metin, yazı tipi, ekran görüntüsü, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

Engineering Faculty / Electrical and Electronics Engineering

**Analysis and Replay Attack Implementation on LoRa Communication Systems**

**Prepared by:**  
Barış ER

**Supervisor:**  
Prof. Dr. Tansu Filik

**Date:**  
January 2025

**Course:**  
EEM461 Z Research in Telecommunications I - Fall 2024

**Table of Contents**

Table of Contents………………………………………………………………………………………… ……….. 1

Abstract……………………………………………………………………………………………………………….. 2

1. Introduction………………………………………………………………………………………………………. 2

2. Theoretical Background………………………………………………………………………………………. 3

2.1 LoRa and LoRaWAN Overview…………………………………………………………………………….. 3

2.2 Chirp Spread Spectrum (CSS) Modulation……………………………………………………………. 3

2.3 Symbols, Chip and Spreading Factor…………………………………………………………………… 4

2.4 LoRa Modulation in Ebyte Modules…………………………………………………………….. ........... 5

2.5 ESP32 Microcontrollers………………………………………………………………………………………. 5

2.6 Software Defined Radio(SDR) ……………………………………………………………………………… 6

2.7 Replay Attack and Security Considerations……………………………………………………………… 6

2.8 GNU Radio and Platform.io…………………………………………………………………………………. 7

2.9 Security Vulnerabilities in IoT………………………………………………………………………………. 7

3. Experimental Setup……………………………………………………………………………………………. 8

4. Methodology……………………………………………………………………………………………………… 9

4.1 Establishing Communication Between Ebyte Modules…………………………………………… 9

4.2 Signal Recording and Quality Analysis with URH……………………………………………………. 10

4.3 Implementing a Basic Replay Attack……………………………………………………………………. 10

4.4 Dynamic Signal Capture and Manipulation…………………………………………………………… 11

4.5 Signal Analysis with Python Scripts……………………………………………………………………… 12

5. Results and Discussion………………………………………………………………………………………. 12

6. Conclusion and Future Work……………………………………………………………………………….. 13

Resources……………………………………………………………………………………………………………. 14

**Abstract**

This project explores the security vulnerabilities of LoRa communication by implementing and analyzing a replay attack scenario. The initial phase involved establishing communication between two LoRa modules (Ebyte E22-900T-22D) via ESP32 microcontrollers to understand the basic functionality and structure of LoRa communication layers. Based on this understanding, a replay attack was designed and executed.

The attack was implemented dynamically using GNU Radio, where captured packets were stored in files and selectively replayed during runtime. Python scripts were developed to further analyze and manipulate the captured data. Additionally, tools such as Platform.io and GNU Radio Companion were utilized to streamline the development and testing process.

The findings of this project demonstrate the vulnerabilities of LoRa communication systems in physical layer and highlight the importance of addressing these security challenges, particularly in IoT applications. Future work could focus on extending the scope of the analysis beyond the physical layer to include higher layers of the LoRaWAN protocol stack, addressing their vulnerabilities and exploring potential security enhancements.

**1. Introduction**

The rapid growth of IoT (Internet of Things) devices has led to the widespread adoption of low-power wide-area network (LPWAN) technologies, with LoRaWAN being one of the most prominent protocols. LoRaWAN's capability to provide long-range communication with low power consumption has made it a popular choice for IoT applications, particularly in smart cities, agriculture, and industrial automation. However, as with any communication technology, the security of LoRaWAN remains a critical concern.

This project investigates the vulnerabilities in the LoRa communication stack, specifically focusing on the physical (PHY) layer. Initially, the communication between two LoRa modules (Ebyte E22-900T-22D) was established using ESP32 microcontrollers, and the key aspects of the LoRa PHY layer were analyzed. Based on these analyses, a replay attack was designed and executed to assess potential security weaknesses in LoRa communication.

The tools used for this project included Platform.io for ESP32 development and GNU Radio for dynamic signal processing and packet manipulation. A dynamic GNU Radio flowchart was designed to capture, store, and replay packets in real-time, enabling a comprehensive analysis of the replay attack. Independent python scripts are also used to analyse stored packages.

