UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Simulating attacks to evaluate Intrusion Detection Systems in Industrial Control Systems

Dissertation presented by
Gorby Nicolas KABASELE NDONDA , Syméon MALENGREAU

for obtaining the Master's degree in
Computer Science

Supervisor(s)
Ramin SADRE

Reader(s)
Ramin SADRE, Olivier PEREIRA , Benjamin HESMANS

Academic year 2015-2016

Contents

Chapter 1

Introduction

1.1 Context

In the domain of computer security we have seen a dramatical increase over the past year in the number of attacks against Industrial Control Systems and their networking infrastructure. Industrial Control System are systems used in industrial facilities such as power plants, oil factory, water treatment... Historically, ICS were isolated systems, they were not connected to other networks. Gradually this isolation got tore down as ICS got connected to corporate network to optimize the business process. This increase in connectivity had as a consequence to bring traditional IP network vulnerabilities to ICS networks leading to this increase in the number of attacks against ICS. In response to this, researchers have turned their attention to the development of protection system, such as Intrusion Detection Systems (IDS) targeting this kind of environments. Because of the nature of these networks, getting traffic from real ICS is a difficult task. Indeed, due to security concerns, operators and companies are not keen to release traces and logs from their systems. In consequence of what evaluating IDS becomes a hard task in the context of ICS. In order to perform a proper evaluation, IDS have to be tested against real trace including attack trace.

1.2 Goal

In this context, the goal of this thesis is to create synthetic traces as realistic as possible for various types of attack in Industrial Control System. These generated traffic traces will be then used to evaluate Intrusion Detection System. To achieve this goal, we can therefore divide it in two subgoals, creating an ICS simulator to generate realistic traces containing attack traffic and then using it to evaluate IDS.

We have created a simulator representing a coal power-plant. The simulator generates MODBUS/TCP traces. We have implemented some attack which we run on the simulator. These traces where then tested on two IDSs, an open source signature based IDS called Snort and an IDS we have implemented ourselves. This IDS uses whitelists to detect malicious packet while Snort identify attacks based on a knowledge base represented as rules. We believe that by testing the traces against IDSs using diametrically opposed approach to detect attacks, result would be more meaningful.

In the end we will see if simulating attacks is possible and if it is a proper way to test IDS in the context of ICS.

1.3 Structure

- In chapter 2, we give description about how Industrial Control System works, the different parts that compose them, the protocols that are used. We also explain some famous attacks targeting ICS and discuss the differences between IT networks and ICS networks and what consequences it holds when it comes to protection.
- In chapter 3, we give theoretical background about Intrusion Detection System, the desirable properties they should have, the different types of IDS and the existing approaches to detect attacks.
- In chapter 4, we describe some SCADA specific IDSs. We also explain how the IDS we have implemented works.
- In chapter 5, we present our simulation, the architecture we have used and how have we tried to make it as realistic as possible. Then we discuss the attacks we have implemented.
- In chapter 6, we explain our methodology for our test and we present the analysis of the results we obtained. After analyzing the results, we will try to answer the main question of our work about the utility and feasibility of attack simulation against IDS in the context of ICS.

Code and implementation desing/choice is available at the following repository.

Chapter 2

Industrial Control System

Before digging into protection system and attack simulation in the context of ICS we have to be able to understand all the different components that compose those systems. This section is dedicated to build the foundation necessary to have a good understanding of ICSs.

The order is as follows, we first explain the historical background of Industrial Control System. Then, we present some attacks that have targeted ICS and show the elements proving that this threat has increased. Next, we explain how ICS are working and the components that interact with each other within these systems. Finally, we focus on the security aspect of ICS and why they are hard to protect.

2.1 History

2.1.1 Background

In the task of describing the history of Industrial Control System (ICS), the first question would be to find a correct definition of what an ICS is. We could define an ICS as follows *Industrial Control Systems (ICS) are command and control networks and systems designed to support industrial processes* [1]. But this definition limits ICS to modern industrial systems. We could consider an informal definition to be any system used to control industrial processes of any matter.

Industrial processes were born with the existence of factory processes in the mid 1800, but we can find regulators (with clocks, oil lamps, water tanks, . . .) in early Greek and Arabic societies. But it's only in the 17th and 18th century that we can find the first automatic system to help in various domains to control furnace, or help to sail ships, etc. The discovery of steam and coal have been at the base of the first industrial revolution with the creation of various industrial facilities. We can consider that it is during this period that we see "modern" industrial facilities appear. Then technology evolved by using electricity, which created more complex industrial control system with the creation of relay logic (or what we will later call PLC)[2].

