Advanced security aspects on Industrial Control Network

PhD in Computer Science

Andrea Carcano

Advisor: Prof. Alberto Trombetta

Co-Advisor: Dr. Igor Nai Fovino

Università degli Studi dell'Insubria

Dipartimento di Scienze Teoriche e Applicate

February 2013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Security threats are one of the main problems of this computer-based era. All systems making use of information and communication technologies (ICT) are prone to failures and vulnerabilities that can be exploited by malicious software and agents. In the latest years, Industrial Critical Installations started to use massively network interconnections as well, and what it is worst they came in contact with the public network, i.e. with Internet. The net effect of such new trend, is the introduction of a new interleaved and heterogeneous architecture combining typical information system (e.g. data bases, web-servers, web-applications and web-activities), with real-time elements implementing the control functions of industrial plants. If, from a certain point of view, the advantages of such complex architectures are several (remote management functions, distributed control and management systems, on the fly monitoring etc.), on the other hand they introduce a new layer of exposure to malicious threats.

Industrial networks are responsible for process and manufacturing operations of almost every scale, and as a result the successful penetration of a control system network can be used to directly impact those processes. Consequences could potentially range from relatively benign disruptions, such as the disruption of the operation (taking a facility offline), the alteration of an operational process (changing the formula of a chemical process), all the way to deliberate acts of sabotage that are intended to cause harm. For example, manipulating a certain processes could cause pressure within a boiler to build beyond safe operating parameters. Cyber sabotage could result in injury or loss of life, including the loss of critical services (blackouts, unavailability of vaccines, etc.) or even catastrophic explosions. This happens due to the fact that most of the Critical Infrastructure use industrial control systems.

Over the past decade, there have been numerous incidents that have been identified as the result of a cyber incident. In *2000*, a man in Australia who was rejected for a government job was accused of using a radio transmitter to alter electronic data within a sewerage pumping station, causing the release of over two hundred thousand gallons of raw sewage into nearby rivers.

In *August 2006* the Browns Ferry nuclear plant in Alabama, U.S. was manually shut down because a number of reactor recirculation pumps failed [2]. It was later found that this failure had occurred due to an overload of network traffic. Some form of unspecified vulnerability existed which allowed for the overload of traffic to cause the system to be unresponsive. Whether or not the overload of network traffic was caused because of a Denial of Service (DoS) attack is not stated, however it is a plausible cause.

In *2007*, there was the Aurora Project: a controlled experiment by the Idaho National Laboratories (INL) in United States, which successfully demonstrated that a controller could be destroyed via a cyber attack. The vulnerability allowed hackers, which in this case were authorized security researchers at the INL, to successfully open and close breakers on a diesel generator out of synch, causing an explosive failure. In September 2007, CNN reported on the experiment, bringing the security of our power infrastructure into the popular media.[23]

In *2009* it is reported that Chinese and Russian spies penetrated the U.S. electrical power grid, leaving behind potentially disruptive software programs [3]. A senior, but unnamed U.S. official, claimed that both Chinese and Russian spies had attempted to map U.S. critical infrastructure, using network mapping tools. The U.S. Government has remained vague on technical details. In 2009 in Dallas, U.S, a hospital security guard, Jessie William McGraw (aka GhostExodus) took advantage of his position to install malware on hospital machines and also control the heating, ventilation and air-conditioning (HVAC) system. McGraw was caught as he uploaded videos and images to numerous websites.

In *June of 2010* the Stuxnet computer worm was identified by Belarus-based security firm, VirusBlokAda. Stuxnet. Stuxnet is a threat targeting a specific industrial control system likely in Iran, such as a gas pipeline or power plant. The goal of Stuxnet is to sabotage that facility by reprogramming programmable logic controllers (PLCs). It uses four zero-days in total to infect and spread, looking for SIMATIC WinCC and PCS 7 programs from Siemens, and then using default SQL account credentials to infect connected Programmable Logic Controllers (PLCs) by injecting a rootkit via the Siemens fieldbus protocol. Stuxnet then looks for automation devices using a frequency converter that controls the speed of a motor. If it sees a controller operating within a range of 8001200 Hz, it attempts to sabotage the operation. [25] Stuxnet drew the attention of the mass media through the fall of 2010 and it immediately raised the industrys awareness and illustrated exactly why industrial networks need to dramatically improve their security measures.

The aim of this thesis is to analyze the main vulnerabilities concerning Industrial Control Systems and explore different solutions to protect them. The interconnectivity of Industrial Control Systems with corporate networks and the Internet has significantly increased the threats to critical infrastructure assets. Meanwhile, traditional IT security solutions such as firewalls, intrusion detection systems and antivirus software are relatively ineffective against attacks that specifically target vulnerabilities in SCADA protocols. The thesis is organized as follows: the second chapter provides a brief overview about Industrial Control Networks and Critical Infrastructures. In the third chapter the details of the most important SCADA protocols, i.e. Modbus is presented. In the fourth chapter a deep analysis of the related works is done. The fifth chapter presents a threat model from an attacker point view with some experimental results about a malware developed for Modbus protocol. The sixth chapter shows a secure version of the Modbus SCADA protocol that incorporates integrity, authentication, non-repudiation and anti-replay mechanisms. In chapter seven is presented a new intrusion detection approach called ”Critical State Approach” used to develop an Intrusion Detection System for Industrial Control Networks. In the eighth chapter, after describing our testbed, the experimental results are presented and discussed.

Chapter 2

Industrial Control Networks

Before attempting to secure an industrial network, it is important to understand what an industrial network really is. Industrial Control Systems are often referred to in the media as SCADA, for example, which is both inaccurate and misleading. An industrial network is most typically made up of several distinct areas, which are simplified here as a business network or enterprise, business operations, a supervisory network, and process and control networks. SCADA, or Supervisory Control and Data Acquisition, is just one specific piece of an industrial network, separate from the control systems themselves, which should be referred to as Industrial Control Systems (ICS) or Distributed Control Systems (DCS) [5]. Each area has its own physical and logical security considerations, and each has its own policies and concerns. In this work will focus our attention only in that ICS referred as Critical Infrastructure considering all the aspect that are not considered in a general IT environment.

2.1 Critical Infrastructure

European Commission(EC) define Critical infrastructures as those physical and information technology facilities, networks, services and assets which, if disrupted or destroyed, would have a serious impact on the health, safety, security or economic well-being of citizens or the effective functioning of governments in European Union (EU) countries [1]. It's easy to understand how for example the production of energy is much more important in modern society than the production of decorative object. The proper manufacture and distribution of electricity can directly impact our safety by providing heat in winter or by powering irrigation pumps during a drought. The proper manufacture and distribution of chemicals can mean the difference between the availability of flu vaccines and pharmaceuticals and a direct health risk to the population. The European commission presented detailed classifications where it's possible to see how critical infrastructures extend across many sectors: transport and distribution, energy, utilities, health, food supply and communications, as well as key government services:

- *Utilities:* Water, gas, oil, electricity, and communications are critical infrastructures that rely heavily on industrial networks and automated control systems. Because the disruption of any of these systems could impact our society and our safety, they are listed as critical by European Commission; because they use automated and distributed process control systems, they are clear examples of industrial networks. Of the common utilities, electricity is often separated as requiring more extensive security. In the United States, Canada and Europe, it is specifically regulated to standards of reliability and cyber security. Oil and gas refining and distribution are systems that should be treated as both a chemical/hazardous material and as a critical component of our infrastructures.

- *Nuclear Facilities:* Nuclear facilities represent unique safety and security challenges due to their inherent danger in the fuelling and operation, as well as the national security implications of the raw materials used. This makes nuclear facilities a prime target for cyber attack, and it makes the consequences of a successful attack more severe. As such, nuclear energy is heavily regulated in the European Commission by the European Nuclear Safety Regulators Group (ENSREG). The ENSREG is an independent, authoritative expert body created in 2007 following a decision of the European Commission in an attempt to guarantee the safe operation of nuclear facilities and to protect people and the environment. This includes regulating the use of nuclear material including by-product, source, and special nuclear materials, as well as nuclear power. Another organization operating actively on the nuclear field is the Intentional Atomic Energy Agency (IAEA). The IAEA is the world's center of cooperation in the nuclear field. The IAEA was created in 1957 in response to the deep fears and expectations resulting from the discovery of nuclear energy.

- *Chemical:* Manufacture and distribution represent specific challenges to securing an industrial manufacturing network. Unlike the utility networks (electric, nuclear, water, gas), chemical facilities need to secure their intellectual property as much as they do their control systems and manufacturing operations. This is because the product itself has a tangible value, both financially and as a weapon. For example, the formula for a new pharmaceutical could be worth a large sum of money on the black market. The disruption of the production of that pharmaceutical could be used as a social attack against a country or nation, by impacting the ability to produce a specific vaccine or antibody. Likewise, the theft of hazardous chemicals can be used directly as weapons or to fuel illegal chemical weapons research or manufacture. For this reason, chemical facilities need to also focus on securing the storage and transportation of the end product.

Some critical elements in these sectors are not strictly speaking 'infrastructure', but are in fact, networks or supply chains that support the delivery of an essential product or service. For example the supply of food or water to the major urban areas is dependent on some key facilities, but also a complex network of producers, processors, manufacturers, distributors and retailers. As claimed before the purpose of this work is to focus on the protection of Industrial Control network with a particular attention of those considered Critical Infrastructure.

2.2 Industrial Control Network

The most notable trend regarding Industrial Network over the past five years is the move towards networks at all levels. At lower levels networks provide higher reliability, visibility, and diagnosability, and enable capabilities such as distributed control, diagnostics, safety, and device interoperability. At higher levels, networks can leverage internet services to enable automated scheduling, control, and diagnostics; improve data storage and visibility. During this work we had the chance to collaborate with some of the most important companies regarding oil, gas, water and electricity. In order to don't reveal critical information regarding this companies in the following section we present a general schema that take consideration all the information gathered in these years. Figure 2.1

- **Control Network:** it contains all the PLCs. It is directly interfaced with the field network, i.e. the network interconnecting the sensors, and the actuators which physically perform the process tasks on the system. Additionally, it sends to the Process Network the data gathered from field devices.
- **Process Network:** the scope of the process network is to allow the interconnection between the different subsystems of the production process. Moreover, it must guarantee the remote access to such systems from other networks, and the delivery of the process data to a dedicated zone called DMZ. Process Network hosts the the SCADA system and all the operator station where is possible manage the whole power plant: sending control commands, reading and evaluate measurements and parameters. Such a network must satisfy the following goals:
  - Quality of service: the packets sent through the network has to be delivered in time.
  - Network survivability: the network must guarantee a minimum level of resilience against failures.
- **Demilitarized Zone(DMZ):** this area hosts a set of "data exchange" servers called historians, which receive data from the process network and make them available to the operators which work in the Corporate Intranet.

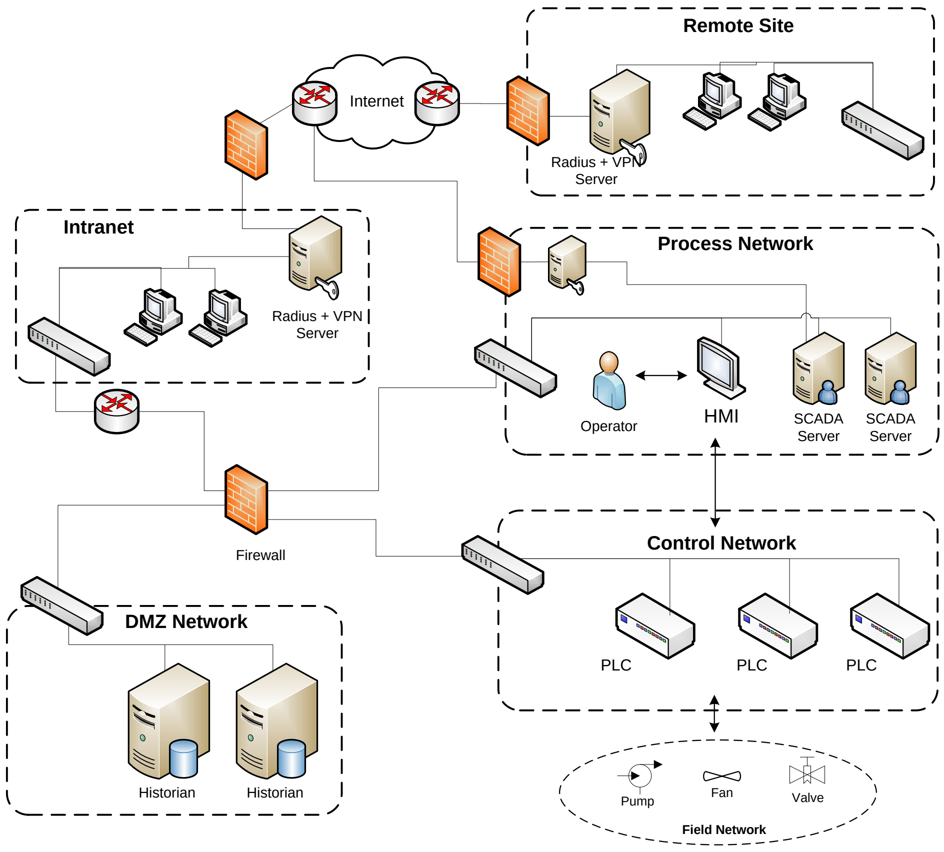


Figure 2.1: A firewalling architecture used to separate SCADA network and external networks.

- **Intranet:** It is the generic Intranet of the Company that operates the Power Plant. It is usually based on a classical Windows domain architecture.
- **Firewall + VPN:** it is used to separate the process network and the control network from the rest of the world. In a similar configuration is important to define exactly the flow directions allowed as shown in Table 2.1.
- **Remote Site:** The operators hosted into a remote site can connect themselves to the local Intranet through a site to site VPN, showing their credentials to a Radius authentication server.

Due to the importance of the process and control network usually the design and configuration of this networks is provided from third parties high specialized in automation i.e. (abb, simens). In order to guarantee the full support in case of malfunctioning is always required a direct access to the process network form this third parties. The VPN-firewall allow to create an host-to site VPN connection between the PC of the operator and the firewall.

Direction	**Allowed**
Intranet → DMZ	YES
Intranet → Process network	NO
Intranet → Control network	NO
DMZ → Intranet	No (allowed only replies to flows originally started from the Intranet)
DMZ → Process network	NO (allowed only replies to flows originally started from the Process Network) in the case in which the SCADA server is separated from the historian and is hosted into the process network YES if historian and SCADA servers are hosted all into the DMZ
DMZ → Control network	NO
Process network → DMZ	YES
Process network → Intranet	NO
Process network → Control network	YES
Control network	it is allowed only the flow with the Process network(in both direction); all the other flows are forbidden.

Table 2.1: Firewall rules table

2.2.1 SCADA - Supervisory

Process network is vendors including ABBconsidered the core of the industrial network and is characterized by a typical architecture called Supervisory Control and Data Acquisition (SCADA). SCADA is a technology that enables a user to collect data form one or more distant facilities and/or send limited control instructions to those facilities. The architecture is mainly composed by the elements in figure 2.2.

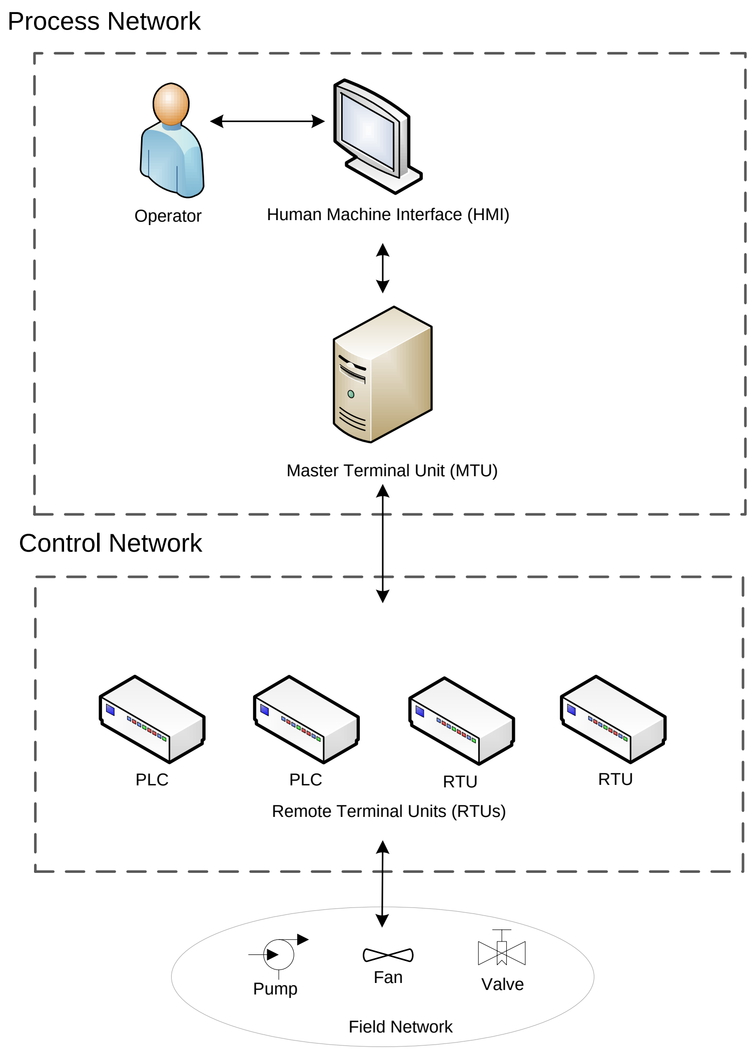

Process Network
Operator
Human Machine Interface (HMI)
Master Terminal Unit (MTU)
Control Network
PLC
PLC
RTU
RTU
Remote Terminal Units (RTUs)
Pump
Fan
Valve
Field Network

Figure 2.2: Industrial Control Network Components

In the following a description of all the elements composing the SCADA system is provided.

PLC - Programmable Logic Controller A programmable logic controller (PLC) [4] is a special form of microprocessor-based controller that uses a programmable memory to store instructions and to implement functions such as logic, sequencing, timing, counting and arithmetic in order to control machines and processes and are designed to be operated by engineers with perhaps a limited knowledge of computers and computing languages. They are not designed so that only computer programmers can set up or change the programs. Thus, the designers of the PLC have pre-programmed it so that the control program can be entered using a simple, rather intuitive, form of language [4]. PLCs are similar to computers but whereas computers are optimized for calculation

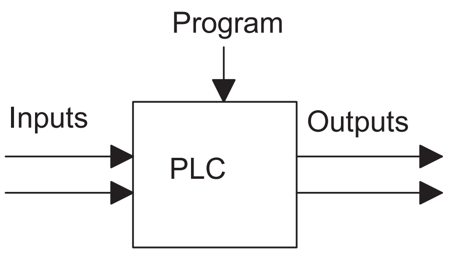


Figure 2.3: A programmable logic controller

and display tasks, PLCs are optimized for control tasks and the industrial environment:

1. Rugged and designed to withstand vibrations, temperature, humidity and noise.

2. Have interfacing for inputs and outputs already inside the controller.

3. Are easily programmed and have an easily understood programming language which is primarily concerned with logic and switching operations.

The first PLC was developed in 1969. They are now widely used and extend from small self-contained units for use with perhaps 20 digital inputs/outputs to modular systems which can be used for large numbers of inputs/outputs, handle digital or analogue inputs/outputs, and also carry out proportional-integral-derivative control modes. Typically a PLC system has composed by: processor unit, memory, power supply unit, input/output interface section, communications interface and the programming device.

- The *processor unit* or *central processing unit (CPU)* is the unit containing the microprocessor and this interprets the input signals and carries out the control actions, according to the program stored in its memory, communicating the decisions as action signals to the outputs.

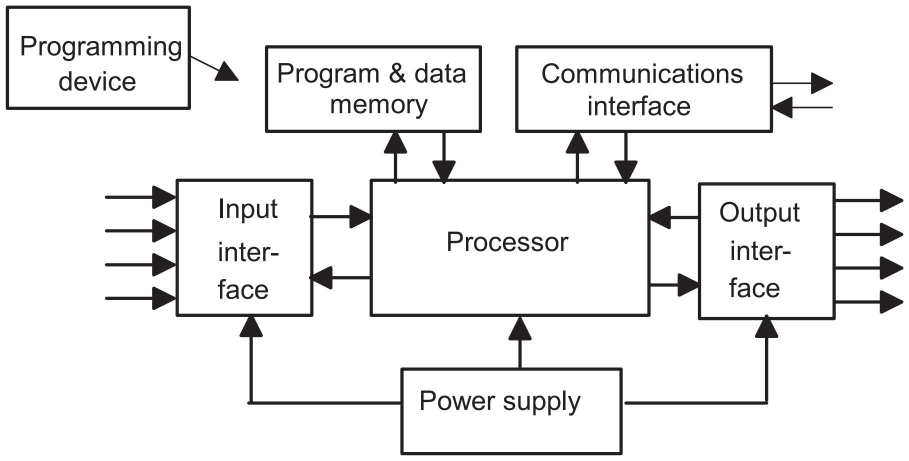


Figure 2.4: The PLC system

- The *power supply unit* is needed to convert the mains a.c. voltage to the low d.c. voltage (5 V) necessary for the processor and the circuits in the input and output interface modules.
- The programming device is used to enter the required program into the memory of the processor. The program is developed in the device and then transferred to the memory unit of the PLC.
- The *memory unit* is where the program is stored that is to be used for the control actions to be exercised by the microprocessor and data stored from the input for processing and for the output for outputting. PLCs
- The *input and output sections* are where the processor receives information from external devices and communicates information to external devices. The inputs might be from switches, temperature sensor or flow sensor. The outputs might be to motor starter coils, solenoid valves, etc. Input and output devices can be classified as giving signals which are discrete, digital or analogue. Devices giving discrete or digital signals are ones where the signals are either off or on; a switch is a device giving a discrete signal, either no voltage or a voltage. Digital devices can be considered to be essentially discrete devices which give a sequence of on-off signals. Analogue devices give signals whose size is proportional to the size of the variable being monitored. For example, a temperature sensor may give a voltage proportional

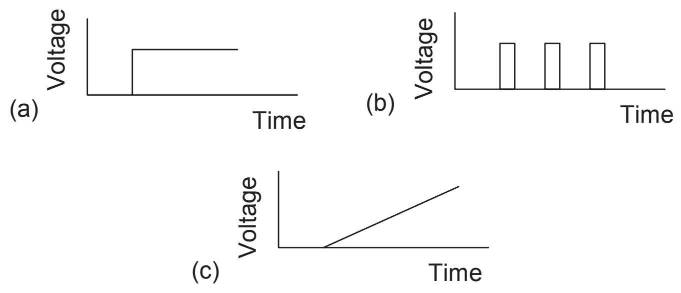


Figure 2.5: Signals: (a) discrete, (b) digital, (c) analogue

to the temperature.

- The communications interface is used to receive and transmit data on communication networks from or to other remote PLCs. It is concerned with such actions as device verification, data acquisition, synchronization between user applications and connection management.

RTU - Remote Terminal Unit A Remote Terminal Unit (RTU) typically resides in a substation or other remote location. RTUs monitor field parameters and transmit that data back to a central monitoring stationtypically either a Master Terminal Unit (MTU), or a centrally located PLC, or directly to an HMI system. RTUs, therefore, include remote communications capabilities, consisting of a modem, cellular data connection, radio, or other wide area communication capability. RTUs and PLCs continue to overlap in capability and functionality, with many RTUs integrating programmable logic and control functions, to the point where an RTU can be thought of as a remote PLC.

Human machine Interface(HMI) Human machine interfaces (HMIs) [5] are used as an operator control panel to PLCs. HMIs replace manually activated switches, dials, and other controls with graphical representations of the control process and digital controls to influence that process. HMIs allow operators to start and stop cycles, adjust set points, and perform other functions required to adjust and interact with a control process. Because the HMI is software based, they replace physical wires and controls with software parameters, allowing them to be adapted and adjusted very easily. HMIs are modern software applications running on modern operating systems, and as such they are capable of performing many functions. They act as a bridge between the human operator and the complex logic of one or more PLCs, allowing the operator to function on the process rather than on the underlying logic that performs the function and to control many functions across distributed and potentially complex processes from a centralized location. To accomplish this, the user interface will graphically represent the process being controlled, including sensor values and other measurements, and visible representation of output states (which motors are on, which pumps are activated, etc.). Humans interact with the HMI through a computer console, and as such must authenticate to the HMI system using password protection. Because HMIs provide supervisory data (visual representation of a control processs current state and values) as well as control (i.e., set point changes), user access may lock out specific functions to specific users. The security of the industrial process therefore relies heavily on access control and host security of the HMI. The HMI, in turn, interacts with one or more PLCs and/or RTUs, typically using industrial protocols.

Data Historians In some installations is possible to find the Data Historian Server in the Process Network. A Data Historian is a specialized software system that collects point values and other information from industrial devices and stores them in a purpose-built database. Most industrial asset vendorsincluding ABB, Arreva, Emerson, GE, Rockwell, Siemens, and othersprovide their own proprietary Data Historian systems. In addition, there are third-party industrial Data Historian vendors, such as Canary Labs (www.canarylabs.com), Modis (www.modius.com), and OSIsoft (www.osisoft.com), which interoperate with third-party assets and even integrate with third-party Data Historians in order to provide a common, centralized platform for data historization and analysis. Data points that are historized and stored within a Data Historian are referred to as tags and can represent almost anythingthe current frequency of a motor or turbine, the rate of airflow through an heating, ventilation, the total volume in a mixing tank, the specific volumes of injected chemical catalysts in a tank, etc. Tags can even represent human-generated values, such as production targets and acceptable loss margins.

As claimed at the beginning of this section because the information stored within a Data Historian is used by both industrial operations and business management, Data Historians are often replicated across an industrial network. This can represent a security risk, as a Data Historian in a less secure zone (i.e., the business network) could be used as a vector into more secure zones (i.e. DMZ).

At the time of this writing, OSIsoft holds a dominant position in the Historian market, with 65% market penetration in global industrial automated systems.[6] The OSIsoft PI System integrates with many IT and OT systems including other Data Historians, and as such is a premium target for attack.

Industrial networks operate differently from enterprise networks and use specialized devices including RTUs and/or PLCs, HMIs, Supervisory Management Workstations, Data Historians, etc. These devices utilize specialized protocols to provide the automation of control loops, which in turn make up larger industrial control processes. In the next chapter we will examine the most important SCADA protocols.

Chapter 3

Industrial Network Protocols

Understanding how industrial networks operate requires a basic understanding of the underlying communications protocols that are used, where they are used, and why. There are many highly specialized protocols used for industrial control system, most of which are designed for efficiency and reliability to support the operational requirements of SCADA systems. Unfortunately, this means that most industrial protocols don't include any feature or function that is not absolutely necessary. In particular they don't include any security features such as authentication and encryption, both of which require additional overhead.

Industrial Network Protocols are often referred as SCADA and/or fieldbus protocols. SCADA protocols are primarily used for the communication of supervisory systems, whereas fieldbus protocols are used for the communication of field devices. However, most of the protocols have the ability to perform both functions, and so will be referred in this thesis more generically as Industrial Network Protocols.

Industrial Network Protocols are real-time communications protocols, developed to interconnect the systems, interfaces, and instruments that make up an industrial control system. Most were designed initially to communicate serially over RS-232, RS-485, or other serial connections but have since evolved to operate over Ethernet networks using routable protocols such as TCP/IP. There are literally dozens of industrial protocols, developed by specific manufacturers for specific purposes. In this thesis we will go more in-depth with one of the most used Industrial protocol called Modbus.

