Caso de Estudio #1: Modbus

MC4002 Seguridad en Sistemas Ciberfísicos

Luis Bonilla

Febrero 2026

Índice

1. Introducción

1.1. ¿Qué es Modbus TCP/IP?

Modbus es un protocolo de comunicación de capa de aplicación, ubicado en el nivel 7 del modelo OSI, que proporciona comunicación cliente/servidor entre dispositivos conectados a través de diferentes tipos de buses o medios de comunicación. Introducido en 1979, Modbus se ha convertido en el estándar de facto para sistemas de control industrial (ICS).

El protocolo Modbus TCP/IP es la implementación del protocolo Modbus sobre redes Ethernet, utilizando TCP como protocolo de transporte en el puerto 502. Esta versión mantiene la misma estructura de mensajes que Modbus serial, pero utiliza el encabezado MBAP (Modbus Application Protocol) en lugar del checksum de error.

1.2. ¿Por qué es inseguro?

Modbus TCP presenta múltiples vulnerabilidades de seguridad inherentes a su diseño:

- **Sin encriptación**: Todos los datos se transmiten en texto plano, permitiendo que cualquier atacante con acceso a la red pueda interceptar las comunicaciones.
- **Sin autenticación**: No se requieren credenciales para leer o escribir en los registros del dispositivo.
- **Sin verificación de integridad**: Los paquetes pueden ser modificados en tránsito sin detección.
- **Sin autorización**: Cualquier cliente que conozca la dirección IP puede enviar comandos al servidor.

1.3. Objetivo del laboratorio

El objetivo de este laboratorio es demostrar las vulnerabilidades del protocolo Modbus TCP/IP mediante:

1. Configuración de un servidor Modbus y comunicación con clientes
2. Análisis de tráfico de red para observar la falta de encriptación
3. Uso de herramientas de seguridad (Nmap, Scapy) para interactuar con dispositivos Modbus
4. Captura y reproducción de paquetes Modbus

1.4. Entorno de Laboratorio

Para este laboratorio se utilizaron dos máquinas virtuales:

Máquina	**Sistema Operativo**	**Rol**	**Dirección IP**
VM1	Ubuntu Linux	Servidor Modbus	192.168.1.66
VM2	Kali Linux	Atacante/Analizador	192.168.1.64

Cuadro 1: Configuración del entorno de laboratorio

2. Breaking Modbus

2.1. Instalación de paquetes

2.1.1. En Ubuntu (Servidor Modbus)

Se instalaron los paquetes necesarios para ejecutar el servidor Modbus:

```
sudo apt update
sudo apt install python3-pip wireshark -y
sudo pip3 install pymodbus
```

Listing 1: Instalación de paquetes en Ubuntu

2.1.2. En Kali Linux (Atacante)

Se instalaron las herramientas de análisis y cliente Modbus:

```
# Instalar Ruby (requerido para modbus-cli)
sudo apt update
sudo apt install ruby ruby-dev -y

# Instalar modbus-cli
sudo gem install modbus-cli

# Instalar Scapy
sudo pip install scapy
```