From this point in time, the next big modification in ICS occurs in the 1990s. At that time ICS networks got to be interconnected with the corporate network. They passed from an isolated, with limited connectivity environment to open systems using network protocols from the corporate network. The goal of this interconnection is to increase efficiency and productivity. The corporation was able to retrieve data for its decision making or simply for storage. But this optimization comes with a high cost and is the root problem of security vulnerability in ICS. By being connected to the corporate network, ICS are also connected to the Internet. Before this point in time, defending an ICS would mean to protect it against physical aggression, network security was provided to obscurity. With this interconnection, the surface of attack regarding ICS networks got expanded such that an attacker does no longer need to physically come attack the

system. He/She can do it from the distance by exploiting known vulnerabilities of the corporate network and then access the ICS through the connection between the two networks[33].

2.1.2 Famous Attacks

We have given a presentation of the "modern" industrial control systems. Those systems work with networks and connected devices centralizing informations (we will explain in details in further section how, concretely, all of this work). We will look at three famous attacks to understand more deeply what can be an attack and what disastrous effects they may have. We have selected three attacks: *stuxnet*, *havex* and *night dragon*.

Stuxnet Stuxnet is a famous malware that was used against iranian industrial system trying to attack the centrifuge of the uranium enrichment facility. The attack was planned by the USA and Israel in their will to stop Iran from trying to fabricate nuclear weapon[39].

The first version of the virus was brought to the iranian nuclear facility with an infected USB key. One of the engineer of the system, by plugging his/her infected key in a computer of the system, contaminated the whole facility. Later Stuxnet was given the capability of self-replication, it was able to make itself appear as a legitimate driver to install itself (using zero-day vulnerabilities of windows operation system)[40].

But how does it work? First let's look at what was the target of the virus: the centrifuges. The centrifuges are an important component in the uranium enrichment facility. Stuxnet had two main attack vectors to destroy those centrifuges: over-speed the centrifuge and over-pressure them. The virus was designed to over pass the *cascade protection* of the centrifuges[40].

Concretely Stuxnet is what we call a *man-in-the-middle* virus. It will wait for the industrial process to reach specific condition and will discard the traffic to send its own traffic to different components. With this attack, the virus creates legitimate traffic, discarding the real traffic[40].

The Stuxnet virus is only targeting particular SCADA configurations, infecting PLCs[39].

What were the consequences of this attack? The virus was detected in 2010 by a Belorussian anti-virus company. The Stuxnet destroyed centrifuges with its attack, in consequence of what iranians had to stop the system to change the centrifuges often, slowing down the whole enrichment process.

Havex The Havex malware was designed as a malware to access information of the ICS. It is "a remote access Trojan that was originally used for general-purpose espionage and evolved into a criminal tool set"[41].

There were three way for the malicious software to be spread. They used phishing techniques, infected version of ICS vendors and sometimes they even provided infected version of ICS software installer[41].

The Havex software, once installed on the system, would perform a scan attack to discover all ICS components (we will explain later in this work how does a scan attack work). The attacker, in the end, would gain access to lots of sensitive information about the attacked ICS[41].

Night Dragon Night dragon (called in this manner because the attack came from China), was a "coordinated covert and targeted cyberattacks have been conducted against global oil, energy, and petrochemical companies."[42].

Mostly companies web-server were attacked and compromised by SQL-injection attacks or by phishing methods. Once the servers were compromised the attackers gained access to username and password such tat he/she was able to communicate with the infected machines.

Lots of sensitive informations about those companies were stolen. If we have selected this attack, it was to show that sometimes threat does not only come from the industrial facilities themselves but the company that manages them.

Comments Although we described some attacks against ICS, we would like to convince the reader about the seriousness of the problem. In 2015, Dell released an annual threat report[30]. The report states:

> SCADA attacks increased from 91,676 in January 2012 to 163,228 in January 2013, and 675,186 in January 2014.

As you can see, there has been a significant rise of cyberattacks on SCADA[1] networks. Actually there is a high chance that the number of attacks could be way more. Indeed, an another report from the Industrial Control System Cyber Emergency Response Team, which is an entity working with the US government to reduce risks in critical infrastructure, explained that approximately half of the incidents against ICS vendors were not reported[31]. The report from Dell also states that the number of attacks should continue to rise:

> This lack of information sharing combined with the vulnerability of industrial machinery due to its advanced age means that we can likely expect more SCADA attacks to occur in the coming months and years.

Compared to traditional attacks which have more of a financial purpose, attacks against SCADA system tends to be more political, so was Stuxnet. Because of the nature of the targeted system, a failure can lead to immeasurable damages way beyond economical ones such as life loss or environmental causality. These elements show that adding security to ICS is extremely important. Even though security comes with some constraints, it is absolutely necessary.

