University Of Piraeus
Department Of Digital System

VULNERABILITIES OF THE MODBUS PROTOCOL

Submitted by
Evangeliou I. Evangelia,

University of Piraeus, Athens, Greece
February 2018

Master's Supervisor: Xenakis Christos,

Associate Professor at University of Piraeus,
Department of Digital Systems

Περιεχόμενα

1. Introduction

Industrial Control Systems (ICSs), such as those in power grid including “Supervisory and Data Acquisition (SCADA)” systems, constitute the fundamental elements in the operation of modern power systems, providing a wide range of applications that range from applications in large control center to substations and controls of individual components. While originally ICSs had minimal networking capabilities, they have incorporated Ethernet modems and packet switched communications, to support communications with large number of devices.

One of the most commonly used protocols in industrial control systems is the Modicon Communication Bus (Modbus). It is a method used for transmitting information over serial lines between electronic devices. Typically is used to transmit signals from instrumentation and control devices back to a main controller or data gathering system, for example a system that measures temperature and humidity and communicates the results to a computer. Modbus is often used to connect a supervisory computer with a remote terminal unit (RTU) in supervisory control and data acquisition (SCADA) systems. It was not designed to provide security. At this thesis, it is confirmed that the Modbus protocol is vulnerable to many attacks.

This thesis is divided into eight chapters. Chapter 2 gives a brief description of the existing Modbus protocol and the Function Codes that are used. Chapter 3 reviews related work on Modbus vulnerabilities and attack detection systems. Chapter 4 gives a short abbreviation for Modbus and chapter 5 refers to vulnerabilities and classified attacks that exist. Chapter 6 introduces the theory behind the intrusion detection mechanisms and implements all function codes at Finite State Machines. Chapter 7 concludes the thesis.

2. Modbus Protocol

The Modbus protocol was developed in 1979 by Modicon, Incorporated, for industrial automation systems and Modicon programmable controllers. It has since become an industry standard method for the transfer of discrete/analog I/O information and register data industrial control and monitoring devices. Modbus is now a widely-accepted, open, public-domain protocol that requires a license, but does not require royalty payment to its owner.

Modbus is an application layer messaging protocol, positioned at level 7 of OSI network model. It provides client/server communication between devices connected on different types of buses or networks. It is currently implemented using:

- TCP/IP over Ethernet
- Asynchronous serial transmission over a variety of media (wire: EIA/TIA-232-E, EIA-422, EIA/TIA-485-A, fiber, radio, etc.)
- Modbus PLUS, a high speed token passing network

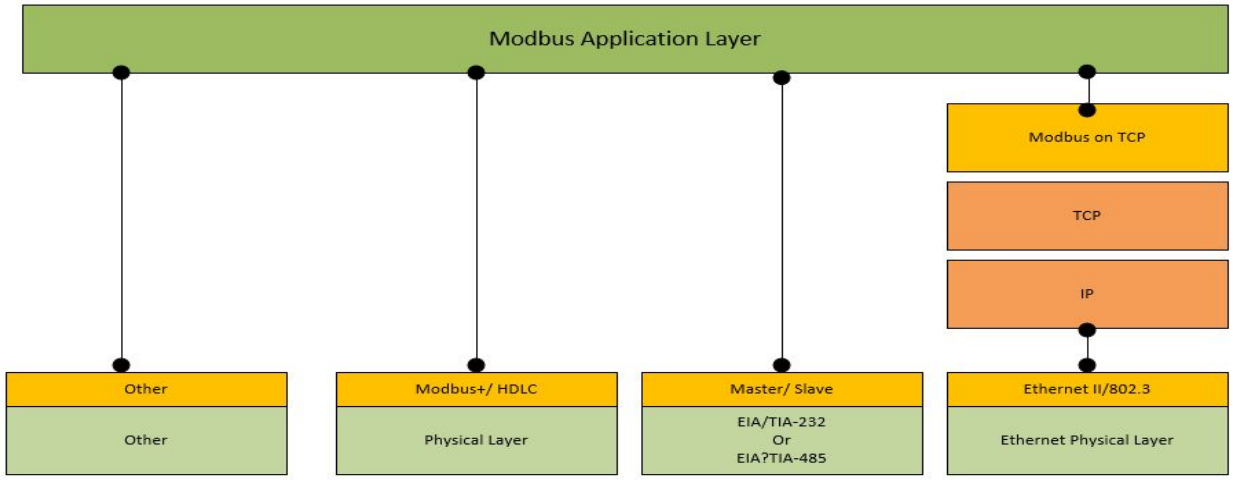

Figure 1: Modbus Communication stack

Modbus protocol allows an easy communication within all types of network architectures. Every type of devices (PLC, HMI, Control Panel, Driver, Motion Control, I/O Device etc.) can use Modbus protocol to initiate a remote operation and communication can be done as well either on serial line or on Ethernet TCP/IP networks.

2.1 Modbus description

The Modbus protocol defines a simple protocol data unit (PDU) independent of the underlying communication layers. The mapping of Modbus protocol on specific buses or network can introduce some additional fields on the application data unit (ADU). Modbus communications are two types: (i) query/response (communications between **master/slave**), or (ii) **broadcast** (a master sends a command to all the slaves). A Modbus transaction comprises a single query or response frame, or a single broadcast frame. A Modbus frame message contains the address of the intended receiver, the command the receiver must execute and the data needed to execute the command.

For the communication between Modbus devices, a **master-slave** (*client-server*) technique is used in which only one device (the *master/client*) can initiate transactions, which are called queries. The other devices (slaves/servers) respond by supplying the requested data to the master, or by taking the action requested in the query. A *slave* is any peripheral device which processes information and send its output to the master using Modbus. I/O transducer, valve, network drive or any other measuring device can play the role of slave, as it was referred above. A typical master device is a host computer running

appropriate application software, while other devices may function as both clients (master) and servers (slaves).

The Modbus application data unit (ADU) is built by the master/client that initiates the Modbus transaction. Masters can address individual slaves, or can initiate a broadcast message to all slaves. Slaves return a response to all queries addressed to them individually, while they cannot respond to broadcast queries, and do not initiate messages on their own – they only respond to queries from the master. A master's query will consist of a slave address (or a broadcast address), a function code defining the requested action, any required data, and an error checking field. A slave's respond consists of fields confirming the action taken, any data to be returned, and an error checking filed. If no error occurs, the slave's response contains the data as requested, but in case of an error in the received query, or if the slave is unable to perform the requested action, the slave will return an exception message as its response. The error check field of the slave's message frame allows the master to confirm that the contents of the message are valid. Traditional Modbus message are transmitted serially and parity checking is also applied to each transmitted character in its data frame.

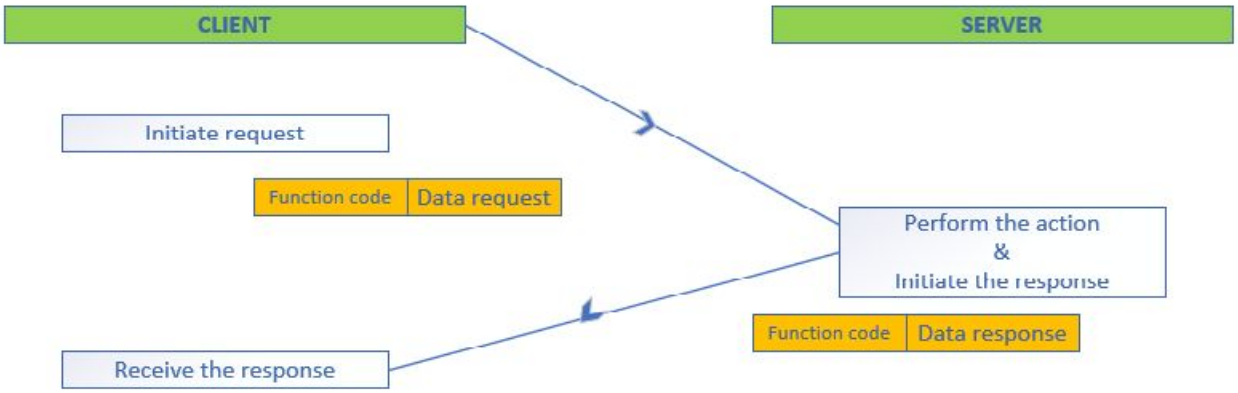

Figure 2: Modbus transaction (error free)