Listing 2: Instalación de paquetes en Kali

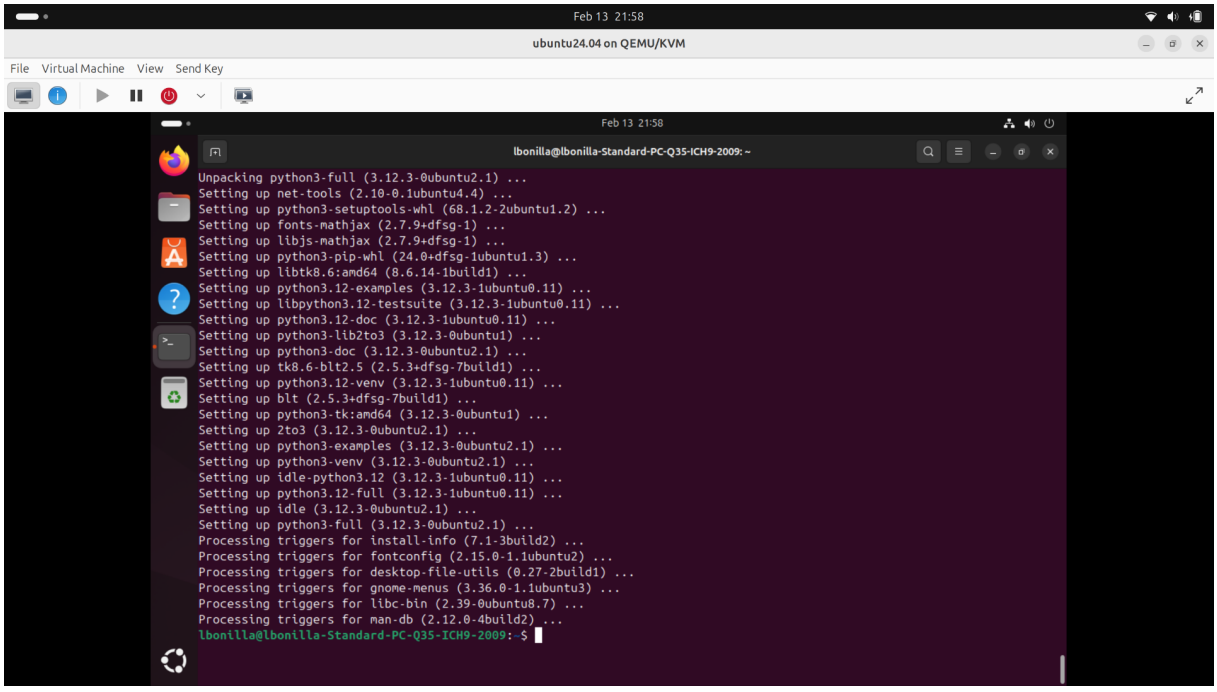

Figura 1: Instalación de paquetes en la máquina virtual

2.2. Configuración del Servidor Modbus

Se creó un script en Python para iniciar un servidor Modbus TCP asíncrono utilizando la biblioteca pyModbus:

```
#!/usr/bin/env python
from pymodbus.server.async import StartTcpServer
from pymodbus.device import ModbusDeviceIdentification
from pymodbus.datastore import ModbusSequentialDataBlock
from pymodbus.datastore import ModbusSlaveContext, ModbusServerContext

# Crear datastore con datos de prueba
store = ModbusSlaveContext(
    di = ModbusSequentialDataBlock(0, [17]*100),  # Discrete Inputs
    co = ModbusSequentialDataBlock(0, [17]*100),  # Coils
    hr = ModbusSequentialDataBlock(0, [17]*100),  # Holding Registers
    ir = ModbusSequentialDataBlock(0, [17]*100))  # Input Registers

context = ModbusServerContext(slaves=store, single=True)

# Identificacion del servidor
identity = ModbusDeviceIdentification()
identity.VendorName = 'PyModbus Inc.'
identity.ProductCode = 'PM'
identity.VendorUrl = 'https://github.com/riptideio/pyModbus'
identity.ProductName = 'Modbus Server'
identity.ModelName = 'PyModbus'
identity.MajorMinorRevision = '1.0'

print("Starting Modbus server...")
StartTcpServer(context, identity=identity, address=("0.0.0.0", 502))
```

Listing 3: Código del servidor Modbus (Modbus_server.py)

El servidor se inició con el siguiente comando:

```
sudo python Modbus_server.py
```