While this project primarily focuses on the PHY layer, future work will extend the scope to higher layers of the LoRaWAN protocol stack. This will allow for a broader assessment of LoRaWAN’s security landscape and contribute to the development of more robust IoT communication systems.

**2. Theoretical Background**

**2.1** **LoRa and LoRaWAN Overview**

LoRa (Long Range) is a proprietary low-power wide-area network (LPWAN) technology developed by Semtech. It operates at the physical (PHY) layer and utilizes Chirp Spread Spectrum (CSS) modulation to achieve long-range communication with low power consumption. LoRaWAN, on the other hand, is a network protocol designed to operate on top of LoRa. It manages medium access control (MAC), security, and application-layer communication.

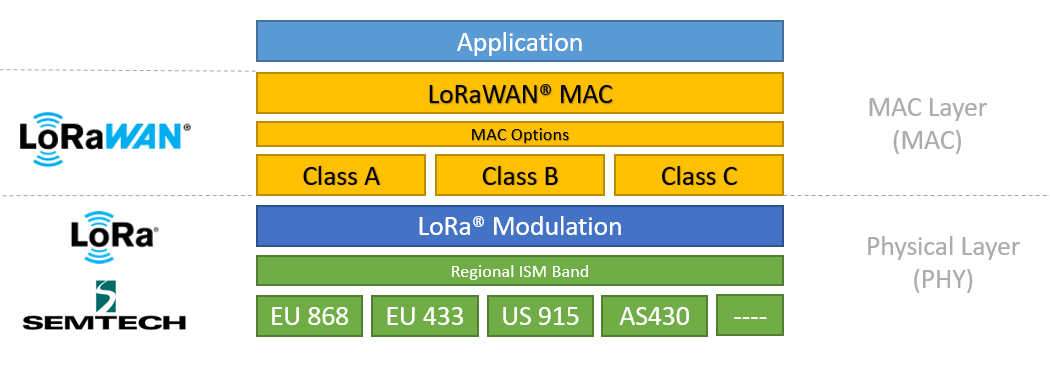


Figure 1: LoRa Network Stack

LoRa primarily focuses on the PHY layer, providing robust data transmission techniques such as CSS modulation. LoRaWAN builds on this by addressing higher-layer functionalities, including encryption, authentication, and network management. Semtech developed LoRa technology, while LoRaWAN is an open standard maintained by the LoRa Alliance.

**2.2 Chirp Spread Spectrum (CSS) Modulation**

Chirp Spread Spectrum (CSS) modulation is a technique that encodes information using wideband linear frequency-modulated chirp signals. A chirp is a signal in which the frequency either increases (up-chirp) or decreases (down-chirp) over time. A chirp is a ramp signal whose slope depends on being up-chirp or down-chirp. CSS spreads the signal across a wide frequency band, providing robustness against noise and interference while allowing long-range communication with low power consumption.

