not break anything. They simply fed a system more of what it was already willing to accept, faster than it was prepared to handle, or in a context it never thought to question. The defenses, correspondingly, are not about stronger locks but about a system learning to doubt its own inputs.

7.5.1 Throttling the Flood

The BLE spam attack (Case Study 8) is the clearest case, and its defense is one this thesis observed in two states. The attack works because mobile operating systems parse and react to every BLE advertising packet they hear, in the name of frictionless pairing. Early versions of the attack were severe enough to crash and reboot iPhones outright, a flaw catalogued as CVE-2023-42941 [47]. The fix Apple shipped is instructive

precisely because of what it is not: the pairing convenience was not removed, and the advertising channel was not authenticated. Instead, the operating system was taught to rate-limit — to cap how many pairing prompts it would raise in a given window, so that a flood of advertisements could no longer translate into an unbounded flood of notifications [47].

The empirical result in Case Study 8 captured both sides of that fix. On unpatched devices the attack was catastrophic; on current, rate-limited devices the same attack produced only interface clutter, never a crash. This is the defining property of rate-limiting as a countermeasure: it does not eliminate the attack, but it caps the damage, converting a system-level collapse into a bounded annoyance. The same logic answers the beacon-flood variant (Case Study 6), where the defense is again not to stop listening for beacons but to limit how aggressively the interface reacts to a sudden multitude of them.

7.5.2 Validating the Source of Input

Throttling limits the volume of trusted input. The deeper fix is to question the legitimacy of the input at all. Two case studies point to this. The captive-portal attack (Case Study 9) succeeded because the operating system's own connectivity check faithfully displayed whatever page answered it, with no scrutiny of whether that page was trustworthy. The BadUSB attack (Case Study 10), discussed further in the next section, succeeded because the system accepted a device's bare claim to be a keyboard. In both, the failure was the same: input was acted upon before its source was validated.

The countermeasure is to insert that validation. For captive portals, modern operating systems increasingly sandbox the portal in a restricted mini-browser with no access to saved credentials or other sites, and warn that the connection is unencrypted — narrowing what a malicious portal can plausibly ask for. The broader principle, sometimes called a zero-trust posture toward input, is that a convenience feature should treat unsolicited input as untrusted by default, validating its source or constraining its privileges before acting, rather than extending trust automatically and revoking it only after something goes wrong.

7.5.3 Why This Class of Defense Is Different

It is worth drawing out why these defenses feel unlike those in Section 7.3. Cryptographic countermeasures aim to make an attack impossible — a forged frame is

rejected, a captured handshake cannot be cracked. Rate-limiting and input validation rarely achieve that. They make an attack survivable: the BLE flood still floods, but the phone no longer dies; the rogue portal still appears, but it can no longer reach the user's saved passwords. For the convenience features that define the modern IoT and mobile experience — features that exist precisely because they act on input automatically — this is often the only kind of defense available, since the alternative is to remove the convenience entirely. It is a pragmatic acceptance that some attack surface cannot be eliminated, only contained.

7.6 Physical and Policy Controls

The defenses so far have lived in firmware, protocols, and configuration. The final two root causes resist all of them. No cryptographic scheme stops an attacker who has physical access to an unlocked machine, and no patch fixes a human being who chooses to type their password into a convincing fake. These are the cases where the answer is not code but policy, hardware, and people.

7.6.1 Defending Against Physical Interface Attacks

The BadUSB attack (Case Study 10) is the purest example of a threat that software cannot see. The injection succeeded in under six seconds, and antivirus and firewall did nothing — not because they failed, but because from the operating system's perspective nothing abnormal occurred. A keyboard was attached and keys were pressed. There is no malicious file to scan and no suspicious network connection to flag; the input is, by definition, legitimate.

Because the threat is invisible at the software layer, the defenses operate at the hardware-policy layer instead. The most direct is USB device control — often called endpoint device control or USB whitelisting — which the literature identifies as the standard countermeasure for this attack class [60]. The critical insight, repeatedly stressed in the practitioner literature, is that blocking USB storage does not block BadUSB: the malicious device never presents as storage, but as a keyboard, so storage-focused policies miss it entirely [60]. Effective controls instead operate on the Human Interface Device class itself — for example, refusing to accept a newly attached keyboard unless its hardware identifier appears on an approved list, so that a device masquerading as a keyboard but lacking a recognized vendor ID is denied before it can type a single

character [60]. A complementary, behavioral approach detects the attack by its signature rather than its identity: a genuine human types at human speed, while an injection device emits hundreds of keystrokes per second, and a host that flags or throttles input arriving faster than any person could produce it can catch even a whitelisted-looking device [61].

Alongside these technical controls sits the simplest and most often-cited mitigation of all: lock the screen. The BadUSB attack requires an unlocked, logged-in session; a workstation that locks automatically when unattended removes the precondition the entire attack depends on. It is a policy measure, not a technical one, and it costs nothing.

The same physical-bound logic, in a gentler form, applies to the infrared and proximity-RF attacks elsewhere in the thesis. Infrared (Case Study 7) requires direct line of sight, and RFID and NFC (Case Studies 3 and 4) require the reader to be brought within centimetres of the credential. These are not strong protections — the thesis showed all three being defeated — but they do mean the attacker must achieve physical proximity, and controlling physical access (shielding badges, limiting reader exposure, situating IR-controlled devices out of casual sightlines) remains a meaningful, if partial, layer.

7.6.2 The Human Factor: Awareness as a Control

The evil portal (Case Study 9) sits outside every technical category in this chapter. It did not defeat a cipher, flood a buffer, or exploit a configuration. It convinced a person. The user connected to an open network named to look familiar, was shown a page designed to look like their own router, and typed their credentials into it. Every system involved behaved exactly as designed. The only thing that failed was the human judgment at the end of the chain.

