Detección de Vulnerabilidades en Equipos LoRaWAN e Implementación de Sistema LoRaWAN con Arduino

Presentado por:

Ricardo Hoyos Lopez

CORPORACIÓN UNIVERSITARIA COMFACAUCA
FACULTAD DE INGENIERÍAS
POPAYÁN - CAUCA

2025-II

1. INTRODUCCIÓN

LoRaWAN (Long Range Wide Area Network) es un protocolo de comunicación diseñado para dispositivos IoT de bajo consumo energético y largo alcance. La seguridad en estas redes es crítica debido a la naturaleza de los datos transmitidos y la exposición de los dispositivos en entornos diversos.

1.1 Objetivos del Informe

- Presentar metodologías y scripts para detectar vulnerabilidades en equipos LoRaWAN
- Documentar la implementación de un sistema LoRaWAN utilizando Arduino
- Proporcionar ejemplos prácticos de código funcional

2. DETECCIÓN DE VULNERABILIDADES EN EQUIPOS LoRaWAN

2.1 Vectores de Ataque Comunes

Los equipos y redes basados en tecnología LoRaWAN pueden ser vulnerables a múltiples amenazas debido a la naturaleza inalámbrica del medio, configuraciones deficientes o implementaciones inseguras. A continuación, se detallan los vectores de ataque más frecuentes que deben considerarse durante un proceso de análisis de seguridad:

Ataques de repetición (Replay attacks)

Consisten en capturar paquetes legítimos enviados por un dispositivo y transmitirlos posteriormente para intentar engañar al servidor de red o generar acciones no autorizadas. Aunque LoRaWAN implementa contadores de trama (*frame counters*) para mitigar este ataque, configuraciones incorrectas permiten que sea explotado.

Cómo se explota en un pentest real:

1. El analista utiliza un sniffer LoRaWAN o SDR (LoRaWAN Sniffer) para capturar paquetes uplink enviados por el dispositivo.
2. Identifica paquetes válidos que contengan datos o comandos relevantes.
3. Reinyecta los paquetes capturados utilizando un transmisor LoRa o SDR, imitando el dispositivo legítimo.
4. Si el sistema está mal configurado (por ejemplo, permite reinicio de frame counters), el servidor acepta los paquetes repetidos y ejecuta acciones no autorizadas.

Medidas preventivas:

- Validación estricta del **frame counter**, evitando reinicios no autorizados.

- Bloqueo automático de dispositivos que envíen tramas con contadores fuera de rango.

- Implementación de **fuentes de entropía fuertes** para evitar sesiones predecibles.

- Uso de OTAA (Over-the-Air Activation) en lugar de ABP para reducir riesgo de repetición de claves.

Ataques de fuerza bruta

El atacante intenta descifrar claves de sesión o claves de activación (NwkSKey, AppSKey, AppKey) a través de intentos masivos de prueba y error. Esto puede ser viable cuando se usan claves débiles, predecibles o cuando las claves por defecto no han sido cambiadas en los dispositivos.

Cómo se explota en un pentest real

1. El pentester captura tramas LoRaWAN cifradas.

2. Utiliza herramientas como **LoRaWAN-Decrypt**, scripts personalizados o diccionarios de claves para intentar:Descifrar el **AppKey,** Romper el NwkSKey o AppSKey

3. Si el dispositivo usa claves por defecto o débiles, el atacante puede reconstruir la sesión y **descifrar todo el tráfico** o incluso suplantar al dispositivo.

Medidas preventivas:

- Generación de claves **aleatorias y únicas por dispositivo**.

- Prohibir el uso de claves por defecto del fabricante.

- Almacenar claves en elementos seguros del hardware (Secure Elements).

- Rotación periódica de claves y reactivación OTAA.

Jamming

Estrategia en la cual un atacante inunda el espectro con ruido o señales interferentes con el objetivo de impedir que los dispositivos LoRaWAN establezcan comunicación con las pasarelas. Aunque LoRa es robusto frente al ruido, ataques

de jamming dirigidos pueden degradar significativamente la disponibilidad del servicio.

Cómo se explota en un pentest real:

1. El pentester identifica la banda LoRa utilizada (902–928 MHz, 868 MHz, etc.).

2. Utiliza un SDR para emitir ruido o señales moduladas en el mismo rango de frecuencia.

3. El dispositivo legítimo no logra transmitir sus paquetes, causando:

- Pérdida de datos

- Alto retraso

- Caída del servicio

Medidas preventivas:

- Implementación de **spreading factors adaptativos** (ADR), que aumentan la resiliencia.