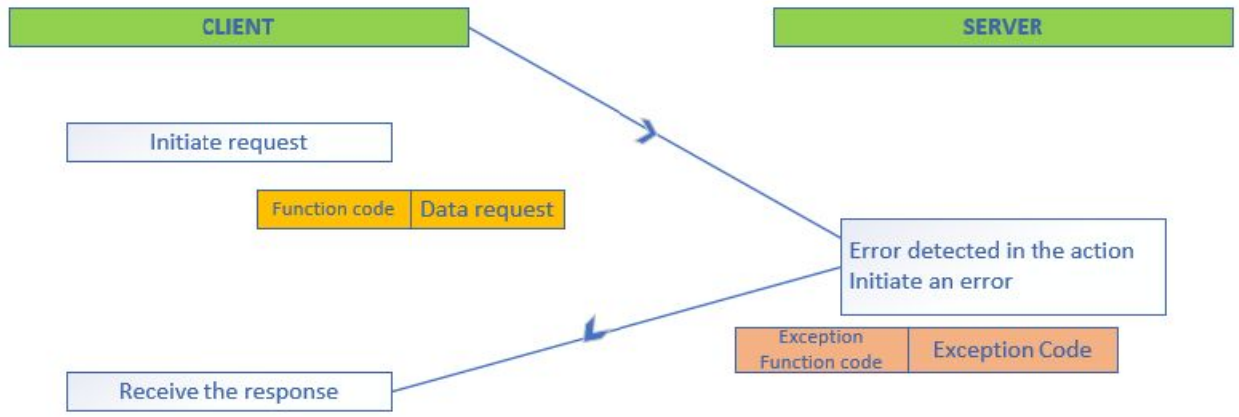

Figure 3: Modbus transaction (error exception)

Modbus, as an application protocol, defines rules for organizing and interpreting data, but remains simply a messaging structure, independent of the underlying physical layer. As it is shown below at figure 4, it defines protocol data unit (PDU) and some times, some additional fields are introduced on Application Data Unit (ADU), which are Additional Address and Error Check. In general, it is easy understandable, freely available and accessible to anyone and is supported by many manufacturers.

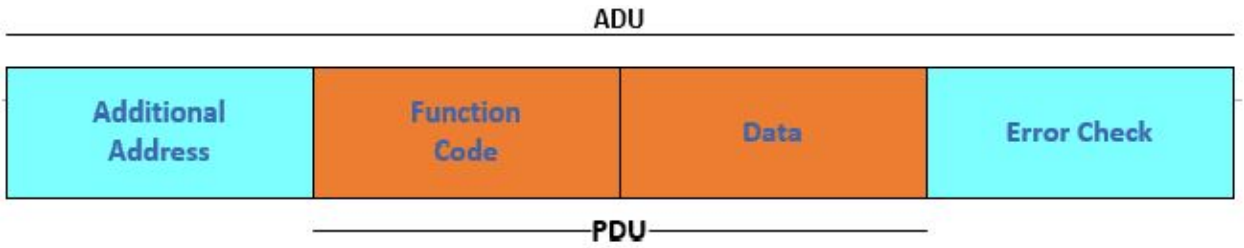

Figure 4: Modbus Structure

- The *Function Code* field is coded in one byte and its valid codes are between 1 ... 255 decimal. In case, that a client sends a message to a server, the *Function Code* has information about the kind of action to perform. Besides, *Function Code* "0" is not valid. Sub-function codes are added to some function codes to define multiple actions. Function and sub-function are referred analytically at chapter 2.3.
- The *data field* in a message sent from a client to a server devices contains additional information that server uses to take the action defined by the function code. Register addresses, quantity of items to be handled and the count of actual data bytes in the field are some items included at *data field*. In case the server does not require any additional information, the data field can be nonexistent (of zero length).

2.2 MODBUS TCP/IP

Modbus TCP/IP (also Modbus TCP) is simply the Modbus RTU protocol with a TCP interface that runs on Ethernet. The Modbus messaging structure is the application protocol that defines the rules for organizing and interpreting the data independent of the data transmission medium. TCP/IP refers to the Transmission Control Protocol and Internet Protocol, which provides the transmission medium for Modbus TCP/IP messaging. In other words, TCP/IP allows block of binary data to be exchanged between computers. It is a world-wide standard that serves as the foundation for the World Wide Web.

The primary function of TCP is to ensure that all packets of data are received correctly, while IP makes sure that messages are correctly addressed and routed. The TCP/IP combination is merely a transport protocol, and does not define what the data means or how the data is to be interpreted. The interpretation of data is a job of the application Modbus protocol.

To sum up, Modbus TCP/IP uses TCP/IP and Ethernet to carry the data of the Modbus message structure between compatible devices, and combines a physical network (Ethernet), with a networking standard (TCP/IP) and a standard method of representing data (Modbus as the application protocol). Simply, Modbus TCP/IP message is a Modbus communication encapsulated in an Ethernet TCP/IP wrapper. In practice, Modbus TCP embeds a standard Modbus data frame into a TCP frame, without the Modbus checksum.

The Modbus commands and user data are themselves encapsulated into the data container of a TCP/IP telegram without being modified in any way. However, the Modbus error checking field(checksum) is not used, as the standard Ethernet TCP/IP link layer checksum methods are used to guaranty data integrity. Further, the Modbus frame address field is supplanted by the unit identifier in Modbus TCP/IP, and becomes part of the Modbus Application Protocol (MBAP) header. The function code and data fields are absorbed in their original form.

Thus, a Modbus TCP/IP Application Unit (ADU) takes the form of a 7 bytes header, which consists of transaction identifier, protocol identifier, length field and unit identifier, and the protocol data unit, which consists of function code and data. ADU is embedded into the data field of a standard TCP frame and sent via TCP to system port 502. This port is specifically reserved for Modbus applications and all servers and clients receive and listen Modbus data via it.

The Modbus Application Protocol (MBAP) header has length of 7 bytes and its fields are:

- *Transaction/invocation identifier (2 bytes):* It is used for transaction pairing when multiple messages are sent along the same TCP connection by a client without waiting for a prior response.
- *Protocol identifier (2 bytes):* It is always 0 for Modbus services and other values are reserved for future extensions.
- *Length(2 bytes):* It is a byte count of the remaining fields and includes the unit identifier byte, function code byte, and the data fields.

- *Unit identifier (1 byte):* It is used to identify a remote server located on a non TCP/IP network (for serial bridging). In a tyical Modbus TCP/IP server application, the unit ID is set to 00 or FF, ignored by server, and simply echoed back in the response.

2.3 Function Codes

The Modbus protocol defines several function codes, each of which corresponds to a specific command. Specially, there are three categories of MODBUS Function codes. They are:

Public Function Codes

- Are well defined function codes
- Guaranteed to be unique
- Validate by the MODBUS.org community
- Publicly documented
- Have available conformance test
- Include both defined public assigned function codes as well as unassigned function codes reserved for future use

User-Defined Function Codes

- There are two ranges of user-defined function codes, i.e. 65 to 72 and from 100 to 110 decimal
- User can select and implement a function code that is not supported by he specification
- There is no guarantee that the use of the selected function code will be unique
- If the user wants to re-position the functionality as a public function code, he must initiate an RFC to introduce the change into the public category and to have a new public function code assigned
- MODBUS Organization, Inc. expressly reserves the right to develop the proposed RFC

Reserved Function Codes

- Function Codes currently used by some companies for legacy products and that are not available for public use

PUBLIC function codes {127 – 111}
User-Defined Function Codes {110 - 101}
PUBLIC function codes {100 – 73}