No firmware update addresses this, which is why security awareness is itself a control rather than a soft adjunct to the real defenses. The relevant awareness is specific and teachable: legitimate networks do not ask for a password through a pop-up after connecting; a captive portal requesting the network's own administrative credentials is a contradiction that should raise immediate suspicion; and an open, unencrypted network is inherently untrustworthy for anything sensitive. These are learnable habits. The technical measures from Section 7.5 — sandboxed captive-portal browsers, unencrypted-network warnings — raise the floor, but they cannot replace a user who

recognizes the deception for what it is. In the layered model that follows, the human is correctly understood not as the weakest link to be worked around, but as a control to be strengthened like any other.

7.7 A Layered Model: Defense-in-Depth

The preceding sections each addressed a single root cause with a single class of countermeasure. That separation is useful for analysis, but it misrepresents how defense actually works. No real system is protected by one measure. Each of the countermeasures examined here has a gap, and the gaps are covered not by perfecting any one defense but by layering several so that the failure of one is caught by the next. This is the principle of defense-in-depth, and the case studies in this thesis make the argument for it almost by themselves.

Consider what each layer leaves uncovered. Cryptographic authentication (Section 7.3) defeats forgery and replay, but does nothing against a user who is socially engineered into surrendering a credential outright. Correct configuration (Section 7.4) closes the default-key and weak-passphrase gaps, but a perfectly configured system can still be flooded into a denial of service. Rate-limiting (Section 7.5) contains that flood, but does not stop a physical keystroke injection. Physical and policy controls (Section 7.6) address the injection and the human, but only if the screen-lock policy is actually enforced and the user actually internalizes the training. Every layer is necessary; none is sufficient. The WPA2 case study (Case Study 5) shows the constructive version of the same point: even there, two independent layers had to align for the attack to land — the protocol had to permit the deauthentication and the passphrase had to be weak. The PMF-protected router broke the first layer and the whole chain collapsed, regardless of password strength.

A practical consequence follows for anyone deploying these systems. Security should not be sought in a single decisive control but distributed across the stack: authenticate at the protocol layer, harden at the configuration layer, throttle and validate at the input layer, and constrain at the physical and human layer. An attacker must defeat every layer in sequence; the defender must hold only one. That asymmetry, finally, is the one that favours the defender — and it exists only when the layers are actually present together.

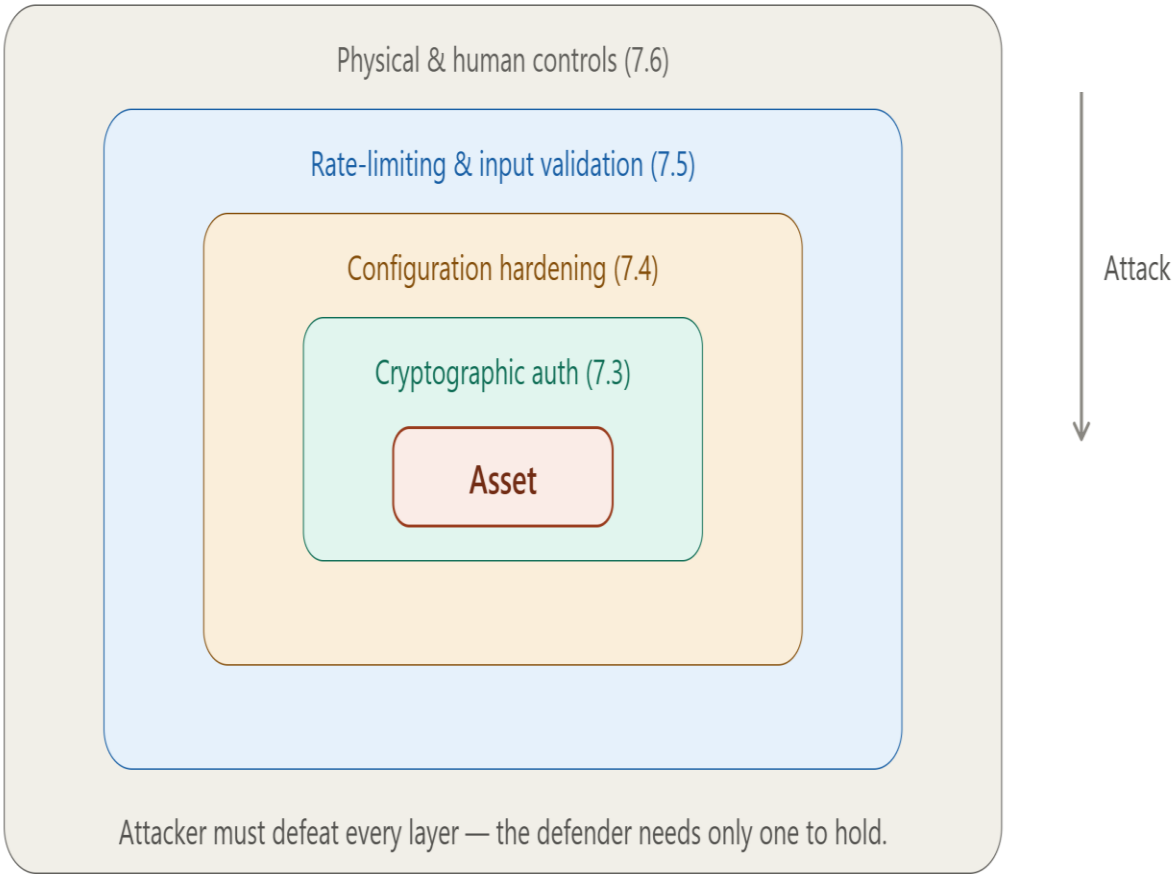

Figure 7.1: The defense-in-depth model. Each layer maps to a section of this chapter; an attacker must defeat all of them, while the defender needs only one to hold.

Table 7.1 draws the chapter together. It maps each empirical attack from Chapter 5 to the root cause it exploited and the principal countermeasure that addresses it, making explicit the throughline from demonstrated weakness to recommended defense.