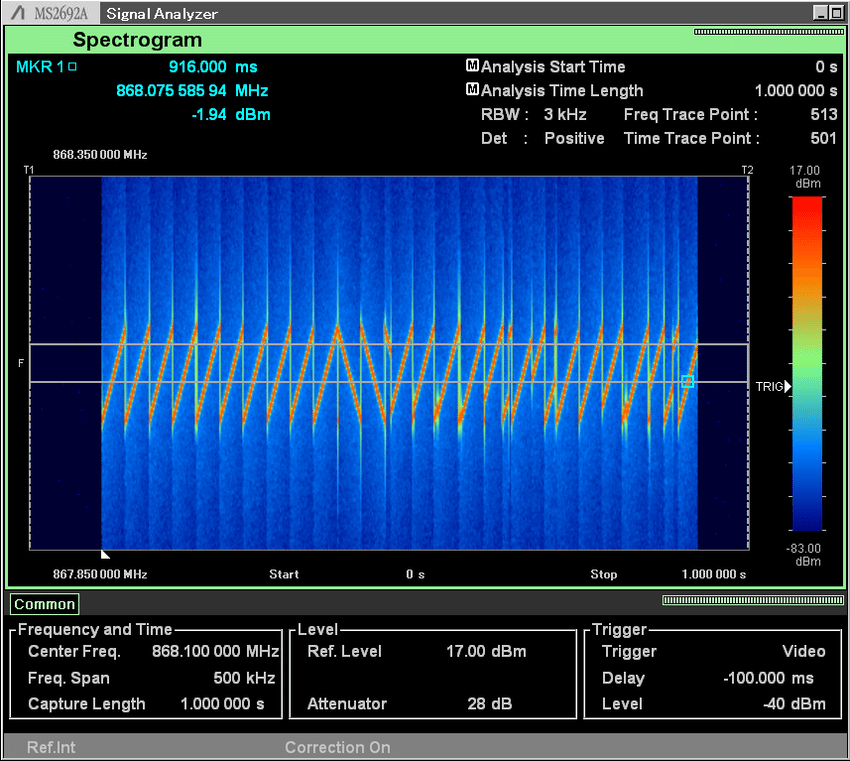


Figure 2: A Typical Message Structure of LoRa on Spectogram

The chirp signal can be expressed mathematically as:

s(t) = A × cos[2π × (f₀ + (B / 2T) × t) × t + φ], 0 ≤ t < T

Where:

* **s(t):** Instantaneous signal value at time t.
* **A:** Amplitude of the chirp signal.
* **f₀:** Starting frequency (center frequency).
* **B:** Bandwidth of the chirp signal.
* **T:** Duration of the chirp.
* **t:** Time variable.
* **φ:** Initial phase of the signal.

**Explanation of the Parameters**

1. **Amplitude (A):** Determines the signal's power level.
2. **Frequency (f₀):** The initial frequency of the chirp. For an up-chirp, the frequency increases from f₀ to f₀ + B; for a down-chirp, it decreases from f₀ + B to f₀.
3. **Bandwidth (B):** The range of frequencies over which the chirp is spread. A larger bandwidth provides greater resilience to interference and noise.
4. **Duration (T):** The total time of the chirp signal. Longer chirp durations improve processing gain but increase transmission time.
5. **Phase (φ):** The initial phase of the signal, which determines the starting point of the oscillatory cycle.

**Instantaneous Frequency**

The instantaneous frequency of the chirp signal can be described as:

F(t) = f₀ + (B / T) × t, 0 ≤ t < T

This equation shows that the frequency increases linearly with time for an up-chirp and decreases linearly for a down-chirp.

**2.3 Symbols, Chip and Spreading Factor**

A chirp represents a symbol and A chip is a segment of the chirp signal. Each symbol is mapped to a sequence of chips, which are distributed over the bandwidth. Amount of chips in one chirp is decided by Spreading Factor(SF). SF defines the trade-off between range and data rate. A higher SF increases range but also increases transmission time. For instance:

SF = 7: Each symbol carries 7 bits and spans 27=1282^7 = 12827=128 chips.

SF = 12: Each symbol carries 12 bits and spans 212=40962^{12} = 4096212=4096 chips

metin, çizgi, yazı tipi, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 3: Relation Between Chips and Symbols

CSS is a fundamental component of LoRa's ability to provide reliable and efficient communication in challenging environments: Spreading the signal over a wide frequency range reduces susceptibility to narrowband interference while it is suitable for IoT applications requiring long battery life.

**2.4 LoRa Modulation in Ebyte Modules**

The Ebyte E22-900T-22D module incorporates Semtech's LoRa chip and supports two primary modes of operation: transparent mode and fixed mode.

**Transparent Mode:**

* Transmits raw data directly, bypassing additional protocol layers.
* Ideal for basic communication setups and reverse engineering as it simplifies packet structure.

**Fixed Mode:**

* Adds a predefined protocol structure for data transmission, including addressing and control features. However, it still lacks encryption.

Transparent mode was chosen in this project to simplify the implementation of replay attacks and to facilitate a deeper understanding of LoRa's physical layer.