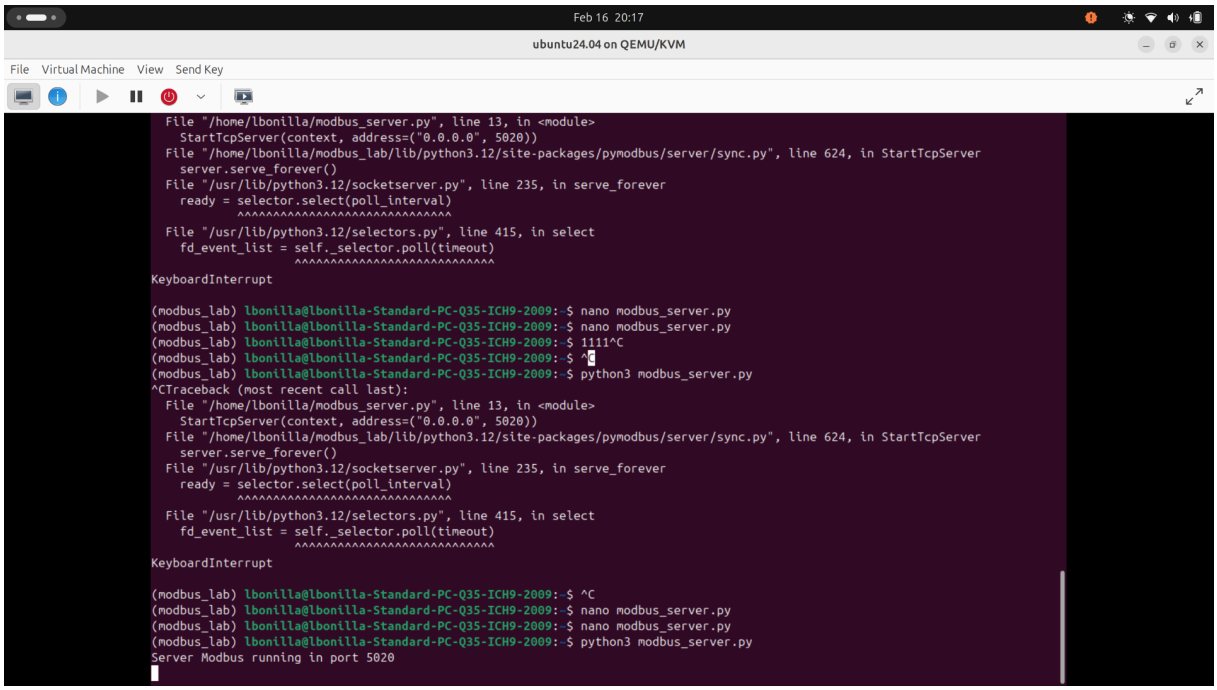

Figura 2: Inicialización del Servidor Modbus en Ubuntu

2.3. Comunicación con modbus-cli

Desde la máquina Kali, se utilizó la herramienta `modbus-cli` para comunicarse con el servidor Modbus y demostrar la falta de autenticación.

2.3.1. Lectura de Coils

```
modbus read 192.168.1.66 %M1 5
```

Listing 4: Lectura de coils usando modbus-cli

Resultado obtenido:

```
%M1      1
%M2      1
%M3      1
%M4      1
%M5      1
```

2.3.2. Escritura de valores (sin autenticación)

Se demostró que es posible escribir valores sin ningún tipo de autenticación:

```
# Escribir valor 0 en la posicion 1
modbus write 192.168.1.66 1 0

# Verificar el cambio
modbus read 192.168.1.66 1 5
```