3.1 Modbus protocol

Modbus is the oldest and perhaps the most widely deployed industrial control communications protocol. It was designed in 1979 by Modicon (now part of Schneider Electric) that invented the first Programmable Logic Controller (PLC). Modbus has been widely adopted as a de facto standard and has been enhanced over the years into several distinct variants. Modbus success stems from its relative ease of use: it communicates raw messages without restrictions of authentication or excessive overhead. It is also an open standard, is freely distributed, and is widely supported by members of the Modbus Organization, which still operates today.

Modbus is an application layer messaging protocol, meaning that it operates at layer 7 of the OSI model. It allows for efficient communications between interconnected assets based on a request/reply methodology. It can be used by extremely simple devices such as sensors or motors to communicate with a more complex computer, which can read measurements and perform analysis and control. o support a communication protocol on a simple device requires that the message generation, transmission, and receipt all require very little processing overhead. This same quality also makes Modbus suitable for use by PLCs and RTUs to communicate supervisory data to a SCADA system. Because Modbus is a layer 7 protocol, it can operate independently of underlying network protocols, and it has allowed Modbus to be easily adapted to both serial and routable network architectures. As shown in Figure 3.1, MODBUS is currently implemented using several buses or networks:

- 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.

3.1.1 MODBUS Application Data Unit

The MODBUS protocol specification [20] defines the format of a MODBUS message which is called Application Data Unit (ADU). Each Application Data Unit (ADU) is composed by the following fields:

- **Protocol Data Unit (PDU):** it contains the core information for the MODBUS protocol and it is independent of the underlying communication layers.

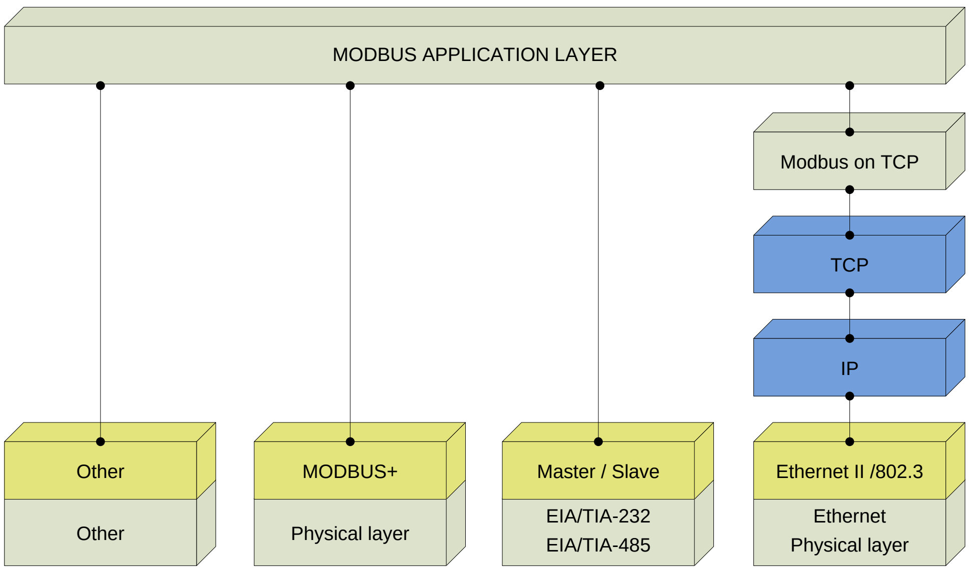


Figure 3.1: MODBUS Communication Stack

- **Additional fields:** they are related to the communication protocol used by the layers below.

The Figure 6.1 shows a general MODBUS Frame regardless of the underlying layers.

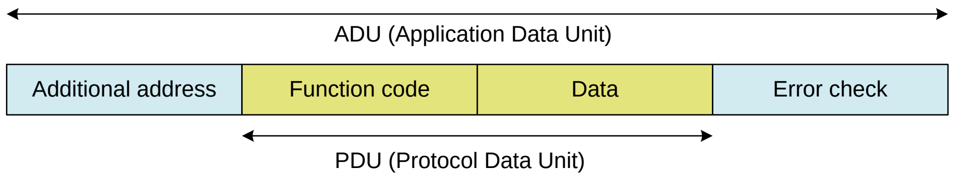


Figure 3.2: General MODBUS Frame

The original MODBUS specification included two possible transmission modes:

- **RTU Mode:** is the most common implementation, using binary coding and CRC error-checking.
- **ASCII Mode:** is less efficient, uses ASCII coding and uses less effective LRC error checking.

The two modes are incompatible, so a device configured for ASCII mode cannot communicate with one using RTU. MODBUS/TCP is a much more recent development, created to allow MODBUS ASCII/RTU protocol to be carried over TCP/IP-based networks. MODBUS/TCP embeds MODBUS messages inside TCP/IP frames.

MODBUS over Serial Line ADU

The MODBUS over Serial Line ADU has the fields in Figure 3.3.

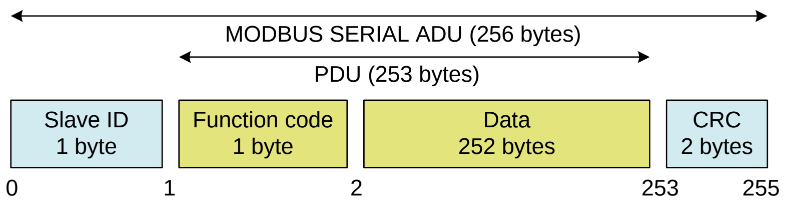


Figure 3.3: Serial Application Data Unit (ADU)

- **Slave ID:** Slave identificator (1 byte $2^8 = 256$; from 0 to 255).
- **Function Code:** defines the type of action required (1 byte $2^8 = 256$; from 0 to 255).
- **Data:** contains additional information that are used to take the action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field.
- **CRC:** the cyclic redundancy check is an error control function.

MODBUS over TCP/IP network ADU

The MODBUS over TCP/IP network ADU has the fields in Figure 3.4.

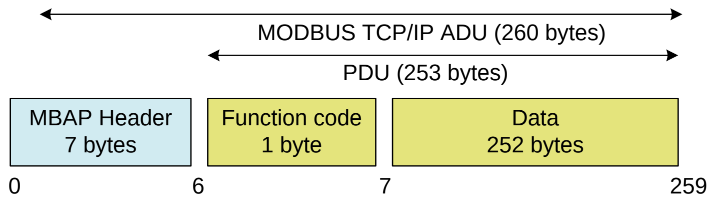


Figure 3.4: TCP/IP Application Data Unit (ADU)

The Protocol Data Unit (PDU) is the same as in MODBUS over Serial Line, but there is an additional header called MODBUS Application Protocol header (MBAP header). It contains

some useful fields for the TCP/IP protocol.
The header is 7 bytes long an it has the following structure:

Transaction Identifier 2 bytes	**Protocol Identifier** 2 bytes	**Length** 2 bytes	**Unit Identifier** 1 byte

Table 3.1: MPAB Header Fields

- **Transaction identifier:** identification of a MODBUS Request/Response transaction. Initialized by the master and recopied by the slave from the received request.
- **Protocol Identifier:** 0 = MODBUS protocol. Initialized by the master.
- **Length:** Number of following bytes.
- **Unit Identifier:** Identification of a remote slave. It is typically used to communicate to a MODBUS serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line.

3.1.2 MODBUS Protocol Data Unit

The Protocol Data Unit (PDU) is the only part that does not change in the ADU because it is independent of the underlying communication layers. The MODBUS protocol specification [**?**] defines three PDUs:

Request Protocol Data Unit

Function Code 1 byte	**Request Data** 252 bytes

Table 3.2: Request Protocol Data Unit

- **Function Code:** defines the type of action required.

- **Data:** contains additional information used to take the action defined by the function code.

Response Protocol Data Unit

Function Code	**Response Data**
1 byte	252 bytes

Table 3.3: Response Protocol Data Unit

- **Function Code:** the function code corresponding to the request.
- **Data:** the response specific data.

Exception Response Protocol Data Unit

Exception Function Code	**Exception Code**
1 byte	1 bytes

Table 3.4: Exception Response Protocol Data Unit

- **Exception Function Code:** the function code corresponding to the request + 0x80 (128), which means the most significant bit set to 1.
- **Data:** a code specifying the exception.

These Protocol Data Units (PDUs) are used in different MODBUS transactions as explained in the next section.

3.1.3 MODBUS Transactions

The MODBUS protocol follows a master/slave (client/server) architecture where a master will request data from the slave. The master can also ask the server to perform some actions. Only the master can initiate transactions or queries. The other devices, slaves, respond by supplying the requested data to the master or by taking the action requested in the query. There are two kinds of transactions:

MODBUS Transactions (error free)

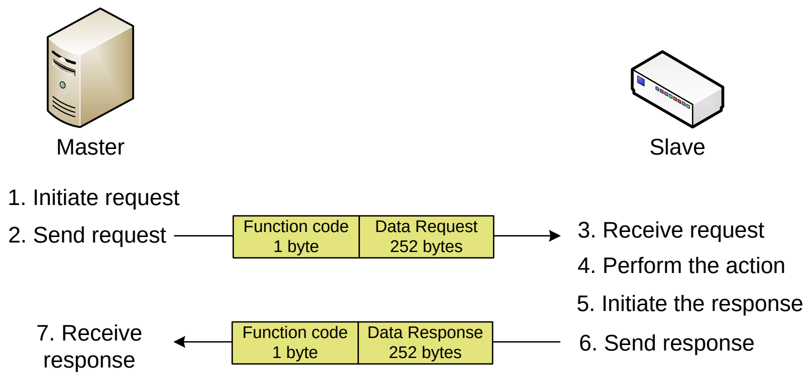


Figure 3.5: MODBUS transaction (error free)

The master node builds the request message, then sends it to the slave. The slave receives the request than search the requested data or performs the requested action. After it builds the response message and sends it to the slave.

MODBUS Transactions (exception response)

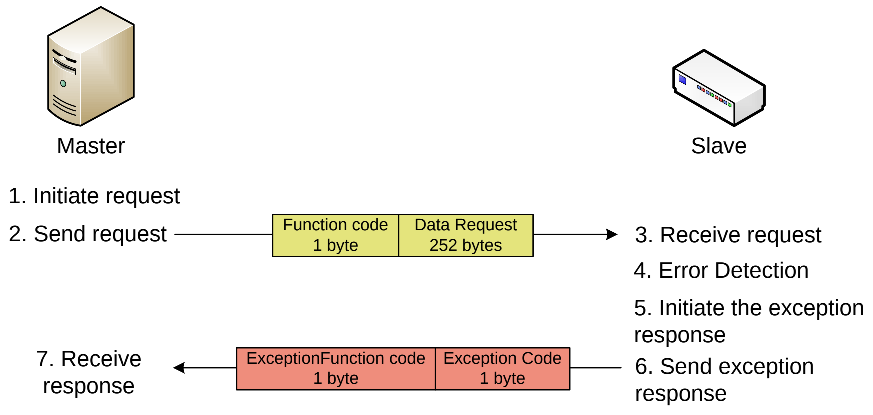


Figure 3.6: MODBUS transaction (exception response)

The master node builds the request message, then sends it to the slave. The slave receives the request, but an exception occurs while processing the function code. The slaves will return a function code that is equivalent to the original function code with the most significant bit set to logic 1 and an exception function code which specifying the exception occurred.

3.1.4 MODBUS Data Model

MODBUS transactions always perform a set of actions by reading or writing to a set of four data types. Table 3.5 describes the four data formats used by the MODBUS application layer.

Primary Table	**Object Type**	**Type of**	**Comments**
Discrete Input	Single bit	Read-Only	can be provided by an I/O system.
Coils	Single bit	Read-Write	can be alterable by an application program.
Input Registers	16-bit word	Read-Only	can be provided by an I/O system.
Holding Registers	16-bit word	Read-Write	can be alterable by an application program.

Table 3.5: MODBUS Data Model

The Table 3.1.4 explains the same data models, but from the point of view of the I/O system:

Inputs	Digital Inputs	Discrete Inputs	1 bit
	Analog Inputs	Input Registers	16 bits
Outputs	Digital Outputs	Colis	1 bit
	Analog Outputs	Holding Registers	16 bits

3.1.5 MODBUS Function Codes

Function code field in the MODBUS ADU is 1 byte field. The MSB(Most Significant Bit) is 0 for Request and Response and 1 for Exception Response so the function codes are $2^7 = 128$ (0-127). There are three categories of MODBUS Functions codes. They are :

- **Public Function Codes(1-65,72-100,110-127):** well defined function codes, validated by the MODBUS-IDA.org community, publicly documented, have available conformance test.
- **User-Defined Function Codes(65-72,100-110):** user can select and implement a function code that is not supported by the specification.
- **Reserved Function Codes:** function Codes currently used by some companies for legacy products and that are not available for public use.

The Table 3.6 shows the public function codes and sub-codes: In the following are presented some important and illustrative function codes.

0x01 Read Coils (01)

Reads the ON/OFF status of discrete outputs (coils) in the slave.

Request			
Function code	1 byte	0x01	01
Starting Address	2 bytes	0x0000-0xFFFF	00-65535
Quantity of coils	2 bytes	0x0001-0x07D0	1-2000

Response			
Function code	1 byte	0x01	01
Byte Count	1 bytes	n	n
Coil Status	n bytes	values	values

n = Quantity of coils / 8, if the reminder is $\neq 0$ then n=n+1

Exception Response		
Function code	0x81	Function Code + 0x80
Exception code	0x01,02,03,04	Exception code

Read Coils Example:

Read Coils 20-36 example.

Request		
Function code	0x01	01
Starting Address	0x0013	19
Quantity of coils	0x0013	17

01	0x01	Read Coils	1 bit
02	0x02	Read Discrete Inputs	1 bit
03	0x03	Read Holding Registers	16 bits
04	0x04	Read Input Registers	16 bits
05	0x05	Write Single Coil	1 bit
06	0x06	Write Single Register	16 bits
07	0x07	Read Exception Status	
08	0x08	Diagnostics	
	00 (0x00)	Return Query Data	
	01 (0x01)	Restart Communications Option	
	02 (0x02)	Return Diagnostic Register	
	03 (0x03)	Change ASCII Input Delimiter	
	04 (0x04)	Force Listen Only Mode	
	10 (0x0A)	Clear Counters and Diagnostic Register	
	11 (0x0B)	Return Bus Message Count	
	12 (0x0C)	Return Bus Communication Error Count	
	13 (0x0D)	Return Bus Exception Error Count	
	14 (0x0E)	Return Slave Message Count	
	15 (0x0F)	Return Slave No Response Count	
	16 (0x10)	Return Slave NAK Count	
	17 (0x11)	Return Slave Busy Count	
	18 (0x12)	Return Bus Character Overrun Count	
	19 (0x14)	Clear Overrun Counter and Flag	
11	0x0B	Get Comm Event Counter	
12	0x0C	Get Comm Event Log	
15	0x0F	Write Multiple Coils	1 bits
16	0x10	Write Multiple registers	16 bits
17	0x11	Report Slave ID	
20	0x14	Read File Record	
21	0x15	Write File Record	
22	0x16	Mask Write Register	16 bits
23	0x17	Read/Write Multiple registers	16 bits
24	0x18	Read FIFO Queue	16 bits
43	0x2B	Encapsulated Interface Transport	
	13 (0x0D)	CANopen General Reference Req and Resp PDU	
	14 (0x0E)	Device Identification	

Table 3.6: MODBUS function codes

Response		
Function code	0x01	01
Byte Count	0x03	3
Coil status(byte 1) 27-20	0xCD	1100 1101
Coil status(byte 2) 35-28	0x6B	0110 1011
Coil status(byte 3) 38-36	0x05	0000 0001

The registers values red by the master are:

27	26	25	24	23	22	21	20
1	1	0	0	1	1	0	1
35	34	33	32	31	30	29	28
0	1	1	0	1	0	1	1
*	*	*	*	*	*	*	36
0	0	0	0	0	0	0	1

The Function Code “**0x02 Read Discrete Inputs (02)**” has exactly the same message format.

0x03 Read Holding Registers (03)

Reads the 16-bit values of holding registers in the slave.

Request			
Function code	1 byte	0x03	03
Starting Address	2 bytes	0x0000-0xFFFF	00-65535
Quantity of Registers	2 bytes	0x0001-0x007D	1-125

Response			
Function code	1 byte	0x03	03
Byte Count	1 bytes	n	n
Register Status	n bytes	values	values

n = Quantity of Registers * 2

Exception Response		
Function code	0x83	Function Code + 0x80
Exception code	0x01,02,03,04	Exception code

Read Holding Registers Example:

Read Holding Registers 108-110 example.

Request		
Function code	0x03	03
Starting Address	0x006B	107
Quantity of registers	0x0003	3

Response		
Function code	0x03	03
Byte Count	0x06	6
Register status(byte 1) 108 HI	0x02	
Register status(byte 2) 108 LO	0x2B	
Register status(byte 3) 109 HI	0x00	
Register status(byte 1) 109 LO	0x00	
Register status(byte 2) 110 HI	0x00	
Register status(byte 3) 110 LO	0x64	

The registers values red by the master are: HR[108] = 0x022B = 555, HR[109] = 0x0000 = 0, HR[108] = 0x0064 = 100.
The Function Code “**0x04 Read Input Registers (04)**” has exactly the same message format.

0x05 Write Single Coil (05)

Write a single coil to either ON or OFF in the slave.

Request			
Function code	1 byte	0x05	05
Output Address	2 bytes	0x0000-0xFFFF	00-65535
Output value	2 bytes	0x0000-0xFF00	0-65280

Response			
Function code	1 byte	0x05	05
Output Address	2 bytes	0x0000-0xFFFF	00-65535
Output value	2 bytes	0x0000-0xFF00	0-65280

Exception Response		
Function code	0x85	Function Code + 0x80
Exception code	0x01,02,03,04	Exception code

Write Single Coil Example:

Write coil 173 ON example.

Request		
Function code	0x05	05
Output Address	0x00AC	172
Output value	0xFF00	ON (0x0000 is OFF)

Response		
Function code	0x05	05
Output Address	0x00AC	172
Output value	0xFF00	ON (0x0000 is OFF)

0x0F Write Multiple Coils (15)

This function code is used to write each coil in a sequence of coils to either ON or OFF. The request message specifies the coil references to be written. The requested ON/OFF states are specified by contents of the request data field. A logical '1' in a bit position of the field requests the corresponding output to be ON. A logical '0' requests it to be OFF. The normal response returns the function code, starting address, and quantity of coils written.

Request			
Function code	1 byte	0x0F	15
Starting Address	2 byte	0x0000 to 0xFFFF	
Quantity of Outputs	2 byte	0x0000 to 0x07B0	0-1968
Byte Count	1 byte	n	
Outputs Value	n byte	values	values

n = Quantity of Outputs / 8, if the reminder is $\neq 0$ then n=n+1

Response			
Function code	1 byte	0x0F	15
Starting Address	2 byte	0x0000 to 0xFFFF	
Quantity of Outputs	2 byte	0x0000 to 0x07B0	

Exception Response		
Function code	0x8F	Function Code + 0x80
Exception code	0x01,02,03,04	Exception code

Write Multiple Coil Example:

Request to write a series of 10 coils starting at coil 20.

Request			
Function code	1 byte	0x0F	0F
Starting Address	2 bytes	0x0013	19
Quantity of Outputs	2 bytes	0x000A	10
Byte Count	1 bytes	0x02	2
Outputs Value (reg. 27-20)	1 bytes	0xCD	1100 1101
Outputs Value (reg. 29-28)	2 bytes	0x01	0000 0001

Response			
Function code	1 byte	0x0F	0F
Starting Address	2 bytes	0x0013	19
Quantity of Outputs	2 bytes	0x000A	10

The binary bits correspond to the outputs in the following way:

27	26	25	24	23	22	21	20
1	1	0	0	1	1	0	1
*	*	*	*	*	30	29	28
0	0	0	0	0	0	0	1

0x06 Write Single Register (06)

Write a single 16-bit holding register in the slave.

Request			
Function code	1 byte	0x06	06
Register Address	2 bytes	0x0000-0xFFFF	00-65535
Register value	2 bytes	0x0000-0xFFFF	0-65535

Response			
Function code	1 byte	0x05	05
Register Address	2 bytes	0x0000-0xFFFF	00-65535
Register value	2 bytes	0x0000-0xFFFF	0-65535

Exception Response		
Function code	0x86	Function Code + 0x80
Exception code	0x01,02,03,04	Exception code

Write Single Register Example:

Write the value 0x0003 in the holding register 2.

Request		
Function code	0x06	06
Register Address	0x0001	1
Register value	0x0003	3

Response		
Function code	0x06	06
Register Address	0x0001	1
Register value	0x0003	3

0x10 Write Multiple registers (16)

This function code is used to write a block of contiguous registers (1 to 123 registers) in a remote device. The requested written values are specified in the request data field. Data is packed as two bytes per register. The normal response returns the function code, starting address, and quantity of registers written.

Request			
Function code	1 byte	0x10	16
Starting Address	2 byte	0x0000 to 0xFFFF	
Quantity of Registers	2 byte	0x0000 to 0x07B0	0-123
Byte Count	1 byte	n	n
Registers Value	n byte	values	values

n = Quantity of Registers * 2

Response			
Function code	1 byte	0x10	16
Starting Address	2 byte	0x0000 to 0xFFFF	
Quantity of Registers	2 byte	0x0000 to 0x07B0	

Exception Response		
Function code	0x90	Function Code + 0x80
Exception code	0x01,02,03,04	Exception code

Write Multiple registers Example

Here is an example of a request to write two registers starting at 2 to 00 0A and 01 02 hex.

Request			
Function code	1 byte	0x10	16
Starting Address	2 bytes	0x0001	1
Quantity of Registers	2 bytes	0x0002	2
Byte Count	1 bytes	0x04	4
Outputs Value (reg. 01)	2 bytes	0x000A	
Outputs Value (reg. 02)	2 bytes	0x0102	

Response			
Function code	1 byte	0x10	16
Starting Address	2 bytes	0x0001	1
Quantity of Registers	2 bytes	0x0002	2

0x07 Read Exception Status (07)

This function code is used to read the contents of eight Exception Status Coils in a remote device. Coils can be programmed by the user to hold information about the controller's status, for example, "machine ON/OFF", "heads retracted", "safeties satisfied", "error conditions exists", or other user defined flags. The normal response contains the status of the eight Exception Status outputs. The outputs are packed into one data byte, with one bit per output.

Request			
Function code	1 byte	0x07	07

Response			
Function code	1 byte	0x07	07
Output Data	1 bytes	0x00-0xFF	00-255

Exception Response		
Function code	0x87	Function Code + 0x80
Exception code	0x01,04	Exception code

Example:

In the Modicon Controller 484 the 8 status coils are:

257	258-264
Battery Status	User defined

The request/response to read the exception status coils is:

Request		
Function code	0x07	07

Response		
Function code	0x07	07
Output Data	0x6D	0110 1101

The coils values red by the master are:

264	263	262	261	260	259	258	257
0	1	1	0	1	1	0	1

In this example, coil 257 is ON, indicating that the controller's battery is OK.

0x08 Diagnostics (08)

MODBUS function code 08 provides a series of tests for checking the communication system between a Master device and a Slave, or for checking various internal error conditions within a slave. The function uses a two bytes sub-function code field in the query to define the type of test to be performed.

Request			
Function code	1 byte	0x08	08
Sub-Function	2 bytes	0x0000-0xFFFF	
Data	n bytes	values	values

Response			
Function code	1 byte	0x08	08
Sub-Function	2 bytes	0x0000-0xFFFF	
Data	n bytes	values	values

Response		
Function code	0x88	Function Code + 0x80
Exception code	0x01,03,04	Exception code

Sub-function codes

In the following there are some of the most important sub-function codes.

0x00 Return Query Data (00)

The data passed in the request data field is to be returned (looped back) in the response. The entire response message should be identical to the request.

Response			
Sub-Function	2 bytes	0x0000	0
Data	n bytes	Any	Any

Response			
Sub-Function	2 bytes	0x0000	0
Data	N x 2 bytes	Echo Request Data	Echo Request Data

0x04 Force Listen Only Mode (04) and 0x01 Restart Communications Option (01)

The function code "*Force Listen Only Mode*" forces the addressed remote device to enter into the "*Listen Only Mode*". When the remote device enters in Listen Only Mode, all active communication controls are turned off. While the device is in this mode, any MODBUS messages addressed to it or broadcast are monitored, but no actions will be taken and no responses will be sent. The only function that will be processed after the mode is entered will be the Restart Communications Option function (function code 8, sub-function 1). The Figure 3.7 shows the two MODBUS operational mode and how to pass from the normal "on-line mode" to the "Listen Only Mode".

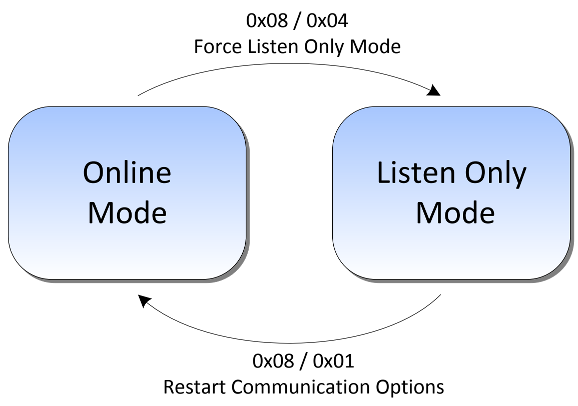


Figure 3.7: Listen Only Mode

The remote device could be re-activated by the command "Restart Communications Option". When such device receives this command it performs a re-initialization: the remote device port must be initialized and restarted, and all of its communications event counters are cleared. The successful completion of the restart will bring the port on-line. This function is the only one that brings the port out of Listen Only Mode. If the port is currently in Listen Only Mode, no response is returned, otherwise if the port is not currently in Listen Only Mode, a normal response is returned. The request message with the function code "Force Listen Only Mode" has the following structure and response is not required.

Request			
Sub-Function	2 bytes	0x0004	04
Data	1 byte	0x0000	

The request/response message transaction for the function code "Restart Communications Option" is shown below. A request data field contents of FF 00 hex causes the port's Communications Event Log to be cleared. Contents of 00 00 leave the log as it was prior to the restart. The response is only a request echo.

Request			
Sub-Function	2 bytes	0x0001	01
Data	N x 2 bytes	0x0000 or 0xFF00	

Response			
Sub-Function	2 bytes	0x0001	01
Data	N x 2 bytes	Echo Request Data (0x0000 or 0xFF00)	

In the MODBUS slave device there is a flag to identify the current mode of the device. For example in the Modicon 184/384 the 4^{th}bit of the Diagnostic Register is called “Force Listen Only Mode” and if it is ON the device is in Listen Only Mode and if it is OFF the device is in On line Mode.

0x02 Return Diagnostic Register (02)

The diagnostic register is used in a MODBUS device to store some diagnostic information about the device itself. The contents of the remote device’s 16-bit diagnostic register are returned in the response.

Request			
Sub-Function	2 bytes	0x0002	02
Data	2 byte	0x0000	

Response			
Sub-Function	2 bytes	0x0002	02
Data	2 byte	Diagnostic Register Content	

Return Diagnostic Register Example

The diagnostic register is a normal 16-bit register, but each bit has a special diagnostic meaning. For example in the Modicon 184/384 the Diagnostic Register bits have the following meanings:

0	Continue on Error
1	Run Light Failed
2	T-Bus Test Failed
3	Asynchronous Bus Test Failed
4	Force Listen Only Mode
5-6	Not Used
7	ROM Chip 0 Test Failed
8	Continuous ROM Checksum Test in Execution
9	ROM Chip 1 Test Failed
10	ROM Chip 2 Test Failed
11	ROM Chip 3 Test Failed
12	RAM Chip 5000-53FF Test Failed
13	RAM Chip 6000-67FF Test Failed, Even Addresses
14	RAM Chip 6000-67FF Test Failed, Odd Addresses
15	Timer Chip Test Failed

Counter Function Codes 0x0B(11) - 0x12(18)

The sub-function codes from 0x0B(11) to 0x12(18) are used to return the counter values. Each counter counts a specific kind of messages. In the MODBUS specification [**?**] are defied eight counters. The Table 3.7 shows a list of MODBUS counters.