**2.5 ESP32 Microcontrollers**

The ESP32 is a dual-core microcontroller with integrated Wi-Fi and Bluetooth capabilities, making it a popular choice for IoT applications. Its low cost, ease of programming, and processing power make it an excellent platform for prototyping LoRa-based systems. In this project, ESP32 microcontrollers were used to interface with the Ebyte LoRa modules and facilitate communication between them.



Figure 4: Esp32 Used in This Experiment

**2.6 Software Defined Radio(SDR)**

Software Defined Radio (SDR) is a versatile communication system where traditionally hardware-implemented components, such as mixers, filters, and modulators, are instead implemented using software. This flexibility allows SDR to process, analyze, and modify radio signals dynamically in real-time, making it a crucial tool for wireless communication research and security testing. In this project, SDR bridges the gap between hardware and software, enabling tasks like reverse engineering LoRa's chirp structure, testing replay attack feasibility, and exploring potential vulnerabilities in the LoRa physical layer. By capturing and manipulating chirp-modulated signals, SDR provides the necessary platform to evaluate LoRa's resilience to attacks while offering insights into its encoding mechanisms.

|  |  |
| --- | --- |
| HackRF One SDR Geliştirme Kartı Geliştirme Kartları Motorobit -  Motorobit.com | RTL-SDR Blog V3 R860 RTL2832U 1PPM TCXO SMA Yazılım Tanımlı Radyo (Yalnızca  Kilitli) : Amazon.com.tr: Bilgisayar |
| Figure 5: HackRF Used in This Experiment | Figure 6: RTL-SDR Used in This Experiment |

**2.7 Replay Attack and Security Considerations**

A replay attack is a form of cyberattack where an adversary intercepts legitimate communication and retransmits it to deceive the recipient. However, the success of a replay attack depends on the target system's security mechanisms. For instance, many systems implement measures such as timestamps, session identifiers, or nonces to prevent the reuse of intercepted messages.

In the context of LoRa communication, the presence or absence of these mechanisms is often unclear at the outset. Therefore, testing whether the attack is feasible becomes a critical step in the research process. This project explored the vulnerabilities of LoRa's physical layer by dynamically capturing and replaying packets in real-time. The primary goal was to assess whether the LoRa PHY layer includes mechanisms that could detect or block replayed messages.

The testing process revealed that LoRa's PHY layer, in its basic implementation (e.g., transparent mode), does not inherently include features such as timestamps or message authentication. As a result, replay attacks could bypass these layers without resistance. However, systems using LoRaWAN might implement security measures at higher layers, which could potentially thwart such attacks. This highlights the importance of conducting experiments to identify and evaluate potential vulnerabilities in the system under test.

metin, ekran görüntüsü, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 7: Anatomy of a Replay Attack

**2.8 GNU Radio and Platform.io**

**GNU Radio:** GNU Radio is an open-source software toolkit for signal processing, widely used in software-defined radio (SDR) applications. Its flowchart-based design allows for dynamic signal manipulation, making it ideal for implementing replay attacks. In this project, GNU Radio was employed to capture LoRa packets, store them in files, and replay them in real time. This approach provided flexibility in testing the vulnerabilities of LoRa communication.

**Platform.io:** Platform.io is a development environment extention in Visual Studio Code for embedded systems, providing seamless integration with hardware platforms such as ESP32. It was used in this project to program the ESP32 microcontrollers and manage firmware development. Its robust library support and project management capabilities made it an efficient tool for prototyping.

**2.9 Security Vulnerabilities in IoT**

With the rapid adoption of IoT devices, security has become a critical concern. LPWAN technologies like LoRa are particularly vulnerable to attacks due to their lightweight protocols and resource-constrained devices. Common security weaknesses include:

* Lack of encryption in the PHY layer
* Weak or absent authentication mechanisms
* Susceptibility to packet replay, eavesdropping, and spoofing attacks

This project specifically focuses on the vulnerabilities in the PHY layer of LoRa communication, demonstrating how replay attacks can exploit the lack of built-in security measures. Future work aims to address these vulnerabilities and explore security mechanisms across all layers of the LoRaWAN protocol stack.