Listing 5: Escritura y verificación de valores

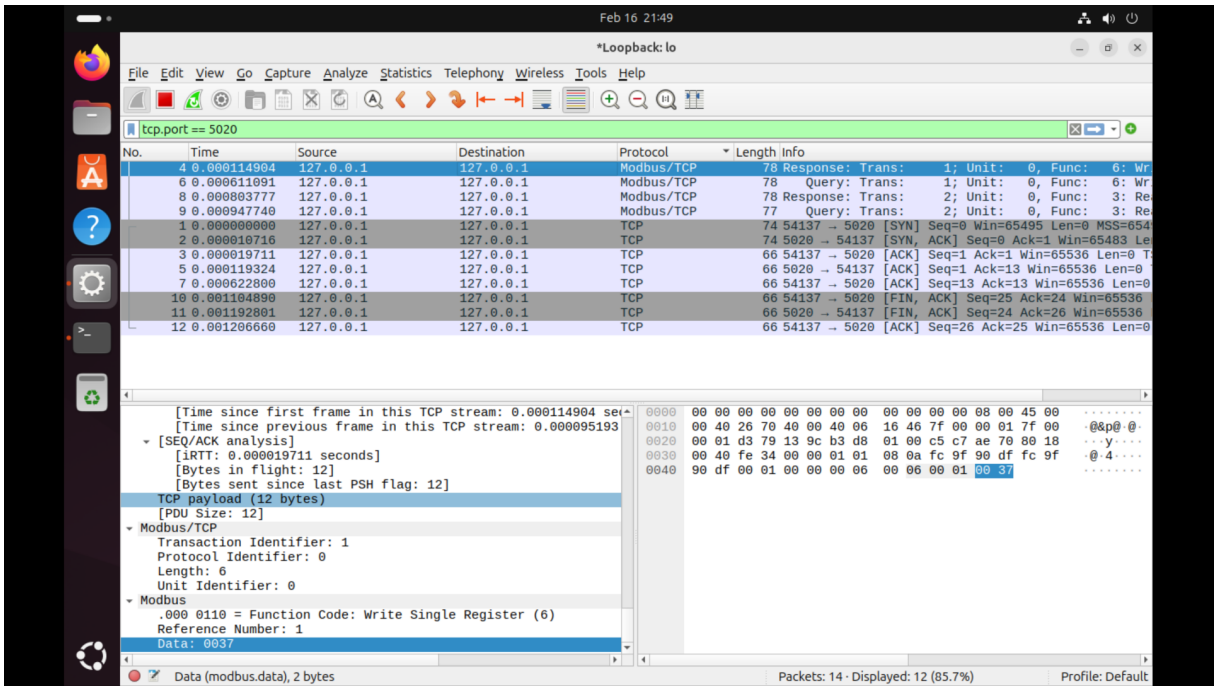

Figura 3: Comunicación con el servidor Modbus usando modbus-cli

2.4. Escaneo con Nmap

Se utilizó Nmap para descubrir dispositivos Modbus en la red y obtener información del servidor.

2.4.1. Descubrimiento de hosts

```
nmap -sP 192.168.1.0/24
```

Listing 6: Escaneo de red con Nmap

2.4.2. Escaneo de puertos

```
nmap -A 192.168.1.66 -p-
```

Listing 7: Escaneo completo de puertos

El resultado mostró el puerto 502/tcp abierto con el servicio `mbap` (Modbus Application Protocol).

2.4.3. Script de descubrimiento Modbus

Se utilizó el script NSE de Nmap para obtener información detallada del dispositivo:

```
nmap 192.168.1.66 -p 502 --script modbus-discover.nse
```

Listing 8: Uso del script modbus-discover

Resultado:

```
PORT     STATE SERVICE
502/tcp  open  Modbus
| Modbus-discover:
|   sid 0x1:
|     error: SLAVE DEVICE FAILURE
|_    Device identification: PyModbus Inc. PM 1.0
```

Este resultado demuestra que la información de identificación del dispositivo (VendorName, ProductCode, etc.) está expuesta a cualquier atacante en la red.

3. Using Python and Scapy to Communicate over Modbus

3.1. Configuración de Scapy para Modbus

Scapy es una herramienta de manipulación de paquetes que permite construir, enviar, capturar y analizar paquetes de red. Para soportar el protocolo Modbus, se instalaron módulos adicionales del framework smod.

```
# Descargar framework smod
git clone https://github.com/enddo/smod.git

# Copiar modulos al directorio de Python
sudo mkdir -p /usr/lib/python2.7/dist-packages/Modbus/
sudo cp smod-master/System/Core/*.py /usr/lib/python2.7/dist-packages/
   Modbus/
```

Listing 9: Instalación de módulos Modbus para Scapy

3.2. Deshabilitación de RST automático

Linux envía paquetes RST automáticamente para conexiones TCP forjadas. Para evitar interferencias, se deshabilitó esta funcionalidad:

```
sudo iptables -A OUTPUT -p tcp --tcp-flags RST RST -s 192.168.1.64 -j
   DROP
```

Listing 10: Regla iptables para deshabilitar RST

3.3. Creación de paquetes Modbus con Scapy

Se utilizó Scapy interactivamente para crear y analizar paquetes Modbus:

```
>>> from Modbus.Modbus import *
>>> ip = IP(src='192.168.1.64', dst='192.168.1.66')
>>> tcp = TCP(sport=12345, dport=502, flags='S')
>>> pkt = ip/tcp/ModbusADU()/ModbusPDU01_Read_Coils()
>>> pkt.show()
```