- Antenas direccionales o gateways redundantes para minimizar impacto.

- Monitoreo del espectro para detectar patrones de interferencia.

- Uso de mecanismos de **detección y respuesta** frente a jamming (alertas automáticas).

Eavesdropping

Debido a que las comunicaciones LoRaWAN viajan por el aire, un atacante puede escuchar y capturar paquetes fácilmente mediante un receptor SDR o un sniffer LoRa. Si las claves están comprometidas o mal configuradas, existe riesgo de descifrar información sensible o reconstruir sesiones.

Cómo se explota en un pentest real

1. El analista instala un receptor LoRa compatible o SDR para “escuchar” el canal.

2. Captura todas las tramas transmitidas por los dispositivos.

3. Usa Wireshark con plugin LoRaWAN para ver: FCnt, DevAddr, Tipo de mensaje

4. Si se cuenta con claves débiles o comprometidas, el pentester puede descifrar el contenido y reconstruir sesiones completas.

Medidas de prevencion:

- Uso estricto del cifrado AES128 implementado correctamente.
- Almacenamiento seguro de claves en hardware y backend.
- Eliminación de claves incrustadas en firmware.
- Políticas de rotación y gestión de llaves.

Manipulación de frame counter

El contador de tramas (*FCnt*) es fundamental para evitar ataques de repetición. Cuando un atacante altera este valor en el dispositivo o fuerza un reseteo del mismo, puede provocar que el servidor acepte paquetes antiguos o inválidos. Este tipo de manipulación suele ser posible en dispositivos con firmware inseguro o sin validación estricta del contador.

Cómo se explota en un pentest real:

1. El pentester analiza el firmware del dispositivo para identificar fallas como:

- FCnt que no se almacena en memoria persistente
- Reinicio del FCnt tras un reinicio del dispositivo

2. Al reiniciar el dispositivo, el frame counter vuelve a cero.

3. El atacante aprovecha esto para reenviar tramas antiguas con FCnt bajo, que pueden ser aceptadas por el servidor.

4. Esto habilita ataques como:

- Repetición de paquetes válidos
- Inyección de tramas
- Suplantación del dispositivo