2.2 Basics

In this section we are going to go deeper in the understanding of ICS. This will be a quick compilation of different documents to give a brief and complete overview of all components of an ICS.

2.2.1 First insight

Industrial Control System is a generic term, that points out at every element to control and monitor industrial facilities. They are used in facilities such as power plant, waste control facilities, manufacturing process. In those ICS we can find different kind of control system which are: SCADA (Supervisory Control And Data Acquisition), DCS (Distributed Control System). The one of the key idea behind Industrial Control System is the instrumentation. Instead of having someone manually operating some component of the facility such as valves, the operations are automated through field devices equipped of sensors and actuators directly attached to the said component. These field devices can either be Programmable Logic Controller or Remote Terminal Unit.

2.2.2 SCADA

SCADA systems are highly distributed systems used to control geographically dispersed assets, often scattered over thousands of square kilometers, where centralized data acquisition and control are critical to system operation.[3].

SCADA is the main system that will centralize informations of multiple systems to control the different parameters of it.

[1] We will explain it later.

2.2.3 DCS

DCS are used to control industrial processes[. . .]. DCS are integrated as a control architecture containing a supervisory level of control overseeing multiple, integrated sub-systems that are responsible for controlling the details of a localized process.[3]

Reader may wonder what is the main difference between SCADA systems and DCSs. The main difference is that SCADA systems are more oriented towards the centralization of information while DCS take care of the process being run in the industrial system[5]. To simplify, in DCS system, the process will be run by the system itself (but a monitoring of the system is obviously possible), while on the SCADA system, engineer will have the responsibility, based on the data they have collected to take the different decisions (but still, as for DCS, the reality is not black and white, there exists some automation in SCADA too). The DCS will use mostly PLCs, which they will periodically check the parameters of those PLCs to obtain a feedback.

2.2.4 PLCs

PLC is one of the key component of industrial systems. They are defined as *computer-based solid-state devices that control industrial equipment and processes.[3]*. They are used both in SCADA and DCS. They communicate with the system and they monitor the different variables. PLC usually holds different states in registers that the system can access. PLCs are connected to actuators and sensors which interact with the physical world, the PLC acts as controller for logical operation. There is an other kind of field device called Remote Terminal Units which are now basically the same as the PLCs.

2.2.5 HMI and Control Server

There are two other elements we will refer to in this work which are HMI (Human-Machine Interface) and Control Server. The HMI is responsible of providing in real time the current status of the processes. It gives a graphical representation of the systems to an operator and it allows him/her to interact with different component of the system. It can define settings, control objective and override automatic control operations[3].

A control server is the host that will act as an interface between the HMI and the field devices. It will send command to field devices and process collected data. It is the control server that hosts the SCADA system software.

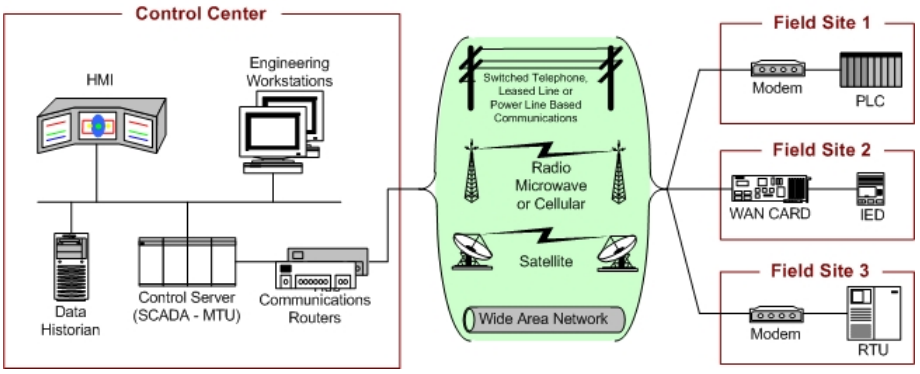

Figure 2.1: Scada system architecture[3]

2.2.6 Modbus/TCP

There exists different protocols in ICS, such as DNP3, MODBUS or PROFIBUS[3]. In this thesis, we will focus on the MODBUS protocol. This protocol has been widely adopted in many

infrastructures because of its simplicity[12].

