Report for the CPS Final Project

Andrea Auletta
andrea.auletta@studenti.unipd.it
Niccolò Zenaro
niccolo.zenaro@studenti.unipd.it

February 16, 2025

Contents

Abstract

Modbus is a commonly used protocol in Supervisory Control And Data Acquisition (*SCADA*) environments for monitoring, control and data acquisition. Despite its wide popularity, Modbus is not secure because when it was developed and adopted (1979) security was not considered to be a concern in isolated Industrial Control Systems (*ICS*), thus is not designed to be secure like modern IT networks. Among the various attacks, 4 different taxonomies can be identified to facilitate formal risk analysis efforts for clarifying the nature and the scope of the security threats on Modbus systems and networks.

1 Introduction and objectives

1.1 Modbus

The Modicon Commuication Bus (*Modbus*) protocolo operates in a master-slave or server-client based model. The master devices initiates the queries while the slave devices respond to all such queries. Masters can either send a broadcast message to all the slaves or indivdually poll a specific device. All the experiments run in this work, like in the original paper [1] are focused on TCP/IP implementation, while Modbus protocol can be implemented also on top of several communication networks like serial or UDP.

Modbus TCP messages are wrapped in TCP/IP header and transmitted over an ethernet-based Modbus network between different devices. The *Modbus Slaves* listen for incoming TCP connections on port 503, that has been selected by us, and once a connection has been established the Protocol Data Unit (*PDUs*) are exchanged and encapsulated in TCP messages. The protocol itself can be broken down into six sections:

- Transaction Identifier: a 2-byte field that is used to correlate request and responses; it is easily predictable due to poor randomization.
- Protocol identifier: a 2-byte field that for Modbus is always set to 0.
- Length: a 2-byte field that indicates the length of remaining bytes in the payload.
- Unit Identifier: 1 byte that is used to identify the specific slave at an IP address.

- Function Code: is a 1-byte field that indicates the action requested my master. For example, reading and writing coils, registers or holding registers.
- Data: this field has variable length, with values associated with the various function codes.

1.2 Attack identification

In general attacks on Modbus systems and networks can exploit protocol's specifications, i.e. they are common to all Modbus systems/networks that are conform to the protocol specifications. The attack identification methodology used by the authors [1] involves an analysis of each protocol, that leads to four groups or threat categories: *interception*, *interruption*, *modification*, *fabrication*. The main targets for the Modbus protocol are the master, the field device, the serial communication links and messages. Attacks were implemented based on the possibility of the system to have a Modbus sniffer and a packet injector, that also could block, modify or fabricate arbitrary Modbus messages or sequences of messages.

Fifteen attacks that exploit TCP protocols have been recognized, and as said require access to the master device, network communication path or field device. The most serious attacks are those that disable or bypass the master unit and seize control of field devices, this type of attacks affect the integrity and the availability of the messages or of the network; on the other hand attacks on confidentiality involve obtaining information on the network or on slave devices by simply reading messages.

1.3 Objectives

In this work we tried to emulate the various attacks defined by the authors, precisely we emulate one attack for each taxonomy identified in the paper. We built our own Modbus simulator with python libraries and then we produced python scripts for each attack. More precisely, *interception*, *interruption*, *modification* and *fabrication* identified in the original paper are described and implemented as:

- Interception: *Passive reconnaissance* involves passively reading Modbus messages or network traffic, intercepting the messages and reading field device data.

- Interruption: *TCP FIN flood* is an interruption attack that aims to launch spoofed TCP packets with the FIN flag set after a legitimate message from Modbus server to Modbus client, in order to close the TCP connection or cause important delays.

- Modification: *Response Delay* involves delaying a response message so that the master receives out-of-date information from slaves, and is done sniffing and modifying field device, sending the modified packet with a delay.

- Fabrication: *Rogue Interloper* is a sort of man-in-the-middle attack where a MITM device can sniff and fabricate messages.

2 System setup

This work was entirely done with python scripts, crafting master and slaves with `PyModbus` and working on TCP level with `Scapy`. Although these are two of the most famous python labraries for dealing with TCP Modbus packets, we found out that their documentation is often incomplete and not accurate, with most of the informations difficult to retrieve.

Wireshark was employed for sniffing the entire connection on the `Loopback interface` and on `port 503`, in order to obtain a feedback on what actually happened on our Modbus TCP network.

3 Experiments and discussions

In this section we describe the experiments done to emulate the attacks. We assume that the attacker knows IP address and the port where the devieces are listening. First of all, in all the experiments we have a master and a slave except for the Fabrication where we need only the server. The slave is a simple Modbus server that listens on port 503 and the master is a Modbus client that sends requests to the slave. So run the `slave.py` script on a terminal to start the server listening. In the file `master.py` there are three different function used to read, write coils and registers:

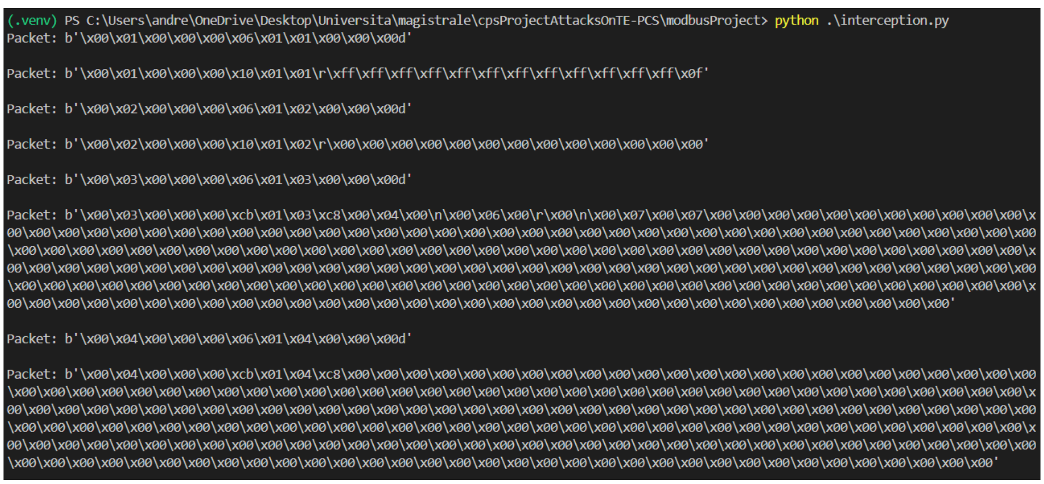
- master_read(): reads the data from all the registers of the slave;

- master_connected(time_wait): the master will be connected for time_wait seconds to the slave, it will read coils and will be disconnected time_wait seconds;
- master_write(): writes values in the holding registers of the slave.

One of these functions is called based on the attack type (change it in the main function of the file). The master have to be run in a different terminal.

3.1 Interception attack

The goal of this attack is to intercept a sent message from the master to the slave. In the master, the function `master_read()` is called. Before calling the master, run the `interception.py` script that will sniff the packet and print it. The attacker will listen to the network and when it found the packet with the right characteristics, it will print it. In the image 1 we can see that



```
(.venv) PS C:\Users\andre\OneDrive\Desktop\Universita\magistrale\cpsProjectAttacksOnTE-PCS\modbusProject> python .\interception.py
Packet: b'\x00\x01\x00\x00\x00\x06\x01\x01\x00\x00\x00d'

Packet: b'\x00\x01\x00\x00\x00\x10\x01\x01\r\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\x0f'

Packet: b'\x00\x02\x00\x00\x00\x06\x01\x02\x00\x00\x00d'

Packet: b'\x00\x02\x00\x00\x00\x10\x01\x02\r\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

Packet: b'\x00\x03\x00\x00\x00\x06\x01\x03\x00\x00\x00d'

Packet: b'\x00\x03\x00\x00\x00\xcb\x01\x03\xc8\x00\x04\x00\n\x00\x06\x00\r\x00\n\x00\x07\x00\x07\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

Packet: b'\x00\x04\x00\x00\x00\x06\x01\x04\x00\x00\x00d'

Packet: b'\x00\x04\x00\x00\x00\xcb\x01\x04\xc8\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

Figure 1: Results of the interception attack

there are 4 operations requested by the master and 4 responses from the slave. Here there is a conversion of the first packet present in the image:

1. Transaction ID = x00x01 = 1;
2. Protocol ID = x00x00 = 0;
3. Length = x00x06 = 6;

4. Unit ID = x01 = 1;
5. Function Code = x01 = 1 (read coils);
6. Data = x00x00x00;

We found the function codes and their assignd functions in the following site codes

3.2 Fabrication attack

The goal of this attack is to fabricate a fake message and send it to the server in order to change the value of a register, more precisely we substituted the value of the register 1 with 1337 of the holfing registers. For this attack, after starting the slave, run the `fabrication.py` script that will send the packet to the server. In the following image 2 we can see the register of interest before the attack: A packet with the following characteristics is sent to the

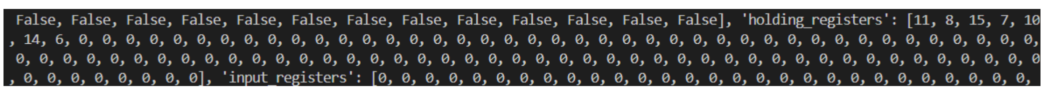


Figure 2: Register before the fabrication attack

server:

1. Transaction ID = x12x34 (chosen randomly);
2. Protocol ID = x00x00 = 0;
3. Length = x00x06 = 6;
4. Unit ID = x01 = 1;
5. Function Code = x06 = 6 (write single register);
6. Register Address = x00x01;
7. Value = x05x39 = 1337;

And this is the result 3:

```
e, False, False, False, False, False], 'holding_registers': [11, 1337, 14, 5, 10, 9, 14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'in
```

Figure 3: Register after the fabrication attack

3.3 Modification attack

The aim of this attack is to intercept the packet sent by the master to write in multiple registers and modify the value of one of them, and also delaying the response. When a packet with the function code 16 is sent, the attacker will intercept it and modify the value of the register and send it to the server. For this attack, after starting the slave, run the master with the function `master_write()` and `master_read()` to see the registers as in figure. Then run the `modification.py` script and with the master read the registers again to see the modification. In the following image 4 we can see the register of interest before the attack:

```
se, False, False, False, False, False, False, False, False, False, False, False, False, False, False], 'ho
lding_registers': [0, 0, 0, 0, 5, 6, 7, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 4: Register before the modification attack

After the modification, the value of the register is changed as shown in the following image 5:

```
e, False, False, False], 'holding_registers': [0, 4369, 0, 0, 5, 6, 7, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'input registers': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure 5: Register after the modification attack

In this case the original packet sent by the master is not blocked, but the aim is achieved anyway because the value of the register is changed starting from the packet sent by the master.

4 Conclusions

References

[1] Peter Huitsing et al. “Attack taxonomies for the Modbus protocols”. In: *International Journal of Critical Infrastructure Protection* 1 (2008), pp. 37–44.