User-Defined Function Codes {72 - 66}
PUBLIC function codes {65 - 1}

Table 1: MODBUS Function Code Categories

2.3.1 Public Function Codes Definition

				Function Codes			
				code	*Sub code*	*(hex)*	*Section*
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	02		02	6.2
		Internal Bits Or Physical coils	Read Coils	01		01	6.1
			Write Single Coil	05		05	6.5
			Write Multiple Coils	15		0F	6.11
	16 bits access	Physical Input Registers	Read Input Register	04		04	6.4
		Internal Registers Or Physical Output Registers	Read Holding Registers	03		03	6.3
			Write Single Register	06		06	6.6
			Write Multiple Registers	16		10	6.12
			Read/Write Multiple Registers	23		17	6.17
			Mask Write Register	22		16	6.16
			Read FIFO queue	24		18	6.18
	File record access		Read File record	20		14	6.14
			Write File record	21		15	6.15
Diagnostics			Read Exception status	07		07	6.7
			Diagnostic	08	00-18,20	08	6.8
			Get Com event counter	11		OB	6.9
			Get Com Event Log	12		0C	6.10
			Report Slave ID	17		11	6.13
			Read device Identification	43	14	2B	6.21
Other			Encapsulated Interface Transport	43	13,14	2B	6.19

Notification: There is an analytical description for all Public Function Codes at "MODBUS APPLICATION PROTOCOL", Section 6. All Public Function Codes have been designed at Finite State Machines, below, at chapter 6 "*Detection Engine".*

3. Related Work

The Modbus/TCP Fuzzer (MTF) [1] is one of the first methods for discovering MODBUS protocol faults by providing unexpected input and monitoring for exceptions. There is an increasing need to test Modbus protocol implementations for security vulnerabilities, as devices are accessible via internet. MTF implementation was an important first step towards getting insights on the resilience of these systems against new threats arising from the internet. The fuzzer operates in phases so as to minimize the generated traffic and reduce testing time in order to reduce network noise and remain as stealth as possible based on the knowledge it collects. It was tested against eight Modbus implementations. Specifically, the Modbus/TCP Fuzzer (MTF) can be used to test both the master and slave sides of the protocol. Armed with the open specification of the Modbus/TCP protocol, the research team implemented an informed fuzzer that constructs almost-valid protocol packets with erroneous inputs. Firstly, MTF builds a list of possible attack cases described already in the published literature in the form of memory map boundaries and supported functions and then using this collected knowledge perform a guided fuzzing so as to reduce the injected frames and the number of tests. The major concerns for MTF were about DoS attack, because it was identified as inability to open new socket to the SUT, and as socket timeout errors, which indicated that the SUT was not responsive.

Trying to limit the wide set of threats, the majority of Modbus security mechanisms use detection techniques. Particularly, [2] proposes a set of signature based intrusion detection rules for the Modbus/TCP, Modbus over Serial Line protocols. A total of 50 rules are described with details on protocol requirements and construction of the IDS signature. These rules were developed to detect malicious activity on industrial control system MODBUS communication networks and are intended for use with SNORT Intrusion Detection System [3]. SNORT is a rule based open source network intrusion detection and intrusion prevention tool, which collects and logs network traffic, analyzes network traffic searching for rule violations, and alerts the administrator of suspicious activity. Quickdraw [4] is a Snort preprocessor and another set of Snort rules developed for industrial control systems using the MODBUS/TCP, DNP3, and Ethernet/IP communication standards. The quick draw rules include alerts for

invalid device configuration attacks, coil and register read and write attacks, high traffic volume attacks, malformed MODBUS application data unit (ADU) content attacks, unresponsive device scenarios, and port and function code scanning attacks.

Another approach that attempts to address the limitation at power transmission grid is through the hybrid network IDS for protective digital relays. In [5], the authors have proposed a specification-based intrusion detection signature based on the execution of the hybrid automata that specify the communication rules and physical limits the system should obey. The proposal constitutes a novel use of network intrusion detection systems (NIDSs) tailored to detect attacks against networks that support micro-processor based on hybrid controllers and packet-switched communications that implement power grid protection schemes. The hybrid set of IDS rules is a blending of network communication signatures with physical constraints. The research team, firstly, created the hybrid automation that characterizes system's behavior, including both physical limits and communication patterns, and then, implemented the IDS rules using Bro Network Security Monitor, which included IP packet parsers for Modbus and DNP3 protocols. It was supposed 3 attack scenarios: injecting malicious packets, isolating the power transformer and imitating the Master controller's behavior. In [6], a deeper analysis at vulnerabilities and attacks to the huge heterogeneous network of smart grid is proposed such as solutions for these security challenges. The greater and greater number of intelligent devices, the customer data security, IP usage and commercial off-the-shelf hardware and software, the implicit trust between traditional power devices and the lifetime of power systems are some of the vulnerabilities which break the confidentiality and integrity of transmitted data. As consequence, the attackers exploit these vulnerabilities and cause different levels of damage. The authors classify into three categories the attacks: component-wise, protocol-wise and topology-wise. Besides, it is referred the major differences between IT and grid network security objectives which necessitate the need for new security solutions specific for smart grid network. IDS and IPS, vulnerabilities assessment, better authentication and authorization techniques for TLS and IPSec, VPN and PKI usage are only some of the needed measures proposed at this paper.

In [7], another approach is described in order to analyze protocols, applications and networks. "Attack Tree" technique, as it is known, initially described by Bruce Schneider and, although "fault trees" have long been an accepted system analysis technique, this methodology first applied to the domain information security Dr Dobb's Journal article in 1999. The first published application of attack trees to a network protocol was "An Attack Tree for the Border Gateway Protocol", which is currently under consideration by the IETF Routing Protocol Security working group. Building on these approaches, the project team relied heavily on attack trees to support later vulnerability analysis and testing of MODBUS/TCP-based devices, with the goal of identifying flaws that could result in the greatest damage to SCADA systems. The primary benefit of using attack trees is the focus analysis on measurable attack goals against real-world devices, networks, and protocol implementations. Another observation of experimental observation was that the trees significantly improved other lab members' ability to find new exploits in SCADA systems. Last but not least, the trees were useful for selecting the most appropriate mitigation for cutting off an avenue attack.

Modbus Protocol, as it is known, is vulnerable to flooding attacks. These commands involve injection of commands that result in disrupting the normal operation of the control system. The research team with

head Sajal Bhatia[8], showed that an anomaly-based change detection algorithm and signature-based Snort threshold module are capable of detecting Modbus flooding attacks. In comparison these intrusion detection techniques, the signature-based detection requires a carefully selected threshold value and anomaly-based change detection algorithm may have a short delay before detecting the attacks detecting on the parameters used. Moreover, a network traffic dataset of flooding attacks on the Modbus control system protocol was generated. Besides, Gao W. and Morris T. in [9] have described a set of 28 cyber attacks against industrial control systems which use MODBUS application layer network protocol. This paper also describes a set of standalone and state for signature based intrusion detection system rules which can be used to detect cyber attacks and to store evidence of attacks for post incident analysis. Standalone rules parse a single MODBUS packet looking for a match to a specific signature and if the signature is present in the parsed packet, then the packet is classified as a match and an alert is issued. State based rules require knowledge from previous MODBUS packets or from another source, such as a process sensor. The lack of digital signature or other means to ensure the integrity of network frames in industrial systems leads to many vulnerabilities between the HMI/MTU and the PLC. The paper was implemented with the MODBUS application layer protocol.

4. Abbreviaton

ADU Application Data Unit
HDLC High level Data Link Control
HMI Human Machine Interface
IETF Internet Engineering Task Force
I/O Input/Output
IP Internet Protocol
MAC Media Access Control
MB Modbus Protocol
MBAP Modbus Application Protocol
PDU Protocol Data Unit
PLC Programmable Logic Controller
TCP Transmission Control Protocol

5. Vulnerabilities & Attacks

5.1.1 Vulnerabilities

MODBUS is the most widely used SCADA Protocol. Supervisory Controls and Data Acquisition (SCADA) protocols are communications protocols designed for the exchange of control messages on industrial networks. Over the past three decades, several hundred of these protocols have been developed for

serial LAN, and WAN-based communications in a wide variety of industries including petrochemical, automotive, transportation and electrical generation/ distribution.