Table 7.1: Mapping Empirical Attacks to Root Causes and Countermeasures

Case Study (Attack)	Root Cause Exploited	Principal Countermeasure	Section
CS1 — Fixed-code gate (CAME)	Static, replayable signal	Rolling code / single-use credential	7.3.2
CS2 — Rolling-code shutters (Somfy)	Replayable signal (permissive resync window)	Conservative synchronization window	7.3.2
CS3 — RFID badge (HID Prox)	Static credential, no authentication	Authenticated / single-use credential	7.3.1–7.3.2
CS4 — NFC card (MIFARE Classic)	Weak/default configuration (default keys)	Provision unique non-default keys	7.4.1
CS5 — Wi-Fi WPA2 (deauth + crack)	No frame authentication; weak passphrase	PMF / WPA3-SAE; strong credentials	7.3.1, 7.3.3, 7.4.1
CS6 — Beacon flood + WPS recon	Implicit trust in input; unneeded surface	Rate-limiting; disable unused features	7.5.1, 7.4.2
CS7 — Infrared (AC/TV)	No origin authentication	Authentication; line-of-sight control	7.3.1, 7.6.1
CS8 — BLE advertising spam	Implicit trust in input (unbounded)	OS rate-limiting	7.5.1
CS9 — Evil portal (rogue AP)	Human factor; implicit input trust	Sandboxed portal; user awareness	7.5.2, 7.6.2
CS10 — BadUSB (HID injection)	Physical access; implicit device trust	USB/HID device control; screen-lock	7.6.1

The table is the chapter's argument in miniature. Ten attacks, spanning six radio and physical technologies, reduce to five root causes and a correspondingly small set of countermeasures — and not one of those countermeasures is exotic. Every defense recommended here already exists, most of them standardized, several of them observed working in the thesis's own experiments. The gap that this work documents is not an absence of solutions. It is the distance between the defenses that are available and the defenses that are actually deployed in the field — a gap that, as Chapter 6 argued, is where the real-world risk lives.

8. Conclusions

This final chapter draws the investigation to a close. It returns to the question posed at the outset, assesses how fully the research met the objectives it set for itself, and reflects on what the results mean beyond the individual experiments. The chapter first summarizes the work and its principal findings, then measures those findings against the original research objectives and articulates the study's central contribution. It closes by considering the legal and educational implications of accessible penetration-testing hardware, acknowledging the limitations that bound the conclusions, and identifying the directions in which this line of work can productively continue.

8.1 Summary of the Research

This thesis set out to test a single, practical proposition: that the security of the IoT perception layer can now be meaningfully assessed — and meaningfully attacked — with one inexpensive, pocket-sized tool, rather than the laboratory of specialized equipment such work has traditionally required. The investigation pursued that proposition empirically, building a body of hands-on penetration tests around the Flipper Zero and a small set of supporting hardware, and then situating the results within the broader threat landscape.

The work unfolded in three movements. The empirical core (Chapter 5) documented ten case studies of practical attacks against real devices, spanning the wireless and physical technologies that define the perception layer: fixed- and rolling-code Sub-GHz systems, low- and high-frequency RFID and NFC, 802.11 Wi-Fi, infrared, Bluetooth Low Energy, captive-portal social engineering, and direct USB keystroke injection. Where the laboratory could not safely or legally reproduce certain threats — moving vehicles, large-scale infrastructure — the analysis turned to documented real-world attacks through open-source intelligence (Chapter 6). Finally, the experimental findings were synthesized into a structured set of countermeasures (Chapter 7), organized not by technology but by the small number of root causes the attacks repeatedly exposed. This concluding chapter draws those threads together and reflects on what they mean.

8.2 Key Findings

Across ten attacks and six distinct technologies, a handful of findings recur with enough consistency to be treated as the central results of the work.

Most attacks did not break cryptography — they exploited its absence or its misconfiguration. This is the most consistent thread in the entire body of experiments. The CAME gate, the RFID badge, and the infrared appliances had no meaningful authentication to break. The MIFARE card and the WPA2 network had strong cryptography that was simply undermined by default keys and a weak passphrase respectively. Genuine cryptanalysis was rarely necessary and rarely attempted; the path of least resistance ran through missing authentication and poor configuration almost every time. The implication is that the perception layer's insecurity is, for the most part, not a hard mathematical problem awaiting a solution but a deployment problem with solutions already in hand.

Adding intelligence to a device does not secure it. Several targets were "smart" or could be made so — the gate with an optional Wi-Fi module, the shutters paired with a home-automation hub, the appliances with companion apps. In every case the underlying radio channel remained exactly as exposed as it was before the connected features were added. A networked front-end layered on top of an unauthenticated RF channel does nothing to secure that channel; it merely adds a second attack surface above it. The perception layer's weaknesses sit beneath the "smart" features and are not addressed by them.

Configuration is as decisive as protocol design. Roughly half of the successful compromises succeeded not because the protocol was weak but because the deployment was. A sound cipher guarded by default keys, a strong encryption standard protected by a dictionary-present password — these are not protocol failures, and no protocol upgrade fixes them. This finding reframes responsibility: securing these systems is at least as much the task of the installer and operator as of the protocol designer.

Accessibility, not sophistication, defines the practical threat. The attacks that worked most reliably were not the most technically advanced; they were the most accessible. Every exploit in Chapter 5 was carried out with a sub-$200 device and freely available firmware, by a single operator, often in seconds. The threats that demand

specialist expertise and laboratory equipment — differential power analysis against KeeLoq, for instance — remain real but rare in the field. The risk that actually matters is the one that is cheap, fast, and requires no expertise, and that is precisely the risk this tool embodies.

8.3 Meeting the Research Objectives

Chapter 1 set out four objectives. It is worth returning to them directly, since a thesis should be judged in part on whether it did what it said it would.