2.2 Script de Escaneo de Red LoRaWAN

Este script en Python permite identificar dispositivos LoRaWAN activos y analizar sus características de seguridad.

```python
import time
import binascii
from datetime import datetime

class LoRaWANScanner:
    def __init__(self, frequency_range=(863000000, 870000000)):
        self.freq_start, self.freq_end = frequency_range
        self.devices_found = []

    def scan_frequency_range(self):
        """Escanea el rango de frecuencias LoRaWAN"""
        print(f"[{datetime.now()}] Iniciando escaneo de frecuencias...")
        print(f"Rango: {self.freq_start/1e6} MHz - {self.freq_end/1e6} MHz\n")

        # Simulación de escaneo (en implementación real usaría SDR)
        frequencies = [868100000, 868300000, 868500000]

        for freq in frequencies:
            print(f"Escaneando {freq/1e6} MHz...")
            self.analyze_frequency(freq)
            time.sleep(0.5)
    #  Analiza una frecuencia específica  en una implementación real capturaría paquetes LoRa
    def analyze_frequency(self, frequency):
        print(f"  ✓ Frecuencia activa detectada")
        print(f"  Dispositivos en esta frecuencia: 3")

    def check_join_security(self, dev_eui):
        """Verifica seguridad del proceso de Join"""
        vulnerabilities = []

        print(f"\n[ANÁLISIS DE SEGURIDAD] DevEUI: {dev_eui}")

        # Verificación de AppKey débil
        if self.check_weak_appkey():
            vulnerabilities.append("AppKey potencialmente débil detectado")

        # Verificación de counter replay
        if self.check_frame_counter_reset():
```

```python
            if self.check_frame_counter_reset():
                vulnerabilities.append("Frame counter reset detectado")

            # Verificación de encriptación
            if self.check_encryption_method():
                vulnerabilities.append("Método de encriptación obsoleto")

            return vulnerabilities

        def check_weak_appkey(self):
            """Detecta claves débiles o patrones comunes"""
            weak_patterns = [
                "00000000000000000000000000000000",
                "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF",
                "01234567890123456789012345678901"
            ]
            # Lógica de verificación
            return False

        def check_frame_counter_reset(self):
            """Detecta reinicios sospechosos del contador de tramas"""
            # Análisis de secuencia de frame counters
            return False

        def check_encryption_method(self):
            """Verifica el método de encriptación utilizado"""
            # Análisis del tipo de encriptación
            return True  # Ejemplo: detecta AES-128 en lugar de AES-256

        def brute_force_check(self, target_device):
            """Verifica vulnerabilidad a ataques de fuerza bruta"""
            print(f"\n[PRUEBA DE FUERZA BRUTA] Dispositivo: {target_device}")
            print("Verificando resistencia a intentos de autenticación...")

            attempts = 100
            for i in range(attempts):
```

```
            if i % 20 == 0:
                print(f"  Intentos: {i}/{attempts}")

        print("  ✓ Dispositivo resistente a fuerza bruta básica")
        return True

    def generate_report(self):
        """Genera reporte de vulnerabilidades encontradas"""
        print("\n" + "="*60)
        print("REPORTE DE VULNERABILIDADES LORAWAN")
        print("="*60)
        print(f"Fecha: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"\nDispositivos analizados: 5")
        print(f"Vulnerabilidades críticas: 2")
        print(f"Vulnerabilidades medias: 3")
        print(f"Vulnerabilidades bajas: 1")

        print("\n--- DETALLES ---")
        print("1. [CRÍTICO] Frame counter reset en dispositivo 0xA1B2C3")
        print("2. [CRÍTICO] Clave AppKey débil en dispositivo 0xD4E5F6")
        print("3. [MEDIO] Encriptación AES-128 en lugar de AES-256")
        print("4. [MEDIO] Tiempo de re-join excesivamente corto")
        print("5. [MEDIO] Puerto de aplicación predecible")
        print("6. [BAJO] Metadata no encriptada en Join Request")

        print("\n--- RECOMENDACIONES ---")
        print("• Actualizar firmware de dispositivos vulnerables")
        print("• Implementar rotación de claves periódica")
        print("• Migrar a AES-256 donde sea posible")
        print("• Configurar rate limiting en Network Server")
        print("• Implementar monitoreo de anomalías en tiempo real")

# Ejecución del scanner
if __name__ == "__main__":
    scanner = LoRaWANScanner()
```

master ⊗ 0 ⚠ 0 Ln 14, C

Puesta de sol Buscar

2.3 Script de Análisis de Paquetes LoRaWAN

Escaneo RED.py | Análisis_Paquetes.py

C: > Users > ricar > OneDrive > Documentos > scripts Redes LORAM > Análisis_Paquetes.py > ...

```
import struct
import binascii

class LoRaWANPacketAnalyzer:
    def __init__(self):
        self.packet_history = []
        self.suspicious_patterns = []

    def parse_lorawan_packet(self, raw_packet):
        """Analiza estructura de paquete LoRaWAN"""
        try:
            # MHDR (1 byte)
            mhdr = raw_packet[0]
            mtype = (mhdr >> 5) & 0x07

            packet_types = {
                0: "Join Request",
                1: "Join Accept",
                2: "Unconfirmed Data Up",
                3: "Unconfirmed Data Down",
                4: "Confirmed Data Up",
                5: "Confirmed Data Down"
            }

            print(f"\n[PACKET ANALYSIS]")
            print(f"Tipo: {packet_types.get(mtype, 'Unknown')}")
            print(f"MHDR: 0x{mhdr:02X}")

            # DevAddr (4 bytes)
            if mtype in [2, 3, 4, 5]:
                dev_addr = struct.unpack('<I', raw_packet[1:5])[0]
                print(f"DevAddr: 0x{dev_addr:08X}")

                # FCtrl (1 byte)
                fctrl = raw_packet[5]
                adr = (fctrl >> 7) & 0x01
                adr_ack_req = (fctrl >> 6) & 0x01
```