Listing 11: Creación de paquetes en Scapy

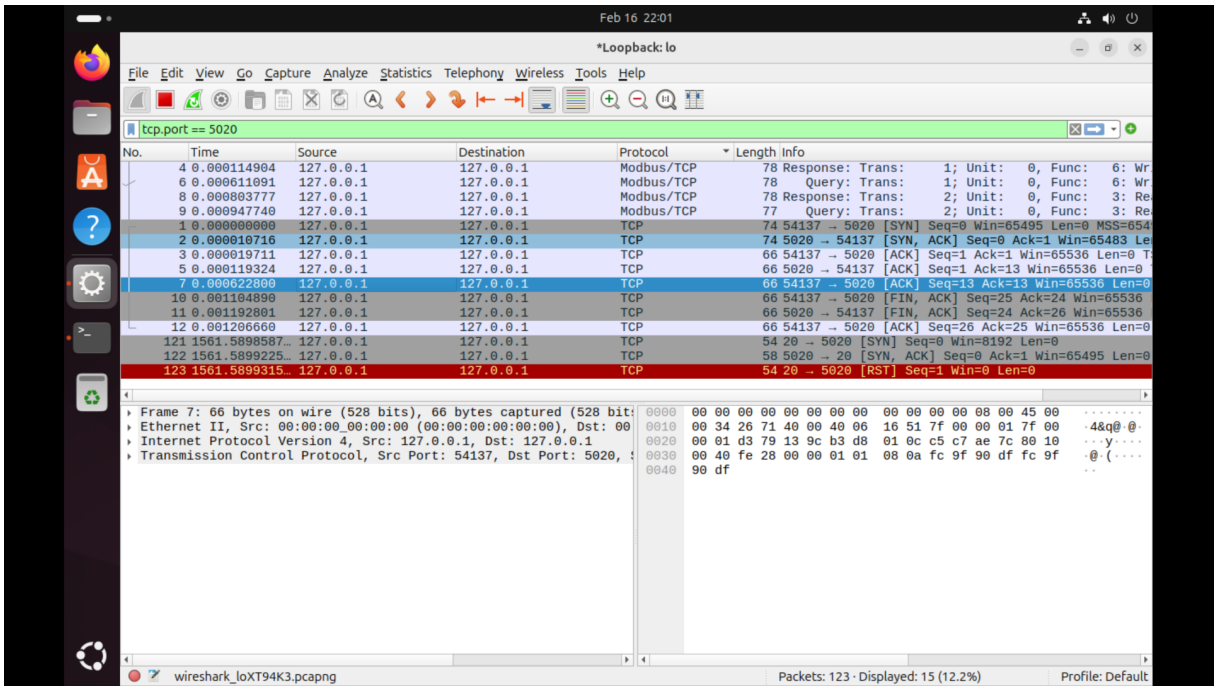

Figura 4: Creación de paquetes Modbus en Scapy

3.4. Script de comunicación Modbus con TCP Handshake

Se desarrolló un script completo que establece una conexión TCP, envía solicitudes Modbus y procesa las respuestas:

```
from scapy.all import *
from Modbus.Modbus import *
import random

# Configuracion
srcIP = '192.168.1.64'
srcPort = random.randint(1024, 65535)
dstIP = '192.168.1.66'
dstPort = 502
seqNr = random.randint(444, 8765432)
ackNr = 0

def tcpHandshake():
    global seqNr, ackNr
    ip = IP(src=srcIP, dst=dstIP)
    SYN = TCP(sport=srcPort, dport=dstPort, flags='S', seq=seqNr)
    pktSYNACK = sr1(ip / SYN)

    ackNr = pktSYNACK.seq + 1
    seqNr = seqNr + 1
    ACK = TCP(sport=srcPort, dport=dstPort, flags='A', seq=seqNr, ack=
    ackNr)
    send(ip / ACK)
    return ip / ACK

def connectedSend(pkt):
    pkt[TCP].flags = 'PA'
```

```
    pkt[TCP].seq = seqNr
    pkt[TCP].ack = ackNr
    send(pkt)

# Establecer conexion
ConnectionPkt = tcpHandshake()

# Crear y enviar solicitud Modbus
ModbusPkt = ConnectionPkt/ModbusADU()/ModbusPDU01_Read_Coils()
ModbusPkt[ModbusADU].unitId = 1
ModbusPkt[ModbusPDU01_Read_Coils].quantity = 5

connectedSend(ModbusPkt)
```

Listing 12: Script de comunicación Modbus con Scapy

El script demuestra cómo es posible forjar paquetes Modbus completamente funcionales utilizando Python y Scapy, sin necesidad de autenticación.

4. Replaying Captured Modbus Packets

4.1. Captura de tráfico con Wireshark

Se utilizó Wireshark para capturar el tráfico Modbus generado por el script de Nmap:

```
nmap 192.168.1.66 -p 502 --script modbus-discover.nse
```

Listing 13: Generación de tráfico Modbus para captura

En Wireshark se aplicó el siguiente filtro para visualizar únicamente el tráfico Modbus:

```
ip.addr== 192.168.1.66 && tcp.port == 502
```

4.2. Exportación de paquete como Hex Stream

Para reproducir un paquete capturado, se exportó como cadena hexadecimal:

1. Seleccionar un paquete Modbus Query (Request)
2. Click derecho → Copy → ...as a Hex Stream

4.3. Importación y manipulación en Scapy

El paquete capturado se importó en Scapy para su análisis y modificación:

```
>>> from Modbus.Modbus import *
>>> import binascii

# Convertir hex stream a binario
>>> raw_pkt = binascii.unhexlify('000c298f792c000c29f27ece...')

# Crear paquete Scapy
>>> Modbus_pkt = Ether(raw_pkt)
>>> Modbus_pkt.show()
```