As the reader will see, the MODBUS protocol does not have any security capabilities (authentication, encryption,...). This is true for most of the SCADA protocols. The explanation is that many of those protocols were designed for serial, LAN communications. The only security they provided was through isolation[13]. Indeed, many protocols for SCADA network have been developed, between 150-200 protocols[35]. Most of them were created by private companies. Now there exists open standard protocols. The requirements for those protocols were mainly performance oriented so that they could cope with the constraints of the network. But as we mentioned earlier, with time, control systems got connected to corporate network for business reasons. With this new connection, communication protocols that were previously used evolved. For example, MODBUS/TCP is an evolution of MODBUS which runs over TCP. Because the isolation got torn down, those systems became vulnerable to typical network attacks.

Architecture MODBUS[2] is an application layer protocol. It is based on client/server architecture. The client sends request to the server and according to the request the server will perform an action and send a reply. Typically, in a SCADA network, the PLCs are the MODBUS server and a master server act as the client. Like many application layer protocol, MODBUS has a reserved system port which is 502.

Message Format The MODBUS protocol specifies three messages:

- MODBUS Request
- MODBUS Response
- MODBUS Exception Response

1-byte function code	n-bytes request/response data

Figure 2.2: MODBUS Request/Response message format

When the client sends a request, it chooses a function code to specify what kind of action it wants the server to perform. Function code takes one byte in a MODBUS message. Upon receiving this request, the server will take the appropriate action and respond back. The response will contain the function code corresponding to the request it replies to.

A subtlety of the MODBUS protocol is that even if 8 bits are used for the function code, the client can only use value between 1 and 127 as function code. Indeed, MODBUS uses value above 127 to warn the client that an error has occurred. If the server can cannot complete a request because of an error, it will send a message containing the function code of the request incremented by 128 and an exception code to specify the cause of the error.

For each of these messages, MODBUS entities prepend an 7 bytes header composed of the following fields:

Transaction Identifier: This field is used by the client to identify each request, this identifier must be unique. The server echoes the transaction identifier in its response, so that the client can match responses to requests even if they do not arrive in the order of request sending.

[2]When we used MODBUS it means MODBUS/TCP

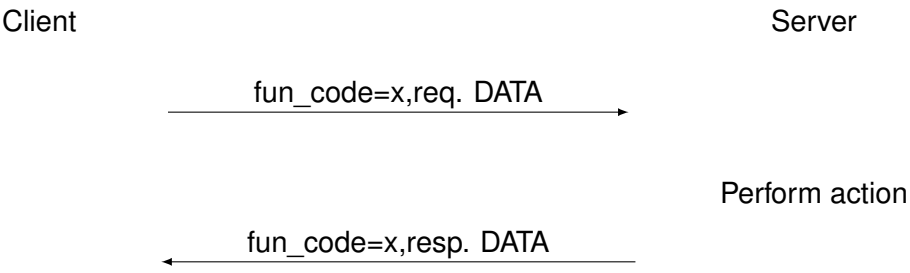

Figure 2.3: Modbus exchange

1-byte exception function code	1-byte exception code

Figure 2.4: MODBUS Exception message format

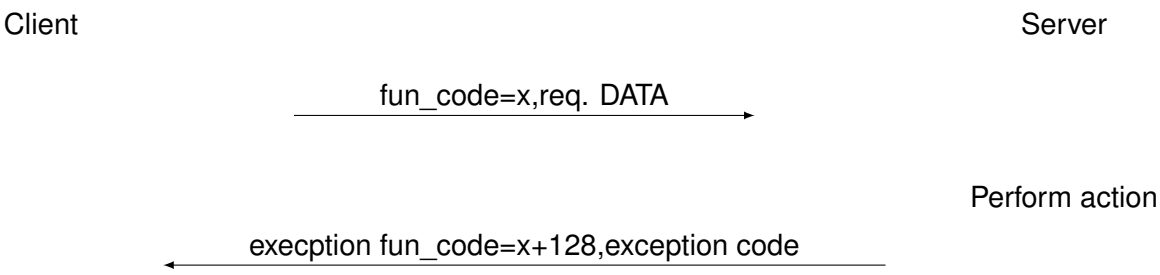

Figure 2.5: MODBUS exception

Protocol Identifier: always 0 for MODBUS (reserved for future extensions)

Message Length: Specifies the number of byte that follows in the message.

Unit Identifier: This field is used to identify a server in a non TCP/IP network. In a TCP/IP network, it is set to 0 or 255 and is ignored and echoed by the server.

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

Figure 2.6: MODBUS Header

Data Modelling MODBUS server keeps different registers. Each of these tables have specific characteristic. Data stores which are in read and write mode can be altered by the client using a

Name	Size	R/W
Discrete input	1-bit register	Read only
Coil	1-bit register	Read and Write
Input Register	16-bit register	Read only
Holding Register	16-bit register	Read and Write

Table 2.1: MODBUS Data Model

function code. The other ones can be altered via an I/O system but not by the client. Depending on the characteristics of the register, the purpose differs. For example, a coil register acts as an ON/OFF switch while an input register can be used to store the temperature of a pool in a power plant.