The first objective was the **identification of protocol vulnerabilities** across the perception layer. This was met across the full range named at the outset and somewhat beyond it. Sub-GHz fixed and rolling codes (Case Studies 1 and 2), RFID and NFC (Case Studies 3 and 4), Wi-Fi (Case Studies 5 and 6), infrared (Case Study 7), and BLE (Case Study 8) were each examined and their weaknesses categorized by type and exploitability — and the scope extended in practice to captive-portal social engineering (Case Study 9) and USB HID injection (Case Study 10), two vectors that proved too central to the real threat picture to omit.

The second objective was the **execution of practical penetration tests**. Ten controlled, ethically bounded experiments were designed and carried out against representative devices, using the Flipper Zero with supporting hardware. The attacks named in the introduction — signal replay, tag cloning, denial of service, and Wi-Fi deauthentication — were all performed, alongside others that emerged as the work developed: handshake capture and offline cracking, BLE flooding, credential harvesting, and keystroke injection.

The third objective was **impact analysis**. Each experiment closed with an explicit assessment of consequences and an assigned risk level, framed in operational terms: unauthorized physical access (the gate, the badge, the card), denial of service (the jamming, the deauthentication, the BLE flood), credential compromise (the WPA2 crack, the evil portal), and total system control (the BadUSB injection). The summary table in Chapter 5 consolidates these into a single comparative view.

The fourth objective was the **formulation of countermeasures**. Chapter 7 met this not as a loose list of fixes but as a structured argument, tracing each demonstrated

weakness back to one of five root causes and forward to a corresponding defense, and culminating in a mapping table and a layered defense-in-depth model. All four objectives, then, were met, and two of them — the range of vulnerabilities and the breadth of attacks — were exceeded as the practical work revealed vectors the original plan had not anticipated.

8.4 Original Contribution: Reframing the Threat Model

The introduction was careful to state what this work did not claim: it did not set out to discover previously unknown vulnerabilities. The weaknesses in these protocols are documented, some of them for over a decade. The contribution lies elsewhere, and the completed body of work confirms it.

Prior research has tended to examine these protocols in isolation, each with its own specialized instrument — a software-defined radio for Sub-GHz, a Proxmark for RFID, a custom rig for one attack class at a time. That fragmentation produces an accurate picture of each protocol but a misleading picture of the adversary, because it implies a level of equipment and expertise that real-world attacks no longer require. This thesis took the opposite approach: a single, commercially available, sub-$200 instrument applied across the entire perception layer, by one operator, within a single coherent methodology.

The result is empirical evidence for a shift in the threat model itself. The same pocket-sized device opened a gate, cloned an access badge, cracked a Wi-Fi password, controlled appliances, flooded mobile devices, harvested credentials, and compromised a workstation — a span of attacks that, a decade ago, would have demanded a workbench of dedicated hardware and the expertise to drive it. By demonstrating that span on one cheap tool, the work provides concrete grounding for a claim often asserted but less often shown: that cross-protocol, cyber-physical attacks have moved out of the specialist laboratory and into the reach of a far broader population.

That reframing is the central contribution, and its consequences are not merely technical. A threat model built on the assumption that such attacks are rare, expensive, and expert-bound will systematically under-defend against an adversary for whom they are none of those things. The honest framing throughout the experiments — recording where attacks failed as carefully as where they succeeded, as when the PMF-protected

router defeated the deauthentication and the rolling-code shutters resisted naïve replay — strengthens rather than weakens this contribution. It shows that the accessibility being documented is real and bounded, not exaggerated: the tool is powerful precisely where systems are left undefended, and it meets its limits exactly where sound, available defenses have been deployed. That boundary is the practical heart of the thesis.

8.5 Legal, Educational, and Policy Implications

The accessibility this thesis documents is double-edged, and the same property that makes the Flipper Zero a useful research instrument makes it a contested object in public policy. The introduction promised to reflect on these implications, and the empirical findings give that reflection a firmer footing than commentary alone could.

The clearest illustration is the regulatory episode examined in Chapter 6: the 2024 Canadian proposal to ban the device outright, framed around its supposed role in vehicle theft. The experiments in this thesis bear directly on that framing. The tool trivially defeated a fixed-code gate, but the rolling-code systems that protect essentially every modern car resisted naïve replay exactly as designed. A capture-and-replay device of this kind is not, on the evidence gathered here, the instrument of mass automotive theft it was portrayed as. This is a concrete case of policy aimed at a recognizable object rather than the underlying insecurity — and banning the tool would have left every genuinely vulnerable system, the fixed-code installations and the default-keyed access cards, exactly as exposed as before. The honest conclusion is that the device's danger is consistently overstated where systems are well-designed and consistently understated where they are not; regulation that ignores this distinction will misfire in both directions.

There is an educational dimension that cuts the other way. The very accessibility that worries regulators is what makes a tool like this valuable for teaching and defensive research. Vulnerabilities that were once abstract — a slide describing replay attacks, a paragraph on unauthenticated management frames — become tangible when a student can capture a signal and watch a gate open, or send a deauthentication frame and watch a device drop offline. The same property that lowers the barrier for an attacker lowers it for a learner and a defender. Throughout this work the device functioned as exactly that: an instrument for understanding weaknesses concretely enough to reason about their defenses, which is the necessary first step toward fixing them.

The dual-use character is therefore not a flaw to be regulated away but a fact to be managed. A penetration-testing tool, like the knowledge it embodies, can be turned to attack or to defense, and the appropriate response is the one this thesis tried to model: clear ethical boundaries, permission-based testing, honest reporting of both successes and failures, and a focus on the defensive lesson rather than the offensive thrill. The policy conclusion that follows is measured: a response focused on closing the underlying vulnerabilities these tools expose is likely to prove more effective than one focused on restricting the tools themselves.

8.6 Limitations

Several limitations bound the conclusions of this work, and stating them plainly is part of presenting the findings honestly.