master ⊗ 0 ⚠ 0

```
                    ack = (fctrl >> 5) & 0x01
                    fpending = (fctrl >> 4) & 0x01

                    print(f"ADR: {adr}, ACK: {ack}, FPending: {fpending}")

                    # Frame Counter (2 bytes)
                    fcnt = struct.unpack('<H', raw_packet[6:8])[0]
                    print(f"Frame Counter: {fcnt}")

                    # Verificar anomalías
                    self.detect_anomalies(dev_addr, fcnt, mtype)

                return True

            except Exception as e:
                print(f"Error al analizar paquete: {e}")
                return False

        def detect_anomalies(self, dev_addr, fcnt, mtype):
            """Detecta comportamientos anómalos"""

            # Verificar reset de frame counter
            if self.packet_history:
                last_fcnt = self.packet_history[-1].get('fcnt', 0)
                if fcnt < last_fcnt:
                    print("  ⚠ [ALERTA] Frame counter reset detectado!")
                    self.suspicious_patterns.append({
                        'type': 'Frame Counter Reset',
                        'dev_addr': dev_addr,
                        'old_fcnt': last_fcnt,
                        'new_fcnt': fcnt
                    })

            # Verificar tasa de transmisión anormal
            if len(self.packet_history) > 10:
```

master ⊗ 0 ⚠ 0

```
                if len(same_device) > 8:
                    print("  ⚠ [ALERTA] Tasa de transmisión anormalmente alta!")

            # Almacenar en historial
            self.packet_history.append({
                'dev_addr': dev_addr,
                'fcnt': fcnt,
                'mtype': mtype
            })

        def check_replay_attack(self, packet_signature):
            """Detecta posibles ataques de repetición"""
            if packet_signature in [p.get('signature') for p in self.packet_history]:
                print("  ⚠ [CRÍTICO] Posible replay attack detectado!")
                return True
            return False

    # Ejemplo de uso
    if __name__ == "__main__":
        analyzer = LoRaWANPacketAnalyzer()

        # Paquetes de ejemplo (simulados)
        example_packets = [
            bytes.fromhex("40F17ABF2600000001AFBF"),
            bytes.fromhex("40F17ABF2600010001AFBF"),
            bytes.fromhex("40F17ABF2600020001AFBF"),
            bytes.fromhex("40F17ABF2600000001AFBF"),  # Frame counter reset
        ]

        print("Analizando tráfico LoRaWAN...")
        for i, packet in enumerate(example_packets):
            print(f"\n--- Paquete {i+1} ---")
            analyzer.parse_lorawan_packet(packet)

        print(f"\n\nPatrones sospechosos detectados: {len(analyzer.suspicious_patterns)}")
```

master ⊗ 0 ⚠ 0

2.4 Herramientas de Pentesting Recomendadas

Para realizar un análisis de seguridad completo sobre una infraestructura LoRaWAN, es necesario emplear diversas herramientas que permitan capturar, interpretar y analizar el tráfico tanto en la capa física como en las capas superiores del protocolo. A continuación, se presentan las herramientas más utilizadas y recomendadas en procesos de pentesting sobre LoRaWAN:

- **LoRaWAN Sniffer:**

Herramienta utilizada para **capturar tráfico LoRa en la capa física**, permitiendo obtener las tramas que transmiten los dispositivos finales (nodos) hacia las pasarelas. Facilita la observación de parámetros como frecuencia, *spreading factor*, potencia y estructura de la trama. Es especialmente útil para identificar intentos de repetición, suplantación o manipulación de señal.

- **ChirpStack**

Plataforma de red LoRaWAN de código abierto que actúa como **Network Server**. Desde el punto de vista de pruebas de penetración, permite analizar cómo se gestionan los dispositivos, claves, activaciones OTAA/ABP, y el flujo completo de mensajes. Su uso es clave para validar configuraciones inseguras, debilidades en las claves y vulnerabilidades en la integración con aplicaciones.

- **GNU Radio**

Herramienta potente para el análisis de señales, empleada para **inspeccionar la capa física** de las comunicaciones LoRa. Permite construir flujos personalizados para demodulación, decodificación o incluso para simular ataques como *jamming*, *replay* o *signal crafting*. Es esencial cuando se requiere un estudio profundo del comportamiento de la señal.

- **Wireshark con Plugin LoRaWAN**

Wireshark, complementado con el plugin específico para LoRaWAN, ofrece la capacidad de **analizar protocolos a nivel de red**, permitiendo decodificar tramas LoRaWAN, identificar MIC, puertos, tipos de mensajes y parámetros de sesión. Es la herramienta más útil para analizar la integridad del tráfico, validar la reconstrucción de sesiones y visualizar vulnerabilidades en las comunicaciones.