The implementation of this data modeling is left to the device manufacturer, but the MODBUS specification states that the logical reference numbers are unsigned integer with the index beginning at zero[6]. So, the only requirement is that there is a mapping between the logical reference and the physical address. Each registers are identified by a 16-bit reference.

Function Codes Function codes are at the core of the MODBUS protocol. These functions have different uses, data access, diagnostic, etc. MODBUS distinguishes three categories of function codes, public, user-defined and reserved. Public function codes are well-defined, documented and guaranteed to be unique. This category envelops function code already assigned and function code reserved for future use. User-defined function codes designate functions implemented by the user but not supported by the specification. The last category corresponds to the function code used by some companies that are not available for public use.

Among the function code, we have the following:

Function Code	Description
01	Read Coil
02	Read Discrete Input
03	Read Holding Registers
04	Read Input Registers
05	Write Single Coil
06	Write Single Holding Register
15	Write Multiples Coils
16	Write Multiple Holding Registers

Table 2.2: MODBUS basic function code

2.3 Protection Issues

In early ICS, the networks as we know them nowadays did not exist, such that ICSs were independent working entities. So protection, back in those days, meant mostly to protect infrastructure against physical attack. It is only recently that those systems have been plugged to the traditional IP network. Such that the vulnerabilities inherent to traditional IP networks are now a part of ICS networks. So nowadays, they are exposed to network attacks.

Those systems have particularities that make their protection quite challenging. In this part we are going to expose the difficulties of protection of ICS and some classical attacks scheme that can be used to attack the ICS.

2.3.1 Difficulty of protection

Because of their nature and their use, ICSs have a lot of constraints that make their protection more difficult than a traditional network. Typically, failures are relatively tolerated in traditional network. Sometimes simply rebooting a network component such as a server or a router allows the network to recover from failure. This way of doing things is not possible in an ICS. The cost of failure is way more important in an ICS. We will discuss some of these constraints that make the protection of ICS so hard.

Modification The first, and main difficulty of ICS is the fact that those systems are difficult to modify and upgrade. It is due to multiple factors.

First, those systems cannot be easily stop. Stopping such system can have great consequences. For instance, stopping a nuclear power plant may cause an outage on a city energy network, or stopping a production plant might result in huge loss of money. This makes those systems difficult to test and to upgrade or update. In the end, it often happens that unpatched versions of program, with known vulnerabilities are used in ICS.

The second factor, is the need for compliance. The consequence of a bug can be terrible in a plant, it can cause delay, or outage, or even physical damage of the system. Also, in another category of problem, no owner of ICS wants to see their informations displayed and the architecture of their systems openly showed to the public. So this need for verification and compliance reduces the capacity of modification.

Error management Another serious problem to take into consideration, is the fact that the system is often time-critical. It cannot accept jitter or delay. A command sent to an actuator must take effect immediately and data sent by a field device is only valid if it represents, in real-time, the current state of a part of the system. We can also point that failure tolerance is lower in ICSs than in IT networks because of that time-critical property. A communication that does not operate properly because of packet loss for example can have severe impact. If for example, we wanted to sort out traffic by dropping malicious packet, there must not be false positive. It means that legitimate packet must never be, in any circumstance, considerated as a malicious packet. If false positive happens, this can disrupt the course of execution of the whole system and corrupt the process.

Resources The last problem we want to speak about is the resources limitations. We point out three limitations:

- Field devices used in ICSs are embedded systems that have limited resources. It is not possible to apply traditional, resource demanding, security mechanism such as encryption.
- Resource can be located such that it is hard to access them physically to operate on them.
- Compared to IT network components which last for 3 to 5 years with the fast growth of technology, ICS components have a long lifetime, from 10 years to 20 years, which results in relatively old component[36].

Field device runs continuously without being stopped once during their lifetime so they suffer of memory fragmentation. Adding the fact that they have low resource make them more prone to buffer-overflow attacks[34].

2.3.2 Differences

We have explained some constraints that show why it is hard to ensure security in a ICS network, but even without those constraints the simple fact that ICSs and IT networks do not serve the same purpose prevent traditional security mechanisms to be as efficient in IT networks than in ICS networks. Security policies in IT network focus mainly on authentication, integrity and confidentiality. In ICS network, because it has a direct impact on the physical world, availability and data integrity should be considered as more important than confidentiality. Also, the protection of field device is as important as control server. In IT networks, it is often not the case where central servers are protected but not edge clients [33]. In ICSs it is not the case, field device are quite vulnerable due to their long lifetime, so leaving unprotected or not enough protected could lead to their infection and violating the previously evoked properties.