The evaluation was conducted on a specific, consumer-grade testbed in a single environment. The devices examined are representative of common perception-layer hardware, but they are not exhaustive, and results obtained against them do not automatically generalize to enterprise-grade equipment, industrial control systems, or devices with security features absent from the samples tested. The findings describe what was observed on this hardware, not a universal claim about every implementation of each protocol.

The scope was also shaped deliberately by the ethical framework of Chapter 4. Certain attacks were demonstrated as methods rather than executed to completion — the MFkey32 reader attack, for instance, was documented but not run, since the target did not require it and attacking unrelated hardware fell outside the permitted scope. These were principled choices, but they mean some attack paths are analyzed rather than empirically confirmed in this work.

A further limitation concerns currency. Several findings are tied to the state of devices at the time of testing — most visibly the BLE results, where the catastrophic crash documented historically had already been contained by operating-system rate-limiting on the patched hardware available. Security is a moving target; an attack that fails against a current device may have succeeded months earlier, and the reverse is equally possible as new weaknesses emerge.

8.7 Future Work

The single-device methodology that defines this thesis is also the natural starting point for extending it. The Flipper Zero proved capable across a wide span of the perception layer, but the experiments also exposed where its integrated hardware reaches its limits — and those limits point directly to the most promising next steps.

The most immediate is **hardware augmentation**. Some findings were bounded by the tool rather than the target: the BLE reconnaissance in Case Study 8 had to be offloaded to a smartphone because the ESP32-S2 board has no Bluetooth radio, and the 868 MHz jamming in Case Study 2 failed because the device's front-end is optimized for 433 MHz. Pairing the Flipper with purpose-built boards would close the first kind of gap. A genuine step up, however, would be a wideband software-defined radio such as the HackRF: paired with GNU Radio, it handles the 868 MHz band, the full-duplex attacks like RollJam that are awkward on the Flipper, and the TPMS capture that the foundational research in Chapter 6 performed with exactly that class of tool. Such an extension also serves a sharper analytical purpose. This thesis rests on a claim about accessibility — that a sub-$200 device reaches attacks once reserved for specialist equipment. Mapping where that cheap tool reaches its ceiling, and an intermediate-tier instrument like the HackRF becomes necessary, would turn a single data point into a graded map of capability against cost, sharpening the threat-model reframing this work began.

Three further directions follow from the results. Having watched Protected Management Frames defeat the deauthentication attack, subjecting a fully **WPA3-SAE network** to the same battery of attacks would document where the modern standard holds and where it does not. The **application-layer BLE attacks** surveyed in Chapter 6 but not performed here — GATT-level connection attacks — would extend the Bluetooth work from denial of service into active device control. And the **temporal dimension** deserves attention: since an attack's success proved tied to a device's patch state, a longitudinal study re-running a fixed suite of attacks over time would capture how this landscape actually shifts.

8.8 Closing Remarks

The central finding of this thesis is not that the devices around us are insecure — that much was already known — but that demonstrating their insecurity no longer requires

the resources it once did. A single tool, costing less than a smartphone and fitting in a pocket, opened gates, cloned credentials, cracked passwords, and seized control of a workstation, across protocols that a decade ago would each have demanded their own specialized equipment and expertise.

Yet the same experiments that proved this also drew its limits. The well-configured router held. The correctly implemented rolling code held. The patched phone held. Every defense that failed in these pages had a known, available, often standardized remedy that simply had not been applied. That is the quiet conclusion beneath the dramatic one: the perception layer's insecurity is not, for the most part, a frontier of unsolved problems. It is a backlog of unapplied solutions. The tools to attack these systems have become cheap and accessible; the tools to defend them have been available all along. Closing the distance between the two is less a research challenge than a matter of will — on the part of manufacturers, installers, and users alike — and it is there, rather than in any single exploit, that the real work of securing the Internet of Things remains to be done.

References

[1] IoT Analytics, "State of IoT 2025: Number of Connected IoT Devices Growing 14% to 21.1 Billion Globally," IoT Analytics Research, 2025. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/ [Accessed: 10-Feb-2026].

[2] SonicWall, "2024 SonicWall Mid-Year Cyber Threat Report," SonicWall Capture Labs, 2024. [Online]. Available: https://www.sonicwall.com/threat-report/ [Accessed: 10-Feb-2026].

[3] Flipper Devices Inc., "Flipper Zero Official Documentation," 2024. [Online]. Available: https://docs.flipper.net [Accessed: 13-Feb-2026].

[4] R.-S. Neculai, I.-C. Bogdan, M. Alexandru, and D.-N. Robu, "Access Management in IoT: Implementing a Secure Access Control System," in Proc. 15th Int. Conf. on Communications (COMM), 2024, pp. 1–6. doi: 10.1109/COMM62355.2024.10741422

[5] R. Barry, The FreeRTOS Reference Manual: API Functions and Configuration Options. Real Time Engineers Ltd., 2017.

[6] K. Chandu, R. Gorrepotu, K. N. Swaroop, and M. Dasari, "Performance Analysis of Sub-GHz System for IoT Applications," International Journal of Electrical and Electronic Engineering & Telecommunications, vol. 10, no. 2, pp. 125–132, 2021. doi: 10.18178/ijeetc.10.2.125-132

[7] Microchip Technology Inc., "KeeLoq Code Hopping Encoder HCS301," Data Sheet DS21143B, 2001. [Online]. Available: https://ww1.microchip.com/downloads/en/devicedoc/21143b.pdf [Accessed: 17-Feb-2026].