SCADA protocols have much vulnerability. MODBUS is one of the most vulnerable ones to cyber attacks. The MODBUS/TCP protocol implementation contains too much vulnerability that could allow an attacker to perform reconnaissance activity or issue arbitrary commands. Below are referred the basic sections/classes of MOSBUS/TCP protocol vulnerabilities:

1. **Lack of Confidentiality**: All MODBUS messages are transmitted in clear text across the transmission media.
2. **Lack of Integrity**: There is no integrity checks built into MODBUS application protocol. As a result, it depends on lower layer protocols to preserve integrity.
3. **Lack of Authentication**: There is no authentication at any level of the MODBUS protocol. One possible exception is some undocumented programming commands.
4. **Simplistic Framing**: MODBUS/TCP frames are sent over established TCP connections. While such connections are usually reliable, they have a significant drawback. TCP connection is more reliable than UDP but the guarantee is not complete.
5. **Lack of Session Structure**: Like many request/response protocols (I.e. SNMP, HTTP, etc.) MODBUS/TCP consists of short-lived transactions where the master initiates a request to the slave that results in a single action. When combined with the lack of authentication and poor TCP initial sequence number (ISN) generation in many embedded devices, it becomes possible for attackers to inject commands with no knowledge of the existing session.

These vulnerabilities allow an attacker to perform reconnaissance activity on the targeted network. The first vulnerability exists because a MODBUS slave device may return Illegal Function Exception responses for queries that contain an unsupported function code. An unauthenticated, remote attacker could exploit this vulnerability by sending crafted function codes to carry out reconnaissance on the targeted network.

An additional reconnaissance vulnerability is due to multiple illegal Address Exception responses generated for queries that contain an illegal slave address. An unauthenticated, remote attacker could exploit this vulnerability by sending queries that contain invalid addresses to the targeted network and gathering information about network hosts from returned messages.

Another vulnerability is due to lack of sufficient security checks in the MODBUS/TCP protocol implementation. The protocol specification does not include an authenticated mechanism for validating communication between MODBUS master and slave devices. This flaw could allow an unauthenticated, remote attacker to issue arbitrary commands to any slave device via a MODBUS master.

The MODBUS/TCP protocol contains another vulnerability that could allow an attacker to cause a denial of service (DoS) condition on a targeted system. The vulnerability is due to an implementation error in the affected protocol when processing Read Discrete Inputs request and response messages. . An unauthenticated, remote attacker could exploit the vulnerability by sending request or response parameters that contain malicious values for the data filed option to a system that contains the

vulnerable MODBUS/TCP implementation. As a consequence, the processing of the messages could trigger a DoS condition-attack to the vulnerable system.

Moreover, another attack can be MODBUS TCP packets that exceed the maximum length. MODBUS TCP is commonly used in SCADA and DCS networks for process control. MODBUS limits the size of the PDU to 253 bytes to allow the packet to be sent on a serial line, RS-485 interface. MODBUS TCP prepends a 7-byte MODBUS Application Protocol (MBAP) header to the PDU, and the MBAP_PDU is encapsulated in a TCP packet. This places an upper limit on legal packet size. An attacker creates a specially crafted longer than 260 bytes and sends it to a MODBUS client and server. If the client or server programmed incorrectly, this could lead to a successful buffer overflow or a denial-of-service attack.

5.2 Attacks against industrial control systems

There have been several documented incidents and cyber attacks affecting SCADA systems, which clearly illustrate the above critical infrastructure vulnerabilities. These cyber attacks have produced a variety of financial damage and harmful events to humans and their environment. In this section a set of cyber attacks against industrial control systems (ICS) are referred and grouped into four attack classes; reconnaissance, response and measurement injection, command injection, and denial of service.

1. ***Reconnaissance attacks***

Reconnaissance attacks gather control system network information, map the network architecture, and identify the device characteristics such as manufacturer, model number, supported network protocols, system address and system memory map. Below are referred four reconnaissance attacks against MODBUS servers; the address scan, the function code scan, the device identification attack, and the points scan. The address scan discovers ICS servers connected to a network. The function code scan identifies supported network operations which can be performed for an identified server. The device identification attack allows an attacker to learn a discovered device's vendor name, product code, major and minor revision etc. The points scan allow an attacker to build a device memory map. MODBUS is an open standard and is popular network protocol used for ICS devices. While these attacks are known to function against MODBUS servers the vulnerabilities the attacks exploit are general enough to also likely be found in other network protocols used for ICS.

Industrial control system users often develop standard hardware, software, control scheme parameter configurations which are duplicated throughout a control system. For example, an electric transmission system may use a standard panel which includes the same protective relays used at many substations throughout a single transmission system. A second example is pump stations for a gas pipeline. Pump stations are distributed along the gas pipeline to ensure product flow. Programmable logic controllers (PLC) and the programmation on those controllers will be similar throughout the system.

The combined results of the address scan, function code scan, the device identification attack, and the points scan can be used to generate a signature for MODBUS servers common to a particular company, use case, or vendor. Such signatures can also be used to build a database of vulnerabilities and exploits for each aforementioned category.

2. ***Response and Measurement Injection Attack***

Industrial Control systems commonly use polling techniques to continuously monitor the state of a remote process. Polling takes the form of a query transmitted from the server to the client. The state information is used to provide a human machine interface to monitor the process, to store process measurements in historians, and as part of feedback control loops which measure process parameters and take requisite control actions based upon process state.

Many industrial control system network protocols lack authentication features to validate the origin of packets. This enables attackers to capture, modify and forward response packets which contain sensor reading values. Industrial control system protocols also often take the first response packet to a query and reject subsequent responses as erroneous. This enables to craft response packets and use timing attacks to inject the responses into a network when they are expected by a client.

Response injection attacks take 3 forms. First, response injection attacks can originate from control of programmable logic controller or remote terminal unit, network endpoints which are the servers which respond to queries from clients. Second response injection attacks can capture network packets and alter contents during transmission from server to client. Finally, response injections may be crafted and transmitted by a third party device in the network. In this case, the response there may be multiple responses to a client query and the invalid response may assume prominence due to exploiting a race condition or due to secondary attack such as a denial of service attack which stops the true sever from responding.

3. Command Injection Attacks

Command injection attacks inject false control and configuration commands into a control system. Human operators oversee control systems and occasionally intercede with supervisory control actions. Hackers may attempt to inject false supervisory control actions into a control system network. Remote terminals and intelligent electronic devices are generally programmed to automatically monitor and control the physical process directly as a remote site. This programming takes the form of ladder logic, C code, and registers which hold key control parameters such as high and low limits gating process control actions. Hackers can use command injection attacks to overwrite ladder logic, C code, and remote terminal register settings.

The potential impacts of malicious command injections include interruption process control, interruption of device communications, interruption of device communications, unauthorized

modification of device configurations, and unauthorized modification of process set points. As mentioned in the response injection discussion above much industrial control system network protocols lack authentication features t validate the origin of packets. This enables attackers to capture and alter command packets. Additionally, attackers can craft original command packets and directly inject them into the control system network.

4. Denial of Service Attacks

Denial of service attacks against industrial control system attempt to stop the proper functioning of some portion of the cyber physical system to effectively disable the entire system. As such DOS attacks may target the cyber system or the physical system. DOS attacks against the cyber system target communication links or attempt to disable programs running on system endpoints which control the system, log data, and govern communications. DOS attacks against the physical system vary from the manual opening or closing of valves and switches to destruction of portions of the physical process which prevent operation. This work concentrates on DOS attack against the communication system. MODBUS TCP/IP is a routable protocol which allows other devices to initiate Dos attacks targeted to a victims IP address. A device on the network can become infected with malware and then the infected device can initiate a DOS attack against other devices on the network.

Traffic jamming is a class of DOS attacks in which high volumes of traffic are sent to a network endpoint. Attackers attempt to overwhelm the endpoint by either sending transmissions faster than they can be processed or by sending packets crafted to cause software errors which generate exceptions with crash the network stack, the running program , or the operating system of the targeted device.