These differences, along with the constraint explained, show that it is hard to protect ICS networks and that maybe security mechanism used in IT networks are not fully adapted yet to ICSs. It is in this context that we develop the idea of simulating attacks against IDS.

2.3.3 Classical Attack Scheme

Attackers can spend several months to mount an attack on a target. These attacks can be divided in phases. These phases have different purposes but they are necessary to achieve the attack correctly. The first phase is the *reconnaissance* where the attacker will look for a target. The second phase is the *scanning* where the attacker will try to find vulnerabilities on the target. This phase tends to be the longer one this is when the attacker will gather as much information as possible so that he/she will be able to create exploits with more ease. After this phase, this is where the *exploitation* phase will begin which corresponds to the attack. The attack can have different goals such as gaining information or simply disrupting the operation of the target. For an attack on an ICS, the attacker will try to access the control network through different network directly connected to the control network such as the corporate network. If he/she gets access to the control network, it can also access field devices. Knowledge about ICS common vulnerabilities can be found easily, specially because ICSs are less isolated and use open standard protocols. So during the scanning phase the attacker will look for these vulnerabilities. Depending on the vulnerabilities he/she found, the attacker can choose how to exploit it. Common attack vectors includes[34]:

- Backdoor in network perimeter
- Vulnerabilities in common protocol
- Communication hijacking
- ...

Given access to the control network, the attacker can then manipulate at his/her will the field devices and also the actuators and the sensors. The attacker can for example inject fake data in the communication between the control server and the field devices such that the state given to the operator is not correct which would lead the operator to perform wrong actions.

2.4 Security

In this section we will go through all majors protection mechanisms that exist to protect ICSs against all kind of attacks. We will base this section on 2 documents that collect a lot of these protection mechanisms.

1. ENISA - Protecting Industrial Control System[1]
2. NIST - Guide to Industrial Control Systems Security[3]

2.4.1 Protection Mechanism

In this section we are mainly looking at software based ways to protect ICSs. We will not look at all techniques that focus on physical attacks, trying to protect ICSs from the perspective of a person accessing the system from the inside. Protection mechanisms are divided according to the purpose they want to achieve. We identify three purposes of protection mechanisms, resisting attacks, detecting attacks and recovering from attacks. We will look at the following protection mechanisms used to resist attacks: *Authentication*, *Access Control* and *Communication Protection*.

Authentication The authentication "*describes the process of positively identifying potential network users, hosts, applications, services, and resources using a combination of identification factors or credentials*"[3]. In other words, we want to be able to know who or what we are dealing with when performing any action. Usually, we distinguish three categories of authentication mechanism:

What I know: The authentication will be based on an information that only the user knows.

What I have: The user will be authenticated by a trusted and unique object that he/she owns.

What I am: Biometric features of the user will be used to do authenticated him/her.

There exist different technologies for each category of mechanism such as password, challenge/response or even eye-scan authentication system. A good authentication system usually involves two of the three categories. Indeed, many networks, not only SCADA networks, use passwords as authentication mechanism. This kind of authentication is not secure enough by itself, password can be retrieved through social engineering attacks such as phishing attacks or they can be cracked if they are not well chosen. Adding a new layer provides security in depth. These mechanisms are useful when trying to control that a person trying to perform a particular operation or to access sensitive data has the right to do so.

Access Control The authentication process will give us the possibility to implement access control. This is to ensure that the person that accesses a particular resource has the right to do so.

To ensure access control we have different technologies we can use. We can use, for instance, VLAN to divide the network and grant access of some part to some kind of users and the other parts to other kind.

Communication Protection The last protection mechanism we will present here is the communication protection. To ensure that the communication remains private and that nobody can eavesdrop them. The traditional way to perform communication protection is to encrypt them. In this way only the right person can understand the communication that are transmitted inside of the system given that he/she possesses the right key to decrypt. Virtual Private Networks use encryption to provide an access from a remote network to private network. VPNs allow sending information through public networks thanks to tunneling technologies. In the case of ICS systems, the VPNs can be used to provide secure access between the corporate network and the ICS network.

Chapter 3

Intrusion Detection System

In this section, we describe the desirable properties of Intrusion Detection System. We then give some theoritical background about IDS. We present different approaches that Intrusion Detection System takes to detect malicious operation. We also present Snort, an open source IDS, that we used to test our traces.