[8] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. Wichers Schreur, and B. Jacobs, "Dismantling MIFARE Classic," in Computer Security – ESORICS 2008, LNCS vol. 5283, Springer, 2008, pp. 97–114. doi: 10.1007/978-3-540-88313-5_7

[9] P. Stirparo, J. Loeschner, and M. Cattani, "Bluetooth Technology: Security Features, Vulnerabilities and Attacks," European Commission, Joint Research Centre (JRC), Tech. Rep. JRC 68414, 2012. doi: 10.13140/RG.2.2.23401.49760

[10] Bluetooth Special Interest Group (SIG), "Bluetooth Core Specification Version 5.3," 2021. [Online]. Available: https://www.bluetooth.com/specifications/specs/core-specification-5-3/ [Accessed: 20-Feb-2026].

[11] M. Kim and T. Suh, "Eavesdropping Vulnerability and Countermeasure in Infrared Communication for IoT Devices," Sensors, vol. 21, no. 24, art. 8207, 2021. doi: 10.3390/s21248207

[12] B. G. Drăghici, A. E. Dobre, M. A. Jibotean, D. Ionică, O. P. Stan, and L. C. Miclea, "Assessment of Deauthentication Threats in IoT: An Empirical Analysis of ESP8266 Deauther and Flipper Zero," in Proc. 29th Int. Conf. on System Theory, Control and Computing (ICSTCC), Cluj-Napoca, Romania, 2025, pp. 661–666. doi: 10.1109/ICSTCC66753.2025.11240388

[13] IEEE, "IEEE Standard for Information Technology — Local and Metropolitan Area Networks — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 4: Protected Management Frames," IEEE Std 802.11w-2009, 2009.

[14] M. Frustaci, P. Pace, G. Aloi, and G. Fortino, "Evaluating Critical Security Issues of the IoT World: Present and Future Challenges," IEEE Internet of Things Journal, vol. 5, no. 4, pp. 2483–2495, 2018. doi: 10.1109/JIOT.2017.2767291

[15] S. Kamkar, "Drive It Like You Hacked It: New Attacks and Tools to Wirelessly Steal Cars," presented at DEF CON 23, Las Vegas, NV, Aug. 2015. [Online]. Available: https://samy.pl/defcon2015/ [Accessed: 05-Mar-2026].

[16] L. Csikor, H. W. Lim, J. W. Wong, S. Ramesh, R. P. Parameswarath, and M. C. Chan, "RollBack: A New Time-Agnostic Replay Attack Against the Automotive Remote Keyless Entry Systems," ACM Transactions on Cyber-Physical Systems, vol. 8, no. 1, pp. 1–25, 2024. doi: 10.1145/3627827

[17] A. Karygiannis, B. Eydt, G. Barber, L. Bunn, and T. Phillips, "Guidelines for Securing Radio Frequency Identification (RFID) Systems," National Institute of Standards and Technology, NIST Special Publication 800-98, 2007.

[18] K. Finkenzeller, RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication, 3rd ed. Chichester, U.K.: John Wiley & Sons, 2010.

[19] A. Francillon, B. Danev, and S. Capkun, "Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars," in Proc. Network and Distributed System Security Symposium (NDSS), San Diego, CA, 2011.

[20] A. Juels, "RFID Security and Privacy: A Research Survey," IEEE Journal on Selected Areas in Communications, vol. 24, no. 2, pp. 381–394, 2006. doi: 10.1109/JSAC.2005.861395

[21] S. S. Hassan, S. D. Bibon, M. S. Hossain, and M. Atiquzzaman, "Security Threats in Bluetooth Technology," Computers & Security, vol. 74, pp. 308–322, 2018. doi: 10.1016/j.cose.2017.03.008

[22] M. Newlin, "Multiple Bluetooth HID UAF / Keystroke Injection Vulnerabilities (CVE-2023-45866)," SkySafe Vulnerability Disclosure, Dec. 2023. [Online]. Available: https://github.com/skysafe/reblog/blob/main/cve-2023-45866/README.md [Accessed: 11-Mar-2026].

[23] MITRE, "Adversary-in-the-Middle: Evil Twin (T1557.004)," MITRE ATT&CK Enterprise, 2025. [Online]. Available: https://attack.mitre.org/techniques/T1557/004/ [Accessed: 13-Mar-2026].

[24] M. Vanhoef and F. Piessens, "Advanced Wi-Fi Attacks Using Commodity Hardware," in Proc. Annual Computer Security Applications Conference (ACSAC), 2014, pp. 256–265. doi: 10.1145/2664243.2664260

[25] O. Nakhila, E. Dondyk, M. F. Amjad, and C. Zou, "User-Side Wi-Fi Evil Twin Attack Detection Using SSL/TCP Protocols," in Proc. 12th IEEE Consumer Communications and Networking Conference (CCNC), 2015, pp. 239–244. doi: 10.1109/CCNC.2015.7157983

[26] S. Viehböck, "Brute Forcing Wi-Fi Protected Setup: When Poor Design Meets Poor Implementation," Whitepaper, 2011. [Online]. Available: https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf [Accessed: 18-Mar-2026].

[27] US-CERT / CERT-CC, "Vulnerability Note VU#723755: Wi-Fi Protected Setup (WPS) PIN Brute Force Vulnerability," 2011. [Online]. Available: https://www.kb.cert.org/vuls/id/723755 [Accessed: 18-Mar-2026].

[28] K. Nohl, S. Krißler, and J. Lell, "BadUSB — On Accessories That Turn Evil," presented at Black Hat USA, Las Vegas, NV, Aug. 2014.

[29] MITRE, "Hardware Additions (T1200)," MITRE ATT&CK Enterprise, 2025. [Online]. Available: https://attack.mitre.org/techniques/T1200/ [Accessed: 24-Mar-2026].

[30] D. J. Tian, A. Bates, and K. R. B. Butler, "Defending Against Malicious USB Firmware with GoodUSB," in Proc. 31st Annual Computer Security Applications Conference (ACSAC), 2015, pp. 261–270. doi: 10.1145/2818000.2818040

[31] F. J. Muñoz-Ruiz, A. J. Di-Bartolo, F. Broncano-Morgado, B. M. Ramírez-Gabardino, and M. Ávila, "Exploring the Real Capabilities of the Flipper Zero," Engineering Proceedings, vol. 123, no. 1, art. 6, 2026. doi: 10.3390/engproc2026123006

[32] justcallmekoko, "ESP32Marauder," GitHub repository, 2023. [Online]. Available: https://github.com/justcallmekoko/ESP32Marauder [Accessed: 28-Mar-2026].