N.	Name	Sub-fc	Meaning
1	Bus Message Counter	0x0B(11)	Quantity of messages that the slave has detected
2	Bus Communication Error Counter	0x0C(12)	Quantity of CRC errors that the slave has detected
3	Bus Exception Error Counter	0x0D(13)	Quantity of MODBUS exception responses that the salve has returned
4	Slave Message Counter	0x0E(14)	Quantity of messages that the slave has processed
5	Slave No Response Counter	0x0F(15)	Quantity of messages addressed to the slave for which it returned no response.
6	Slave NAK Counter	0x10(16)	Quantity of messages addressed to the slave for which it returned a Negative Acknowledge (NAK)
7	Slave Busy Counter	0x11(17)	Quantity of messages addressed to the slave for which it returned a Slave Device Busy exception response
8	Overrun Counter	0x12(18)	Quantity of messages addressed to the slave that it could not handle due to a character overrun condition

Table 3.7: MODBUS Counters

Each Sub-function code form 0x0B(11) to 0x12(18) has the same name: “Return Register Name” and returns to the master the counter 16-bit value.
The request/response transaction is very simple: the response data field returns the quantity of messages that the remote device has detected on the communication system since its last restart, clear counters operation, or power up.

Request			
Sub-Function	2 bytes	0x000B	12
Data	1 byte	0x0000	

Response			
Sub-Function	2 bytes	0x000B	12
Data	1 byte	Total Message Count	

Example Sub-function code

Here is an example of a request to remote device to Return Query Data. This uses a subfunction code of zero (00 00 hex in the two-byte field). The data to be returned is sent in the two-byte data field (A5 37 hex).

Request			
Function code	1 byte	0x08	08
Sub-function code	2 byte	0x0000	00
Data	2 bytes	0xA537	

Response			
Function code	1 byte	0x08	08
Sub-function code	2 byte	0x0000	00
Data	2 bytes	0xA537	

3.1.6 MODBUS Exception Code

When a master device sends a request to a slave device it expects a normal response. One of four possible events can occur from the master’s query:

- If the slave device receives the request without a communication error, and can handle the query normally, it returns a normal response.

- If the slave does not receive the request due to a communication error, no response is returned. The master program will eventually process a timeout condition for the request.
- If the slave receives the request, but detects a communication error (parity, LRC, CRC, ...), no response is returned. The master program will eventually process a timeout condition for the request.
- If the slave receives the request without a communication error, but cannot handle it (for example, if the request is to read a non-existent output or register), the slave will return an exception response informing the master of the nature of the error.

The exception response message has two fields that differentiate it from a normal response:

Function Code Field: In a normal response, the slave echoes the function code of the original request in the function code field of the response. All function codes have a most significant bit (MSB) of 0 (their values are all below 80 hexadecimal). In an exception response, the slave sets the MSB of the function code to 1. This makes the function code value in an exception response exactly 80 hexadecimal higher than the value would be for a normal response. With the function code MSB set, the master's application program can recognize the exception response and can examine the data field for the exception code.

Data Field: In a normal response, the slave may return data or statistics in the data field. In an exception response, the slave returns an exception code in the data field. This defines the slave condition that caused the exception.

Exception Code Example:

Response			
Function Code	1 byte	0x01(Read Coils)	01
Starting Address	2 byte	04A1	1185
Quantity of colis	1 byte	0x01	

Exception Response		
Function code	0x81	Function Code + 0x80
Exception code	0x02	Illegal Data Address

The master device sends a function code (01) "Read Coils". It requests the status of the output at address 1185 (04A1 hex). If the output address is non-existent in the server device, the server will return the exception response with the exception code shown (02). This specifies an illegal data address for the slave. The Table 3.8 shows the MODBUS exception codes list.

01	Illegal Function	The function code received in the query is not an allowable action for the slave. This may be because the function code is not implemented in the unit selected
02	Illegal Data Address	The data address received in the query is not an allowable address for the slave. More specifically, the combination of reference number and transfer length is invalid.
03	Illegal Data Value	A value contained in the query data field is not an allowable value for the slave. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect.
04	Slave Device Failure	An unrecoverable error occurred while the slave was attempting to perform the requested action.
05	Acknowledge	Specialized use in conjunction with programming commands. The 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	Slave Device Busy	Specialized use in conjunction with programming commands. The slave is engaged in processing a long-duration program command. The master should retransmit the message later when the slave is free.
08	Memory Parity Error	Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The slave attempted to read record file, but detected a parity error in the memory. The master can retry the request, but service may be required on the 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 Target Device 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.

Table 3.8: MODBUS Exception Codes

Industrial networks use a variety of specialized fieldbus protocols to accomplish specific tasks, often with careful attention to synchronization and real-time operation. Each protocol has varying degrees of inherent security and reliability, and these qualities should be considered when attempting to secure these protocols. However, because industrial network protocols, in general, lack sufficient authentication or encryption, all are susceptible to cyber attack using relatively simple MITM attacks, which can be used to disrupt normal protocol operations or potentially to alter or otherwise manipulate protocol messages to steal information, commit fraud, or potentially cause a failure of the control process itself.

Chapter 4

Related works

Critical infrastructures in the last ten years evolves from a close environment to a massive interconnected environment but speaking from an attacker point of view they constitute a completely separated world with its own vulnerabilities and attack patterns, different from the traditional ICT world. Regarding the security of Critical Infrastructure Adam and Byres [7] presented an interesting high level analysis of the possible threats of a power plant system, a categorization of the typical hardware devices involved and some high level discussion about intrinsic vulnerabilities of the common power plant architectures. A more detailed work on the topic of SCADA security, is presented by Chandia, Gonzalez, Kilpatrick, Papa and Shenoi [8]. In this work, the authors describe two possible strategies for securing SCADA networks, underlying that several aspects have to be improved in order to secure that kind of architectures. What is evident in the mentioned article is that the most evident lack, in SCADA networks, is the communication protocols used in such systems, (e.g. MODBUS, DNP3 etc.), because they not consider ICT security aspects. Historically, this is due to the fact that when they were designed, the world of industrial control systems was completely isolated from the public networks, and then ICT based intrusion scenarios were considered completely negligible.The recent history showed how is possible to perform complex attack against critical installation by taking advantage of all together security lack in SCADA. The attack was performed not only to steal information but also to cause damage. Falliere, Murchu and Chien [9] presented a detailed analysis of the Stuxnet worm. Stuxnet had as its main target industrial control systems with the goal of modifying the code running in Programmable Logic Controllers (PLCs) in order to make them deviate from their expected behavior. His deviation would be small and only noticeable over a longer period of time. In parallel great effort was put by the Stuxnet creators in hiding those changes from the operators, even imitating legitimate data. Nicholson, Webber, Patel and Janicke [10] take in consideration fundamental aspects of SCADA system security, citing specific examples of vulnerabilities that were discovered and patched in real world systems. The authors make an interesting analysis about both researchers and vendors must propose more sophisticated solution in order to protect critical installations. Cagalaban and Seoksoo [11] to address the limitations in SCADA security, presented a vulnerability assessment methodologies to test the existing SCADA security design and implementation. They provides a relevant analysis of most important issues and a perspective on enhancing security of these systems. Further, they described key requirements and features needed to improve the security of the current SCADA networks. Unfortunately in this work they take in consideration only a really small subset of the possible attack scenarios. SCADA traffic injection for example is one of the most dangerous attacks concerning ICS and must be considered in order to have a complete analysis.

There are currently many types of approaches to improve the security of industrial networks. In particular all of then can be grouped in two categories:

1. **Industrial Network communications protection:** this solutions regards the protection of the direct communication between actuators and SCADA devices. In particular can be reached by the introduction of tunnels around SCADA protocols or by other stander security mechanism.

2. **Network monitoring:** these solutions regard everything concerned the monitoring without modify the configuration and the actual devices deployed inside industrial networks.

Industrial Network communications protection Some works have been done about the security of industrial specialized communication protocols: for example, Majdalawieh, Parisi-Presicce and Wije-sekera [12] presented an extension of the DNP3 protocol, called DNPsec, which tries to address some of the known security problems of such Master-slave control protocols (i.e. integrity of the commands, authentication, non repudiation etc.). Bagaria, Prabhakar, Saquib [13] propose a mechanism to secure existing and future DNP3 networks. Unfortunately all the approach cited above presented a solution that require to be implemented in embedded systems with limited computing resources. Hong, Phuong and Lee [15] have made some interesting test about how securing methods can affect the performance of embedded systems. In this work they propose a different approach for securing SCADA modbus communication taking in consideration alle the physical constraints regarding industrial networks such as low computational power and low latency. Lee and Kim [14] proposed an encryption and decryption method of SCADA traffic. They choose to encrypt the whole data itself and do not consider the aspect of protocol modification. This approach unfortunately doesn't prevent one of the most common issue: replay attack. exchanges keys and uses those keys to encrypt the data flowing on the network. They used DNPSec as framework. Shahzad and Musa proposed an Hybrid-based Cryptography: combination of Symmetric AES and Asymmetric RSA in order to enable confidentiality and authentication. The hybrid cryptosystem is constructed using two separate cryptosystems: (a) a key encapsulation scheme, which is a public-key cryptosystem, and (b) a data encapsulation scheme, which is a symmetric-key cryptosystem. For very long messages encryption/decryption is done by the more efficient symmetric-key scheme, while the heavy processing public-key scheme is used only to encrypt/decrypt a short key value. Considering a standard communication between the SCADA server and the RTU and assuming

1. SCADA server fetch the RTU public key from his table and then generates a fresh session/symmetric key for the data encapsulation scheme.

2. SCADA server encrypts the message under the data encapsulation scheme, using the session/symmetric key just generated.

3. SCADA server again encrypt the session/symmetric under the key encapsulation scheme, using RTU public key.

4. SCADA server send encrypted message and session/symmetric key to RTU.

5. RTU fetches the SCADA server record from RTU-key table and uses it private key to decrypt the session/symmetric contained in the key encapsulation segment. If session/symmetric key decrypted successful than RTU uses this key to decrypt the message contained in the data encapsulation segment.

This method reaches some interesting results regarding encryption/decryption operation and total delay but introduce a single point of failure. All keys, such as symmetric and public, are stored locally in database using MySQL and all the devices in the industrial networks must be reconfigured in order to reach the database.

Network monitoring The increasing usage of standard information and communication technologies (ICT) in SCADA systems enables to use common security. However, as proved in [18], traditional firewalls and Intrusion detection systems fail in detecting attack pattern specifically designed for SCADA protocols. The primary limitation of the traditional intrusion detection and cyber security monitoring solutions is that they have no knowledge or intelligence of SCADA applications and protocols. While attacks against a Microsoft IIS web server using the http protocol are addressed well, an attack on a SCADA control server using the Modbus protocol is not even considered. So the current systems are excellent against script kiddie hackers and other novices that use exploits easily available on the Internet; they are not adequate to detect attacks by a cyber terrorist, disgruntled insider, or skilled hacker with knowledge of SCADA systems. Mander, Nabhani, Wang and Cheung [17] presented a proxy filtering solution aiming at identifying and avoiding anomalous control traffic. The proposed solution is extremely interesting; however, it does not protect the system against two particular scenarios:

1. The scenario in which an attacker is able to inject malicious packets directly in the network segment between the proxy and the remote terminal unit

2. The scenario in which both the proxy and the master have been corrupted and collaborate in order to damage the process network.

Only recently a set of ad-hoc rules and preprocessing modules in particular for intrusion detection systems [19] have been released with the capacity of detecting some attacks to SCADA protocols. With these rules a Network Intrusion Detection System (NIDS) would be able to identify single packet-based attacks. Unfortunately SCADA attacks are rarely based on the exploitation of a single [26] consequently, an attack correlation mechanism would be needed. Gross [27] proposed a mechanism for collaborative intrusion detection (selecticast) that uses a centralized server in order to dispatch among the ID sensors information about activities deriving from suspicious IP addresses. This approach is useful for providing a broader picture regarding suspicious events happening in the monitored system. However, it does not provide any kind of specific technique for identifying high level and complex malicious actions. Ning et al. [28] proposed a model aiming at identifying causal relationships between alerts on the basis of prerequisites and consequences. The approach proposed by Cuppens and Miege in [29] adopts pre and postconditions; unfortunately this technique can generate spurious correlation rules, increasing the noise in the IDS alerting system. The present work present a complete theoretical and applicative framework, introducing the concept of critical state ranking prediction. The idea of detecting attacks by analyzing whether the system is entering into a critical state has some similarities with what has been done in the Fault-Detection and Diagnosis research field. More in detail, the most similar approach is the model-based fault detection, which makes use of the so called limit-value-based supervisory functions, for monitoring measurable variables in search for invalid values or, in case of automatic protection functions, for triggering counteractions to protect the process. [30] [31] [32] These approaches cannot be adopted to discriminate between cyber-attacks and accidental faults, and do not provide an easily computable critical state proximity metric, in this PhD thesis a large section is dedicated to the calculation of critical state proximity, in particular we propose different methodologies considering computational performance. Situational awareness is another research field that brings some resemblance with the work presented in this thesis. Situational awareness is a key component in critical environments and a large number of academic works have investigated it. [33] Generally, the main drawback of situation awareness approaches lies in the huge quantity of data to be processed and in the lack of effective techniques for the subsequent evaluation of events that could be sources of threat. Significant work in this area has been surveyed in [34]. Tailoring an intrusion detection system to the specifics of critical infrastructures as claimed before can significantly improve the security of such systems. Considering that SCADA refers to a physical industrial process that follows well known comportment is another key point for increase detection accuracy. Linda et al presented the first Intrusion detection System for SCADA based on neural networks [35]. Thorough a learning phase and two different neural networks algorithms they were able to learn the behavior of the the industrial network and detect anomaly traffic. This approach is deeply applicable when the SCADA network is small and heterogeneous. Unfortunately the real world cases show how in this field it's easy to find different technologies and protocols. Another issue regards the only use of network traffic without any use of SCADA protocols parser to better understand the behavior of the system. In this work we introduce the concept of learning phase based of the knowledge the SCADA protocols.

Chapter 5

Threat Model

Control systems have many characteristics that are different from traditional IT systems in terms of risks and operational priorities thus render unique performance and reliability requirements besides the use of operating systems and applications being unconventional to typical IT personnel. [36] Even where security is well defined, the primary goal in the Internet is to protect the central server and not the edge client. In process control, an edge device, such as PLC or smart drive controller, is not necessarily merited less importance than a central host such as data historian server [37], as they are on the first frontier facing human lives and ecological environment. These differences between SCADA systems and IT systems demand an adjusted set of security property goals and thus security and operational strategies. In the traditional IT community, the set of common desirable security properties are confidentiality, integrity and availability, or CIA in short. The most important in ITs world is confidentiality and integrity while in control systems is system availability and data integrity as result of human and plant safety being its primary responsibility. In this thesis we prioritize security properties of SCADA systems in the order of its importance and desirability in industry, especially in utilities sector.

5.1 SCADA security properties

To evaluate the design of a secure information system, NIST compiled a set of engineering principles for system security. These principles provide a foundation upon which a more consistent and structured approach to the design, development, and implementation of IT security capabilities can be constructed. However, in light to the difference from control systems and standard IT systems, its necessary to analyze in a different way the securities proprieties.: [36]

1. Timeliness: explicitly expresses the time-criticality of control systems, a given resulted from being real-time system, and the concurrencies in SCADA systems due to being widely dispersed distributed systems. It includes both the *responsiveness* aspect of the system, e.g. a command from controller to a PLC should be executed in real time by the latter, and the timeliness of any related data being delivered in its designated time period, by which, we also mean the *freshness* of data, i.e., the data is only valid in its designated time period. In a more general sense, this property describes that any queried, reported, issued and disseminated information shall not be stale but corresponding to the real-time and the system is able and sensitive enough to process request, which may be of normal or of legitimate human intervention in a timely fashion, such as within a sampling period. In reality, if arrives late or

repeatedly to the specified node, a message is no longer any good, be it a correct command to an actuator or a perfect measurement from a sensor with intact content. As a matter of fact, any replay of data easily breaches this security goal. Moreover, this property also implicitly implies the order of updates among peered sensors, especially if they are observing the same process or correlated processes. The order of data arrival at central monitor room may play an important factor in the representation of process dynamics and affect the correct decision making of either the controlling algorithms or the supervising human operators.

2. Availability : means when any component of a SCADA system, may it be a sensory or servo-mechanical device, communication or networking equipment, or radio channel; computation resource and information such as sensor readings and controller commands etc. that transmits or resides within the system should be ready for use when is needed. Most of SCADA controlled processes are continuous in nature. Unexpected outages of systems that control industrial processes are not acceptable. This desired property for both SCADA systems control performance and security goal requires that the security mechanism employed in SCADA systems shall not degrade the maintainability, operability , and its accessibility at emergency, of the original SCADA system without those security oriented add-ons.

3. Integrity: requires data generated, transmitted, displayed, stored within a SCADA system being genuine and intact without unauthorized intervention, including both its content, which may also include the header for its source, destination and time information besides the payload itself. We can image how disastrous the consequence can be, if a control command is redirected to an actuator other than its intended receiver or fake or wrong source information of a sensor measurement being reported to the central controller. The protocol must prevent an adversary from constructing unauthentic messages, modifying messages that are in transit, reordering messages, replaying old messages, or destroying messages without detection.

4. Confidentiality: refers to that unauthorized person should not have any access to information related to the specific SCADA system. *Confidentiality* in SCADA system assume a lower priority compared to *Integrity*. SCADA systems measure and control physical processes that generally are of a continuous nature with commands and responses are simple and repetitive. Thus the messages in SCADA systems are relatively easy to predict. Hence confidentiality is secondary in importance to data integrity. However, the confidentiality of critical information such as passwords, encryption keys, detailed system layout map and etc. shall rank high

when it comes to security concerns in industry. Applicable reinforcement should be imposed in this aspect. Also, the information regarding physical content flowed within the control algorithm may be subject to leaking critical message to side channel attacks. The drastic difference in the ordering of desired security properties is mostly due to that SCADA systems are demanded to be real-time operating and continuously functioning.

5. Graceful Degradation: requires the system being capable of keeping the attack impact local without a further escalating into a full scale or a full system cascading event. Again, all these desired security properties are not mutual exclusive but closely related. For example, by breaching integrity, an attacker can change control signals to cause a device malfunction which might ultimately affect the availability of the network.

5.2 Attacks

Starting from the securities properties defined above we focus the attention on the cyber attacks against SCADA system, assuming that the basic physical security is provided. In particular, the SCADA server or Master Terminal Unit must be physically secure, i.e., we assume there are no direct physical tampering on the server where the main control and estimation algorithms reside. Brute force physical sabotage such as cutting wires and cables from communication and power supply are not considered in this thesis. Furthermore, we assume that the PLCs are programmed securely. Cyber attacks on SCADA system can take routes through Internet connections, intranet or process network connections and or connections to other networks, to the layer of control networks then down the level of field devices. More specifically, the common attack vectors are:

- Backdoors and holes in network perimeter
- Vulnerabilities in common protocols[38]
- Attacks on field devices
- Communications hijacking and Man-in-the-middle attacks
- Synchronization Attack

Starting from the vector infections described above we have analyzed attacks that can fall into the following categories[36].

5.2.1 Attacks in Hardware

Attacker might gain unauthenticated remote access to devices and change their data set points. This can cause devices to fail at a very low threshold value or an alarm not to go off when it should. Another possibility is that the attacker, after gaining unauthenticated access, could change the operator display values so that when an alarm goes off, the human operator is not noticed of it. This could affect the safety of people in the vicinity of the plant. The main issue in preventing cyber attacks on hardware is access control.

5.2.2 Attacks on Software

SCADA system employs a variety of software to meet its functionality demands. Also there are large databases reside in data historians besides many relational database applications used in cooperate and plant sessions. Data historians contain vital and potentially confidential process information2.2.1. These data are not only indispensable for technical reasons, but also for business purposes, such as electricity or gas pricing. The most common implementation flaw in this kind of software is buffer overflow such as format string, integer overflow and etc. The fact that most control applications are written in C requires to take extra precaution with this vulnerability. In the following we will see in details some attacks that direct impact this kind of software.

No Privilege Separation in Embedded Operating System

Most of the embedded Operating System run as a monolithic kernel with applications implemented as kernel tasks, This means that all tasks generally run with the highest privileges and there is little memory protection between these tasks. A example is VxWorks. VxWorks was the most popular embedded operating system in 2005 and claimed 300 million devices in 2006 [39], which is a platform developed by Wind River Systems and has since been acquired by Intel.

Buffer Overflow

Many attacks that use buffer overflow generate anomalies functionality inside the program. Some general methods manipulating function pointer. The effect of such attacks can take forms such as resetting passwords, modifying content, running malicious code and so on. The buffer overflow problem in SCADA system takes two fronts. One front is on the workstations and servers which are similar to standard IT systems. For example, WellinTech KingView 6.53 HistorySvr, an industrial automation software for historian, has a heap buffer overflow vulnerability that could potentially become the risk of a Stuxnet type attack if not matched [40] . We have also to consider that many field devices in SCADA systems run for years without rebooting and patching. Therefore, these SCADA components, especially in legacy networks, are subject to accumulated memory fragmentation, which leads to program stall.

SQL Injection

Most industrial database applications can be accessed using Structured Query Language (SQL) statements for structural modification and content manipulation. In light of data historians and web accessibility in current SCADA systems, SQL injection, one of the top Web attacks, has a very strong implication on the security of SCADA system. SQL injection is a code injection technique that exploits a security vulnerability in a website's software. The vulnerability happens when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL commands are thus injected from the web form into the database of an application (like queries) to change the database content or dump the database information. In the case studied in this paper[41], where the store procedure in SQL server (see figure 5.1) is enabled by default an attacker still can get into SCADA system even though two LAN cards are installed. Intentionally malicious changes to databases can cause catastrophic damage.

5.2.3 Attacks in the communication stacks

In this section we analyze the communication layer. In particular on network layer, transport layer, application layer and the implementation of protocols. As we will see the most simple and in the same time high impact attack regarding the application layer due to the implementation protocol. In this thesis we present a malware tailored for SCADA system able to take advantage of SCADA protocols problems.

Network Layer

1. Idle Scan: blind port scan by using a zombie host, often used as a preparation for attack. Both MODBUS and DNP3 have scan functionalities prone to such attacks when they are

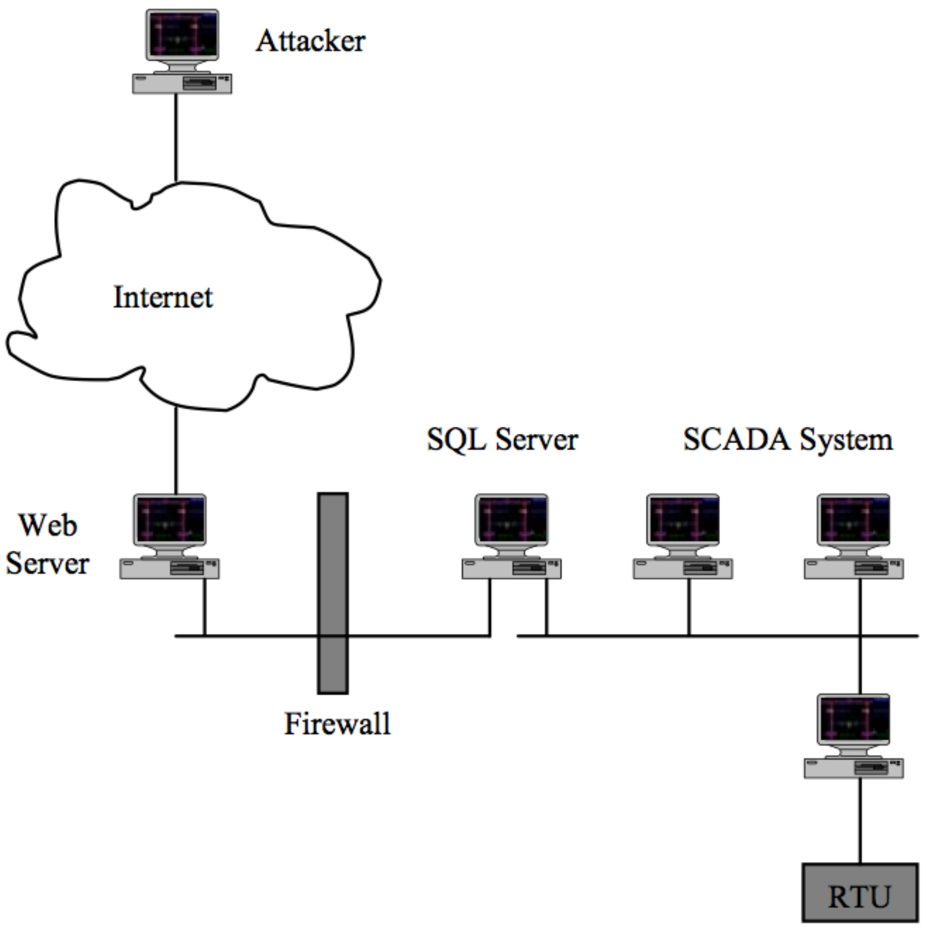


Figure 5.1: SQL Injection Attack

encapsulated for running over TCP/IP.

2. Smurf : is a type of address spoofing, in general, by sending a continuous stream of modified Internet Control message Protocol(ICMP) packets to the target network with the sending address is identical to one of the target computer addresses. In the context of SCADA systems, if a PLC acts on the modified message, it may either crash or dangerously send out wrong commands to actuators.

3. Address Resolution Protocol (ARP) Spoofing/Poisoning: The ARP is primarily used to translate IP addresses to Ethernet Medium Access Control (MAC) addresses and to discover other connected interfaced device on the LAN. The ARP spoofing attack is to modify the cached address pair information. By sending fake ARP messages which contain false MAC addresses in SCADA systems, an adversary can confuse network devices, such as network switches. When these frames are falsefully sent to another node, packets can be sniffed; or to an unreachable host, DoS is launched; or intentionally to an host connected to different actuators, then physical disasters of different scales are initiated. Static MAC address is one of the counter measures. However, certain network switches do not allow static setting for a pair of MAC and IP address. Segmentation of the network may also be a method to alleviate the problem in that such attacks can only take place within same subnet.

4. Chain/Loop Attack: In a chain attack, there is a chain of connection through many nodes as the adversary moves across multiple nodes to hide his origin and identity. In case of a loop attack, the chain of connections is in a loop make it even harder to track down his origin in a wide SCADA system.

Transport Layer

SYN flood is to saturate resources by sending TCP connection requests faster than a machine can process. SCADA protocols, particularly those running over top of transport protocols such as TCP/IP have vulnerabilities that could be exploited by attacker through methodologies as simple as injecting malformed packets to cause the receiving device to respond or communicate in inappropriate ways and result in the operator losing complete view or control of the control device.

Application Layer

Currently, there is no strong security control in protocols used in SCADA systems, such as DNP3, Modbus,Object Linking and Embedding (OLE) for Process Control (OPC), Inter-Control Center Communications Protocol (ICCP). Practically there is no authentication on source and data such that for those who have access to a device through a SCADA protocol, they can often read and write as well. The write access and diagnostic functions of these protocols are particular vulnerable to cyber and cyber induced physical attacks. In the next section 5.3 we discuss in details about Application layer attacks, mainly for two reasons: (a) the potentiality of this kind of attacks in term of damage is considerably higher than in the other attacks, (b) the other attacks considered in this section are in already well-known in literature. As claimed at the beginning of this thesis (section 3.1) we focus our attention on Modbus protocol.