3.1 Goals and properties of IDS

The need for IDS rises from the simple fact that it is hard to guarantee that a system is completely secure. Indeed, a vulnerability is not known until it has been exploited, we can think of the zero day vulnerability. Security flaws can have economic impact but in the case of industrial systems, this impact can go beyond economical aspect. If an attacker could control parts of a nuclear plant, he/she could cause significant damages. Furthermore, because SCADA systems have come to use traditional IP communications, old protocols such as MODBUS which showcase no security capabilities make the system more vulnerable. All these reasons show the importance of IDS. The goal of the IDS will be to detect attacks on a system by monitoring it. In order to evaluate an Intrusion Detection System,[7] proposes the following properties:

Accuracy: The IDS is accurate when it detects attacks and does not trigger false alarm, meaning that it raises an alarm when it should not. Otherwise, it's inaccurate.

Performance: The IDS is efficient when it is able to monitor in real time what is happening in the system. It does not miss a single event.

Completeness: The IDS is complete when it is able to detect all the attacks.

Fault Tolerance: An IDS is fault-tolerant if it is itself resistant to attacks, particularly denial-of-service.

Timeliness: Alert should be triggered fast enough so that the security officer has the possibility to react to the attack.

All these properties are desirable and allow distinguishing between a good and a bad IDS.

3.2 Type of IDS

As mentioned earlier, the goal of an IDS is to detect attack by monitoring a system. If the system monitored is a host in a network, then we talk about *Host based Intrusion Detection System*. If the system monitored consists of a network then we talk about *Network based Intrusion Detection System*

3.2.1 Host based Intrusion Detection System

Host based IDS (HIDS) runs on every host of the network. They monitor traffic incoming and outgoing from the host. HIDS were the first IDS created, as primary target of attack were mainframe computer with users local to the system[7]. In addition to monitor traffic, HIDS monitors file system, login activity,... Therefore they are able to trace all user activities on a host. Because they are host based, if an other host than the one they are running on is compromised they won't detect it. Worse, if the host itself is corrupted then they become useless, meaning that if an attack occurs, alert must be generated while an attacker is taking control of the host. Otherwise, the attacker can remove all trace of attack. Among HIDS we can cite Tripwire[28] or OSSEC[29].

3.2.2 Network based Intrusion Detection System

Compared to HIDS, Network based IDS (NIDS) are placed in strategic position in the network, for example in a Demilitarized Zone, so that it can monitor all the incoming and outgoing traffic of the network. Since the NIDS monitor all the traffic for a network, it should be fast enough when analyzing packet to avoid missing packet as much as possible. There exist several NIDS such as Snort[25] or Bro[26].

3.3 Paradigm

To achieve its purpose, an IDS can use two ways of detection. The detection can be based on an attack signature or on a behavior of the system that differs from the usual one. The latter is denoted as *signature or knowledge based IDS* and the latter is denoted as *behavior based*.

3.3.1 Signature Based

A signature based IDS will use its knowledge about attacks and vulnerabilities. When IDS detects a certain pattern matching with an attack it is aware of, it will trigger an alarm. For example, if an host receives many `SYN` packet and no response to the `SYN/ACK`, the IDS will detect a *SYN flood* attack. In a signature based IDS such as Snort, attacks are described via a specific language to write rules. The IDS only detects attack it is aware of, this means that attacks unknown to the IDS will not be detected. In term of properties, we can see that these kind of IDS are not complete as they can miss some attacks. It will depend on the knowledge of the IDS, the bigger the knowledge of the IDS is, the more complete the IDS will be [7]. The main advantage of these IDS is their accuracy, they generate little false alarm. Furthermore, because the attacks are well-known, they provide high insight and detailed information about what is happening in the system during an attack.

3.3.2 Behavior Based

We can distinguish two states for a behavior based IDS: a learning phase and a monitoring phase. During the learning phase, the IDS will generate a model from different sources of information. This model represents the behavior of the system. An example of behavior can be something as simple as a user that log on its computer every morning business day for two hours on. Once the model has been generated, the IDS will begin to monitor the system. If the IDS observes a deviation in the behavior in the system from the model, it raises an alert. In the example above, if the user log in a Sunday night, IDS will think that something is off. By using behavior based detection, the IDS can even detect new attacks which make them more complete than signature based IDS. But behavior based IDS suffers from two weaknesses. First,

they lack of accuracy. Indeed, it is not always possible to depict optimally the whole behavior of the system. The duration of the learning phase is also important. If it is too short, the IDS might miss some events considered usual and treat them as abnormal when monitoring. These reasons have as consequence high false alarm rate of the behavior based IDS and a lack of completeness if the learning phase is too short. The second drawback of these IDS come from the fact that if an attack occurs while the IDS is in the learning phase, the attack won't be detected in the monitoring phase as it will be considered as normal[7].