**3. Experimental Setup**

The experimental setup for this project was designed to investigate the vulnerabilities of the LoRa communication protocol to replay attacks. It involved the integration of several hardware and software components to ensure accurate signal capture, manipulation, and transmission under controlled conditions.

On the **hardware side**, the following components were used:

* **ESP32 Microcontrollers:** These served as the primary controllers for interfacing with the LoRa modules and executing replay attack scenarios. The ESP32 units were programmed with custom firmware to manage data transmission and reception.
* **Ebyte E22-900T-22D LoRa Modules:** Configured to operate within the 868 MHz ISM band, these modules facilitated the wireless communication between the devices.

|  |  |  |
| --- | --- | --- |
| **ESP32 Pin** | **Ebyte E22 Pin** | **Function/Role** |
| 3.3V | VCC | Provides power to the Ebyte E22 module. |
| GND | GND | Ground connection. |
| GPIO21 | M0 | Selects the operation mode (Mode Control Pin 0). |
| GPIO22 | M1 | Selects the operation mode (Mode Control Pin 1). |
| GPIO23 | AUX | Monitors the module's status (AUX State Pin). |
| GPIO16 | RXD | Serial data input (from ESP32 to Ebyte module). |
| GPIO17 | TXD | Serial data output (from Ebyte module to ESP32). |

Table 1:Hardware Connections

Proper soldering and wiring were performed to ensure stable and noise-free connections.

* **HackRF One:** This versatile software-defined radio (SDR) was employed for transmitting captured replay signals back to the LoRa network.
* **RTL-SDR:** A low-cost SDR device was used for receiving and recording the transmitted LoRa chirp signals for further analysis.

HackRF One and RTL-SDR are connected to computer through USB ports.

On the **software side**, several tools and programming environments were used:

* **GNU Radio:** This open-source software defined radio platform was utilized to design and implement the signal processing pipeline. The pipeline included RTL-SDR-based signal reception, delay mechanisms for replay scenarios, and HackRF-based retransmission. Additionally, the setup incorporated low-pass filtering and FFT-based visualization for signal analysis.
* **Python Scripts:** Custom Python scripts were developed to handle the analysis and visualization of captured .raw files. This included plotting time-domain and frequency-domain graphs, as well as generating spectrograms to observe the temporal behavior of LoRa chirp signals.
* **Platform.io:** This development environment was used to program the ESP32 microcontrollers. The firmware included custom logic for sending and receiving packets, as well as analyzing replayed signals.
* **Universal Radio Hacker (URH):** URH was utilized to capture and record the transmitted signals during initial tests. Its primary purpose was to provide a detailed visualization of the signal structure and quality, offering insights into how data was transmitted over the LoRa modules.

The experimental setup was systematically configured as follows:

1. LoRa modules were initialized and connected to the ESP32 microcontrollers.
2. ESP32 microcontrollers are monitored and debugged through the Visual Studio Code Platform.io extension whenever necessary.
3. GNU Radio workflows were created to capture and process LoRa chirp signals using the RTL-SDR. These workflows incorporated delay blocks to retransmit the signals with controlled timing.
4. The captured signals were stored in .raw files and later replayed through the HackRF One to simulate real-world attack scenarios.
5. Replay signals were analyzed using a combination of Python scripts and GNU Radio visualizations to evaluate the response of the LoRa modules.

**4. Methodology**

The methodology for this project involved multiple stages to establish and analyze LoRa communication, implement replay attacks, and dynamically process and manipulate transmitted messages. Each stage builds upon the previous to explore vulnerabilities in the LoRa physical layer and evaluate the system's resilience to signal-based attacks.

**4.1 Establishing Communication Between Ebyte Modules**

The first phase focused on enabling basic communication between two Ebyte E22-900T-22D LoRa modules using ESP32 microcontrollers. Proper soldering and wiring were performed to ensure stable hardware connections, and the modules were configured to operate in transparent mode for simplicity. Two separate codes were developed and uploaded to the ESP32 microcontrollers, tailored for the transmitter (TX) and receiver (RX) modules.