5.3 Modbus - Attack

ModBUS is an application layer messaging protocol, positioned at level 7 of the Open Systems Interconnection(OSI) model (in the case of ModBUS over TCP), that provides client/server communication between devices connected on different types of buses or networks. Communications can be (i) query/response type (communication between a master and a slave), or (ii) broadcast response type where the master send a command to all the slaves . A transaction comprises a single query and single 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 eventually the data needed for the execution of such command. Modbus/TCP basically embeds a Modbus frame into a TCP frame (see details in section 3.1). The ModBUS protocol, as the DNP3 and the ProfiBUS protocol, have been conceived when the subject of ICT security was not relevant for the process control systems. For that reason, when designed, aspects as Integrity, Authentication, no-repudiation etc. were not taken into consideration. More in details, such protocols presents security problems:

1. Do not apply any mechanism for checking the integrity of the command packets sent by a master to a slave.

2. Do not perform any authentication mechanism between master and slaves.

3. Do not apply any anti-repudiation mechanisms to the master.

In the next section, in the light of such considerations, we will present some attack scenarios which will take advantage of such lacks.

5.3.1 Experimental environment

In contrast to alternative works which use modeling approaches to reconstruct the underlying network, thanks to a collaboration project with a Power Company, we recreated in a protected environment, as shown in 5.2, the architecture of a typical power plant plus a set of additional infrastructures supporting the implementation of our tests in a systematic and scientific way.

More in detail, such an environment is constituted by:

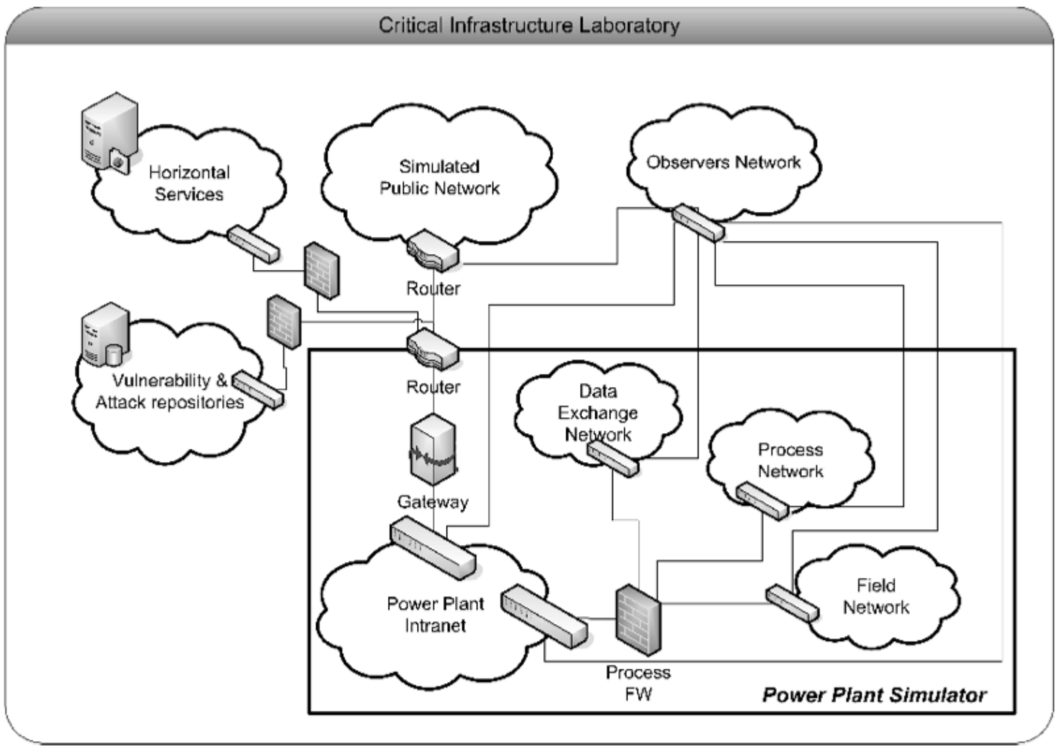


Figure 5.2: High level laboratory environment schema

Power Plant Backbone: it is composed of all the network devices which al- low the different sub-net of the Power Plant to communicate(3Layer switches, Process Network Firewall, Routers, Internet Firewall.

Field Network: it is the network interconnecting the sensors and the actu- ators which directly interact with the Power Plant Electro-Mechanic devices.

Process Network: this network hosts all the SCADA systems. By using these systems, the Plant Operators manage the whole Power Plant, sending control commands to such sensors in the Field Network and reading Plant Measurements and Parameters.

Data Exchange Network: this area hosts a set of data exchange servers, which receive data from the process network and make them available to the operators which work in the Power Plant

Intranet.

Power Plant Intranet: this is the branch of the Company network that provides intranet services to the Power Plant Operators. It is used not only in order to conduct office work, but also to keep remotely under control the Power Plant, by accessing, through a VPN(Virtual Private Network) authentication, the DMZ(Demilitarized Zone) and the Process Network of a target Power Plant.

Public Network: this network simulates the rest of the world (i.e. Internet). In the latest years, as stated before, several critical infrastructures have started to use, in order to provide new services, the public network as communication channel. For that reason, the simulation of such network, is extremely important in order to analyze possible new attack profiles.

Observer Network: its a network of sensors which is used in order to gather information about the system during the experiments.

Horizontal Services network: it provides the usual feature of backup, disaster recovery etc.

Vulnerability and Attack repositories systems: it contains a set of DataBases and analysis system allowing to analyze the collected data. The whole laboratory environment reproduce all the relevant characteristics of a typical power plant; for example, the windows domain of the Power Plant Intranet, has the same identical security and domain policies of a real one which we had the chance to analyze during our research activity, the Process firewall is the same used by default in the power plants of the power company with which we have collaborated, with the same filtering rules and configurations, etc. Such complex testing architecture, has has allowed us to test attack scenario too complex to be represented in a simulated environment and too heavy to be realized in a production facility.

5.3.2 Scada Malware and Attack Scenarios

Starting from the considerations we have made in previous section we identified two possible scenarios in which a tailod made malware could be effective and create serious damages to a critical control system. Since our experimental laboratory is at the moment tailored for recreating the environment of a Power Plant, in the following, we will consider the effects of our attack tests on such kind of systems. As we describe in the following, we concentrate our attention on a particular SCADA protocol, ModBUS, and the malwares we have developed, take advantages of some conceptual and architectural vulnerabilities of such protocol.

ModBUS Malware DoS Scenario

Originally ModBus was conceived in order to be used over serial cable. In such a context, clearly, the risk of external interferences on the communication channel between the master and the slave, were considered practically negligible (at least if we do not consider electromagnetic interferences and physical interruption of the cables). In other words, under such constraints, this closed systems was considered strongly reliable. The porting of the ModBUS protocol over TCP has obviously introduced new layers of complexity in managing the reliable delivery of control packets in an environment strongly real time and, moreover, has opened a new possibility for attackers really motivated in causing damages to the target system.

Attack scope The scope of DoS attack is to desynchronize the communication between Master and Slave and, if possible, completely avoid the communication stream between Master and Slaves. In the light of what presented before, in order to damage the control communication stream, it should be sufficient to per- form some sort of packet-storm against the Master or the set of slaves of the control system. A generic packet storm generator could be normally identified by some Network Intrusion Detection Sensor, or by some firewall anomaly de- tection engine. Ideally, if the packet-storm recreates the same traffic shape of ModBUS traffic, it should be possible to circumvent the monitoring systems, while reaching the scope of avoiding communication between master and slaves.

Attack implementation We have implemented a particular kind of malware, which, once launched, tries to discover the ModBus slaves connected to the same network of the infected machine, and which starts to send them a huge set of ModBUS packets, trying to overload their network bandwidth. More in detail, this malware is composed of the following logical unit:

- A *Packet builder*, which forges in the proper manner ModBUS over TCP packets.
- A *Discovery engine*, which explores the network in order to identify the IP addresses of the Modbus slaves.

- A *Packet deliverer*, which sends in an optimized way the previously forged packets to the target slaves, in order to saturate the bandwidth as soon as possible.

Such a malware, without a proper infection trigger is only an optimized Mod- bus packet generator which have as unique scope sending out a huge number of packets to all the slaves it is able to identify. Such a malware could be effective only when the attacker is able to launch directly the malicious code on a PC connected directly to the field or to the process network of a SCADA system. This scenario is reasonably acceptable (for example the attacker could be simply a disgruntled employee or operator having a direct access to the control system devices), however it will rarely be the first attack option for an internal attacker. Here below we describe other scenarios which can be used instead by an external attacker.

- *E-mail based spreading Scenario:* Some studies regarding the security policies usually implemented in some Power Companies [42] show how the patching operations of PCs or embedded systems in power plant process networks are e-mail based. In other words, a power plant operator receives an e-mail from the ICT-Security team, containing the patching instruction and the patch to be installed; in such scenario the attacker, after gathering information about the hierarchical organization of the ICT security Team, and about the process operators, forges an e-mail identical to the one usually sent for updating purposes (identical not only in the content, but also in term of headers), having attached the previously described malware instead of a normal patch. In such e-mail the attacker asks the operator to install the attached patch on a target Master, or on a PC in the same network. Once installed, the malware will start to deliver massive amount of ModBus packet to the slave, since master and slave will be desynchronized.

- *Through Phishing Infection:* Phishing attacks are typically mounted in one of the following ways: by means of a faked e-mail, displaying a link which seems to point to a legitimate site, but actually linking to a malicious website; or, by poisoning the victims DNS server, thus making it possible to transparently connect to the malicious server. Usually the scope of such attacks is to steal the user credentials. We modified slightly such scenario: In our case in fact, the fake web-server contains a set of malicious scripts allowing to download and execute our ModBUS malware on the local machine from which the web-page has been accessed. The scenario develops as follows: (a) By the use of a fake e-mail or by poisoning the DNS of the process network, an operator is forced to visit an ad-hoc created web-site (b) A set of scripts on the web-site, using some well known vulnerabilities of Microsoft Internet Explorer, download and execute of the operator PC the ModBUS malware (c) the legal ModBUS traffic is interrupted.

- *ModBUS DOS Worm:* A worm is a very effective technique for launching DoS attacks on Modbus networks. We have used MAlSim [43] [44] to create worms similar to Slam- mer, Nimda and Code Red; however, the payloads of these worms carry code that targets Modbus networks. When- ever the worm infects a new machine in a Modbus network, it attempts to spread using the resources of the new host and then executes the Modbus DoS code. This Modbus worm is the first completely independent DoS malware that targets SCADA systems. The worm, initially located on the Internet, infects personal computers in the corporate intranet. Specifically, the worm exploits all infected personal computers with open VPN connections to the power plant process network (these connections are commonly used to perform remote management operations). The worm then enters through the VPN and infects personal computers in the process network. Finally, the worm targets Modbus slaves in the process network by sending Modbus messages that disrupt masterslave communications.

Experimental results The following tables present the communication delays introduced in master/slave communications by the Modbus DoS malware. The first table presents the effects for a master scan rate of 500 ms and a connection timeout of 1200 ms. The second tables shows the results for a master scan rate of 200 ms and a connection timeout of 500 ms. The Modbus DoS worm is clearly dangerous. This is because the worm could simultaneously infect multiple computers in the process network, increasing the average bandwidth consumption and speeding up network degradation. Our testbed closely models the configuration and countermeasures that are usually implemented in a power plant. These include the operating system versions, firewall configurations, anti-virus software and security policies. The operating systems of SCADA servers in a power plant are rarely patched to avoid the risk of conflicts with SCADA applications software [45]. Also, the process of updating anti-virus signatures is much slower than that in a typical corporate intranet environment; many SCADA servers have old versions of operating systems for which anti-virus software is not avail- able [3]. All the attack scenarios discussed above were executed successfully. The main effect of each attack was to disrupt masterslave communications. Since the slaves usually have their own on-board logic, an interruption in the command and data flow generally has negligible effects under normal operating conditions. However, if an incident occurs (e.g., un- ex-

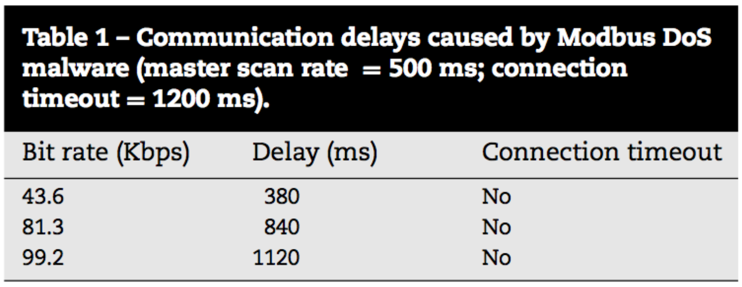


Table 1 – Communication delays caused by Modbus DoS malware (master scan rate = 500 ms; connection timeout = 1200 ms).

Bit rate (Kbps)	Delay (ms)	Connection timeout
43.6	380	No
81.3	840	No
99.2	1120	No

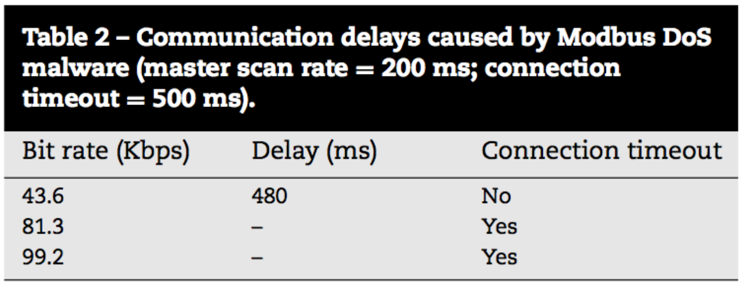


Table 2 – Communication delays caused by Modbus DoS malware (master scan rate = 200 ms; connection timeout = 500 ms).

Bit rate (Kbps)	Delay (ms)	Connection timeout
43.6	480	No
81.3	–	Yes
99.2	–	Yes

pected closing of a valve), the master would be unable to discover the problem because it does not receive data from the slaves. Similarly, it would not be possible to take the appropriate countermeasures if an anomaly or fault were to occur in a control device (e.g., PLC/RTU) because the master would be unable to send the required commands to the slaves. Thus, a disruption in masterslave communications could lead to a potentially disastrous situation in a power plant. Our laboratory testbed incorporates traditional intrusion detection sensors, but these sensors are incapable of detecting Modbus attacks. Of course, the detection capabilities could be enhanced by introducing Modbus-specific rules (e.g., identifying messages with rarely used or anomalous function codes). However, these function codes correspond to valid Modbus commands that are used for diagnostic and maintenance purposes. Consequently, it would be difficult to distinguish between false positives and real attacks.

Modbus worm attack

As mentioned above, the Modbus protocol does not incorporate authentication and integrity mechanisms. When a master sends a message to a slave, the slave executes the command without checking the identity of the sender and the integrity of the message contents. Thus, any attacker with access to the network segment hosting the slaves could send forged Modbus TCP messages that force the slaves to execute unauthorized operations, potentially compromising the system. In the case of a critical infrastructure component such as a power plant, the potential damage could be catastrophic.

Attack scope The goal of the Modbus worm attack is to seize control of the slaves in the process network by exploiting the lack of authentication and integrity mechanisms in the Modbus protocol.

Attack implementation The Modbus worm we implemented is a variant of the Modbus DoS worm presented in Section 5.3.2. After discovering the slaves connected to the same network as the infected machine, the worm sends the slaves a set of correlated Modbus messages with the goal of driving the system to a critical state. The Modbus worm has the following components:

- *Packet builder:* This component fabricates Modbus TCP packets.
- *Discovery engine:* This component explores the network in order to identify Modbus slaves. The information gathered is sent to the strategy and analysis module. The discovery engine is similar to the one presented in the previous section.
- *Strategy and analysis module:* This module uses the information gathered by the discovery engine and built-in heuristics to identify the appropriate attack strategy. The current proof-of-concept strategy is simple; however, it is possible to enhance the module to create complex, coordinated attacks that produce the maximum damage.
- *Packet deliverer:* This component sends fabricated messages to the targeted slaves.

The Modbus worm needs an infection trigger to enter the process network of the targeted SCADA system and do its damage.

Experimental results The Modbus worm was tested in our power plant testbed. Three worm implementations of increasing complexity were developed using MAlSim. In all cases, the worm was able to identify and take control of the slaves.

- *Worm Implementation 1:* This worm uses Modbus messages with function code 15 to force each coil in a slave to the ON or OFF state. The coil addresses range from 0 to 65,535.

Closing or opening the coils can adversely impact the SCADA system and, consequently, the industrial process being controlled.

- *Worm Implementation 2:* This worm targets the input registers of a Modbus slave. It uses Modbus messages with function code 16 to write arbitrary data to a block of contiguous input registers (1 to 123). Once again, this can adversely impact the SCADA system and the industrial process being controlled.

- *Worm Implementation 3:* This worm uses a sequence of two Modbus messages. First, it sends a message with function code 01 to read the state of a sequence of coils in a targeted slave. Next, it sends a message with function code 15 to invert the values of the sequence of coils (Worm Implementation 1). This attack modifies the coil configuration completely.

Our tests highlight the fact that, in order to implement an effective attack on SCADA systems, the malware developer should know at least the high-level details of the system architecture. This knowledge, coupled with the ability to send commands to slaves, enables the malware to potentially seize control of a SCADA system. For example, in the case of a power plant, it would be possible to control the valves that regulate the pressure in a steam cycle power generator, increasing the pressure to a dangerous level or interfering with energy production, potentially causing a blackout. In the case of a pipeline, it would be possible to reduce or block the delivery of gas, oil or water. Furthermore, it is clear that general purpose intrusion detection systems would not be effective. As discussed in the previous scenario, all the commands sent by the malware are legitimate commands. This makes it a priority to develop intrusion detection systems that are cognizant of SCADA protocols, traffic patterns and the operational context.

5.4 Conclusion

The interconnection and integration of ICT systems with SCADA systems in industrial facilities expose critical infrastructure assets to serious threats. In this chapter we show the possible attacks and we analyzed the consequence of each single attack. Starting from the security property described in the section5.1 our experimental tests have shown that the most dangerous attack for a SCADA system is a malware tailored for SCADA protocols that was able to completely circumvent traditional security mechanisms by adopting ad hoc infection and attack strategies, enabling the malware to disrupt or even seize control of vital sensors and actuators.

Unfortunately, the traditional ICT security countermeasures employed in corporate environments are incapable of dealing with SCADA-protocol-specific attacks. Integrity and confidentiality are critical security proprieties concerning SCADA systems. In the next chapter we focus our attention on solution regarding signature and authentication and we propose also a new Intrusion Detection Approach able to detect and mitigate anomalous behavior.

Chapter 6

Secure Modbus

Most SCADA protocols in use today were designed decades ago, when the technological infrastructure and threat landscape were quite different from how they are today. For example, Modbus was originally published in 1979 for a multidrop network with a master/slave architecture. Because Modbus networks were isolated and free from security threats, key aspects such as integrity, authentication and non-repudiation were not taken into consideration in the design of the protocol. In order to guarantee integrity in the next section we focus on Modbus protocols and we propose a new version of this protocol. We do not consider the confidentiality requirement for Modbus messages for two reasons. First, enforcing confidentiality does not mitigate any of the attack scenarios presented above. Second, confidentiality is generally implemented using encryption, which is expensive and introduces considerable overhead that can impact real-time performance.

6.1 Modbus Vulnerabilities

The transportation of Modbus messages using TCP introduces new levels of complexity with regard to managing the reliable delivery of control packets in a process control environment with strong real-time constraints. In addition, it provides attackers with new avenues to target industrial systems. Modbus TCP lacks mechanisms for protecting confidentiality and for verifying the integrity of messages sent between a master and slaves (i.e., it is not possible to discover if the original message contents have been modified by an attacker). Modbus TCP does not authenticate the master and slaves (i.e., a compromised device could claim to be the master and send commands to the slaves). Moreover, the protocol does not incorporate any anti-repudiation or anti-replay mechanisms. The security limitations of Modbus can be exploited by attackers to wreak havoc on industrial control systems. Some key attacks are:

- Unauthorized Command Execution: The lack of authentication of the master and slaves means that an attacker can send forged Modbus messages to a pool of slaves. In order to execute this attack, the attacker must be able to access the network that hosts the SCADA servers or the field network that hosts the slaves. In the previous chapter [38] we show that the attack can be launched by creating malware that infects the network and causes malicious messages to be sent automatically to the slaves.

- Modbus Denial-of-Service Attacks: An example attack involves impersonating the master and sending meaningless messages to RTUs that cause them to expend processing resources.

- Man-in-the-Middle Attacks: The lack of integrity checks enables an attacker who has access to the production network to modify legitimate messages or fabricate messages and send them to slave devices.
- Replay Attacks: The lack of security mechanisms enables an attacker to reuse legitimate Modbus messages sent to or from slave devices.

The best way to address the security threats is to solve them at their origin by attempting to repair the security holes in the Modbus protocol. But such a solution is difficult to implement because it requires significant changes to the control system architecture and configuration. Instead, we adopt a practical approach in which a small number of security mechanisms are introduced into the protocol to protect against the attacks described above.

6.2 Secure Modbus Protocol

A communications protocol is considered secure if it satisfies security requirements such as confidentiality, integrity and non-repudiation. In other words, a secure Modbus protocol should guarantee that:

- No unauthorized entity is allowed to access the content of a message.
- No unauthorized entity is allowed to modify the content of a message.
- No entity is allowed to impersonate another entity.
- No entity is allowed to negate a performed action.
- No entity is allowed to reuse a captured message to perform an unauthorized action.

The original Modbus Serial protocol defines a simple protocol data unit (PDU), which is independent of the underlying communication layer (Figure 6.1). The mapping of Modbus messages to specific buses or networks introduces additional fields in an application data unit (ADU). The Modbus TCP protocol introduces a dedicated Modbus application protocol (MBAP) header. The Slave Address field in a Modbus Serial message is replaced by a one-byte Unit Identifier in the MBAP Header. Also, the error checking field is removed and additional length information is stored in the MBAP header to enable the recipient to identify message boundaries when a message is split into multiple packets for transmission. All Modbus requests and responses are designed so that the recipient can verify that the complete message is received. This is accomplished by simply referring to the function code for function codes whose Modbus PDUs have fixed lengths. Request and response messages with function codes that can carry variable amounts of data incorporate a field containing the byte count.

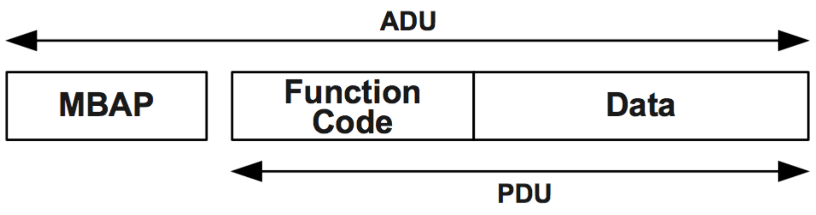


Figure 6.1: Modbus Application Data Unit

The proposed Secure Modbus protocol is intended to satisfy the integrity security requirements defined in the chapter 5.1. in particular we present a details characteristic that must be implemented in Secure Modbus to guarantee the integrity:

- *Authentication:* The integrity mechanism described above does not prevent an attacker from creating a malicious Modbus packet, computing its SHA2 digest, and sending the malicious packet and the digest to the receiver. To address this issue, the Secure Modbus protocol employs an RSA-based signature scheme [46]. Specifically, the originator of the Secure Modbus packet computes the SHA2 digest, signs the digest with its RSA private key, and sends the packet and the signed digest to the receiver. The receiver verifies the authenticity of the digest (and the packet) using the senders public key. Thus, the receiver can ensure that the Secure Modbus packet was created by the purported sender and was not modified en route.

- *Non-Repudiation:* The RSA-based signature scheme also provides a non-repudiation mechanism only the owner of the RSA private key could have sent the Secure Modbus packet.

- *Replay Protection:* The SHA2 hashing and RSA signature schemes do not prevent an attacker from re-using a sniffed Modbus packet signed by an authorized sender. Thus, the Secure Modbus protocol needs a mech- anism that enables the receiver to discriminate between a new packet and a used packet. This is accomplished by incorporating a time stamp (TS) in the Secure Modbus application data unit (Figure 6.2). The time stamp is used by the receiver in combination with an internal time win- dow to check the freshness of the received packet. Our initial solution employed a simple two-byte sequence number and provided all Modbus devices with time windows of limited size to verify freshness. However, this solution was neither elegant nor completely secure. Consequently, our current implementation uses NTP time stamps that facilitate the evaluation of freshness with high precision. Of course, employing NTP time stamps requires an NTP server in the SCADA architecture to provide a reliable clock for all communicating devices.

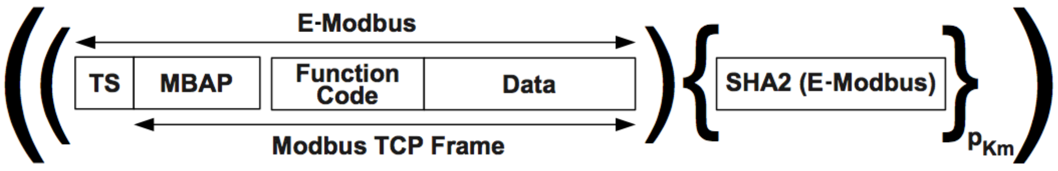


Figure 6.2: Secure Modbus application data unit

The Secure Modbus protocol satisfies the minimum requirements of a secure protocol. However, it is just as important to ensure that the protocol can be implemented efficiently in real-world SCADA environments. Secure Modbus can be readily deployed in SCADA systems with adequate computing resources, network bandwidth and modern, upgradeable slave devices. However, many critical infrastructure assets employ decades-old equipment; therefore, it is important to ensure that legacy systems can be retrofitted (at low cost) to support Secure Modbus. We designed the Modbus Secure Gateway to facilitate the deployment of Secure Modbus in legacy SCADA environments. Figure 6.3 presents a schematic diagram of the Modbus Secure Gateway. It is a dedicated multi-homed gateway that hosts a TCP/IP interface connected to the process network and a set of point-to-point TCP or serial links connected to legacy slaves. The Modbus Secure Gateway operates as follows:

- When it receives a packet on the *process network* interface:
  - It accepts only authenticated Secure-Modbus over TCP traffic from the allowed masters
  - It extracts the Modbus packet from the Secure-Modbus packet.
  - It forwards the packet to the proper slave using the related point-to-point link (serial or tcp).
- When it receives a packet on one of the point-to-point links connected with a slave:

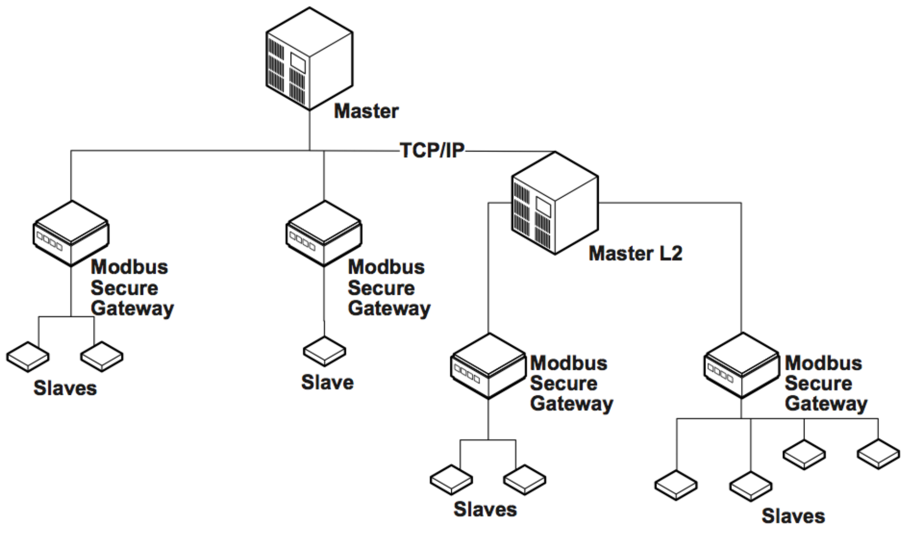


Figure 6.3: Modbus Secure Gateway

- It builds a Secure-Modbus packet containing the received original Modbus packet
- It signs the packet's digest with the private key associated to the related slave
- It forwards the new packet to the proper master through its *process network* interface

Obviously, since the MS-GW constitutes a "single point of failure" in the SCADA architecture, it should be installed only when the "pure Secure Modbus" implementation is not feasible. Next, we summarize the steps involved in sending and verifying a Secure Modbus request message:

1. The Master in accordance with the protocol specifications composes the Modbus request(M_{req}) with the time stamp and the serial slave address.
2. The Master calculates the Modbus request digest and authenticates it with his private key PK_m and sends it to the Slave (or to the MSGW)

$$C = [TS|Modbus]\{SHA2(TS|Modbus)\}p_{Km} \tag{6.1}$$

3. The slave (or the MSGW) validates the Modbus request with the Master's public key (SK_m)

$$M_{req} = Dec\{C, SK_m\} \tag{6.2}$$

4. If the packet has been delivered to a MSGW, it reads the Unit Identifier in the MBAP header and sends the MosBUS request to the right Slave.