Name	Classification
Address Scan	Reconnaissance
Function Code Scan	Reconnaissance
Device Identification	Reconnaissance
Naïve Read Playload Injection	NMRI
Invalid Read Playload Size	NMRI
Naïve False Error Response	NMRI
Sporadic sensor Measurement Injection Attack	NMRI
Slope Sensor Measurement Injection	CMRI
High Slope Measurement Injection	CMRI

High Frequency Measurement Injection	CMRI
Altered System Control Scheme	MSCI
Altered Actuator State	MSCI
Altered Control Set Point	MPCI
Force Listen Only Mode	MFCI
Restart communication	MFCI
Invalid Cyclic Redundancy Code	DOS
MODBUS Slave Traffic Jamming	DOS

6 The detection engine

When a client device sends a request to a server device it expects a normal response. One of four possible events can occur form the client's query for the client's query:

- If the server device receives the request without a communication error, and can handle the query normally, it returns a normal response.
- If the server does not receive the request due to a communication error, no response is returned. The client program will eventually process a timeout condition for the request.
- If the server receives the request, but detects a communication error (parity, LRC, CRC etc) no response is returned. The client program will eventually process a timeout condition for the request
- If the server receives the request without a communication error, but cannot handle it, the server will return an exception response informing the client of nature of the error.

With the exception function codes, the client's application program can recognize the exception response and can examine the data field for the exception code. The below table includes all the exception codes that the intrusion detection system can identify to alert for an error at the Modbus packet transmitting. The list has been produced based on the Public Function Codes which are referred at *Chapter 2.3.1.* At *Chapter 6.1,* there is a graphical state design of all the listed Exception Codes in the process of receiving a request packet.

A list of exception codes is below:

Code	**Name**	**Meaning**
01	Illegal Function	The **function code received in the query is not an allowable** action for the server (or slave). This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server (or slave) is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values.
01A	Force Listen Only Mode	Function Code == 0x08 && sub-function == 0x0004
01B	Restart Communication	Function Code == 0x08 && sub-function == 0x0001

01C	Clear Communication Event Log	Function Code == 0x08 && sub-function == 0x0001 && data == 0xFF00
01D	Change ASCII Input Length	Function Code == 0x08 && sub-function == 0x0003
02	Illegal Data Address	The **data address received in the query is not an allowable address** for the server (or slave). More specifically, the combination of **reference number and transfer length is invalid**. For a controller with 100 registers, the PDU addresses the first register as 0, and the last one as 99. If a request is submitted with a starting register address of 96 and a quantity of registers of 4, then this request will successfully operate (address-wise at least) on registers 96, 97, 98, 99. If a request is submitted with a starting register address of 96 and a quantity of registers of 5, then this request will fail with Exception Code 0x02 “Illegal Data Address” since it attempts to operate on registers 96, 97, 98, 99 and 100, and there is no register with address 100.
03	Illegal Data Value	A **value contained in the query data field is not an allowable value** for server (or slave). This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does NOT mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the MODBUS protocol is unaware of the significance of any particular value of any particular register.
04	Server Device Failure	An unrecoverable error occurred while **the server (or slave) was attempting to perform the requested action.**
05	Acknowledge	Specialized use **in conjunction with programming commands**. The server (or slave) has accepted the request and is processing it, but a **long duration** of time will be required to do so. This response is returned to prevent a timeout error from occurring in the client (or master). The client (or master) can next issue a Poll Program Complete message to determine if processing is completed.
06	Server Device Busy	Specialized use in conjunction with programming commands. The **server (or slave) is engaged in processing a long–duration program command**. The client (or master) should **retransmit** the message **later** when the server (or slave) is free.
08	Memory Parity Error	Specialized use in conjunction **with function codes 20(0x14) and 21(0x15)** and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server (or slave) attempted to read record file, but detected a **parity error in the memory**. The client (or master) can retry the request, but service may be required on the server (or slave) device.
0A	Gateway Path Unavailable	Specialized use in conjunction with gateways, indicates that the **gateway was unable to allocate an internal communication path** from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
0B	Gateway Device Target Failed to Respond	Specialized use in conjunction with gateways, indicates that **no response was obtained from the target device**. Usually means that the device is not present on the network.

6.1 Finite Sate Machine

The set of specification utilized by the detection engine to resolve malicious behavior is expressed through the use of a Finite State Machine (FSM). The Finite State Machine (FSM) is a framework used to model the behavior of a system by means of limited number of states, transitions between states, actions and events. FSM states are characterized as legitimate or malignant, and a transition from one state to another is triggered by the node's operation or actions. Every protocol execution is mapped to the appropriate state (i.e. legitimate or malignant). States that have not been specified are also considered as malignant. The developed specifications are divided into three sets, based on the host-node's communication condition (a) idle, (b) transmitting and (c) receiving states. In the following sections, the FSM states are presented.

The FSM has been designed and implemented based on function codes presented at section C.1 and exception codes presented above. At first, it is presented the generic processing of a MODBUS transaction in server side [figure 5]. Once the request has been processed by a server, a MODBUS response using the adequate MODBUS server transaction is built. A positive MODBUS response leads to a request execution and a negative MODBUS response executes an exception response which gives information concerning the error detection and its reason.

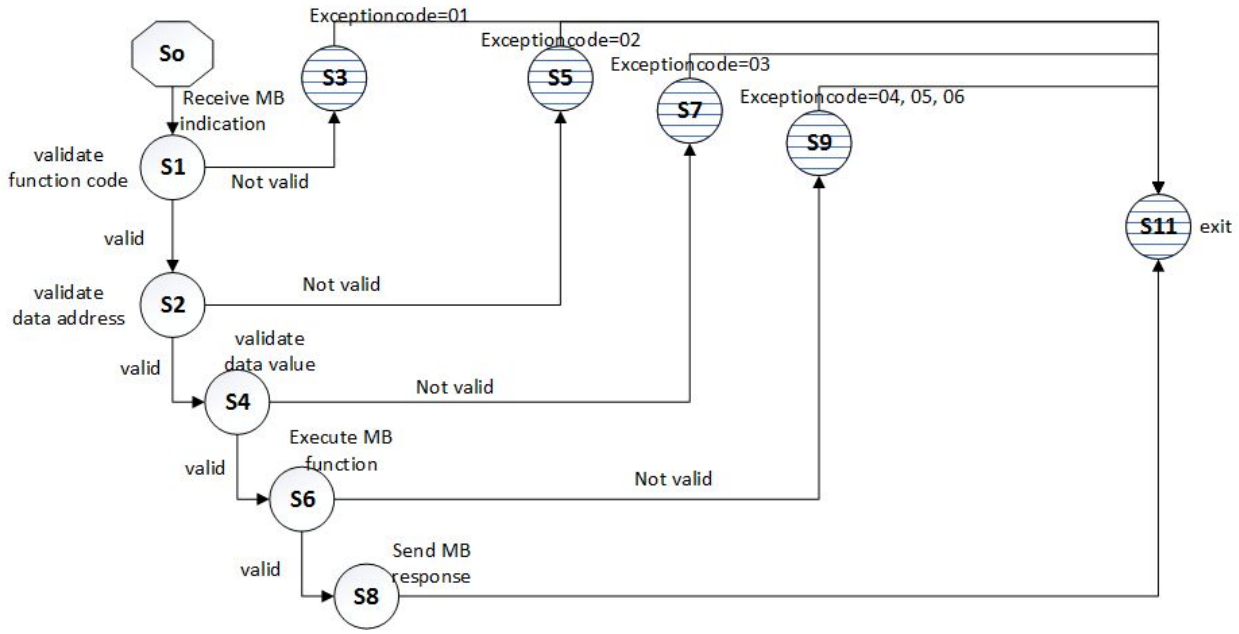

Figure 5: Server Side

The public function codes presented in the following figures [figure 6 - 11] show all positive MODBUS function codes which lead to a request execution. The first state for all diagrams is So, since in this condition the protocol is awaiting for a packet request from master side, and thus, it does not include any final state designating a malicious behavior. At this state, So, the engine is initialized and begins monitoring the protocol data unit (PDU) which include the function code byte and payload. All packets are monitored at state S1 checking if the function code is included in the white list. The white list is limited only to the Public Function Codes referred above. In case the function code does not belong to the white list an alert is generating with exception code 01, state S2. All MODBUS/TCP packets include a protocol identifier field. The protocol identifier is defined as 0 for all MODBUS/TCP packets. Other values are illegal. Exception code 01 also alerts for TCP traffic if the identifier is not 0. In effect, this means any TCP packet with second byte in the payload not equal to 0 is disallowed and should result in an alert.