The TX code allowed the transmission of string or integer data with proper labeling to facilitate reverse engineering and analysis, while the RX code ensured correct processing of incoming data. The system’s communication was tested by sending predefined messages and verifying the expected responses.

**4.2 Signal Recording and Quality Analysis with URH**

Following the establishment of communication, the Universal Radio Hacker (URH) software was utilized to record and analyze the transmitted signals. The primary objective was to examine the quality and structure of the transmitted messages. URH’s capability to directly capture signals provided insights into the integrity of communication under various conditions, serving as a benchmark for subsequent experiments. This analysis established the baseline characteristics of LoRa signals, which were essential for understanding their vulnerabilities.

mor, ekran görüntüsü, renklilik, kalıp, desen, düzen içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 8: LoRa Signal Captured with URH

**4.3 Implementing a** **Basic Replay Attack**

A basic replay attack was designed to test whether the LoRa modules would accept retransmitted signals without any inherent physical layer mechanisms for detecting duplicate messages or ensuring signal integrity. An RTL-SDR was configured to capture the transmitted signals from the TX module, and the captured data was delayed by two seconds using a delay block in GNU Radio. The delayed signals were then retransmitted via HackRF One, emulating a legitimate transmission. The RX module was observed to process these replayed signals without resistance, confirming the lack of security mechanisms at the physical layer.

metin, yazı tipi, çizgi, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 9: GNU Flowchart for Basic Replay Attack Scenario

**4.4 Dynamic Signal Capture and Manipulation**

Building on the basic replay attack, the methodology evolved to a dynamic framework for capturing, storing, and selectively replaying messages in real-time with further modifications to the experimental configuration. The TX module was reprogrammed to send three distinct messages at intervals, which were captured and recorded in raw format by RTL-SDR. The TX module transmitted messages with specific intervals, while the RX module was programmed to respond differently to each message by triggering unique LED patterns. The dynamic interface allowed for the selective replay of captured messages in real-time, demonstrating precise control over the RX module’s responses. Signal analysis during this phase relied on custom Python scripts and GNU Radio tools to validate the feasibility of targeted replay attacks.

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 10: GNU Flowchart for General Replay Attack Scenario

A custom GNU Radio flowchart was designed to manage the dynamic signal capture and replay process, incorporating specialized blocks to ensure real time input from the user. The RTL-SDR Source Block received and digitized raw RF signals, which were then processed by a Low-Pass Filter (LPF) block to remove high-frequency noise and improve signal clarity. Following this, the first Custom Python Block was responsible for capturing the filtered chirp-based messages and saving them as individual files in a designated folder on the computer. These files were timestamped, ensuring that each captured message could be uniquely identified and organized for later use.

The second Custom Python Block was integrated to handle dynamic replay operations. This block interfaced with the HackRF Sink Block to retransmit the captured messages. It also served as the core of the user interface, enabling users to browse the folder containing the saved messages, select specific files, and queue them for replay. The dynamic interaction provided by this block allowed precise control over which messages were replayed, ensuring that the HackRF transmitted the desired signals in real-time. This flowchart design facilitated efficient data handling, from initial message capture to targeted retransmission, demonstrating the feasibility and effectiveness of dynamically controlled replay attacks.

**4.5 Signal Analysis with Python Scripts**

Due to hardware limitations that prevented real-time visualization of spectrograms using QT GUI Sink, custom Python scripts were developed to analyze the captured signals. These scripts focused on spectral analysis by performing Fourier transforms, enabling visualization of the signals in both time and frequency domains. This approach provided critical insights into the chirp signal characteristics and their behavior under various conditions. While the scripts offered a detailed spectral analysis, their scope was limited to visualizing signal properties rather than verifying message integrity.

metin, ekran görüntüsü, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 11: Time and Frequency Domain Representation of a LoRa Signal