[33] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, "Technical Guide to Information Security Testing and Assessment," National Institute of Standards and Technology (NIST), Special Publication 800-115, 2008. doi: 10.6028/NIST.SP.800-115

[34] EC-Council, "Code of Ethics," International Council of E-Commerce Consultants, 2023. [Online]. Available: https://www.eccouncil.org/code-of-ethics/ [Accessed: 02-Apr-2026].

[35] Forum of Incident Response and Security Teams (FIRST), "Common Vulnerability Scoring System v4.0 Specification Document," 2023. [Online]. Available: https://www.first.org/cvss/v4.0/specification-document [Accessed: 04-Apr-2026].

[36] O. Mykhaylova, A. Stefankiv, T. Nakonechny, T. Fedynyshyn, and V. Sokolov, "Resistance to Replay Attacks of Remote Control Protocols Using the 433 MHz Radio Channel," CEUR Workshop Proceedings, vol. 3654, pp. 98–110, 2024. [Online]. Available: https://ceur-ws.org/Vol-3654/paper9.pdf [Accessed: 09-Apr-2026].

[37] R. P. Parameswarath and B. Sikdar, "An Authentication Mechanism for Remote Keyless Entry Systems in Cars to Prevent Replay and RollJam Attacks," in Proc. IEEE Intelligent Vehicles Symposium (IV), 2022, pp. 1725–1730. doi: 10.1109/IV51971.2022.9827256

[38] PushStack, "Reversing the Somfy RTS Protocol," 2014. [Online]. Available: https://pushstack.wordpress.com/2014/04/14/reversing-somfy-rts/ [Accessed: 14-Apr-2026].

[39] B. L. Farris and J. J. Fitzgibbon, "Rolling Code Security System," U.S. Patent 6,810,123 B2, Oct. 26, 2004.

[40] Texas Instruments, "CC1101 Low-Power Sub-1 GHz RF Transceiver," Datasheet SWRS061, 2013. [Online]. Available: https://www.ti.com/lit/ds/symlink/cc1101.pdf [Accessed: 16-Apr-2026].

[41] F. D. Garcia, P. van Rossum, R. Verdult, and R. Wichers Schreur, "Wirelessly Pickpocketing a MIFARE Classic Card," in Proc. 30th IEEE Symposium on Security and Privacy (S&P), 2009, pp. 3–15. doi: 10.1109/SP.2009.6

[42] C. Meijer and R. Verdult, "Ciphertext-Only Cryptanalysis on Hardened MIFARE Classic Cards," in Proc. 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS), 2015, pp. 18–30. doi: 10.1145/2810103.2813641

[43] J. Bellardo and S. Savage, "802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions," in Proc. 12th USENIX Security Symposium, Washington, D.C., 2003, pp. 15–27. [Online]. Available: https://www.usenix.org/conference/12th-usenix-security-symposium/80211-denial-service-attacks-real-vulnerabilities-and [Accessed: 24-Apr-2026].

[44] D. Bongard, "Offline Bruteforce Attack on Wi-Fi Protected Setup (Pixie Dust Attack)," presented at Hack.lu, Luxembourg, 2014.

[45] Black Hills Information Security (R. Felch), "Using Infrared for Hardware Control," 2021. [Online]. Available: https://www.blackhillsinfosec.com/using-infrared-for-hardware-control/ [Accessed: 28-Apr-2026].

[46] Z. Zhou, W. Zhang, S. Li, and N. Yu, "Potential Risk of IoT Device Supporting IR Remote Control," Computer Networks, vol. 148, pp. 307–317, 2019. doi: 10.1016/j.comnet.2018.11.014

[47] C. Reynolds (ECTO-1A), "Crashing iPhones with Flipper Zero and the Story Behind CVE-2023-42941," 2024. [Online]. Available: https://ecto-1a.github.io/AppleJuice_CVE/ [Accessed: 06-May-2026].

[48] M. Cäsar, T. Pawelke, J. Steffan, and G. Terhorst, "A Survey on Bluetooth Low Energy Security and Privacy," Computer Networks, vol. 205, art. 108712, 2022. doi: 10.1016/j.comnet.2021.108712

[49] R. Muthalagu and S. Sanjay, "Evil Twin Attack Mitigation Techniques in 802.11 Networks," International Journal of Advanced Computer Science and Applications (IJACSA), vol. 12, no. 6, pp. 38–41, 2021. doi: 10.14569/IJACSA.2021.0120605

[50] N. A. Hassan and R. Hijazi, Open Source Intelligence Methods and Tools: A Practical Guide to Online Intelligence. New York, NY, USA: Apress, 2018. doi: 10.1007/978-1-4842-3213-2

[51] National Vulnerability Database (NVD), "CVE-2022-27254 Detail," National Institute of Standards and Technology (NIST), 2022. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-27254 [Accessed: 12-May-2026].

[52] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar, "Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study," in Proc. 19th USENIX Security Symposium, Washington, DC, 2010, pp. 323–338. [Online]. Available: https://www.usenix.org/legacy/event/sec10/tech/full_papers/Rouf.pdf [Accessed: 15-May-2026].

[53] ECTO-1A, "AppleJuice," GitHub repository, 2023. [Online]. Available: https://github.com/ECTO-1A/AppleJuice [Accessed: 19-May-2026].

[54] pentestfunctions, "BlueDucky," GitHub repository, 2023. [Online]. Available: https://github.com/pentestfunctions/BlueDucky [Accessed: 22-May-2026].