For legal function codes the monitoring continue with states S6, S13, S15, S31, S19, S22, S26, S33, S50, S36, S43, S47. At this states, the engine checks if the value contained in the query data field is an allowable value for server. In case the value is illegal as described at Exception Codes and as it is shown in the diagrams, the system responds an alert with exception code 03 state S3. If there is not an alert, the engine continues with the data address checking. State S4 with exception code 02 will generated to inform the system that an invalid address is detected. The start_address and end_address are written in memory regions. The end_address may be computed by adding the length address to the start_address.

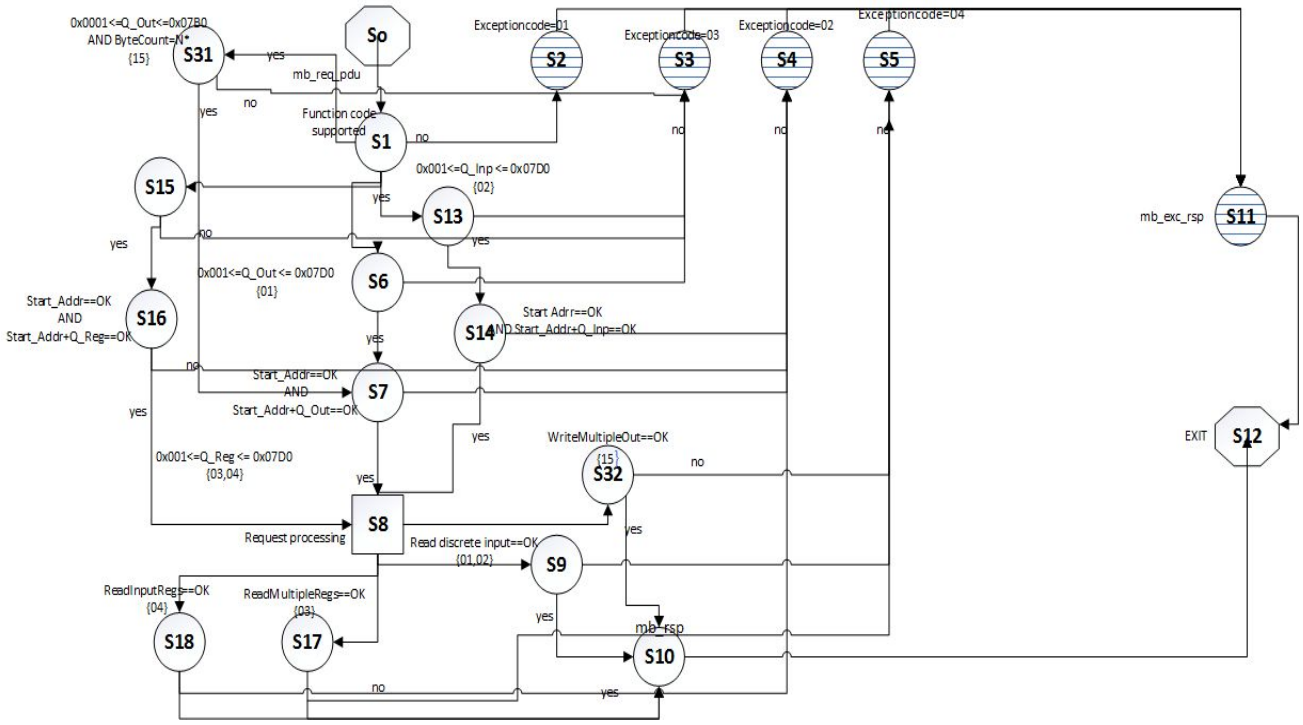

Figure 6: Function Codes 01, 02, 03, 04, 15

One of the most significant states of Finite State Machine is state S8. At this state all requests are processing and no error detection could be occurred. The final state for all diagrams is S12 which is the exit state. At this state the system is terminated. At this state, S12, is concluded state S10 with all positive checked responses and state S11 with negative MODBUS exception responses. All the malicious states are shown filled with lines. The rest no filled with lines states are not malicious. Every Finite State Machine is easily readable because all transmitting and receiving states are reasoned in the figures.