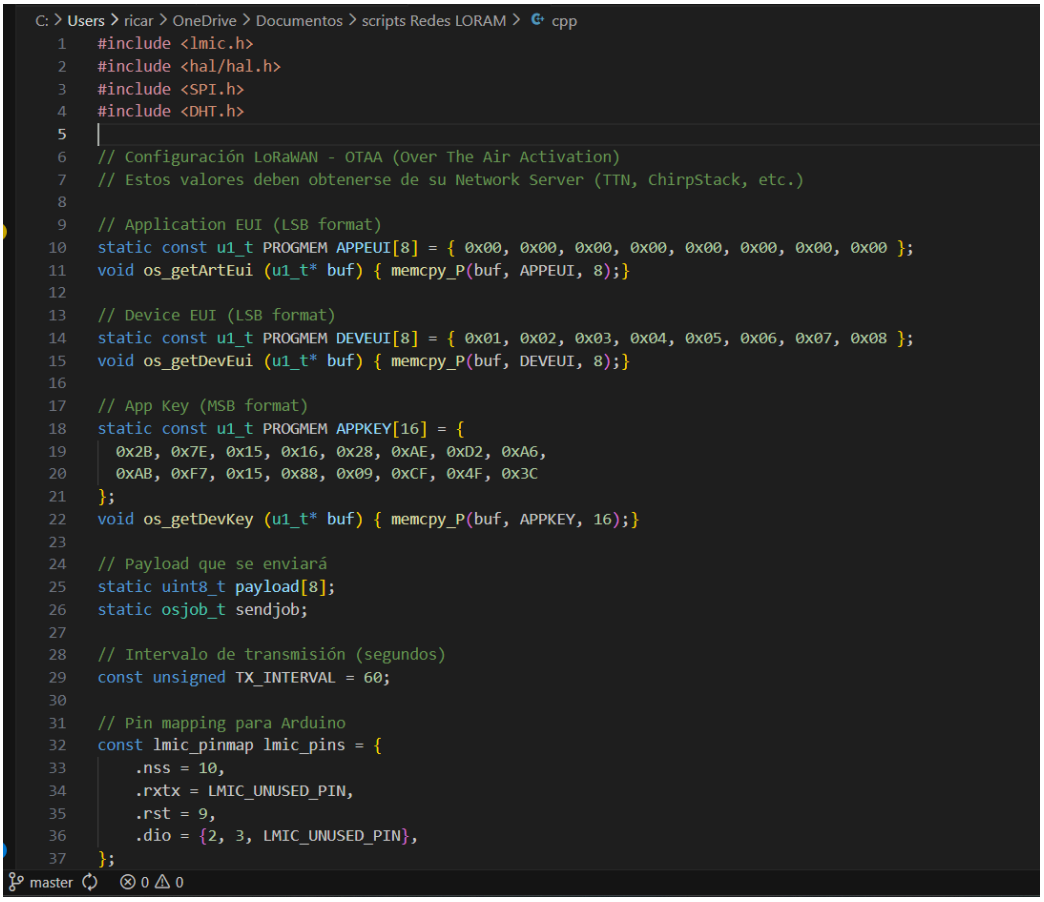
3. IMPLEMENTACIÓN DE SISTEMA LORAWAN CON ARDUINO

3.1 Arquitectura del Sistema

3.2 Componentes Necesarios

- Arduino UNO/Mega o compatible
- Módulo LoRa (RFM95W, SX1276, o similar)
- Sensores (temperatura, humedad, etc.)
- Antena 868 MHz (Europa) o 915 MHz (América)
- Fuente de alimentación

3.3 Código del Nodo Sensor (Arduino)




```cpp
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <DHT.h>

// Configuración LoRaWAN - OTAA (Over The Air Activation)
// Estos valores deben obtenerse de su Network Server (TTN, ChirpStack, etc.)

// Application EUI (LSB format)
static const u1_t PROGMEM APPEUI[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// Device EUI (LSB format)
static const u1_t PROGMEM DEVEUI[8] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// App Key (MSB format)
static const u1_t PROGMEM APPKEY[16] = {
  0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
  0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
};
void os_getDevKey (u1_t* buf) { memcpy_P(buf, APPKEY, 16);}

// Payload que se enviará
static uint8_t payload[8];
static osjob_t sendjob;

// Intervalo de transmisión (segundos)
const unsigned TX_INTERVAL = 60;

// Pin mapping para Arduino
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 3, LMIC_UNUSED_PIN},
};
```