**5. Results and Discussion**

The experiments conducted during this project provided valuable insights into the vulnerabilities of LoRa communication systems, particularly at the physical (PHY) layer. The systematic implementation and testing of replay attacks revealed significant security weaknesses, especially in systems lacking higher-layer protocols like LoRaWAN.  
The first phase of the project successfully established reliable communication between two LoRa modules using ESP32 microcontrollers, providing a foundation for further analysis. Transparent mode facilitated data transmission and detailed packet inspection, confirming that LoRa modules can transmit and receive under controlled conditions.  
Replay attack experiments demonstrated that retransmitted packets were accepted without validation. This confirmed the absence of redundancy or integrity-checking mechanisms, such as message validation protocols, at the PHY layer. Using SDR tools, it became evident that such vulnerabilities could easily be exploited in real-world IoT applications.  
Further analysis, through dynamic capture and replay frameworks, validated how attackers could manipulate LoRa signals in real-time. For example, real-time dynamic replay attack application highlighted the feasibility of targeted replay attacks. Spectral analysis of the chirp signals reinforced the need for security enhancements, emphasizing the risks associated with LoRa's minimalistic PHY-layer implementation.

**6. Conclusion and Future Work**

This project successfully demonstrated the vulnerabilities of LoRa communication systems at the physical layer. By systematically capturing, storing, and replaying LoRa packets, it was evident that the PHY layer lacks features to differentiate legitimate transmissions from malicious retransmissions. This highlights a key area for security enhancement, particularly in resource-constrained IoT environments.

Future work should expand upon this research to address the vulnerabilities in the higher layers of the LoRaWAN stack, such as MAC-layer encryption and authentication protocols. A thorough exploration of cross-layer security, where physical and higher-layer protections work in tandem, could offer a more comprehensive approach to secure IoT communication. Additionally, lightweight integrity-checking mechanisms or adaptive frequency hopping could be explored to mitigate attacks directly at the PHY layer.

In conclusion, the results of this project provide a foundation for improving the security of LoRa systems. By addressing the identified gaps and extending the analysis across protocol layers, this research contributes to the broader effort to secure IoT communication protocols in increasingly connected environments.

**Resources**

**Figures**

Figure 1: <https://www.mesh-net.co.uk/whats-new/>

Figure 2: <https://www.researchgate.net/figure/Spectrogram-of-a-LoRa-packet-Note-the-10-up-sweeps-the-225-down-sweeps-of-the_fig4_319458678>

Figure 3 : https://www.mobilefish.com/developer/lorawan/lorawan\_quickguide\_tutorial.html

Figure 4: <https://www.robotistan.com/esp32-esp-32s-wifi-bluetooth-dual-mode-gelistirme-karti>

Figure 5: https://www.amazon.com.tr/RTL-SDR-Blog-RTL2832U-Yaz%C4%B1l%C4%B1m%C4%B1-Tan%C4%B1mlanm%C4%B1%C5%9F/dp/B0129EBDS2

Figure 6: https://www.motorobit.com/hackrf-one-versiyon-2020-sdr-gelistirme-karti

Figure 7: <https://www.wallarm.com/what/replay-attacks>

Figure 8: Created by author

Figure 9: Created by author

Figure 10: Created by author

Figure 11: Created by author

**Web Resources**

https://fixaj.com/son-yazilarimiz/

<https://www.mobilefish.com/>

<https://gnuradio.org>/

E22-900T22D UserManual

<https://mischianti.org/>

**Referanced Articles**

**1. Joachim Tapparel, Orion Afisiadis, Paul Mayoraz, Alexios Balatsoukas-Stimming, and Andreas Burg,***"An Open-Source LoRa Physical Layer Prototype on GNU Radio,"* in *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Eindhoven, Netherlands, 2020.

**2. Pagilla Manohar Reddy, Archit Kak, Nisha K. S., Abhijith C. B., Sahithya Kattamuri, and Sadanala Manoj Parasuram,***"Detecting, Demodulating & Decoding LoRa,"* in *IEEE - 61001*, Amrita Vishwa Vidyapeetham, Amritapuri, India, 2020.

**3. Alexandre Marquet, Nicolas Montavont, and Georgios Z. Papadopoulos,***"Towards an SDR Implementation of LoRa: Reverse-engineering, Demodulation Strategies and Assessment over Rayleigh Channel,"* in *Computer Communications*, vol. 153, pp. 595–605, 2020.