Note that after verifying that the request is genuine, the Modbus Secure Gateway reads the unit identifier in the MBAP header and sends the Modbus request to the addressed slave. Similar steps are involved when a slave sends a response to the master.

6.3 Secure Modbus Implementation

The basic communication layer between the operating system and the Modbus device is guaranteed by a socket (Level1), which therefore is the basic building block for communication. A Modbus communication is executed by sending and receiving data through sockets. The TCP/IP library provides only stream sockets using TCP and a connection-based communication service. The Sockets are created via the socket () function, which returns a number that is then used by the creator to access the socket. In Figure 6.4 it is possible to the the Highe level architecture of the developed Secure Modbus Module.

The main functions of the TCP/IP level are to manage the communication establishment and ending, and to manage the data flow upon established TCP connections. Through the TCP stream adapter we can set up all the connection parameters for adapting its behavior to the constraints of the system:

- KEEP-ALIVE: client-server applications uses KEEPALIVE to detect inactivity in order to close a connection or to find some communication problem. Shorter timer can cause good connections to be dropped.
- TCP-NODELAY: in order to have a real-time system
- TIME OUT CONNECTIONS: the default time limit before dropping the communicating and establishing a TCP Connection is set to 75 seconds. This default value should be adapted to the real time constraint of the particular application of each implementation case.

The Secure Modbus module is composed of four main parts:

- **Modbus Stream Adapter:** It extracts the Secure-Modbus packet contained in the TCP payload by sending it to the RSA enc/dec, which will verify the authenticity of the SHA digest.

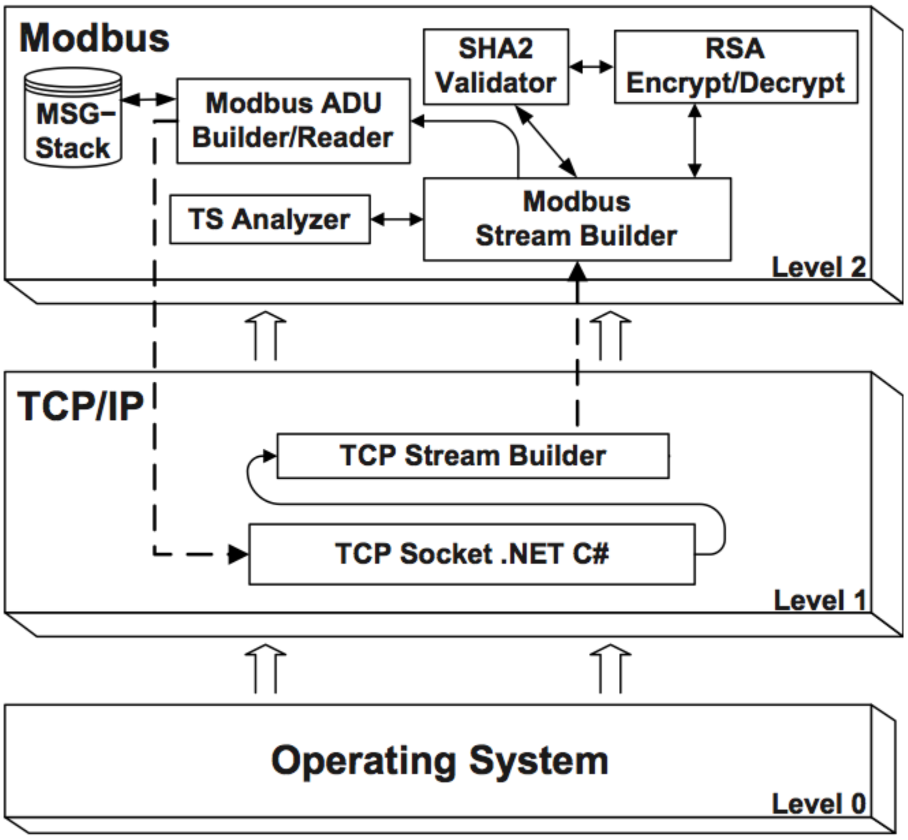


Figure 6.4: High level architecture of the Secure-Modbus Module

The same module, will then send the SHA Digest to the SHA2 validator, to verify the integrity of the packet, and finally it will send the time-stamp to the TS Analyser to verify the freshness of the received command. If all these conditions are satisfied, the module will send the Modbus packet to the related PLC application.

- **RSA enc/dec:** It utilizes the public key of the sender to verify the digest authenticity and the private key of the sender to sign the hash message.
- **SHA2 validator:** It calculates and validates the hash of the Modbus request/response.
- **Modbus ADU Builder:** It builds and manages the Modbus Application Data Unit, communicating with the SHA module and with the RSA enc/dec module for authenticating the

packets.

- **TS Analyser:** It verifies the time stamp validity by using alternatively a time-window or an NTP service (optional).

6.3.1 Experimentals results

We conducted two experiments to evaluate the performance of the Secure Modbus protocol. The first experiment examined the latency resulting from the use of the SHA2 hashing and RSA-based signature schemes. The second examined the increased size of Secure Modbus packets for various function codes.

Modbus		**Secure Modbus**	
Scan Rate	500ms	Scan Rate	500ms
Connection Time Out	1200ms	Connection Time Out	1200ms
Latency	26ms	Latency	27ms

Table 6.1: Communication latency with Modbus and Secure Modbus with a master scan rate of 500ms and a connection timeout of 1200ms

Modbus		**Secure Modbus**	
Scan Rate	200ms	Scan Rate	200ms
Connection Time Out	500ms	Connection Time Out	500ms
Latency	29ms	Latency	31ms

Table 6.2: Communication latency with Modbus and Secure Modbus with a master scan rate of 200ms and a connection timeout of 500ms

Table 6.3.1 compares the communication latency for Modbus TCP and Secure Modbus. The first set of results, corresponding to a master scan rate of 500 ms and a connection timeout of 1,200 ms, show a latency of 26 ms for Modbus and 27 ms for Secure Modbus a negligible difference. A negligible latency difference of 2 ms (29 ms for Modbus TCP and 31 ms for Secure Modbus) is also observed for a master scan rate of 200 ms and a connection timeout of 500 ms. Table 6.3.1 compares the size of Modbus TCP and Secure Modbus packets for four function codes. Secure

Function	Modbus TCP/IP	Sec Modbus	Overhead
Write Coil (0x05)	11 bytes	43 bytes	291 %
Write Reg. (0x06)	12 bytes	44 bytes	267 %
Write Multiple coils (0x0F)	260 bytes	292 bytes	12 %
Write Multiple reg. (0x10)	260 bytes	292 bytes	12 %

Table 6.3: Modbus/TCP and Secure Modbus/TCP packets size, tested with different functions

Modbus packets are larger than the corresponding Modbus TCP packets. However, the increased size is not a significant issue even for SCADA networks with low bandwidth.

6.4 Conclusion

Secure Gateway facilitates the deployment of Secure Modbus in legacy SCADA environments. While the new protocol helps protect against several attacks, it does not address scenarios where an attacker seizes control of a master and sends malicious Modbus messages to slave devices, or where an attacker captures the master units private key and forges malicious Modbus messages that are signed with the stolen key. To address the first attack scenario, will be presented in the nedt chapter a new Intrusion Detection System (IDS) that will identify suspect Modbus messages. Our solution to the second scenario is to use a trusted computing platform to protect key rings.

Chapter 7

Intrusion Detection for Industrial control systems

In general, security can be achieved in three phases: prevention, detection, and correction. Prevention is the most ideal solution, but unfortunately the history shows us it cant be achieved perfectly. Even so, it is a very bad idea to entirely rely on prevention. Its because in case an attacker somehow finds a way to make a security whole on prevention, the cost for fixing the vulnerability and restoring the system back to normal condition must be incredibly expensive in the later phase if there is no preparation for that especially considering critical environments as SCADA systems are. Therefore, security protection systems are better off having ready-to-go correction mechanisms as well. By well-designed correction mechanisms, compromised or malfunctioning systems can be quickly repaired and restored to normal condition. Prevention is effective before successful intrusions. Correction is active after successful intrusions. However, once successful attacks eventually manage to get through prevention, its a matter of time that the whole system is attacked, compromised, and malfunctioned. Thus, we need to have an interim stage such as detection phase, which is positive during intrusion. By a detection mechanism, even if prevention fails to stop intrusion, a protected system can be at least aware of being attacked so that the system can take some actions to reduce the probability of propagating damage and loss.

7.1 Intrusion Detection Techniques

Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices. Incidents have many causes, such as malware (e.g., worms, spyware), attackers gaining unauthorized access to systems from the Internet, and authorized users of systems who misuse their privileges or attempt to gain additional privileges for which they are not authorized. Although many incidents are malicious in nature, many others are not; for example, a person might mistype the address of a computer and accidentally attempt to connect to a different system without authorization. IDS technologies use many methodologies to detect incidents:

- Signature-based
- Anomaly-based
- Stateful protocol analysis

Most IDS technologies use multiple detection methodologies, either separately or integrated, to provide more broad and accurate detection.

7.1.1 Signature-based

A signature is a pattern that corresponds to a known threat. Signature-based detection[47] is the process of comparing signatures against observed events to identify possible incidents. Examples of signatures are as follows:

- A telnet attempt with a username of root, which is a violation of an organizations security policy
- An e-mail with a subject of Free pictures! and an attachment filename of freepics.exe, which are characteristics of a known form of malware
- An operating system log entry with a status code value of 645, which indicates that the hosts auditing has been disabled.

Signature-based detection is very effective at detecting known threats but largely ineffective at detecting previously unknown threats, threats disguised by the use of evasion techniques, and many variants of known threats. For example, if an attacker modifies the malware in the previous example to use a filename of freepics2.exe, a signature looking for freepics.exe would not match it. Signature-based detection is the simplest detection method because it just compares the current unit of activity, such as a packet or a log entry, to a list of signatures using string comparison operations. Signature-based detection technologies have little understanding of many network or application protocols and cannot track and understand the state of complex communications. For example, they cannot pair a request with the corresponding response, such as knowing that a request to a Web server for a particular page generated a response status code of 403, meaning that the server refused to fill the request. They also lack the ability to remember previous requests when processing the current request. This limitation prevents signature-based detection methods from detecting attacks that comprise multiple events if none of the events contains a clear indication of an attack. In applying a signature-based IDS to SCADA, the attack signature database must have different characteristics than signatures in an IT-oriented database. In particular, the signatures have to be correlated with SCAD Aprotocols such asFoundation fieldbus, Modbus, Profinet, ControlNet, and so on. Typical SCADA IDS signature components include IP addresses, transmitted parameters, and protocols.

7.1.2 Anomaly-based

Anomaly-based detection is the process of comparing definitions of what activity is considered normal against observed events to identify significant deviations. An IDS using anomaly-based detection has profiles that represent the normal behavior of such things as users, hosts, network connections, or applications. The profiles are developed by monitoring the characteristics of typical activity over a period of time. In particular regarding the network the important phase in defining the network behavior is the IDS engine capability to cut through the various protocols at all levels. The Engine must be able to process the protocols and understand its goal. Though this protocol analysis is computationally expensive, the benefits it generates like increasing the rule set helps in less false positive alarms. The major drawback of anomaly detection is defining its rule set. The efficiency of the system depends on how well it is implemented and tested on all protocols. Rule defining process is also affected by various protocols used by various vendors. Apart from these, custom protocols also make rule defining a difficult job. For detection to occur correctly, the detailed knowledge about the accepted network behavior need to be developed by the administrators. But once the rules are defined and protocol is built then anomaly detection systems works well. Including malicious activity as part of a profile is a common problem with anomaly-based IDPS products. (In some cases, administrators can modify the profile to exclude activity in the profile that is known to be malicious.) Another problem with building profiles is that it can be very challenging in some cases to make them accurate, because computing activity can be so complex. For example, if a particular maintenance activity that performs large file transfers occurs only once a month, it might not be observed during the training period; when the maintenance occurs, it is likely to be considered a significant deviation from the profile and trigger an alert. Many distinct techniques are used based on type of processing related to behavioral model.

Operational Model (or) Threshold Metric: The count of events that occur over a period of time determines the alarm to be raised if fewer then ”m” or more than ”n” events occur. This can be visualized in Win lock, where a user after n unsuccessful login attempts here lower limit is ”0” and upper limit is n. Executable files size downloaded is restricted in some organizations about 4MB.The difficulty in this submodel is determining m and n.

Markov Process or Markov Model: The Intrusion detection in this model is done by investigating the system at fixed intervals and keeping track of its state. The change of the state of the system occurs when an event happens and the behavior is detected as anomaly if the probability of occurrence of that state is low. The transitions between certain commands determine the anomaly detection where command sequences were important.

Statistical Moments or Mean and Standard Deviation Model: In statistical mean, standard deviation, or any other correlations are known as a moment. If the event that falls outside the set interval above or below the moment is said to be anomalous. The system is subjected to change by considering the aging data and making changes to the statistical rule data base. There are two major advantages over an operational model. First, prior knowledge is not required determining the normal activity in order to set limits; Second, determining the confidence intervals depends on observed user data, as it varies from user to user. Threshold model lacks this flexibility. The major variation on the mean and standard deviation model is to give higher weights for the recent activities.

Machine Learning Based Detection: Machine learning techniques to detect outliers in datasets from a variety of fields were developed by Gardener (use a One-Class Support Vector Machine (OCSVM) to detect anomalies in EEG data from epilepsy patients) and Barbara (proposed an algorithm to detect outliers in noisy datasets where no information is available regarding ground truth, based on a Transductive Confidence Machine (TCM) [48]).Unlike induction that uses all data points to induce a model, transduction, an alternative, uses small subset of them to estimate unknown attributes of test points. To perform online anomaly detection on time series data in [49], Ma and Perkins presented an algorithm using support vector regression. Ihler et al. present an adaptive anomaly detection algorithm that is based on a Markov-modulated Poisson process model, and use Markov Chain Monte Carlo methods in a Bayesian approach to learn the model parameters [50].

7.2 State Analysis Technique

The State-Based Approach is not based on packet signatures techniques, as a classic IDS, but it is based on a representation of the state of the SCADA system. This approach relies on the assumption that the ultimate scope of an attacker is to put the target system into a critical state, therefore by looking at the system evolution and by detecting the occurrence of critical states, it would be possible to automatically detect an attack. Using this approach, the focus is primarily on known variables (the system behavior), rather than on unknown variables (the attacker behavior), which eliminates the risks of false positives and detecting also unknown attacks. This approach is possible in the context of industrial settings, since the system behavior space is fully known and finite. In order to achieve this goal, we have formally defined how to represent a SCADA system. The representation of the system is useful for the creation and update of the System Virtual Image (SVI). The image is a software copy of a real system and evolves in parallel with the real system. The SVI is able to match his status with a list of critical states in order to recognize potential threats to the system. In order to better understand the approach presented in this chapter we explain the critical state concept. Consider the following example: we have a system with a pipe P_1 in which flows high pressure steam. The pressure is regulated by two valves: the valve V_{IN} controls the pipe incoming steam and the valve V_{OUT} controls the pipe outcoming steam. The valve V_{IN} is controlled by the PLC_1 and the valve V_{OUT} is controlled by the PLC_2. The Figure 7.1 shows such scenario.

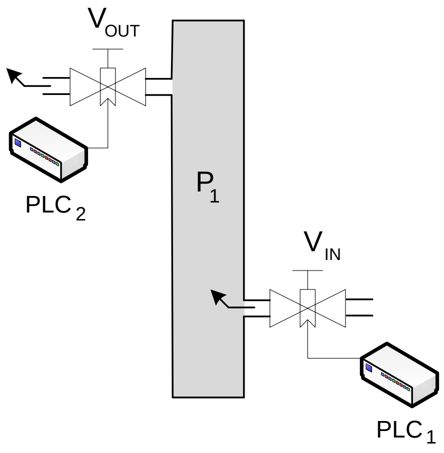


Figure 7.1: Critical State Example

A critical state could be generated by an attacker. For example he is able to send packets on the process network and sends a command to the PLC_2 controlling the valve V_{OUT} in order to force its complete closure, and a command to the PLC_1 controlling the valve V_{IN} in order to maximize the incoming steam. This is a critical state because the pressure in the pipe P_1 may became too high soon and the pipe could explode.

From an operational point of view, the following elements are required for tracking and analyzing the evolution of a system:

- A *system representation language* to describe in a formal way the system under analysis.
- A *system state language* to describe in a formal way the critical states associated to the system under analysis.
- A *state evolution monitor*to follow the evolution of the system.
- A *critical state detector* to check whether the state of the system is evolving toward a defined critical state.
- A *critical state distance metric* to compute how close any state is with respect to the critical states.

7.2.1 System Description and Critical State Representation

A SCADA system, as we described in the chapter 1, is composed by an operator, a Human machine Interface (HMI), one or more Master Terminal Unit (MTU) and some Remote Terminal Units (RTUs) or PLCs. The intention is to represent a SCADA system, i.e. to create a virtual image of the real system. The image of the entire SCADA system consists of the image of each single component such as SCADA Masters or MTUs and SCADA Slaves or PLCs. Representation of the operator and the HMI are of less importance, given the nature of the types of threats the system is intended to protect against (i.e. external attacks). More formally, the system image is a set of images:

$$SYSTEM_{image} = \{Master_{image}, PLC\_1_{image}, ..., PLC\_n_{image}\}$$

. The representation of each PLC or Master station must contain:

- *The architecture of the item:* it is important to represent the information to identify the item, i.e. the IP address and the TCP port, and the information about the item architecture, i.e.

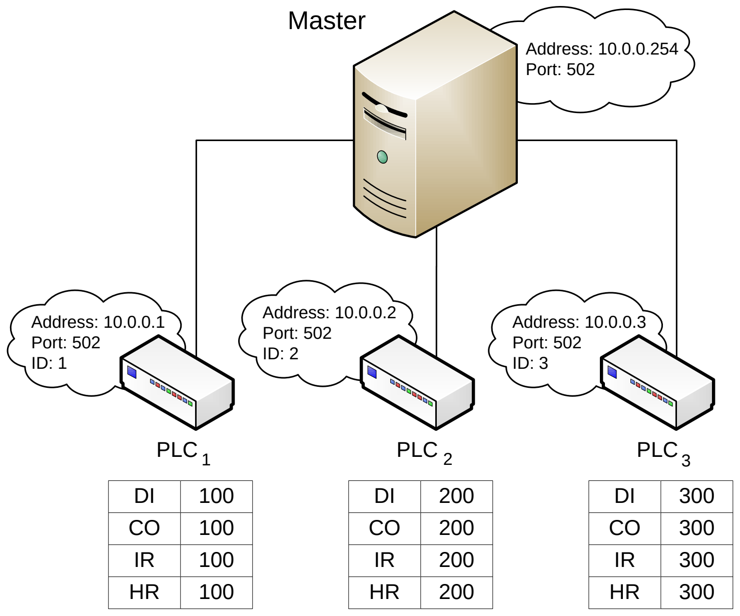
numbers and type of registers.

- *The current status of the item:* the status of a SCADA Slave is the current value of each register. SCADA Masters don't need information about the status.

In the following we present an example of a SCADA system, which uses the MODBUS protocol as communication protocol between masters and slaves.

MODBUS SCADA System Example

Imagine to have the scenario shown in Figure 7.2 with one SCADA Master and three PLCs or SCADA Slaves.



DI	100
CO	100
IR	100
HR	100

DI	200
CO	200
IR	200
HR	200

DI	300
CO	300
IR	300
HR	300

Figure 7.2: MODBUS SCADA System Example

The shown system contains one SCADA Master with the IP address 10.0.0.254 and TCP port 502 (MODBUS default port according to the modbus TCP specification [20]) and three SCADA Slaves:

- The PLC_1 with the IP address 10.0.0.1, TCP port 502 and ID = 1.

- The PLC_2 with the IP address 10.0.0.2, TCP port 502 and ID = 2.
- The PLC_3 with the IP address 10.0.0.3, TCP port 502 and ID = 3.

According to the MODBUS specification [20] each PLC contains Discrete Inputs (DI), Coils (CO), Input Registers (IR) and Holding Registers (HR). In the previous scenario there are 100 DI, CO, IR, HR in the PLC_1, 200 in the PLC_2 and 300 in the PLC_3.
In a MODBUS SCADA Slave, as described in 3, there are also other registers and coils: the "*Exception Status Coils*", the "*Diagnostic Register*" and some "*Counters*", which are used for diagnostic functions. In light of these consideration a possible representation of the SCADA system in figure 7.2 is shown in the UML diagram in Figure 7.3.

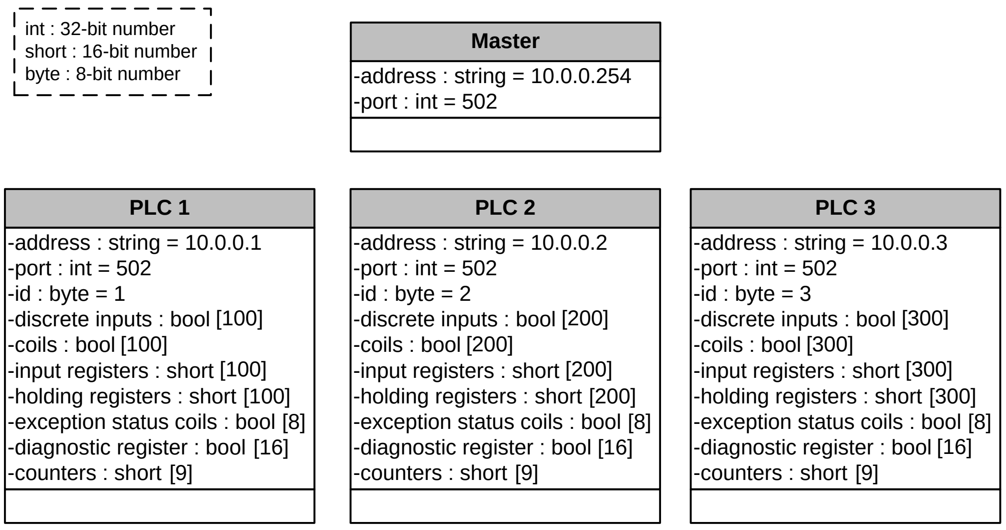

int : 32-bit number
short : 16-bit number
byte : 8-bit number
Master
-address : string = 10.0.0.254
-port : int = 502
PLC 1
-address : string = 10.0.0.1
-port : int = 502
-id : byte = 1
-discrete inputs : bool [100]
-coils : bool [100]
-input registers : short [100]
-holding registers : short [100]
-exception status coils : bool [8]
-diagnostic register : bool [16]
-counters : short [9]
PLC 2
-address : string = 10.0.0.2
-port : int = 502
-id : byte = 2
-discrete inputs : bool [200]
-coils : bool [200]
-input registers : short [200]
-holding registers : short [200]
-exception status coils : bool [8]
-diagnostic register : bool [16]
-counters : short [9]
PLC 3
-address : string = 10.0.0.3
-port : int = 502
-id : byte = 3
-discrete inputs : bool [300]
-coils : bool [300]
-input registers : short [300]
-holding registers : short [300]
-exception status coils : bool [8]
-diagnostic register : bool [16]
-counters : short [9]

Figure 7.3: Representation of the previous system.

The UML diagram above could be implemented in an object oriented language such as C++, C#, Java etc. Each entity in the diagram could be implemented with an object. The object which represents the Master contains only two attributes: a string which represents the IP address and an integer that represents the TCP port number. The other objects represent the Slaves contain many attributes as shown in the Table 7.1

Slave Identification	
address	a string which contains the Slave IP Address
port	a 32-bit number which contains the Slave TCP Port
id	a 8-bit number which contains the Slave id
Registers and Coils	
discrete inputs	an array of boolean values which contains the value of each discrete input
coils	an array of boolean values which contains the value of each coil
input registers	an array of 16-bit numbers which contains the value of each input register
holding registers	an array of 16-bit numbers which contains the value of each holding register
Diagnostic	
exception status coils	an array of boolean values which contains the value of each exception status coil
diagnostic register	a 16-bit number which contains the diagnostic register value
counters	an array of 16-bit numbers which contains the value of each counter

Table 7.1: MODBUS Slave Representation fields

With the structure shown in Table 7.1 is possible to represent the architecture of a SCADA MODBUS Slave and also the current status of the device. The architecture is represented by the slave identification attributes (i.e. address, port and id), the numbers of coils and registers (i.e. array sizes) and the current status is stored in each array cell.

Formal reppresentation

For the formal representation of industrial system states we have specifically defined a new formalized language called Industrial State Modeling Language (ISML). This language supports SCADA systems that use the MODBUS protocol, but it can be easily extended in order to support other industrial protocols. A rule in the ISML has the form $condition \rightarrow action$. $Condition$ is a boolean formula composed of conjunctions of predicates describing what values can be assumed by the different critical components connected to the Programmable Logic Controllers (PLCs). More in detail ISML is defined by a standard BNF notation:

$$\begin{aligned}
\langle rule \rangle &::= \langle condition \rangle \rightarrow \langle action \rangle : \langle level \rangle \\
\langle level \rangle &::= 1|...|5 \\
\langle condition \rangle &::= \langle predicate \rangle \,|\langle predicate \rangle,\ \langle condition \rangle \\
\langle predicate \rangle &::= \mathrm{PLC}[\langle ID \rangle].\langle comp \rangle[\langle \mathit{16bit\_integer} \rangle] \bowtie \langle realvalue \rangle \\
\langle ID \rangle &::= \mathrm{IPaddress : Port} \qquad \bowtie \in \{\leqslant, \geqslant, <, >, =, \neq\} \\
\langle comp \rangle &::= \mathrm{HR}\,|\,\mathrm{IR}\,|\,\mathrm{CO}\,|\,\mathrm{DI}|\mathrm{EX}|\mathrm{DR}|\mathrm{CTR} \quad \langle action \rangle ::= \mathrm{Warning}\,|\,\mathrm{Alert}\,|\,\mathrm{Log}
\end{aligned}$$

The system *state* is defined by the values of the system components. The ISML language has two function:

- Provide a detailed description of the system to monitor, which will be used to generate the virtual system used by the IDS

- Describe *Critical States* that correspond to dangerous or unwanted situations in the monitored system.

For each Critical State it is possible to specify the risk level. The value 1 corresponds to a low risk critical state (e.g. the system is running at less than optimum efficiency). The value 5 corresponds to a critical state dangerous for the system. The system *state* is defined by the values of the system components. Coils and digital outputs can assume digital values: 0 or 1. Input registers(IR) and holding registers(HR) consist of 16 bit variables and can assume values from 0 to 65535.