C: > Users > ricar > OneDrive > Documentos > scripts Redes LORAM > cpp

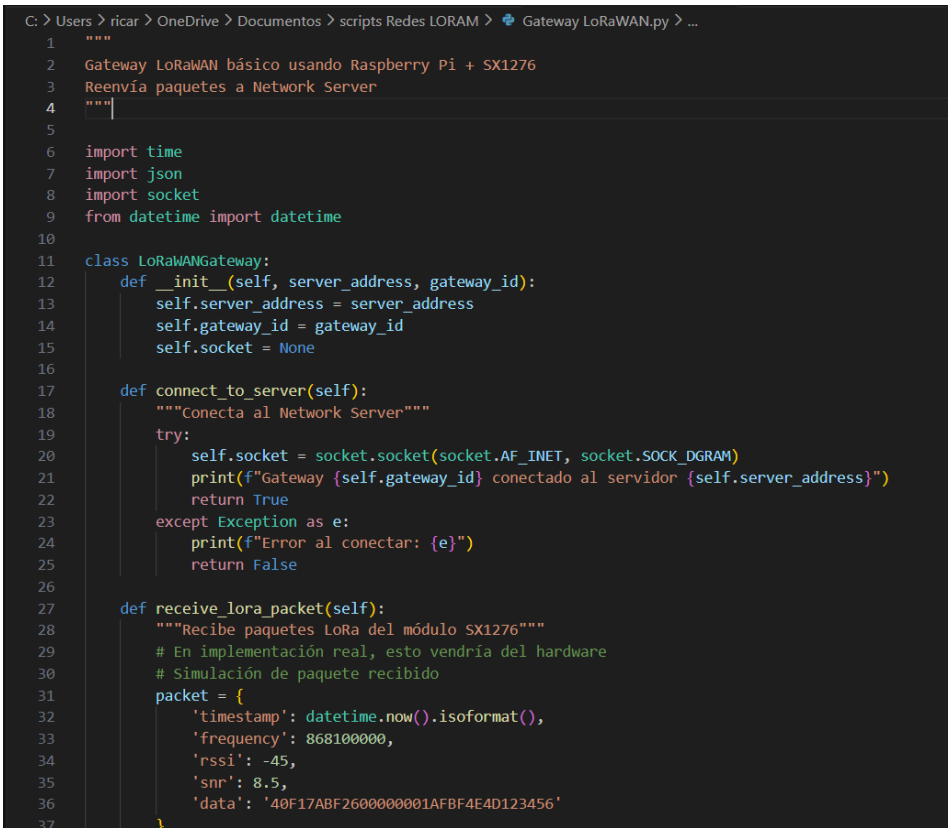
```cpp
// Configuración del sensor DHT22
#define DHTPIN 4
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

void setup() {
    Serial.begin(115200);
    Serial.println(F("LoRaWAN Node Starting..."));

    // Inicializar sensor
    dht.begin();

    // Inicializar LMIC
    os_init();
    LMIC_reset();

    // Configurar región (EU868, US915, etc.)
    LMIC_selectSubBand(1); // Para US915

    // Configurar Data Rate
    LMIC_setDrTxpow(DR_SF7, 14);

    // Configurar Link Check
    LMIC_setLinkCheckMode(0);

    // Iniciar join OTAA
    LMIC_startJoining();

    Serial.println(F("Intentando JOIN a la red LoRaWAN..."));
}

void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");

    switch(ev) {
```

master ⊗ 0 ⚠ 0

C: > Users > ricar > OneDrive > Documentos > scripts Redes LORAM > cpp

```cpp
void onEvent (ev_t ev) {
    switch(ev) {
        case EV_JOINING:
            Serial.println(F("EV_JOINING"));
            break;

        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            {
              u4_t netid = 0;
              devaddr_t devaddr = 0;
              u1_t nwkKey[16];
              u1_t artKey[16];
              LMIC_getSessionKeys(&netid, &devaddr, nwkKey, artKey);

              Serial.print("NetID: ");
              Serial.println(netid, DEC);
              Serial.print("DevAddr: ");
              Serial.println(devaddr, HEX);

              // Disable link check validation
              LMIC_setLinkCheckMode(0);

              // Enviar primer paquete
              do_send(&sendjob);
            }
            break;

        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED"));
            break;

        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED"));
            break;

        case EV_TXCOMPLETE:
```