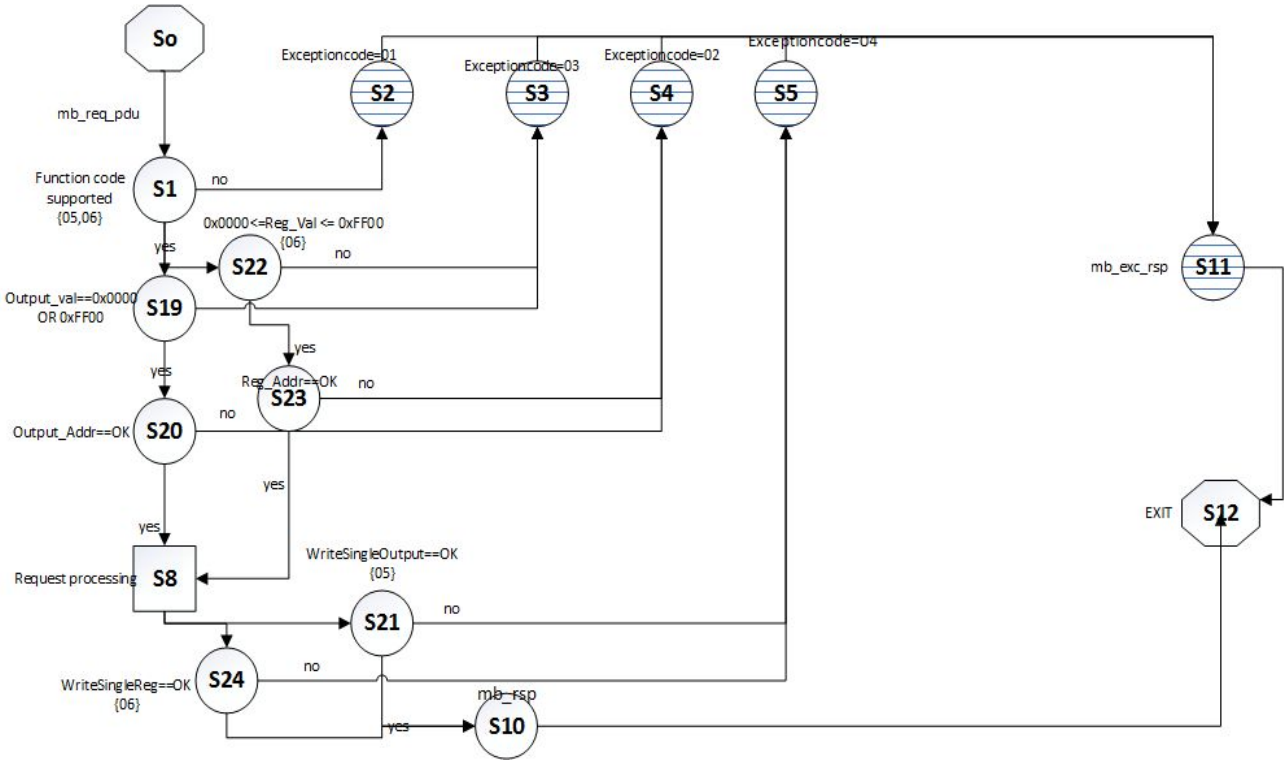

Figure 7: Function Codes 05, 06

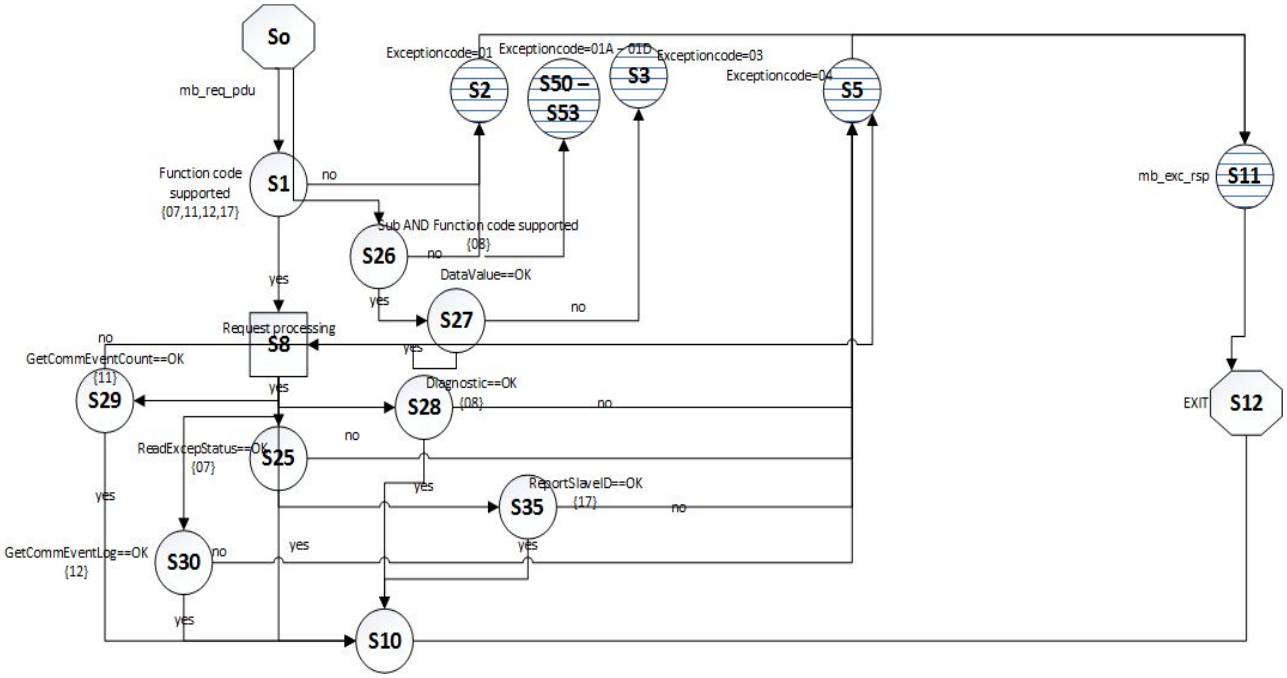

Figure 8: Function Codes 07, 11, 12, 17

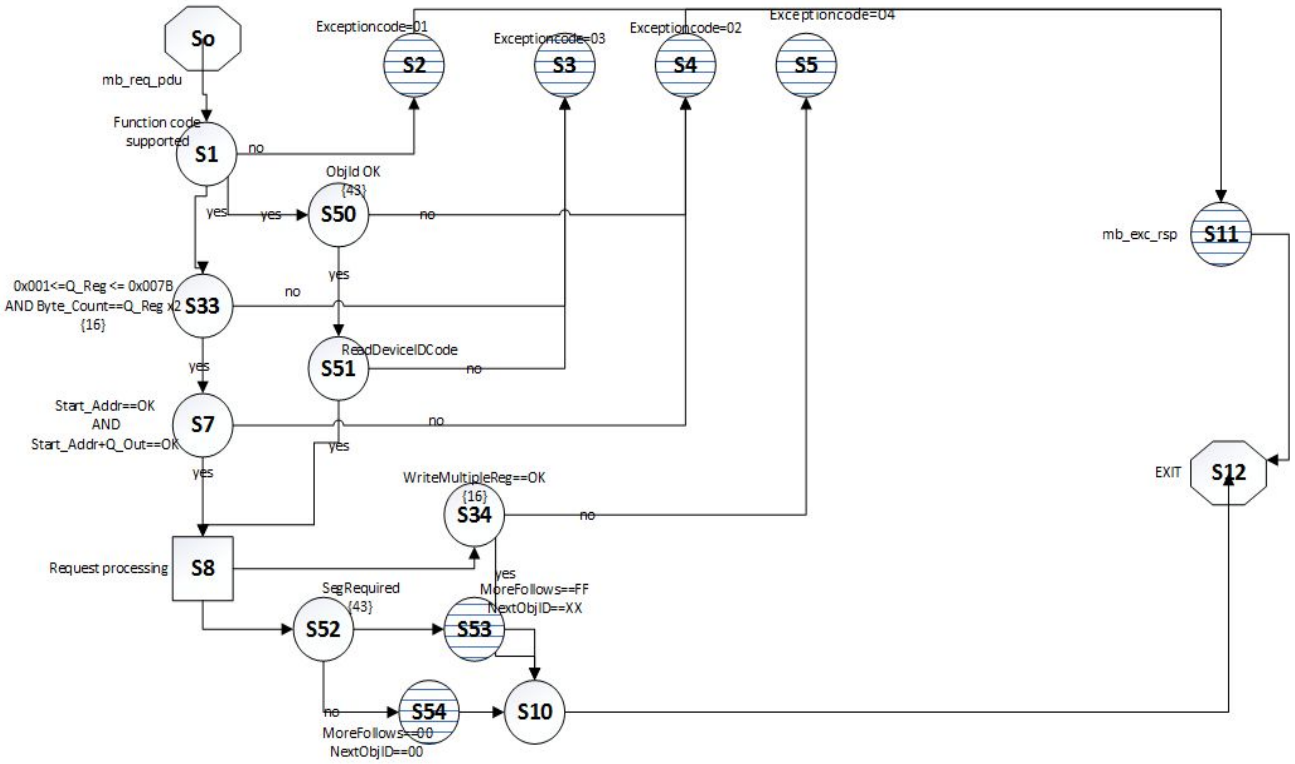

Figure 9: Function Codes 16, 43

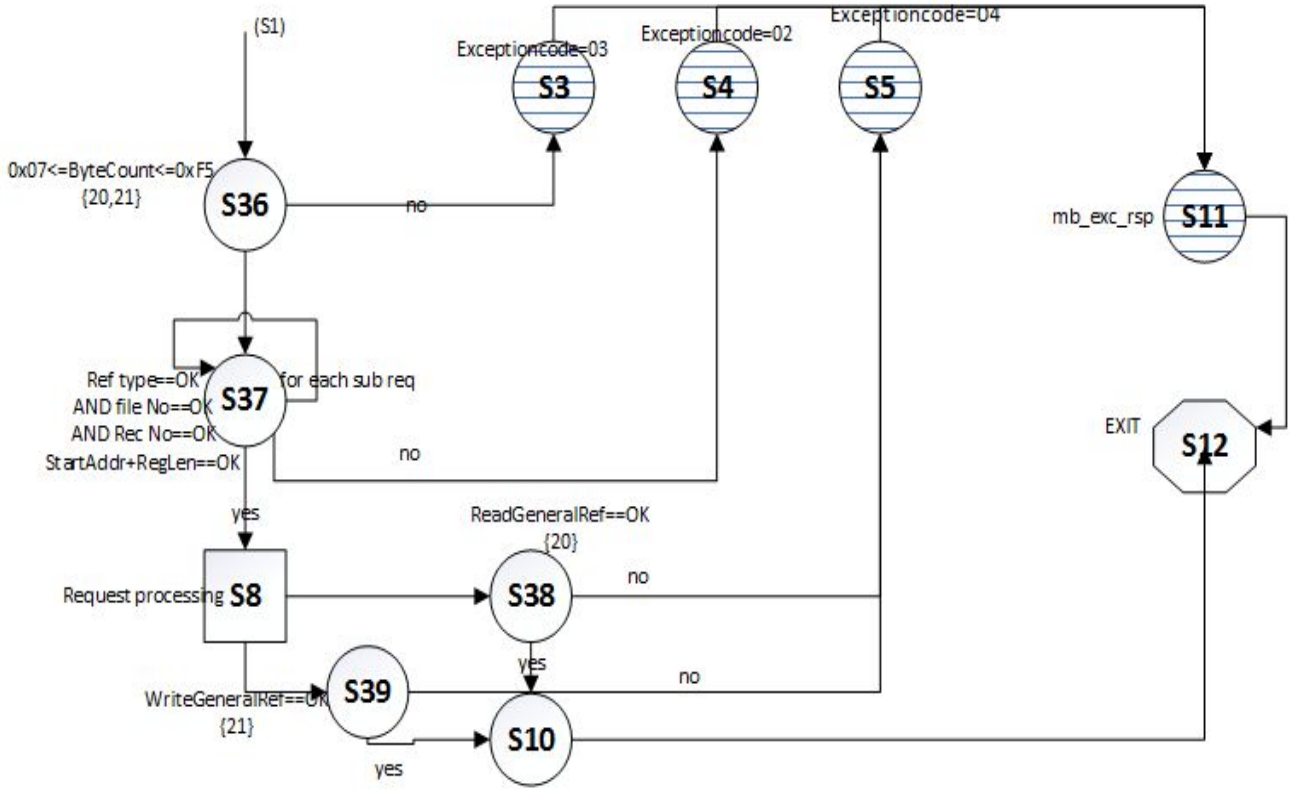

Figure 10: Function Codes 20, 21

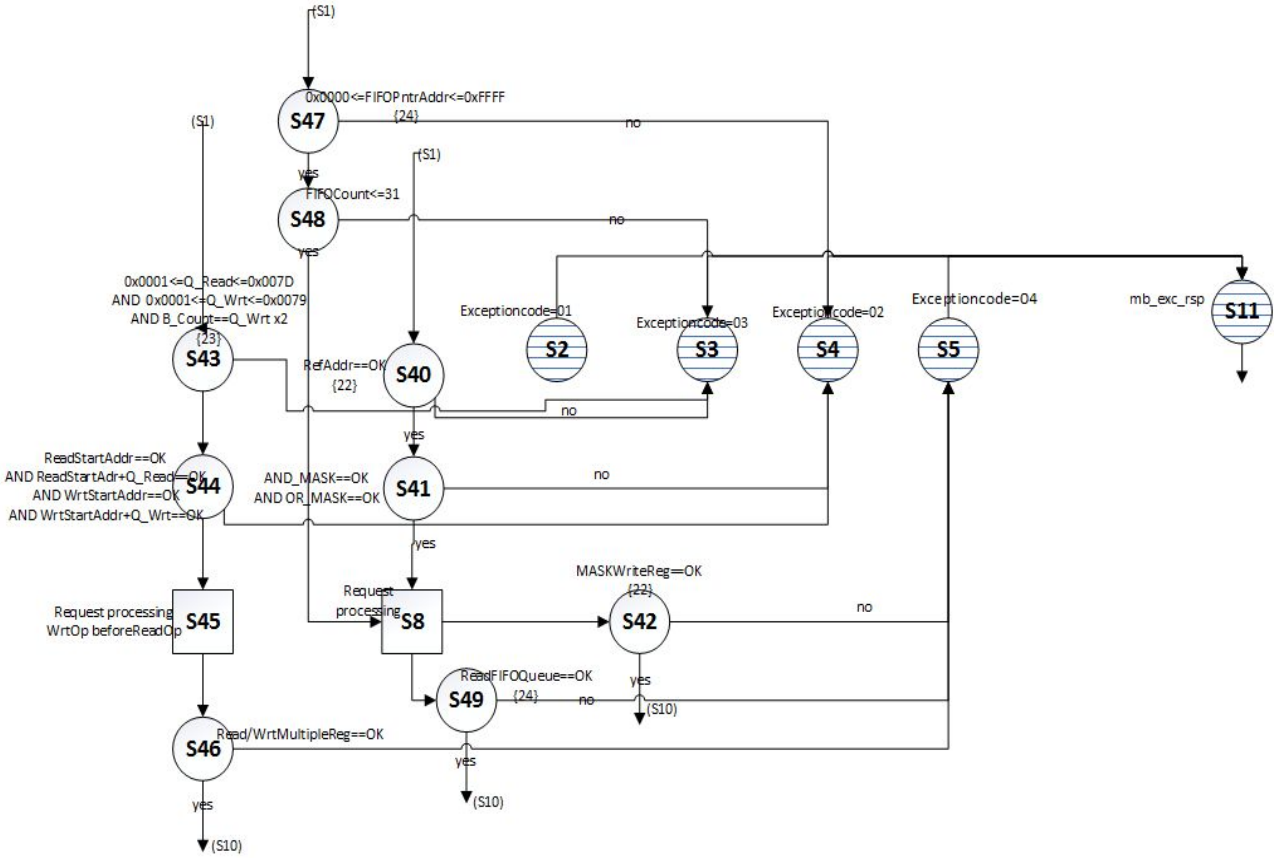

Figure 11: Function Codes 22, 23, 24

7 Conclusion

The use of regular IT networks for industrial control protocols has brought new security challenges to be addressed because of the different uses, requirements and restrictions of that kind of systems. MODBUS TCP/IP is a wide spread protocol that came from legacy serial communication systems. With no build in capabilities protection, monitoring and analysis, it is necessary to achieve a reasonable security level.

This thesis presents much vulnerability which could be exploited by hackers with dangerous results for MODBUS systems, such as network classified attacks. MODBUS TCP/IP is no other than an application protocol. So many intrusion detection engines could be used to analyze and monitor it. At this thesis, many Exception Codes are presented based on the Public Function Codes of the protocol. All these Exception Function Codes are implemented at Finite State Machines in combination with the MODBUS sequence response and the Public Function Codes. The exception codes could be enriched in case this analysis was implemented in a real time industrial system. In industrial systems, such as water, coil and solar power grid, MODBUS protocol is widely used. So an implementation at a system like the above examples could produce more exception codes because the parameters and setting points for the normal function are in variety.