In the following is considered the previous scenario where the system consists of one master and three PLCs. In such scenario two valves and a pipe are added, in order to create the critical state example explained in the beginning of this chapter. This would result in a situation similar to that shown in Figure 7.4.

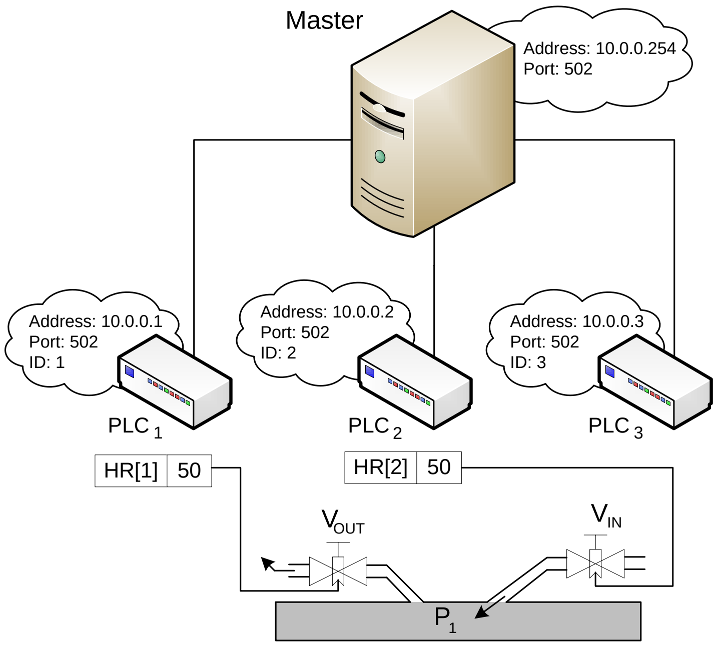


Figure 7.4: SCADA System plus Critical State Situation.

In this architecture we have added the pipe P_1 in which flows steam, the valve V_{IN}, which controls the pipe incoming steam and the valve V_{OUT}, which controls the pipe outcoming steam. In such an example the valves are connected to the SCADA system in the following way:

- V_{IN} **(input steam):** is connected to PLC 10.0.0.1 and the holding register HR[1] contains a value which represents how much the valve is open, i.e. if HR[1] = 100 then V_{IN} is completely open, if HR[1] = 0 then V_{IN} is completely closed, if, like in the picture, HR[1] = 50 then V_{IN} is half closed.

- V_{OUT} **(output steam):**: is connected to PLC 10.0.0.2 and the holding register HR[2] contains a value which represents how much the valve is open; it works in the same way as the valve V_{IN}.

The system is in a critical state when the valve V_{OUT} is too far closed and the valve V_{IN} is too far open, because in a such situation the amount of steam entering the pipe exceeds the amount being released, leading to an increase in pipe pressure and an increased chance of a pipe explosion. If we define "*too far closed*" as a value of 20% for the valve V_{OUT} and "*too far open*" as a value of 70% for the valve V_{IN}, then we can represent the system in a "critical state" using the following boolean expression:

$$\begin{pmatrix} \text{PLC}[10.0.0.10:502].\text{HR}[1] < 1000, \\ \text{PLC}[10.0.0.22:502].\text{IR}[1] > 99 \end{pmatrix} \rightarrow \mathit{Alert}:4$$

The IDS raises an alert if this critical formula is satisfied.

7.2.2 State Evolution Monitor

A *State Evolution Monitor* (SEM) is a software artifact that keeps track of the evolution of the system state. In the presented approach, the formal description of the system, defined by means of the ISML language, is used to create a software virtual image of the monitored subsystem. Each element is represented by a software object reproducing the classical memory map representation of PLCs and Masters. The virtual image contained in SEM is fed using the control traffic exchanged between Masters and Slaves. In other words, relying on the assumption that the control flow between Master and Slaves contains a compact representation of the evolution of the system, by sniffing this traffic it is possible to maintain in the SEM memory a reasonable reproduction of the real system state. Moreover, to guarantee a tight synchronization between the virtual system and the real system, the SEM contains a master emulator for directly querying the PLCs of the monitored system to limit its interference with the system, it is possible to define an "aging time" for every register and coil instantiated into the virtual system, after which the master emulator can perform a direct query. The memory access for updating the virtual system could be potentially expensive. In the implemented prototype all the virtual components are indexed using a hash table, for providing direct access to each element.

7.2.3 Critical State Detection

Every possible state of the system is described by the values of the terms representing the components of the system. A system state with n components can be represented by a vector. The previous example can be represented with states $s \in \mathbb{R}^n$, where the value of the turbine speed is mapped to s_1 and the value of the temperature sensor is mapped to s_2. The set of *critical states*

$CS \subseteq \mathbb{R}^n$ is the set of states *satisfying* the *critical conditions* described as in the Subsection 7.2.1. The SEM can be described by a function $\sigma\colon \mathbb{N} \to \mathbb{R}^n$ providing the state of the system at a certain time t. Following these definitions, we can say that the monitored system is in a critical state iif $\sigma(t)s(t) \in CS$.

Establishing whether a given state s is critical, means to verify if s *satisfies* any of the critical formulas in CS. Given a state s and a critical formula ϕ, verifying whether s satisfies ϕ corresponds to evaluating ϕ with the system components' values represented by s. The time complexity for evaluating ϕ is linear with respect to the number of predicates occurring in it. Even if evaluating a formula can be easily achieved with a simple visit of its syntax tree, we use a different representation in memory of the formula based on *interval constraints*, as described in Section **??**. Evaluating formulas using the constraint-based memory representation has the same complexity as the syntax tree visit. From an operational point of view, the *Critical State Analyzer* checks if the SEM status matches at least one of the specified critical states. If this is the case, it raises an alert, storing the details about the command packets that caused the critical state (if they exist); in this way it would be possible to discriminate between cyber-attacks and faults/physical attacks. It is relevant to note that alerts are triggered by the occurrence of a critical state, and not by a particular "attack pattern"; the critical state description acts as "attack pattern aggregator", by grouping with a single critical state description all the different attacks patterns (known or unknown) that could lead the system into the target critical state. Using this approach it is possible to detect zero-days attacks (i.e. attack patterns not yet discovered) aiming at driving the SCADA system into ***known*** *system critical states*.

7.2.4 Multidimensional metric for CS

In this section, we present a way of predicting whether the system is leading to a critical state. The method is based on the notion of distance from critical states, capturing the concept of critical state proximity. Predicting criticality can be achieved by tracking changes of the distance between the current system state and the critical formulas. The virtual system image described in **??** is used to track the current system state values, and the distance is calculated using these values. The distance notion is parametric with respect to a metric on the system state space. Let $d\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$ be any metric on $\mathbb{R}^n$. In other words, let d be any notion of distance between two system states. In this work two distances are of particular interest:

$$d_1(s,t) = \sum_{i=1}^{n} |s_i - t_i|$$
$$d_v(s,t) = \#\{i \mid s_i \neq t_i\}$$

The distance d_1 is also known in literature as the *Manhattan* distance. The distance d_v counts the number of system components whose values differ among two states. In Example **??**, d_1 computes how close the water temperatures and the fan speed are to critical values. Let $s = (80, 4000)$ be a critical state. Let $u = (40, 4000)$ and $v = (70, 4000)$ be two states. Values $d_1(u,s) = 40$ and $d_1(v,s) = 10$ indicate that the state v is *closer* to s than u, i.e. u is *more secure* than v. Instead, the distance values $d_v(u,s) = 1$ and $d_v(v,s) = 1$ indicate that u and v are equally distant from s. Indeed only one system component (the temperature sensor) has a different value from the state s in both the states u and v. The actual choice of d_1 or d_v depends on the notion of the criticality that has to be captured. When only the number of critical system components matters, the distance d_v is more appropriate. When the actual value of system components is important in order to establish the criticality of the system, then the distance d_1 is more appropriate.

State-Critical States Distance

Given *any* distance function on $\mathbb{R}^n$ (e.g. as d_1 or d_v defined in Section **??**), the notion of distance between a state and a *set of states* can be defined as $d(s, S) = \inf_{t \in S} d(s,t)$[1].

This definition mimics the common sense of distance between a point and a collection of points. The notion of distance between a system state and the set of critical states is defined by $d(s, CS)$. It is crucial to stress that it is completely parametric with respect to the metrics chosen on $\mathbb{R}^n$.

Distance Evaluation

In the following the evaluation of the distance $d(s, CS)$ defined in Section 7.2.4 is presented. It is based on the representation of critical formulas based on interval constraints.

[1]As standard in literature, the expression inf A, where $A \subseteq \mathbb{R}$ is a set of real numbers, denotes the greatest lower bound of A, i.e. inf $A = \max\{x \in \mathbb{R} \mid \forall y \in A.\, x \leqslant y\}$. Moreover, $\inf_{t \in S} d(s,t) = \inf\{d(s,t) \mid t \in S\}$ for any given $s \in S$.

Computing the distance does not follow directly from its definition. For an efficient implementation it is necessary to take advantage of the shape of the set of critical states. The language of critical conditions implies that, for each rule condition, the critical values for every component belong to intervals (bound or unbound) of real numbers. This information is used for computing the distance efficiently as follows.

An interval constraint $C = I_1, \ldots, I_n$ is a sequence of n intervals on $\mathbb{R}$, where n is the number of system components. A constraint specifies a *critical range* for each system component value. A system state s is critical w.r.t. a constraint C if and only if for each $i = 1 \ldots n$, the i-th system component value $s_i \in I_i$. Every critical formula ϕ can be represented as one or more interval constraints. A set of constraints $\{C_1, \ldots C_k\}$ is equivalent to the formula ϕ if for every state s satisfying ϕ there exists at least one constraint C_j such that s is critical w.r.t. C_j. Considering the Example 1, let $\phi = \mathrm{PLC}[10.0.2.31:502].\mathrm{HR}[1] \neq 50$ be a critical formula. It is not possible to find an interval constraint equivalent to ϕ. However, let $C_1 = [-\infty, 49], [-\infty, +\infty]$ and $C_2 = [51, +\infty], [-\infty, +\infty]$ be two constraints. The set of constraints $\{C_1, C_2\}$ is equivalent to ϕ, indeed any state satisfying ϕ also satisfies C_1 *or* C_2, and vice versa.

The notion of equivalent set of constraints is used as the basis for implementing a feasible memory representation of critical formulas. It is possible to scan a critical formula ϕ in order to easily build an equivalent set of constraints. This is done only once during the initialization phase. Considering the Example showed in figure 7.4, let

$$\phi = \begin{pmatrix} \mathrm{PLC}[10.0.2.31{:}502].\mathrm{HR}[1] \geq 60, \\ \mathrm{PLC}[10.0.2.45{:}502].\mathrm{HR}[1] < 5000, \\ \mathrm{PLC}[10.0.2.31{:}502].\mathrm{HR}[1] < 100 \end{pmatrix}$$

Scanning the formula ϕ allows to collect the critical ranges for each system component identifier, and compute the final set of constraints equivalent to ϕ. In this case, the set of equivalent constraints contains only the constraint $C = [60, 100], [-\infty, 5000]$.

Let $\{C_1, \ldots, C_k\}$ be a set of constraints equivalent to a critical formula ϕ. The following equations hold:

$$d(s, \phi) = \min_{i=1}^{k} d(s, C_j) \tag{7.1}$$

$$d(s, C = I_1 \ldots I_n) = \sum_{i=1}^{n} d(s_i, I_i) \tag{7.2}$$

The distance $d(s, CS) = \min_{\phi} d(s, \phi)$ can be calculated using Equation (7.3) and (7.2), implemented with nested iterations on the constraints computed in the initialization phase and on the intervals of each constraint. Time complexity is linear in the number of predicates occurring in the critical formulas.

Equation (7.2) is parametric w.r.t. the actual distance used. Precisely, the function $d(s_i, I_i)$ in the right hand side of the equation stands for both d_1 and d_v. The following definitions allow to easily implement the distance calculation algorithm:

$$d_1(x, I) = \begin{cases} x - \sup I & \sup I \leq x \\ \inf I - x & \inf I \geq x \\ 0 & \text{otherwise} \end{cases} \qquad d_v(x, I) = \begin{cases} 1 & x \notin I \\ 0 & x \in I \end{cases}$$

Where $\inf I$ and $\sup I$ are respectively the lowest and the highest endpoints of the interval I. Summarizing, in order to calculate the distance $d(s, CS)$ between a state and a set of critical states, using the *Manhattan distance* d_1 or the *discrete distance* d_v on $\mathbb{R}$, it is sufficient to calculate a set of interval constraints equivalent to each critical formula during the initialization phase. During the evolution of the system, both the criticality of the current state and the distance from the critical states can be calculated implementing Equations (7.3) and (7.2). The actual choice of the d_v or d_1 depends on the chosen notion of distance.

7.2.5 Threshold detection

In the previous section we have introduced a multidimensional metric providing a parametric measure of the distance between a given state and the set of critical states. This metric can be used for tracking the evolution of a system, indicating its proximity to the set of predefined critical states. The concept of proximity is useful for understanding how close is a state to a critical state, but it is not sufficient. Let's consider the example where a scada system monitor a digital value and a single register respectively a valve and a temperature. The critical formula ϕ defying the critical state cs and a set of system state showed in table 7.2

$$\phi_1 = \begin{pmatrix} \text{PLC[10.0.0.1:502].HR[2]} > 100, \\ \text{PLC[10.0.0.3:502].CO[1]} = 0 \end{pmatrix}$$

State	10.0.0.1	10.0.0.3	Distance(d_1)
s_1	500	1	1
s_2	450	1	1
s_3	60	0	40
s_4	81	0	19
s_5	75	0	25

Table 7.2: Distance from the current state C_1 and the Critical State(CS_1)

The formula ϕ defines a critical scenario where the temperature is more than 100degrees and the drain valve is close. In this system the temperature could reach high value only if the drain valve that let the gas go out is open, otherwise the system risks to explode due to an increasing of the pressure. The states s_1 and s_2 are at distance 1 from the Critical state CS_1 this is basically correct in fact the value of the coil could potentially change from 1 to 0 in a really short timing. This happen because as described in the previous chapter everyone in the network even a potential attacker can send a simple command and change the coil value. The states s_3, s_4 and s_5 show an opposite situation where the drain valve is closed and the temperature is under the limit defined in the cs. The Manhattan distance calculated using the formula d_1 shows how the state s_4 is the closest to the critical state. This distance however doesn't take in consideration the behavior of the system. The temperature of 81 degrees reached in s_4 can refer either to a normal or anomalous behavior of the system. In order to detect which state is potentially anomalous compared with cs we introduce the concept of $Threshold$ calculated through a learning phase.

SCADA system as claimed before control and monitor industrial process that are less dynamic than normal network system. More precisely due to the fact that are monitored physical variable each industrial system is characterize by a similar behavior over a fixed amount of time. This would permit us to calculate a threshold thorough a learning phase for each Critical Formula defined. As claimed before the values of the process parameters, such as measurement and actuator control data, are classified in two types: logical and numerical data. The logical data has two values, namely ON/OFF that are already well considered in the distance calculation. The numerical data represents continuous measurement data (such as motor speed and temperature values). Therefore, the evolution of numerical values are the most significant in order to calculate a proper threshold. The threshold of the critical state cs is performed than as the lower Manhattan distance calculated for each system states reached during the learning phase, if and only if the discrete values satisfying the critical formula. In the previous example the the threshold is set as 19.

Let the critical formula $\phi = \phi_d \cap \phi_c$ where ϕ_d represents the critical formula concerning the digital values and ϕ_c the critical formula regarding continuos values. We can define X_i as a set of state occurred during the learning phase satisfying the critical formula ϕ_d. The threshold is calculated as:

$$Threshold = \inf_{s \in X} d_1(s, \phi) \tag{7.3}$$

Considering the following example as the results of a learning phase, where the physical process parameters P_1, P_2 and T_1, T_2 where P_i represents the status of the $Pumps_i$ and the water level in the $Tank_j$.

P_1	P_2	T_1	T_2	Note
0	0	50	50	$State_1$
0	0	25	95	$State_2$
1	0	61	50	$State_3$
1	0	35	60	$State_4$
1	0	80	60	$State_5$
$P_1 = 1$	$P_2 = 0$	$T_1 > 85$	$T_2 > 90$	Critical(ϕ)

Table 7.3: Evolution of the system during the learning phase

The $State_3$, $State_4$ and $State_5$ are included in our sub-set X due to the fact that the critical formula ϕ_d is satisfied. Considering the definition presented above the threshold concerning the critical formula ϕ is calculated by perform the distance d_1 all the state included in X. The minimum distance is set as the threshold.

P_1	P_2	T_1	T_2	X	d_1
1	0	61	50	$State_3$	64
1	0	35	60	$State_4$	80
1	0	80	60	$State_5$	35

Table 7.4: Distance d_1 for each state included in X

In the next chapter we presents some example where the calculation of the threshold through a learning phase permits us to reach interesting results.

7.3 IDS Implementation

We present in this section a detailed explanation about how the *"State Based Intrusion Detection System"* works. The IDS prototype has been implemented in C# (MS.NET framework version 3.5) in an MS-Windows environment. From the architectural point of view, it was implemented according to a modular approach. There are five big modules:

- Loader
- Scada Protocol Sensor (SPS)
- System Virtual Image (SVI)
- Analyzer

Each module is divided into many sub-modules with different functions. In what follow we will explain in details all the modules and sub-modules and their functions. The Figure 7.5 shows a diagram of the IDS architecture.

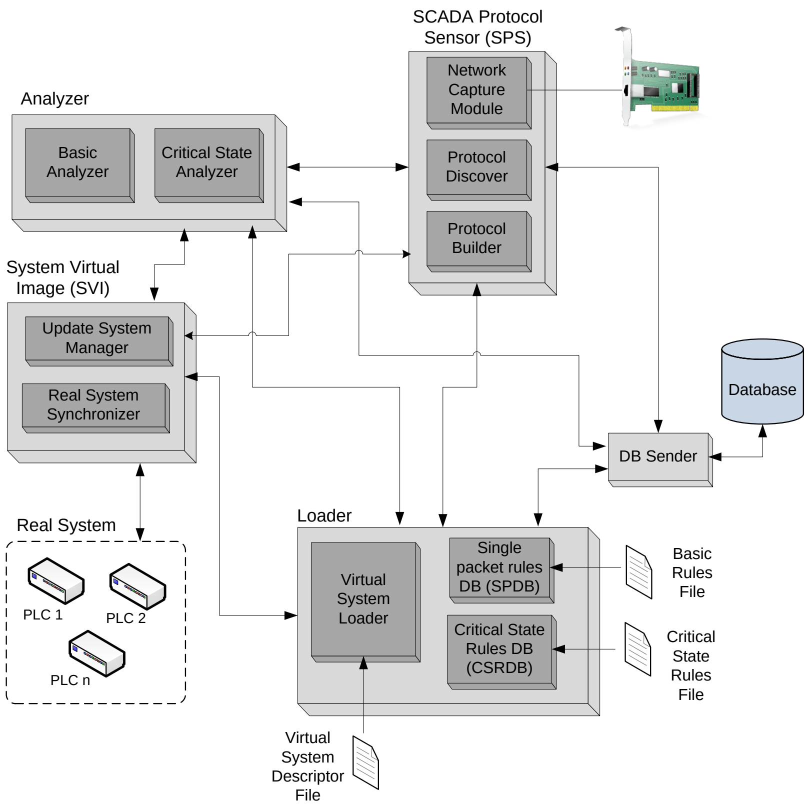


Figure 7.5: "*State Based IDS*" Architecture

We have already shown some aspects of this architecture, but in this section we will explain in detail how they were implemented.

7.3.1 Loader

The loader module is in charge to read the XML configuration files and to initialize the whole system.

Virtual System Loader The "*Virtual System Loader*" reads the XML Virtual System File Descriptor and creates the data structure to represent the SCADA system. The XML file is written by the IDS administrator following the SSML language rules (See Section **??**) and it describes the architecture of the real system. The IDS administrator can write the XML file manually or using an IDS tool called “*Critical Infrastructure Creator*”. The structure of the XML Virtual System File Descriptor is described by the following XML Schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="infrastructure">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="master" type="masterType"/>
          <xsd:element name="plc" type="plcType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="masterType">
    <xsd:attribute name="address" type="xsd:string"/>
    <xsd:attribute name="port" type="xsd:unsignedInt"/>
  </xsd:complexType>
  <xsd:complexType name="plcType">
    <xsd:attribute name="address" type="xsd:string"/>
    <xsd:attribute name="port" type="xsd:unsignedInt"/>
    <xsd:attribute name="id" type="xsd:unsignedByte"/>
    <xsd:sequence>
      <xsd:element name="discrete_inputs" type="coilType"/>
      <xsd:element name="coils" type="coilType"/>
      <xsd:element name="input_registers" type="regType"/>
      <xsd:element name="holding_registers" type="regType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="regType">
    <xsd:attribute name="numbers" type="xsd:unsignedShort"/>
    <xsd:attribute name="def_value" type="xsd:unsignedShort"/>
  </xsd:complexType>
  <xsd:complexType name="coilType">
    <xsd:attribute name="numbers" type="xsd:unsignedShort"/>
    <xsd:attribute name="def_value" type="xsd:boolean"/>
  </xsd:complexType>
</xsd:schema>
```

The XML Schema contains a root node called ”infrastructure”.The root node contains a sequence of sub-nodes, each sub-node could be a “master” or a “plc” node. A “master node” has two attributes, address and port, while a “plc node” has address, port, id and four child nodes, i.e. discrete inputs, coils, input registers and holding registers. In order to better understand, let’s consider the example in Figure 7.6.

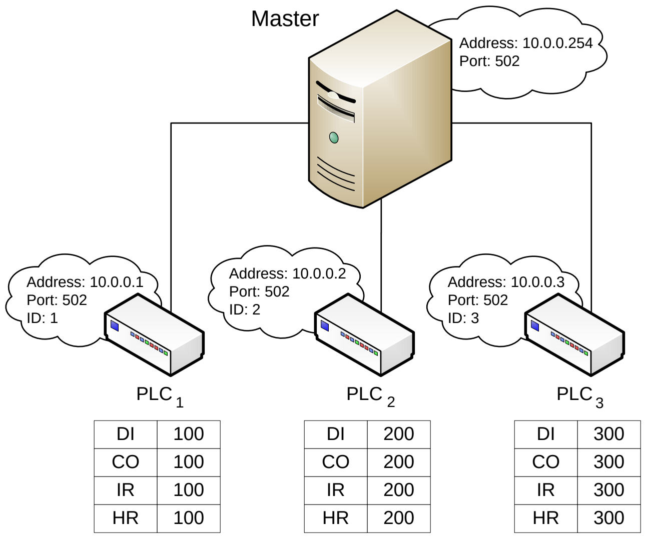


Figure 7.6: MODBUS SCADA System Example

The XML Virtual System File Descriptor for such scenario is the following:

```
<infrastructure>
  <master address="10.0.0.254" port="502" />
  <plc address="10.0.0.1" port="502" id="1" >
    <discrete_inputs numbers="100" def_value="false" />
    ...
    <holding_registers numbers="100" def_value="0"/>
  </plc>
  <plc address="10.0.0.2" port="502" id="2" >
    <discrete_inputs numbers="200" def_value="false" />
    ...
    <holding_registers numbers="200" def_value="0" />
  </plc>
  <plc address="10.0.0.3" port="502" id="3" >
```

```
    <discrete_inputs numbers="300" def_value="false" />
    ...
    <holding_registers numbers="300" def_value="0" />
  </plc>
</infrastructure>
```

The *Virtual System Loader* is an XML parser, which takes in input an XML file and creates the data structure which represents the "*Virtual System Image*". For Example the system presented above generates the following data structure (Figure 7.7). The previous XML Virtual

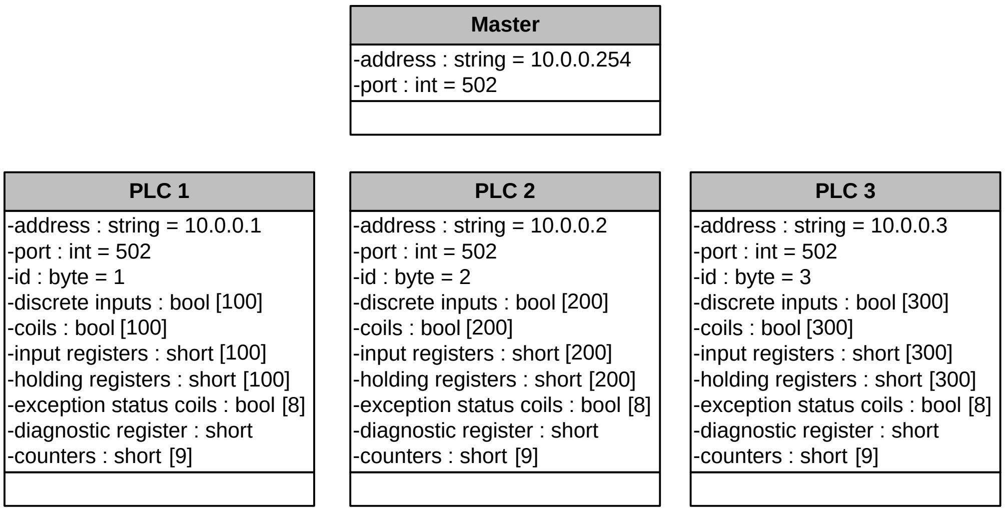

Master
-address : string = 10.0.0.254
-port : int = 502
PLC 1
-address : string = 10.0.0.1
-port : int = 502
-id : byte = 1
-discrete inputs : bool [100]
-coils : bool [100]
-input registers : short [100]
-holding registers : short [100]
-exception status coils : bool [8]
-diagnostic register : short
-counters : short [9]
PLC 2
-address : string = 10.0.0.2
-port : int = 502
-id : byte = 2
-discrete inputs : bool [200]
-coils : bool [200]
-input registers : short [200]
-holding registers : short [200]
-exception status coils : bool [8]
-diagnostic register : short
-counters : short [9]
PLC 3
-address : string = 10.0.0.3
-port : int = 502
-id : byte = 3
-discrete inputs : bool [300]
-coils : bool [300]
-input registers : short [300]
-holding registers : short [300]
-exception status coils : bool [8]
-diagnostic register : short
-counters : short [9]

Figure 7.7: SVI Stored in IDS Memory

System File Descriptor generates a Virtual System with one "master object" and three "plc objects". The master class has only two attributes: address and port, while the plc class has address, port, id and four arrays to store the values of discrete inputs, coils, input registers and holding registers. Moreover the plc class contains arrays with pre-fixed size for exception status coils, counters and the diagnostic register attribute.

Critical State Rule Loader The "*Critical State Rule Loader*" reads the Critical State Rules File and creates the data structure to store the rules in IDS memory. As specified in Chapter 7 a critical state rule is a boolean expression such as:

$$\text{PLC}[10.0.0.2:502].\text{CO}[1] == 0 \rightarrow Alert:3$$

The data structure used to represent boolean expression in memory is a sort of "*Binary Decision Tree (BDT)*". In such kind of BDT there are two types of nodes:

- **Boolean Operator Node:** it represents a boolean operator such as AND, NOT.
- **Condition Node:** it represents a condition like "PLC[10.0.0.2].CO[1] == 0" expressed whit the ISML (See Section 7.2.1).

The data structure is a binary tree in which the internal nodes are only "*Boolean Operator Node*" and the leaves are only "*Condition Node*". The UML diagram in Figure 7.8 shows the classes involved in the "*Binary Decision Tree (BDT)*" implementation.

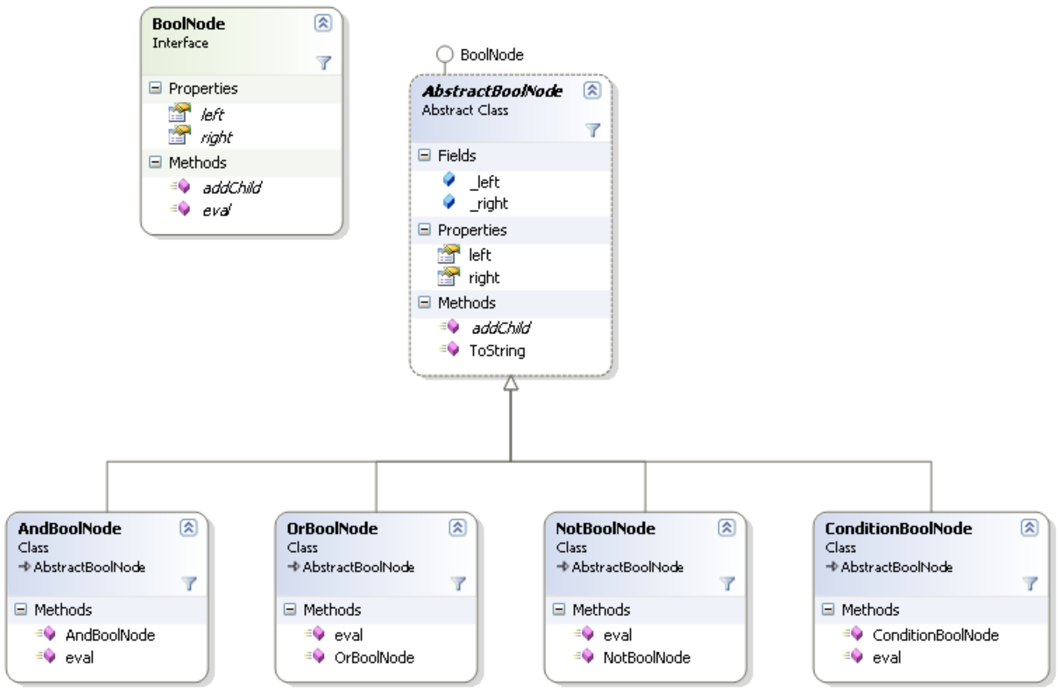


Figure 7.8: The "*Binary Decision Tree (BDT)*" data structure UML diagram

BoolNode: this interface represents a general description of a tree node with a left and a right child and two methods: “addChild” to add a child to the node and “eval” for the evaluation of the node, i.e. it returns the boolean value of the node (true or false). Each node in the tree has to implement the interface “*Bool Node*”.

AbstractBoolNode: this abstract class contains fields, properties and methods common to each kind of node. For example every node has a left and right child and the methods “addChild” and “ToString” are common method for any type of node.

AndBoolNode: this class represent the logic operator AND. The most important method in this class is the “eval” for the evaluation of the and node.

The pseudo-code of such method is:

```
bool eval()
{
    bool v1 = left.eval();
    if (!v1)
        return false;
    else
    {
        bool v2 = right.eval();
        if (!v2)
            return false;
        else
            return true;
    }
}
```

The method recursively calls the eval function on the left child. If the value returned by the left child is false, is useless to check the right child and the function return false. If the value returned by the left child is true, the method recursively calls the eval function on the right child. If the value returned is false the method return false, otherwise the method return true. **NotBoolNode:** this class represent the logic operator NOT. The “eval” method is very simple, in fact the pseudo-code is:

```
bool eval()
{
    if (left!=null)
```

```
        return !left.eval();
    else
        throw new Exception();
}
```

ConditionBoolNode: this class represent a condition expressed with the ISML. The eval method is completely different:

```
bool eval()
{
    bool ret;
    switch (condition_type)
    {
        case equal:
            ret = (reg_val == comp_val);
        break;
        case lt:
            ret = (reg_val < comp_val);
        break;
        case gt:
            ret = (reg_val > comp_val);
        break;
        case lte:
            ret = (reg_val <= comp_val);
        break;
        case gte:
            ret = (reg_val >= comp_val);
        break;
        case neq:
            ret = (reg_val != comp_val);
        break;
    }
    return ret;
}
```

The method, depending on the type of condition, performs a comparison rather than another one between the value in the registry and the value to be compared. The "*Binary Decision Tree (BDT)*" data structure, which we have already presented is used to store in memory each rule in the Critical State Rules File.

7.3.2 Scada Protocol Sensor (SPS)

The SCADA Protocol Sensor (SPS) is basically the equivalent of a normal network IDS sensor, with the advantage of being able to directly support commonly used SCADA protocols.

Network Capture Module It receives the mirror of all the traffic to and from the field network directly from the SPAN port of the field network switch. The traffic capture is preformed by using the WinPcap [**?**] library. It allows applications to capture network packets bypassing the protocol stack.

Protocol Discover The packets captured by the "Network Capture Module" are analyzed by the "Protocol Discover" which is able to analyze the application level payload of the packet and to recognize if it is a SCADA packet (only MODBUS and DNP are now supported by the IDS) or another packet. If the packet is a SCADA protocol one, the protocol discover will sent it to the "Protocol Builder" otherwise it will be discarded.

Protocol Builder The "Protocol Builder" receives only the SCADA traffic from the "Protocol Discover" and analyzes each packets in order to recognize the packet type and to store the packet information in a the proper object. As we claimed in the MODBUS section **??**, there are two kind of MODBUS packets: requests and responses.

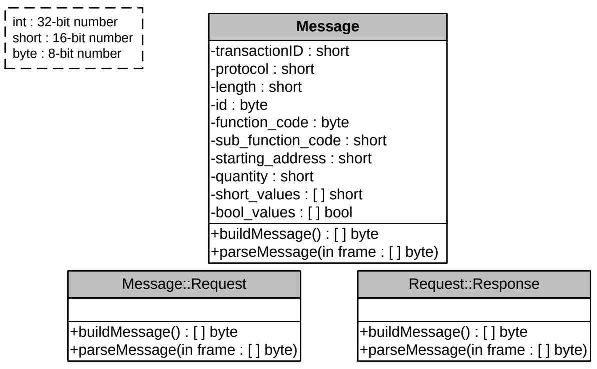

int : 32-bit number
short : 16-bit number
byte : 8-bit number
Message
-transactionID : short
-protocol : short
-length : short
-id : byte
-function_code : byte
-sub_function_code : short
-starting_address : short
-quantity : short
-short_values : [ ] short
-bool_values : [ ] bool
+buildMessage() : [ ] byte
+parseMessage(in frame : [ ] byte)
Message::Request
+buildMessage() : [ ] byte
+parseMessage(in frame : [ ] byte)
Request::Response
+buildMessage() : [ ] byte
+parseMessage(in frame : [ ] byte)

Figure 7.9: MODBUS Messages objects

The Message class contains all the attributes of a MODBUS message, which are common in request and response. The only difference between the two classes Request and Responses are the two methods “buildMessage” and “parseMessage” because each request and response has a different structure. There are also different constructors for every kind of request and response: for example the “Read Coils” request is built with the constructor:

```
public Request(short tID, short protocol, short length, byte id,
byte function_code, short starting_address, short quantity)
```

While the “Read Exception Status Coils” request uses the constructor:

```
public Request(short tID, short protocol, short length, byte id,
byte function_code, short sub_function_code)
```

When the “Protocol Builder” receives a MODBUS packet, it checks if the message is a request or a response, then it checks the function code to parse the message in the correct way. It recognizes the type of message using the TCP source address and the information stored in the System Virtual Image (SVI):

- If the TCP source address is the address of a Master Station (according to the SVI information) then the message is a request.
- If the TCP source address is the address of a Slave Station (according to the SVI information) then the message is a response.

Moreover, the “Protocol Builder” checks the message function code and parse the Request or the Response in the right way.

7.3.3 System Virtual Image (SVI)

The System Virtual Image (SVI) contains the representation of the state of the system under analysis. We have already explained in Section 7.2 how the SVI is build and how it works ,therefore we will not enter into details.

Real System Synchronizer The SVI is updated by periodically querying the real system. The “Real System Synchronizer” contains a SCADA Master simulator which is able to forge MODBUS requests and to parse the responses, then to update the SVI with the information contained in the responses. In the XML Virtual System File Descriptor is possible to specify, for each PLC the update time, e.g. each five seconds. The “Real System Synchronizer” queries the PLC every 5 seconds, asking all the discrete inputs, coils, input registers, holding register, diagnostic register, exception status coils and counters values. Each query is optimized to store much information as possible. For example, if a PLC has 4000 coils, the “Real System Synchronizer” will perform two request: the first for the coils from 0 to 2000 (maximum value allowed for a “Read Coils” request according to the modbus specification [**?**]) and the second from 2000 to 4000.

Update System Manager The “Update System Manager” is composed by two buffer and three thread. There are a Request and a Response buffers and the following three thread:

- **Add Packet Thread:** it receives the request or response packets from the “Protocol Builder” and put them into the right buffer (request or response buffer).

- **System Update and Thread:** it extracts the first request in the “*RequestBuffer*” and tries to find the corresponding response in the “*ResponseBuffer*”; if it has found the response then it updates the System Virtual Image (SVI) and checks if the system is in a critical state.

- **Check Critical States Thread:** it checks periodically if the SVI match with one or more critical states in the Critical State Rules DB (CSRDB).

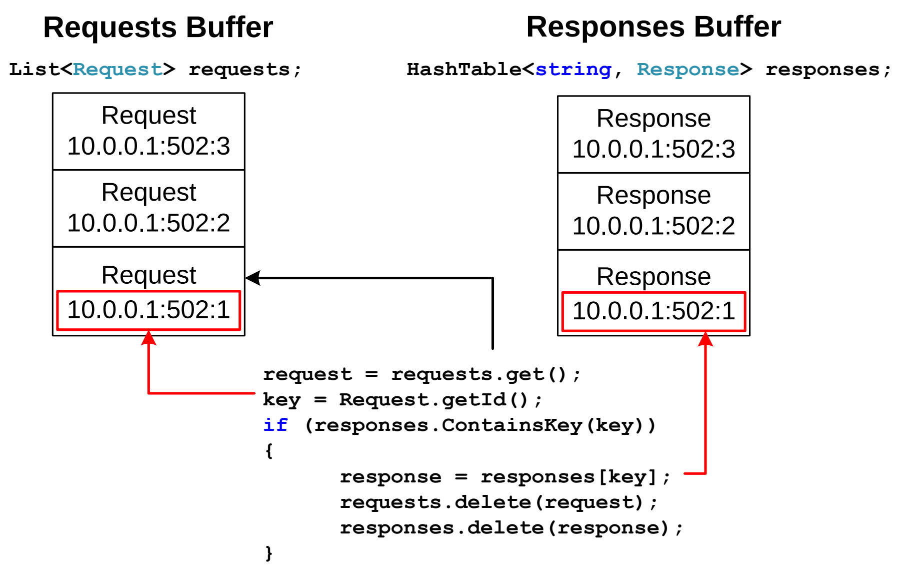
The “RequestBuffer” is a simple list of request objects, while the “ResponseBuffer” is an hash table that maps the response object to associated keys. The keys used to uniquely identify a response object are strings composed by:

```
key = source_address + ":" + source_port + ":" + transaction_id;
```

For example a response key could be:

```
key = "10.0.0.1:502:99";
```

The transaction id is not enough to uniquely identify a response object because the IDS captures the traffic from different PLCs, so it is possible that two PLCs are using the same transaction id at the same time. The “*Add Packet Thread*” is in charge only to puts the incoming packets in the right buffer so it is really simple, but the “*System Update*” and the “*Check Critical States*” threads need a detailed explanation;. For example take into consideration the scenario in Figure 7.10



```
List<Request> requests;
HashTable<string, Response> responses;

request = requests.get();
key = Request.getId();
if (responses.ContainsKey(key))
{
        response = responses[key];
        requests.delete(request);
        responses.delete(response);
}
```

Figure 7.10: Request/Response Buffers

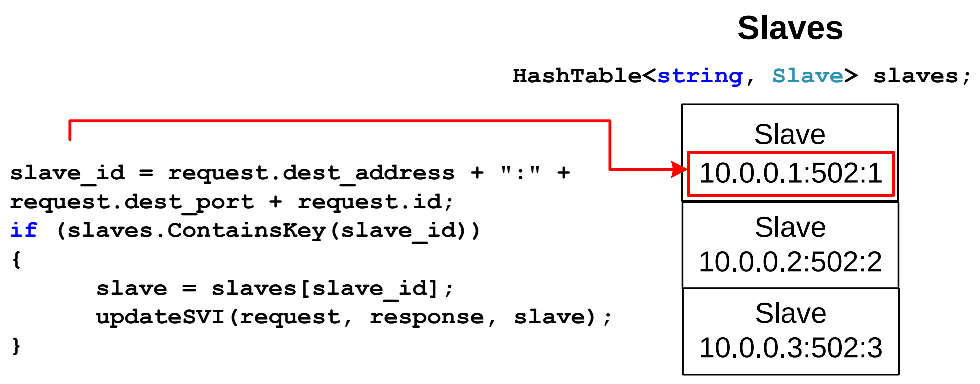
In the example above, when the "*System Update Thread*" takes control there are three packets in the RequestBuffer and three in the ResponseBuffer. The thread gets the first packet from the RequestBuffer and reads the key calling the method getId(). Such method return the following key:

```
key = dest_address + ":" + dest_port + ":" + transaction_id;
```

The key is build in a different way than in the ResponseBuffer because the destination address in a request packet is equal to the source address in the corresponding response packet. When the "*System Update Thread*" has obtained the key it checks if the ResponseBuffer contains such key. If it is true the thread gets the proper response and delete the two messages form their respective buffers.
Moreover the "*System Update Thread*" has to find and update the MODBUS slave or PLC involved in the request/response transaction. The list of the MODBUS slaves is stored again ad an hash table. The keys used to uniquely identify a slave object are strings composed by:

```
key = slave_address + ":" + slave_port + ":" + slave_id;
```

The "*System Update Thread*" finds and updates the slave object as shown in the example in Figure 7.11



```
slave_id = request.dest_address + ":" +
request.dest_port + request.id;
if (slaves.ContainsKey(slave_id))
{
        slave = slaves[slave_id];
        updateSVI(request, response, slave);
}
```

Figure 7.11: Slave Update

The "*System Update Thread*" builds the key of the slave involved in the request/response transaction using the following information contained in the request packet:

```
key = request.dest_address + ":" + request.dest_port + request.id;
```

When the "*System Update Thread*" has built the slave key it checks if the Slave hash table contains such key. If it is true the thread gets the proper slave object and executes the updateSVI and the checkCriticalStates methods. The updateSVI method uses the information in the request/response objects to update the SVI. A part of the pseudo-code of this methods is shown below.

```
void updateSVI(Request req, Response res, ModbusSlave slave)
{
    switch (res.function_code)
    {
        case read_coils:
            short st = req.starting_address;
            for (int i = 0; i < res.bool_values.Length; i++)
                slave.coil[i + st] = res.bool_values[i];
        break;
        case read_discrete_inputs:
            ...
        break;
        ...
    }
}
```

The method reads the function code contained in the response object and executes the correct part of code. For example if the function code is "01 Read Coils" the method reads the starting address and the quantity of coils from the request and the values to updates from the response then perform the proper update into the target slave. The "*Check Critical States Thread*" is a part of the Analyzer Module, therefore it will be described in the following.

7.3.4 Analyzer

As discussed in the previous Section, the "*Analyzer*" performs the "*Signature Based Detection*" and the "*State Based Detection*".

Basic Analyzer The '*'Basic Analyzer*" is very similar to a normal NIDS analyzer. It receives in input the captured packets, then it analyzes the packet and if it match with a signature in the Single packet rules DB (SPDB) it raises an alert.

Critical State Analyzer The "*Crtical State Analyzer*" checks periodically the Virtual System, if the SVI status match with a "Critical State" contained in the Critical State Rules DB (CSRDB) it raises an alert. The "*Crtical State Analyzer*" is an object which contains an array of rules. Each element in this array is a reference to the root node of a "*Binary Decision Tree (BDT)*". The most important method in the "*Crtical State Analyzer*" is the checkCriticalStates which control if SVI status matches with the critical state rules. The pseudo code of such method is really simple because it uses the recursive function "*eval*" that we have already explained in section 7.3.1.