Listing 14: Importación de paquete desde hex stream

El resultado muestra la estructura completa del paquete con todas sus capas (Ethernet, IP, TCP, ModbusADU, ModbusPDU):

```
###[ Ethernet ]###
  dst= 00:0c:29:8f:79:2c
  src= 00:0c:29:f2:7e:ce
###[ IP ]###
  src= 192.168.1.64
  dst= 192.168.1.66
###[ TCP ]###
  sport= 44828
  dport= 502
###[ ModbusADU ]###
  transId= 0x0
  unitId= 0x1
###[ Report Slave Id ]###
  funcCode= 0x11
```

4.4. Modificación y reenvío

Una vez importado, el paquete puede ser modificado y reenviado:

```
# Cambiar Transaction ID
>>> Modbus_pkt[ModbusADU].transId = 0x1234

# Cambiar direccion de inicio
>>> Modbus_pkt[ModbusPDU01_Read_Coils].startAddr = 10

# Verificar cambios
>>> Modbus_pkt[ModbusADU].show()
```

Listing 15: Modificación de campos del paquete

Esta capacidad de capturar, modificar y reproducir paquetes demuestra la vulnerabilidad del protocolo ante ataques de replay.

5. Análisis de Vulnerabilidades

5.1. Vulnerabilidades identificadas

Durante el desarrollo del laboratorio se identificaron las siguientes vulnerabilidades:

Vulnerabilidad	Descripción
Sin Encriptación	Todo el tráfico Modbus TCP se transmite en texto plano. Los valores de registros, comandos y respuestas son visibles para cualquier atacante con acceso a la red.
Sin Autenticación	No se requieren credenciales para conectarse al servidor Modbus. Cualquier cliente puede leer y escribir valores.
Sin Autorización	No existen niveles de acceso. Un cliente puede ejecutar cualquier función Modbus sin restricciones.
Sin Integridad	Los paquetes no incluyen verificación de integridad criptográfica, permitiendo modificaciones no detectadas.
Exposición de información	El protocolo expone información del dispositivo (vendor, modelo, versión) que puede ser usada para identificar vulnerabilidades conocidas.
Replay Attacks	Los paquetes capturados pueden ser reproducidos sin modificación o con cambios en los parámetros.

Cuadro 2: Vulnerabilidades identificadas en Modbus TCP

5.2. Códigos de función Modbus utilizados

FC (Hex)	FC (Dec)	Descripción
0x01	1	Read Coil Status
0x02	2	Read Discrete Inputs
0x03	3	Read Holding Registers
0x04	4	Read Input Registers
0x05	5	Write Single Coil
0x06	6	Write Single Holding Register
0x0F	15	Write Multiple Coils
0x10	16	Write Multiple Holding Registers
0x11	17	Report Slave ID
0x2B	43	Read Device Identification

Cuadro 3: Códigos de función Modbus

6. Conclusiones

En este laboratorio se demostró de manera práctica las vulnerabilidades inherentes al protocolo Modbus TCP/IP:

1. **Falta de seguridad por diseño**: Modbus fue diseñado en 1979 para operar en redes propietarias aisladas, sin considerar amenazas de seguridad modernas.

2. **Facilidad de ataque**: Con herramientas básicas como Nmap, modbus-cli y Scapy, es posible descubrir, comunicarse y manipular dispositivos Modbus sin ninguna barrera de seguridad.

3. **Exposición de información**: Los dispositivos Modbus revelan información de identificación que puede ser utilizada para buscar vulnerabilidades conocidas en bases de datos como ICS-CERT.

4. **Ataques de replay**: La falta de mecanismos de integridad y freshness permite capturar y reproducir comandos legítimos.

5. **Necesidad de controles compensatorios**: Dado que el protocolo no puede ser modificado sin romper compatibilidad, es necesario implementar controles de seguridad a nivel de red (segmentación, firewalls industriales, IDS/IPS específicos para ICS).

7. Referencias

- Ackerman, P. (2017). *Industrial Cybersecurity*. Packt Publishing. Capítulo 2: Insecure by Inheritance.
- Modbus Organization. *Modbus Application Protocol Specification V1.1b3*.
- pyModbus Documentation: `https://github.com/riptideio/pyModbus`
- Scapy Documentation: `https://scapy.net/`
- Nmap NSE Scripts: `https://nmap.org/nsedoc/scripts/modbus-discover.html`