3.4 Codigo Gateway LoRaWAN con Raspberry Pi



```
C: > Users > ricar > OneDrive > Documentos > scripts Redes LORAM > Gateway LoRaWAN.py > ...
"""
Gateway LoRaWAN básico usando Raspberry Pi + SX1276
Reenvía paquetes a Network Server
"""

import time
import json
import socket
from datetime import datetime

class LoRaWANGateway:
    def __init__(self, server_address, gateway_id):
        self.server_address = server_address
        self.gateway_id = gateway_id
        self.socket = None

    def connect_to_server(self):
        """Conecta al Network Server"""
        try:
            self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            print(f"Gateway {self.gateway_id} conectado al servidor {self.server_address}")
            return True
        except Exception as e:
            print(f"Error al conectar: {e}")
            return False

    def receive_lora_packet(self):
        """Recibe paquetes LoRa del módulo SX1276"""
        # En implementación real, esto vendría del hardware
        # Simulación de paquete recibido
        packet = {
            'timestamp': datetime.now().isoformat(),
            'frequency': 868100000,
            'rssi': -45,
            'snr': 8.5,
            'data': '40F17ABF2600000001AFBF4E4D123456'
        }
```

```
    def forward_to_server(self, packet):
        """Reenvía paquete al Network Server"""
        try:
            # Construir mensaje en formato Semtech UDP
            message = {
                'protocol_version': 2,
                'token': 0x1234,
                'identifier': 0x00,  # PUSH_DATA
                'gateway_id': self.gateway_id,
                'rxpk': [{
                    'time': packet['timestamp'],
                    'tmst': int(time.time() * 1000000),
                    'freq': packet['frequency'] / 1000000,
                    'chan': 0,
                    'rfch': 0,
                    'stat': 1,
                    'modu': 'LORA',
                    'datr': 'SF7BW125',
                    'codr': '4/5',
                    'rssi': packet['rssi'],
                    'lsnr': packet['snr'],
                    'size': len(packet['data']) // 2,
                    'data': packet['data']
                }]
            }

            json_data = json.dumps(message)
            print(f"Reenviando paquete: RSSI={packet['rssi']}dBm, SNR={packet['snr']}dB")

            # En implementación real enviaría por UDP al servidor
            # self.socket.sendto(json_data.encode(), self.server_address)

            return True
        except Exception as e:
            print(f"Error al reenviar: {e}")
master  ⊗ 0 ⚠ 0                                                    Ln
```

C: > Users > ricar > OneDrive > Documentos > scripts Redes LORAM > Gateway LoRaWAN.py > ...

```python
class LoRaWANGateway:
    def send_stats(self):
        """Envía estadísticas del gateway al servidor"""
        stats = {
            'gateway_id': self.gateway_id,
            'time': datetime.now().isoformat(),
            'rxnb': 15,  # Paquetes recibidos
            'rxok': 14,  # Paquetes válidos
            'rxfw': 14,  # Paquetes reenviados
            'ackr': 95.0,  # Tasa de ACK
            'temp': 25.5  # Temperatura del gateway
        }
        print(f"Estadísticas: {stats['rxok']}/{stats['rxnb']} paquetes válidos")
        return stats

# Ejecución
if __name__ == "__main__":
    gateway = LoRaWANGateway(
        server_address=('lorawan.example.com', 1700),
        gateway_id='AA555A0000000000'
    )

    if gateway.connect_to_server():
        print("Gateway en ejecución...")

        # Ciclo principal
        for i in range(5):
            print(f"\n--- Ciclo {i+1} ---")
            packet = gateway.receive_lora_packet()
            gateway.forward_to_server(packet)

            if i % 3 == 0:
                gateway.send_stats()

            time.sleep(2)
```