8 References

1. Artemios G. Voyiatzis, Konstantinos Katsigiannis , Stavros Koubias, “A Modbus/TCP Fuzzer for Testing Internetworked Industria Systems”
2. Thomas H. Morris , Bryan A. Jones, Rayford B. Vaughn, Yoginder S. Dandass, “Deterministic Intrusion Detection Rules for MODBUS Protocols”, Hawaii 2013
3. Caswell, B. Bealeand, J. Foster, J. and Faircloth, “Snort2.0 Intrusion Detection”, Syngress, Feb.2003
4. Morris T., Vaughn R., Dandass Y. “A Retrofit Network Intrusion Detection System for MODBUS RTU and ASCII Industrial Control Systems”, Proceedings of the 45th IEEE Hawaii International Conference on System Sciences, January 4-7 2012
5. Georgia Koutsandria, Vishak Muthukumar, Masood Parvania, “A Hybrid Network IDS for Protective Digital Relays in the Power Transmission Grid”, Venice-Italy 2014
6. Fadi Aloula , A. R. Al-Alia , Rami Al-Dalkya , Mamoun Al-Mardinia , Wassim El-Hajjb, “Smart Grid Security: Threats, Vulnerabilities and Solutions”, International Journal of Smart Grid and Clean Energy, 2012
7. E.J. Byres, M. Franz and D. Mille, “The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems”, Lisbon – Portugal 2004
8. Sajal Bhatia, Nishchal Kush, Chris Djamaludin, James Akande, Ernest Foo, “Practical Modbus Flooding Attack and Detection”, New Zealand 2014
9. Wei Gao, Thomas H. Morris, “On cyber attacks and signature based intrusion detection for modbus based industrial control systems”, Journal of Digital Forensics, Security and Law, 2014
10. MODBUS APPLICATION PROTOCOL SPECIFICATION
11. MODBUS MESSAGING TCP/IP IMPLEMENTATION GUIDE
12. https://www.schneider-electric.com/en/faqs/FA168406/
13. https://www.lammertbies.nl/comm/info/modbus.html