```
void checkCriticalStates()
{
    for (int i = 0; i < rules.Count; i++)
        if (rules[i].eval())
            raiseAlert();
}
```

The method simply calls the eval function on each rule in the rules array. If the eval function returns true it means that the system is in a critical state, so the method raises an alert.

Chapter 8

Experimental results

The entire work presented in this thesis was developed with a deep interaction with some of the most important European industry. Eni S.p.a. that is an Italian multinational oil and gas company sponsored part of this research work and give us the possibility to use real industrial installation in order to test the solution presented. The European commission and Enel provides its own know-how and a test bed described in the section **??**. Least but not last the University of Twente in collaboration with Brabant Water one of the public water company in Netherlands, provides real SCADA network traffic about production installation.

In this section several comparative tests on a prototype implementing the presented approach are described. The tests were performed using both real SCADA traffic and a dedicated laboratory able to simulate a real SCADA installation.

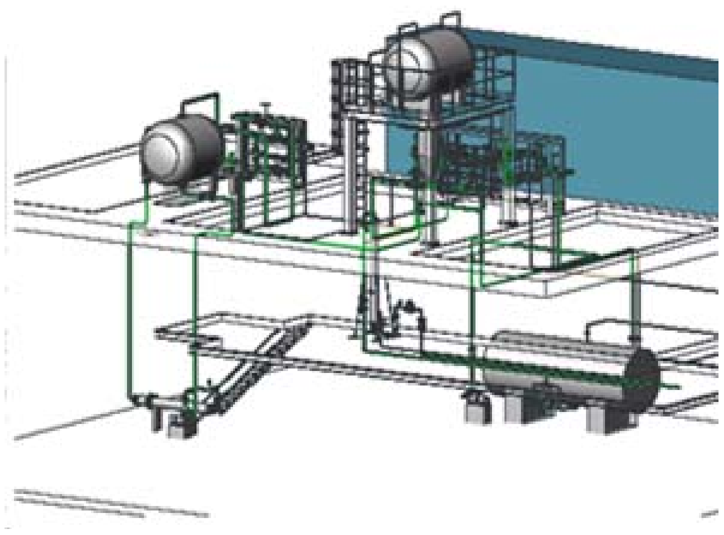


Figure 8.1: Power plant simulator

This laboratory was created thanks to a cooperative research activity between the Research Department of ENEL SPA and the Joint Research Centre of the European Commission. In this protected environment, a complex electromechanical device consisting of pipes, valves, sensors, pumps etc. is used to physically emulate the different states of a real power plant. For the following tests, we have used the Modbus over TCP protocol and PLCs of the ABB AC800 family. Figure 8.1 shows a sketch of the physical electromechanical device used for the simulations, while a detailed description of the experimental environment can be found in [52]. A 3-layer switch is configured with a set of monitor ports where the PLCs are connected. The IDS is linked to the monitor port and it runs on a Windows machine with an AMD Athlon 2.8 GHz processor and 4 GB of memory.

8.1 Boiling Water Reactor Scenario

The *Boiling Water Reactor (BWR) Scenario* in Figure 8.2 shows a nuclear reactor for generating electrical power. The BWR in figure is deliberately simplified. The Reactor Pressure Vessel contains the fuel assemblies and control rods absorbed in light water. The fuel elements heat the water to produce steam. The steam reaches the main turbine through the steam line and makes it rotate. The unused steam is driven to the condenser where it is condensed into water. The resulting water is pumped back to the reactor vessel. The cooling water is maintained at about 75 atm (7.6 MPa, 10001100 psi) so that it boils in the core at about 285 C (550 F).

The PLCs are programmed with the following logic:

PLC 1: it controls the temperature sensor of the water in the vessel and the speed of $Pump_2$. The former values is hold in the register IR_1 and the latter in HR_1. PLC_1 increases the $Pump_2$ speed if the temperature is too high in order to augment the cold water flow and to reduce the vessel temperature.

PLC 2: it controls the pressure sensor connected to IR_2 and the control valve V_{OUT} connected to CO_24, which contains a value representing the valve opening (0 closed - 1 open). When the pressure is too high, PLC_2 opens the valve V_{OUT} to eject steam and to reduce the internal pressure.

PLC 3: it controls the temperature sensor of the condenser connected to IR_3 and the speed of $Pump_1$ linked to HR_3. As PLC_1, when the condenser temperature it too high, PLC_3 increases $Pump_1$ speed in order to augment the cold water flow and to condense the steam.

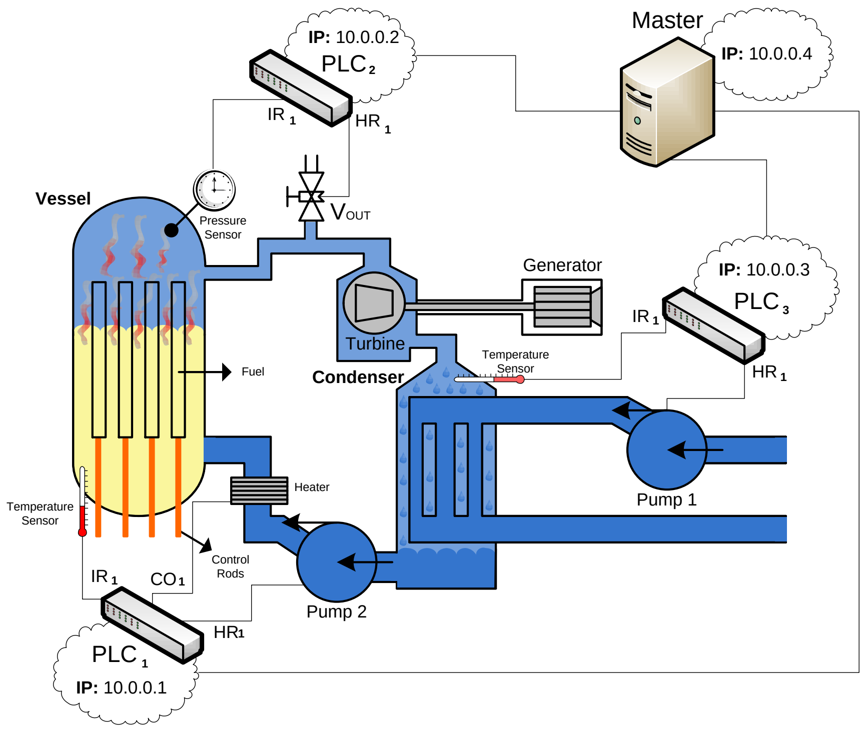


Figure 8.2: Boiling Water Reactor Schema

The formal description of the system using the ISML language is a set of predicates specifying the state of the system, e.g. $Sys =$

$$\{\mathrm{PLC}[10.0.0.1:502].\mathrm{IR}[1] = 200, \dots, \mathrm{PLC}[10.0.0.1:502].\mathrm{HR}[1] = 200,$$
$$\dots \qquad \dots$$
$$\mathrm{PLC}[10.0.0.3:502].\mathrm{IR}[1] = 100, \dots, \mathrm{PLC}[10.0.0.3:502].\mathrm{HR}[1] = 300\}$$

where the values on the right side of the assignments are the initial values of the system, automatically re-synchronized with the real system values thanks to the master emulator.

In the following a set of Critical State Rules representing some possible critical states of the BWR scenario is showed.

$$\begin{pmatrix} PLC[1].IR[1] > 120, \\ PLC[1].HR[1] < 40 \end{pmatrix} \rightarrow Alert:4 \tag{R1}$$

$$\begin{pmatrix} PLC[2].IR[2] > 200, \\ PLC[2].HR[9] < 10 \end{pmatrix} \rightarrow Alert : 2 \tag{R2}$$

$$\begin{pmatrix} PLC[3].IR[3] > 100, \\ PLC[3].HR[3] < 60 \end{pmatrix} \rightarrow Alert : 2 \tag{R3}$$

$$\begin{pmatrix} PLC[2].IR[1] < 70, \\ PLC[1].CO[1] = 0, \\ PLC[1].HR[1] < 1500 \end{pmatrix} \rightarrow Alert : 1 \tag{R4}$$

The rules state the following:

- *R1* - If the temperature is > 120 degree and $Pump_2$ speed is < 40 revolutions per second, then the system is in a critical state because the water temperature is high, but the pump speed is not enough to reduce the temperature in the vessel.

- *R2* - If the pressure is > 200 bar and the valve is close, then the system is in a critical state because the pressure is high, but the valve is not open to eject steam and reduce the internal pressure.

- *R3* - If the temperature is > 100 degree and $Pump_1$ speed is < 60 revolutions per second, then the system is in a critical state because the steam temperature is high, but the pump speed is not enough to reduce the condenser temperature.

- *R4* - If the pressure is under $70atm$, the pre-heater system is turned off and the $Pump_2$ speed is > 1500 revolutions per second the system is in a critical state. In fact in a boiling water reactor, the pressure is created to keep the water boils at the temperature of 285 C. So if the pressure is under the limit and the pump is pumping too much cold water in the system the standard fuel circle in the vessel could can be compromised.

Distance Example: In the following an example of the distance calculation is shown. Consider rule (R1) and assume that PLC[1].IR[1] is mapped to s_1 and PLC[1].HR[1] to s_2. The equivalent set of constraints is the singleton $\{C = [120, +\infty], [-\infty, 40]\}$. Let $s = (100, 45)$ be the current state. The distance between s and the formula is calculated as follows:

$$\begin{aligned} d_1(s_1, [120, +\infty]) &= \inf I_x - s_1 = 120 - 100 = 20 \\ d_1(s_2, [-\infty, 40]) &= s_2 - \sup I_y = 45 - 40 = 5 \\ d_1(s, C) &= \sum_{I_i \in C} d_1(s_i, I_i) = 20 + 5 = 25 \end{aligned}$$

In this example the set of constraints equivalent to R1 contains only one constraint. If the set of constraint has more constraints, the same calculation has to be repeated for every constraint and the distance value is the minimum among the computed values.

Threshold example: Consider the critical states cs_2 and cs_4 defined respectively by the rules (R2) and (R4). Consider also the 7 states occurred during the learning phase showed in the table 8.1

State	$PLC[2].CO[1]$	$PLC[1].CO[1]$	$PLC[2].IR[2]$	$PLC[1].HR[1]$	$d_1(s_i, cs_2)$	$d_1(s_i, cs_4)$
s_1	0	0	80	1100	20	410
s_2	1	0	81	1100	20	511
s_3	0	0	85	900	15	615
s_4	0	0	75	1200	25	305
s_5	0	1	90	400	10	1121
s_6	0	0	91	550	9	971
s_7	1	1	110	600	1	41

Table 8.1: Boiling Water Reactor learning phase

In the last column the Manhattan distance(d_1) between the critical states cs_2, cs_4 and the current state s_i is calculated. Following the definition presented in the section the sub sets X are $X_cs2 = \{s_1, s_3, s_4, s_5, s_6\}$ and $X_cs2 = \{s_1, s_2, s_3, s_4, s_6\}$. For each critical states cs_2 and cs_4 the threshold results respectively:

$$\begin{aligned} Threshold_{cs2} &= 9 \\ Threshold_{cs4} &= 305 \end{aligned}$$

This example show how for each critical state is necessary to define a dedicated threshold based on the behavior of the system.

8.2 Accuracy Analysis

One of the most relevant parameters to be taken into consideration when evaluating an IDS is accuracy, i.e. how well a binary classification test correctly identifies or excludes a condition (in our case the occurrence of a critical state). In the scientific literature of Intrusion Detection, accuracy is commonly measured in terms of False Positives and False Negatives, i.e. considering how many false alerts are raised and how many attacks are not identified. To measure the accuracy of the IDS, we set up the following experiment: a dataset was created by collecting private traffic data in our laboratory for fifteen days. The dataset is made of standard SCADA traffic reflecting normal industrial activities, plus traffic generated by simulating random malicious attacks targeting critical states. The proposed approach is intended as an additional feature to be added to existing IDS, to detect a particular class of attacks against SCADA systems. For that reason, we evaluated its detection accuracy against such family of attacks, i.e. the attacks composed of chains of licit SCADA commands. It is worth noting that the traffic congestion performance affects the accuracy of the IDS. As stressed before, in cases of high network congestion, the virtual system image might be slightly different from the current system state due to packet loss. When that happens, critical state rules are evaluated against a not fully consistent system state, resulting in false positive or false negative alerts. To capture this aspect, we randomly injected bursts of traffic activity with high bandwidth rates.

Table 8.2 provides a clear picture of the number of true and false alarms generated per day. In this example, it is obvious that the true alar highly outnumber the false alarms. Approximately 99% of alerts generated are true positives, while less than 1% of the total alerts are false positives. We remark that accuracy here refers to the specific class of attacks (those composed by licit chains of SCADA commands driving the system into a critical state), for which the critical state base approach has been designed.

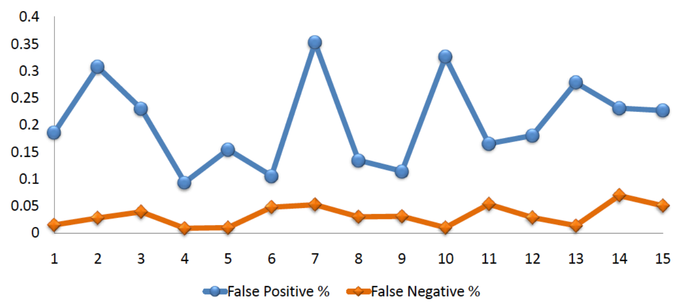


Figure 8.3: Day by day false positive and negative results

Day	Exp. Alerts	False Pos	False Neg	Raised Alerts
1	19872	37 (0.186%)	3 (0.015%)	19906
2	14326	44 (0.307%)	4 (0.028%)	14366
3	17823	41 (0.230%)	7 (0.039%)	17857
4	22457	21 (0.094%)	2 (0.009%)	22476
5	20046	31 (0.155%)	2 (0.010%)	20075
6	18875	20 (0.106%)	9 (0.048%)	18886
7	13351	47 (0.352%)	7 (0.052%)	13391
8	20041	27 (0.135%)	6 (0.030%)	20062
9	22736	26 (0.114%)	7 (0.031%)	22755
10	9824	32 (0.326%)	1 (0.010%)	9855
11	18743	31 (0.165%)	10 (0.053%)	18764
12	24387	44 (0.180%)	7 (0.029%)	24424
13	14728	41 (0.278%)	2 (0.014%)	14767
14	12987	30 (0.231%)	9 (0.069%)	13008
15	19832	45 (0.227%)	10 (0.050%)	19867

Table 8.2: False positive and negative accuracy results.

By observing the system behavior with the CS approach without trying to follow the attacker behavior, a false positive/negative might happen in the case of de-synchronization of the virtual system with the real system. A relevant role is then played by the Master Emulator embedded into the IDS: the faster its synchronization querying time, the lower the risk of de-synchronization of the virtual system. In this respect the previous tests can be taken as a measure of the robustness against false positives/negatives caused by system de-synchronization. On the other side, a faster synchronization query time possibly means higher interference with the monitored system. The trade-off between synchronization query time and system interference is strongly system dependent. Parameters to be considered in defining the proper query time are: (a) the system architecture (e.g. if the system is composed of redundant network connections, the backup lines can be used by the IDS for querying purposes); (b) the system hardware (e.g. the computational and network power of the PLC); and (c) the real time requirements. The evaluation of the correct query time can be considered as part of the usual "system tuning" of every IDS. The right tradeoff between synchronization query time and system interference can be determined on the basis of ad-hoc experimentation.

8.3 Performance Tests

This section describes the tests carried out with a configuration we have denominated "four sub-systems scenario" (FSS). This scenario was implemented for measuring the time delays affecting the IDS. The FSS is composed of four masters connected to 16 PLCs configured with at least 100 different analog and digital IO.

Memory Usage Tests

The evaluation of memory performance is shown in the following. There are two data structures which have a considerable size in the CS IDS: the *Virtual System Image* and the *Rules Representation*. The *Virtual System* Data Structure is a hash table identifying each PLCs with a unique key. The amount of memory required for each PLC object increases linearly with the number of registers into the PLC. The required memory for the entire *Virtual System* increases linearly with the number of PLC in the system. Table 8.3 shows the memory usage for each PLC and Table 8.4 shows the memory usage for a *Virtual System* containing PLCs composed of 65535 registers (maximum value allowed according to the Modbus specification [**?**]).

Registers Num	Size (K/b)
1	0.422
10	1.056
100	1.583
1000	6.856
10000	59.591
30000	176.778
65535	384.991

Table 8.3: PLC Memory Usage

PLCs Num	Size (M/b)
1	0.378
2	0.754
10	3.762
50	18.800
100	37.599
500	187.986
1000	375.970

Table 8.4: VS Memory Usage

The growth is linear and a *Virtual System* composed of 1000 PLCs requires less than 400 Mbytes of memory. The other important data structure is used to represent the "*Signature Based and Critical State Based Rules*". This data structure is a list of lists where each sub-list represents a rule. The required memory for each rule increases linearly with the number of conditions in the rule, and the required memory for the entire rule set increases linearly with the number of rules.

Conditions Num	Size (K/b)
1	0.086
10	0.859
100	8.594
500	42.969
1000	85.938
2000	171.875

Table 8.5: Single Rule Memory Usage

Rules	Size (K/b)
1	0.344
10	3.438
100	34.375
500	171.875
1000	343.750
2000	687.500

Table 8.6: Rules Set Memory Usage

Table 8.5 shows the memory usage for each rule and Table 8.6 shows the memory usage for an entire set of rules (from 1 to 2000 rules, each rule composed of 4 conditions). For both, i.e. single rule and set of rules, the growth is linear and a set of 2000 rules requires less than 600 Kbytes.

Packet Capturing Test

The "*Packet Capturing*" performance of the prototype was tested sending a high number of packets with a very high bit rate. The results are shown in Table 8.7.

Avg. Mbit/s	Avg. Pkts. Captured (out of 400K)	Avg. Pkt. Loss	Avg. Pkt. Loss (%)	Avg. Alerts Raised	Avg. Alert Loss	Avg. Alert Loss (%)
1.215	400000	0	0.0000	200000	0	0.0000
1.476	399994.8	5.2	0.0013	199996.2	3.8	0.0019
1.712	399987.6	12.4	0.0031	199988.8	11.2	0.0056
2.022	399978	22	0.0055	199980.2	19.8	0.0099
2.405	399970.8	29.2	0.0073	199973.8	26.2	0.0131
2.77	399968.4	31.6	0.0079	199968.4	31.6	0.0158

Table 8.7: Packet Capturing and Alerts Raised Tests

In this scenario the IDS is stressed with bursts of 400.000 consecutive packets (with increasing bit-rate). In our tests, the behavior of the IDS appeared reliable, since in the worst case (traffic of 2,77 Mbit/sec) it only lost 0.0079 % of the packets and 0.0158 % of the alerts (in table 8.7 we showed only the alert missed due to packet loss). The packet loss and the alert loss increased with the quantity of traffic as shown in the two charts in figure 8.4. When the traffic rate is under 1.215 Mbit /sec there is no packet loss and when the traffic rate is high (over 2 Mbit/sec) the percentage of packet loss is less than 0.008 %.

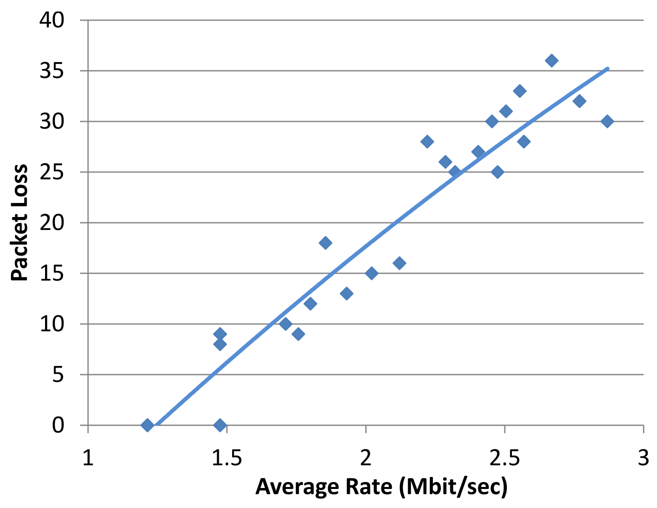


Figure 8.4: Packet Capturing and Alerts Raised Tests

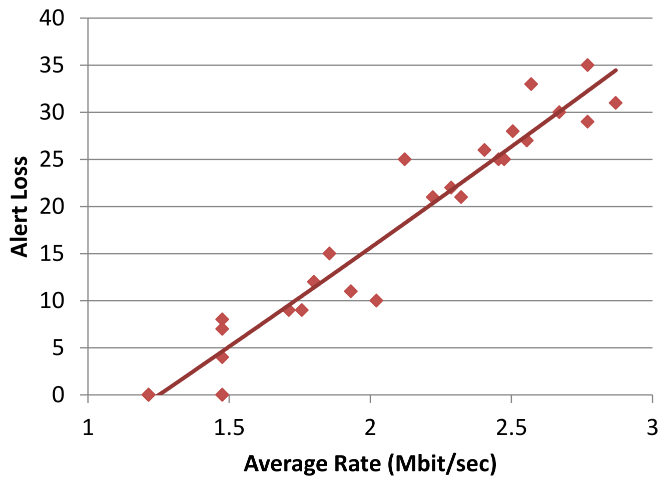


Figure 8.5: Packet Capturing and Alerts Raised Tests

Signature-Based Analyzer Test

The performance of the Signature-Based Analyzer depends on the quantity of rules. In order to test the rule check performance the following experiment has been carried out: the Master Station sends 1000 generic requests and the slave Station responds with the appropriate responses, the IDS captures the messages and checks whether they are licit, according to a rules file containing n rules.

After checking the rules, the IDS can either display a warning on the standard output, or (in a more complex scenario than the one presented above) send the alert to the DB. The results are shown in Figure 8.4 8.5..

Cond.	Avg Time	Avg Time Comm.
10	0.006	0.009
50	0.004	0.012
100	0.006	0.177
250	0.023	0.417
500	0.049	0.572
1000	0.096	0.709
2000	0.194	0.829

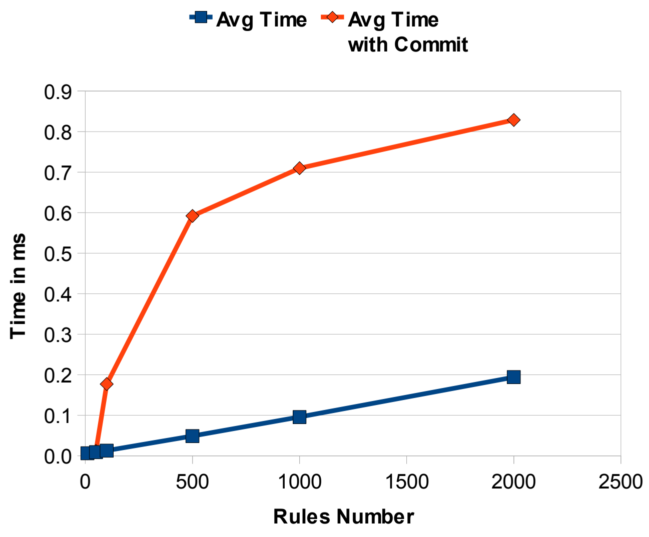


Figure 8.6: Signature-Based Performance Test (average time in ms.)

The elapsed time increases with the quantity of rules as shown in Graph 8.6.

Virtual System Update Tests

The IDS updates the System Virtual Image (SVI) in two steps: it finds the PLC related to the content of the packet and then it updates the Virtual Object which represents the PLC. The first step is not relevant for the performance of the IDS, because the list of PLCs is implemented with a hash table, so the time elapsed to find a PLC (around 0,0042 ms) is almost the same for tables of 1 or of 1000

PLCs. The following test was used for checking the time needed for updating the PLC information: the Master Station sends 1000 requests with the command "Read n coils", the Slave Station answers with responses that contains the n values and the IDS captures the request/response transactions and updates the n values in the Virtual PLC.

N. Coils	Avg. Time
1	0.003
50	0.005
100	0.006
500	0.019
1000	0.034
2000	0.068

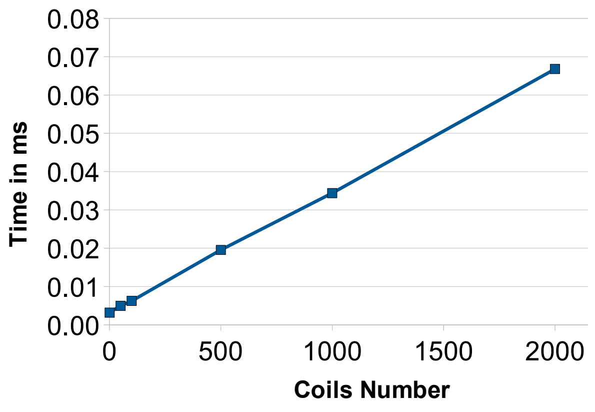


Figure 8.7: Virtual System Update Performance Test and graph (Time in ms.)

The results demonstrate the validity of the proposed approach because even in the worst case, i.e. 2000 coils to update (maximum value allowed according to the Modbus specification [**?**]), the IDS performance is under one millisecond. In addition, the elapsed time increases with the number of coils to update in a linear way as shown in Graph 8.7.

Critical State Rules Analyzer Test

The performance of the Critical State Rules Analyzer depends on two factors: the size of each rule and the quantity of rules. The rule size impact was measured as follows: the Master Station sends 1000 generic requests and the Slave Station answers with the appropriate responses, the IDS captures the request/response transactions and checks whether the Virtual System is entering into a Critical State, according to a rules file that contains only one rule with n conditions (in this case

2 conditions) The impact of the number of rules on the IDS performance was tested increasing the number of rules while keeping fixed the size (2 conditions) of each single rule (results in Table 8.9).

N. Cond.	Avg. Time
2	0.00035
4	0.0004
32	0.0062
128	0.031
256	0.0601
512	0.113
1024	0.246
2048	0.554

Table 8.8: Critical State Rule Analyzer Performance based on condition number

N. Rules	Avg. Time
10	0.0038
50	0.0149
100	0.026
500	0.123
1000	0.329
2000	0.766

Table 8.9: Critical State Rule Analyzer Performance based on number of rules (time in ms.)

It is important to note that the time elapsed for a number of conditions up to 2048 is under one millisecond. This result is satisfactory since it is unlikely to have rules with 2048 conditions. In addition, these tests show that the bottleneck for the IDS performance is the Critical State Rules Analyzer, especially when the quantity of rules is high. In any case the time elapsed is under one millisecond with a number of rules up to 2000.

Distance Performance Test In light of the described implementation, we performed the following tests:

1. **Predicates Test:** the rule set is composed of only one critical formula which has a variable number of predicates (up to 2000) related to the same system component.

2. **System Components Test:** the rule set is composed of only one critical formula which has a variable number of predicates (up to 2000) related to different system components.

3. **Rules Test:** the rule set is composed of a variable number rules (up to 2000).

The results are shown in Table 8.10 for the three parts of the test.

All tests confirm that the time required for calculating the distance is linear with the number of predicates. In test 1, the time elapsed for calculating the distance (Table 8.10.A) is negligible, considering also that a list of 2000 predicates for a single system component is improbable. In test 2, the time spent for calculating the distance (Table 8.10.B) is lower than 6ms which is a good result. In test 3, the time spent for calculating rule distances (Table 8.10.C) is relatively high, but the growth is linear and the maximum time is less than 60ms for a number of rules up to 2000.

Table 8.10.A

# Pred.	Time (ms)
1	0.010
10	0.013
50	0.019
100	0.021
500	0.056
1000	0.093
2000	0.165

Table 8.10.B

# Pred.	Time (ms)
1	0.013
10	0.037
50	0.147
100	0.270
500	1.308
1000	2.547
2000	5.147

Table 8.10.C

# Rules	Time (ms)
1	0.040
10	0.277
50	1.332
100	2.632
500	13.542
1000	28.967
2000	57.386

Table 8.10: Distance Analyzer Performance Test

Chapter 9

Conclusion

The connection of industrial systems to the public network has introduced new security problems in an environment traditionally critical, and ICT security countermeasures are not able to completely protect such systems. This thesis presented a new network detection approach for the detection and mitigation of a particular class of cyberattacks against industrial installations. This technique is based on monitoring the evolution of the state of the protected system and on the analysis of the command packets between master and slaves of a SCADA architecture. The key elements of this technique are the concept of critical state and the observation that an attacker, in order to damage an industrial system, will have to modify its state from secure to critical. The critical state validation, normally hardly applicable in traditional ICT systems, finds its natural application in the industrial control field, where the critical states are generally well-known and limited in number. Moreover, the introduction of the concept of critical state distance allowed to extend the firewall features in the direction of a more complete early warning system.

The results of the tests conducted on a prototype implementing the described approach demonstrated the feasibility and validity of the proposed method. This approach presents some advantages with respect to traditional filtering techniques:

1. Since the network detection system is applied on the basis of the system evolution (something known) and not on the basis of the attack evolution (something unknown), for predefined critical states, this approach allows to block also zero day attacks, i.e., attacks based on unknown techniques.

2. The number of false positives results limited since the traffic is dropped only if the analyzed command will drive the system into a described critical state. There are only two cases in which we can have false positives or false negatives: the case in which a critical state has not been described (and this is an error performed by who configured the intrusion detection rules) or if the real system and its virtual image are desynchronized (and this is due eventually to an error in the configuration of the auto- synchronization time between the real system and the virtual system)

On the other hand, this technique, being conceived to protect strictly the SCADA devices, cannot protect from more traditional ICT attacks such as virus attacks to general purpose ICT systems, etc. For that reason, we see the critical state-based filtering as a technique complementary to the traditional firewall and IDS techniques, helping in enhancing the security of these systems. The configuration of the ruleset is not cheap in term of effort. However, to facilitate this process, we are planning to develop a self-discovery engine able to automatically learn the configuration of the system to be protected.

Bibliography

[1] COMMISSION OF THE EUROPEAN COMMUNITIES Communication form the commission to the council and the European parliament Critical Infrastructure Protection in the fight against terrorism - Brussels, 20.10.2004 COM(2004) 702 final

[2] Nuclear Regulatory Commission. NRC information notice http://www.nrc.gov/ reading-rm/doc-collections/gen-comm/info-notices/2007/ in200715.pdf (cited: November 23, 2012)

[3] Gorman S. Electricity grid in U.S. penetrated by spies The Wall Street Journal. URL: http://online.wsj.com/article/ SB123914805204099085.html; (cited: November 23, 2012)

[4] Gordon Clarke Practical Modern SCADA Protocols - DNP3, IEC 60870.5 and Related Systems

[5] Eric Knapp Industrial Network Security - Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems

[6] OSIsoft OSIsoft company overview http://www.osisoft.com/ 2012 (cited: November 23, 2012).

[7] Creery A., Byres E. *Industrial Cybersecurity for power system and SCADA net- works.* IEEE Industry Application Magazine (July-August 2007)

[8] Chandia R., Gonzalez J., Kilpatrick T., Papa M., Shenoi S. Security Strategies for Scada Networks. First Annual IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection, Dartmouth College, Hanover, New Hampshire, USA, March 19-21 (2007)

[9] N. Falliere, L. O. Murchu, and E. Chien W32.stuxnet dossier Symantec, Tech. Rep., Feb. 2011. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/securityresponse/whitepapers/w32 stuxnet dossier.pdf

[10] A. Nicholson, S. Webber, S. Dyer, T. Patel, H. Janicke SCADA security in the light of Cyber-Warfare Computers & Security - Volume 31, Issue 4, June 2012, Pages 418436

[11] Giovanni Cagalaban, Seoksoo Kim Towards Improving SCADA Control Systems Security with Vulnerability Analysis Parallel and Distributed Computing and Networks Communications in Computer and Information Science Volume 137, 2011, pp 27-32

[12] Majdalawieh M., Parisi-Presicce F., Wijesekera D. Distributed Network Protocol Security (DNPSec) security framework. Proceedings of the 21st Annual Computer Security Applications Conference, Tucson, Arizona, December 5-9 (2005)

[13] Bagaria, S.; Prabhakar, S.B.; Saquib, Z. Flexi-DNP3: Flexible distributed network protocol version 3 (DNP3) for SCADA security International Conference on recent Trends in Information Systems (ReTIS), 2011

[14] Kang, D.; Kim, H. A Proposal for Key Policy of Symmetric Encryption Application to cyber security of KEPCO SCADA Network Future Generation Communication and Networking (FGCN 2007)

[15] Hong, S.; Phuong, T. N.; Lee M. Development of Smart Devices for Secure Communication in the SCADA system

[16] Azuwa, M.P.; Ahmad, R.; Sahib, S.; Shamsuddin, S. A propose technical security metrics model for SCADA systems

[17] Mander, T.; Nabhani T.; Wang, L.; Cheung, R. Data object based security for DNP3 over TCP/IP for increased utility commercial aspects security In Proceedings Power Eng. Soc. Gen. Meeting, Tampa, FL, Jun. 2428, 2007, pp. 18.

[18] Nai Fovino, I.;j Masera M.; Leszczyna, R.; ICT security assessment of a power plant, a case study in Proceedings 2nd Int. Conf. Critical Infrastructure Protect., Arlington, VA, Mar. 2008

[19] Online- Available: $http : //www.snort.org/assets/114/Snort_RH5_SCADA.pdf$ last access 07/10/2012

[20] Modbus-IDA Modbus Application Protocol Specification v1.1b http://www.modbus.org, November 12, 2012

[21] Modbus-IDA MODBUS Messaging on TCP/IP Implementation Guide V1.0b http://www.modbus.org, November 19, 2012

[22] T. Smith The Register. Hacker jailed for revenge sewage attacks http://www.theregister.co.uk/2001/10/31/hacker_jailed_for_revenge_sewage/ October 31, 2001 (cited: November 3, 2012)

[23] J. Meserve CNN.com. Sources: Staged cyber attack reveals vulnerability in power grid http://articles.cnn.com/2007-09-26/us/power.at.risk_1_generator-cyber-attack-electric-infrastructure., September 26, 2007 (cited: November 3, 2012)

[24] Industrial Control Systems Cyber Emergency Response Team (ICS-CERT) ICSA-10-238-01STUXNET MALWARE MITIGATION Department of Homeland Security, US-CERT, Washington, DC, August 26, 2010.

[25] E. Chien, Symantec. Stuxnet: a breakthrough. ,http://www.symantec.com/connect/blogs/stuxnet-breakthrough November 2010 (cited: November 16, 2012)

[26] Slay, S.; Miller, M.; Lessons Learned from the Maroochy Water Breach Springer, 2008, vol. 253, Critical Infrastructure Protection, pp. 7382, IFIP International Federation for Information Processing.

[27] Gross, P.; Parekh, J.; Kaiser, G. Secure selecticast for collaborative intrusion detection systems In Proceedings at International Workshop on DEBS, 2004.

[28] Ning, P.; Cui, Y.; Reeves, D. S.; Constructing attack scenarios through correlation of intrusion alerts In Proceedings at ACM Conference Comput. Commun. Security, Washington, D.C., Nov. 2002, pp. 245254.

[29] Cuppens, F.; Mige, A.; Alert correlation in a cooperative intrusion detection framework In Proceeding 2002 IEEE Symp. Security and Privacy, Washington, DC, 2002, p. 202.

[30] Isermann, R.; Supervision, fault-detection and fault-diagnosis methods. An introduction, Control Engineering Practice, vol. 5, no. 5, pp. 639652, 1997.

[31] Isermann, R.; Process fault detection based on modelling and estimation methodsA survey Automatica, vol. 20, no. 4, pp. 12871298, 1984.

[32] Frank, P. M.; Advanced fault detection and isolation schemes using non linear and robust observers In 10th IFAC Congress, Munich, Germany, 1987, vol. 3, pp. 6368.

[33] Dominguez, C.; Vidulich, M.; Vogel, E.; McMillan, G.; Situation Awareness: Papers and Annotated Bibliography, Armstrong Laboratory, Human System Center, ref. AL/CF-TR-1994-0085, 1994.

[34] Roman, R.; Alcaraz, C.; Lopez, J.; The role of wireless sensor net- works in the area of critical information infrastructure protection Inf. Secur. Tech. Rep., vol. 12, no. 1, pp. 2431, 2007.

[35] Linda, O.; Vollmer, T. ; Manic, M. Neural Network based Intrusion Detection System for critical infrastructures International Joint Conference on Neural Networks, 2009. IJCNN 2009.

[36] Bonnie Zhu, Anthony Joseph, and Shankar Sastry Taxonomy of Cyber Attacks on SCADA Systems Proceedings of CPSCom 2011: The 4th IEEE International Conference on Cyber, Physical and Social Computing, Dalian, China, October 19-22, 2011

[37] Eric Byres, Joel Carter, Amr Elramly, Dan Hoffman Worlds in Collision: Ethernet on the Plant Floor ISA Emerging Technologies Conference, Instrumentation Systems and Automation Society, Chicago, October (2002).

[38] Andrea Carcano, Igor Nai Fovino, Marcelo Masera, Alberto Trombetta SCADA malware, a proof of concept LNCS lecture notes in Computer Science, Springer Verlag 2008

[39] P.J. Pingree The Deep Impact Test Benches 8211 - Two Spacecraft, Twice the Fun Proceedings of IEEE Aerospace Conference, Page 19, 2006

[40] NIST - National Institute of Standard Technology CVE-2011-0406 http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-0406

[41] T. Paukatong SCADA Security: A New Concerning Issue of an In-house EGAT-SCADA 2005 IEEE/PES Transmission and Distribution Conference - Exhibition: Asia and Pacific Dalian, China

[42] Igor Nai Fovino, Marcelo Masera, R. Leszczyna ICT Security Assessment of a Power Plant, a Case Study In: Proceeding of the Second Annual IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection, George Manson University, Arlington, USA 2008

[43] Rafal Leszczyna, Igor Nai Fovino, Marcelo Masera MAlSim. Mobile Agent Malware Simulator In Proceeding of the First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems, Marseille (2008)

[44] Rafal Leszczyna, Igor Nai Fovino, Marcelo Masera Simulating Malware with MAlSim In Proceeding of the 17th EICAR Annual Conference 2008, Laval, France (2008)

[45] M.Masera,I.NaiFovino,R.Leszczyna Security assessmentof a turbo-gas power plant in: M. Papa, S. Shenoi (Eds.), Critical Infrastructure Protection II, Springer, Boston, Massachusetts, 2008, pp. 3140.

[46] R. Rivest, A. Shamir and L. Adleman A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, vol. 21(2), pp. 120126, 1978.

[47] NIST - National Institute of Standard Technologies Guide to Intrusion Detection and Prevention Systems Special Publication 800-94

[48] D. Barbara, C. Domeniconi and J. Rogers, Detecting outliers using transduction and statistical testing ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Philadelphia, PA, Aug. 2003.

[49] J. Ma and S. Perkins Online novelty detection on temporal sequences ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Washington, DC, Aug. 2003.

[50] A. Ihler, J. Hutchins, and P. Smyth Adaptive event detection with time-varying Poisson processes ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), Philadelphia, PA, Aug. 2006.

[51] I. Nai Fovino, A. Carcano, M. Masera and T. de Lacheze Murel Modbus/DNP3 State-based Intrusion Detection System In Proceeding of ANIA 2010 - the IEEE 24^{th} International Conference on Advanced Information Networking and Application, Perth, Australia, April 2010

[52] Igor Nai Fovino, Marcelo Masera, Luca Guidi and Giorgio Carpi An Experimental Platform for Assessing SCADA Vulnerabilities and Countermeasures in Power Plants Conference on Human System Interactions (HSI), 2010