[55] B. Toulas, "Canada to Ban the Flipper Zero to Stop Surge in Car Thefts," BleepingComputer, Feb. 9, 2024. [Online]. Available: https://www.bleepingcomputer.com/news/security/canada-to-ban-the-flipper-zero-to-stop-surge-in-car-thefts/ [Accessed: 28-May-2026].

[56] Electronic Frontier Foundation, "Restricting Flipper Is a Zero-Accountability Approach to Security: Canadian Government Response to Car Hacking," 2024. [Online]. Available: https://www.eff.org/deeplinks/2024/03/restricting-flipper-zero-accountability-approach-security-canadian-government [Accessed: 30-May-2026].

[57] Wi-Fi Alliance, "WPA3 Specification, Version 3.0," Wi-Fi Alliance, 2020. [Online]. Available: https://www.wi-fi.org/ [Accessed: 05-Jun-2026].

[58] D. Harkins, "Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks," in Proc. 2nd Int. Conf. on Sensor Technologies and Applications (SENSORCOMM), 2008, pp. 839–844. doi: 10.1109/SENSORCOMM.2008.131

[59] OWASP Foundation, "OWASP Internet of Things (IoT) Top 10," 2018. [Online]. Available: https://owasp.org/www-project-internet-of-things/ [Accessed: 11-Jun-2026].

[60] Netwrix, "How to Prevent BadUSB, Rubber Ducky, and Flipper Zero Attacks," 2025. [Online]. Available: https://netwrix.com/en/resources/blog/badusb-attack-prevention/ [Accessed: 18-Jun-2026].

[61] H. Mohammadmoradi and O. Gnawali, "Making Whitelisting-Based Defense Work Against BadUSB," in Proc. 2nd Int. Conf. on Smart Digital Environment (ICSDE '18), ACM, 2018, pp. 127–134. doi: 10.1145/3289100.3289121

Appendix A: Supplementary Material

This appendix provides the source code used in the empirical case studies of Chapter 5, included here to support the reproducibility of the experiments without interrupting the main text.

A.1 Wi-Fi Handshake Cracking Commands (Case Study 5)

The following commands document the offline cracking workflow applied to the captured WPA2 handshake in Case Study 5, performed on a Kali Linux workstation (Section 5.5.4). They are included so that the exact tooling and parameters used can be independently verified.

Step 1 — Convert the captured .pcap to the Hashcat hash format (hc22000):

```
hcxpcapngtool -o hash.hc22000 sniffpmkid_1.pcap
```

Step 2 — Launch the dictionary attack with Hashcat (mode 22000, WPA-PBKDF2-PMKID+EAPOL):

```
hashcat -m 22000 -a 0 hash.hc22000 /usr/share/wordlists/rockyou.txt
```

Step 3 — Display the recovered pre-shared key once the hash is cracked:

```
hashcat -m 22000 hash.hc22000 --show
```

Parameter reference: -m 22000 selects the WPA2 (PMKID/EAPOL) hash mode; -a 0 specifies a straight dictionary attack; rockyou.txt is the wordlist of previously breached passwords used as the first-pass dictionary. In this experiment, the pre-shared key was recovered in approximately thirteen seconds.

A.2 Rogue Captive-Portal Phishing Page (Case Study 9)

The following HTML page was served by the ESP32 Evil Portal in Case Study 9, impersonating a TP-Link Archer AX73 router administration panel. The form deliberately submits via an HTTP GET request to the firmware's /get endpoint, so that the entered credentials are captured and displayed on the Flipper Zero's screen (as discussed in Section 5.9.1). The visual styling is included for completeness, as its plausibility is central to the attack.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Router Configuration Panel</title>
<style>
  /* ... styling omitted for brevity ... */
</style>
</head>
<body>
  <div class="topbar">
    <span class="model">TP-LINK ARCHER AX73 | 192.168.0.1</span>
    <span>Firmware: 1.3.0 Build 230912</span>
  </div>
  <div class="main">
    <div class="card">
      <div class="card-header">
        <div class="header-text">
          <h1>Router Admin Panel</h1>
          <p>Management Console — Authentication Required</p>
        </div>
      </div>
      <div class="card-body">
        <div class="alert">
          Your session has expired. Please re-enter your credentials to continue.
        </div>
        <form action="/get" method="GET">
          <label for="email">Administrator Username</label>
          <input type="text" id="email" name="email" placeholder="admin" required>
          <label for="password">Password</label>
          <input type="password" id="password" name="password" required>
          <button class="btn" type="submit">Sign In</button>
        </form>
      </div>
    </div>
  </div>
</body>
</html>
```

A.3 BadUSB Keystroke-Injection Payload (Case Study 10)

The following DuckyScript payload was executed on the Flipper Zero's "Bad KB" module against the Windows workstation described in Case Study 10. The payload is deliberately benign: it opens a command prompt and prints a series of proof-of-concept messages, demonstrating the unauthenticated HID trust model without performing any destructive action.

```
REM ================================================
REM BadUSB Proof-of-Concept Payload
REM MSc Thesis - IoT Security Penetration Testing
REM Target: Windows OS - HID Keystroke Injection
REM Purpose: Demonstrate unauthenticated USB trust
REM ================================================

REM Wait for the OS to enumerate the device as HID keyboard
DELAY 2000

REM Open the Run dialog
GUI r
DELAY 800

REM Launch Command Prompt
STRING cmd
ENTER
DELAY 1200

REM Display proof-of-concept messages
STRING echo ================================================
ENTER
STRING echo [BadUSB DEMO] Unauthorized HID Access Demonstrated
ENTER
STRING echo This device was recognized as a keyboard with no authentication.
ENTER
STRING echo CVE-class USB trust model exploitation - MSc Thesis PoC
ENTER
STRING echo Timestamp:
ENTER
STRING echo %date% %time%
```

```
ENTER
STRING echo ============================================
DELAY 500

REM Pause so the output stays visible for the screenshot
STRING pause
ENTER
```