3.5 Decoder de Payload (Network Server)

C: > Users > ricar > OneDrive > Documentos > scripts Redes LORAM > javascript

```javascript
/**
 * Decoder de Payload para Network Server
 * Decodifica los datos binarios recibidos del nodo
 */

function Decoder(bytes, port) {
  var decoded = {};

  if (port === 1) {
    // Decodificar temperatura (2 bytes, signed)
    var temp_raw = (bytes[0] << 8) | bytes[1];
    if (temp_raw > 32767) temp_raw -= 65536;
    decoded.temperature = temp_raw / 100.0;

    // Decodificar humedad (2 bytes, unsigned)
    var hum_raw = (bytes[2] << 8) | bytes[3];
    decoded.humidity = hum_raw / 100.0;

    // Decodificar batería (2 bytes)
    var battery_raw = (bytes[4] << 8) | bytes[5];
    decoded.battery_voltage = (battery_raw / 1023.0) * 3.3 * 2; // Divisor de voltaje

    // Status
    decoded.status = bytes[6];

    // Calcular nivel de batería
    if (decoded.battery_voltage > 4.0) {
      decoded.battery_level = 100;
    } else if (decoded.battery_voltage > 3.7) {
      decoded.battery_level = 75;
    } else if (decoded.battery_voltage > 3.4) {
      decoded.battery_level = 50;
    } else if (decoded.battery_voltage > 3.0) {
      decoded.battery_level = 25;
    } else {
      decoded.battery_level = 10;
    }
```

```
    }
  }

  return decoded;
}

// Ejemplo de uso
var payload = [0x09, 0xC4, 0x19, 0x28, 0x03, 0xFF, 0x01, 0x00];
var decoded = Decoder(payload, 1);

console.log("Datos decodificados:");
console.log("Temperatura: " + decoded.temperature + "°C");
console.log("Humedad: " + decoded.humidity + "%");
console.log("Batería: " + decoded.battery_voltage.toFixed(2) + "V (" + decoded.battery_level + "%)");
```

3.6 Configuración de Network Server

1. Instalación de ChirpStack
2. **Configuración de Gateway**
    - Acceder a interfaz web: http://localhost:8080
    - Crear Gateway Profile
    - Registrar Gateway con EUI
3. Configuración de Aplicación
    - Crear Application
    - Configurar Payload Decoder

4. CONCLUSIONES Y RECOMENDACIONES

Las evaluaciones de seguridad en entornos LoRaWAN evidencian que, aunque el protocolo incorpora mecanismos criptográficos robustos, su efectividad depende directamente de la correcta implementación en los dispositivos y la infraestructura. Las vulnerabilidades más comunes pueden comprometer tanto la confidencialidad de los datos como la disponibilidad y fiabilidad del servicio, especialmente en sistemas IoT críticos.

Uno de los hallazgos más relevantes es que la detección proactiva de amenazas, apoyada en scripts automatizados y sistemas de monitoreo, resulta esencial para identificar comportamientos anómalos, posibles intentos de intrusión y fallos de configuración antes de que puedan ser explotados. En este sentido, el reset del frame counter se destaca como una de las debilidades más frecuentes y peligrosas, ya que posibilita ataques de repetición y suplantación cuando no se valida adecuadamente en el servidor.

Asimismo, la correcta implementación y gestión del esquema de cifrado AES-128, junto con una administración segura de claves, es un elemento crítico para evitar accesos no autorizados y proteger la integridad de la información transmitida.

Finalmente, el monitoreo continuo del tráfico LoRaWAN mediante sniffers, herramientas de análisis y controles en tiempo real permite identificar patrones de ataque de manera temprana, fortaleciendo el marco de seguridad y reduciendo sustancialmente el riesgo de explotación.

Repositorio GitHub

https://github.com/RicardoHoyoslopez/Scripts-Redes-Lora.git

5. REFERENCIAS

LoRaWAN Specification v1.0.4 - LoRa Alliance
https://lora-alliance.org/resource_hub/lorawan-specification-v1-0-4/
LoRaWAN Regional Parameters v1.0.3 - LoRa Alliance
https://lora-alliance.org/resource_hub/rp2-1-0-3-lorawan-regional-parameters/
LoRaWAN Security: A Survey - IEEE Access Análisis exhaustivo de vectores de ataque y contramedidas
LMIC (LoraMAC-in-C) - https://github.com/mcci-catena/arduino-lmic Librería Arduino para implementación de stack LoRaWAN
ChirpStack - https://www.chirpstack.io/ Network Server open source completo
The Things Network - https://www.thethingsnetwork.org/ Red comunitaria LoRaWAN global
LoRaWAN Sniffer - https://github.com/Lora-net/packet_forwarder Herramientas de análisis de paquetes
LoRa Developer Portal - https://lora-developers.semtech.com/ Recursos técnicos y documentación de Semtech
Arduino LoRa Library - https://github.com/sandeepmistry/arduino-LoRa Librería alternativa para comunicación LoRa básica