3.4 Snort

Snort is a free open source Network Intrusion Detection System. Snort is a signature based IDS, where attacks signature are described via rule in a text file. Snort has a wide community such that many rules have been published and regularly updated. This mitigates the weakness of signature based IDS, with more rules the IDS becomes more powerful.

3.4.1 Description

Snort is composed in 5 components[14]:

Packet Decoder: The packet decoder sniffs the packet and sends them to be preprocessed or directly decoded.

Preprocessors: Preprocessors are plug-ins used to modify packets before being analyzed by the detection engine. This can be useful to remove extra information used by an attacker to bypass the IDS.

Detection Engine: The detection engine is in charge of analyzing incoming packet and detect whether an attack occurs or not. If, a packet does not match a rule, the packet is dropped, otherwise it can be logged or an alert is raised.

Logging and Alerting System: If a packet matches one of the rules, the matching rule decides what to do with the packet, raise an alert or log activity.

Output Modules: These modules are used to specify how to generate the alert and do the logging.

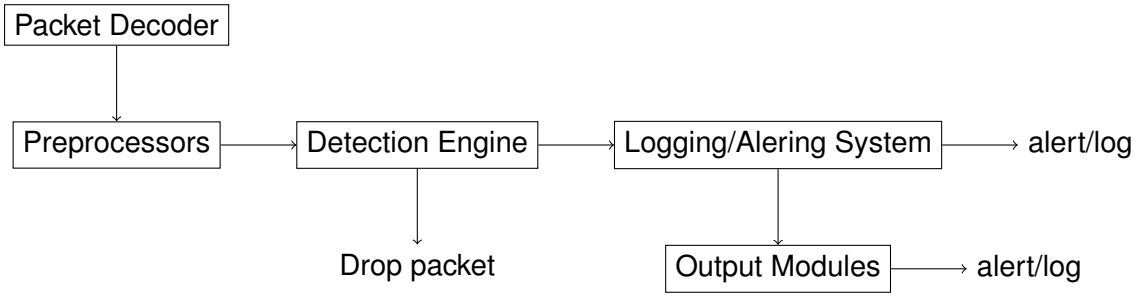

Figure 3.1: Source[14]

Snort provides a MODBUS preprocessor which allows to decode MODBUS protocol packet. The use of preprocessor improves the expressiveness of the rules as protocol field can be directly accessed.

3.4.2 Rules

The rules we used for Snort come from [24]. The rules focus on illegal request, malformed packet size. In MODBUS, each function code follows a specific format so these kinds of rules should be quite efficient. The rules also look for exception message which could denote that something fishy is happening.

```
1 alert tcp $MODBUS_CLIENT any->$MODBUS_SERVER 502
(msg:"Read holding registers";modbus_func:3;sid:2016;rev:1;priority:3;)
3
```

Listing 3.1: Example of Snort Rules

Chapter 4

IDS For SCADA

The need for IDSs in SCADA has been highlighted by an attack such as Stuxnet. There has been some work regarding this subject. Most of them choose a behavior based IDS. The idea is that unlike traditional IP networks, SCADA networks are much more stable. The topology of the network changes more sporadically. Furthermore, communications between the component of the network, are more predictable as the messages are sent in a periodic fashion. All these elements ease the construction of an accurate model for these kinds of network. In this section, we present works that have been done regarding the design of IDS for SCADA network. Reader will see that different solutions have been proposed and that they use different approaches. The common idea behind these propositions is to build model of the system and use that model to detect malicious traffic.

4.1 Related Work

Cheung and al.[13] explains three approaches based on modeling. The first one models the protocol itself through specifications. Indeed, the MODBUS specifications are used to create the model and the traffic deviating from the specifications cause the IDS to raise an alert. The specifications cover the protocol in different ways. One way, is to specify the individual fields defined in the protocol. Typically, for the MODBUS protocol, listing the function code used by a PLC is a first step to model its behavior. Specifications can go further covering multiple fields that are dependent, meaning that the value of a field can be accepted only if the value of an other field has been accepted. Examples of dependent fields are the function code and the length. If the length is not variable for a given function code, by specifying the couple function code and length we can model the message format that are expected for a PLC. If the length varies then, an upper bound and a lower bound is included in the specifications. An other way is to build the specifications covering request/response transaction. Some fields in the response must match the corresponding one in the request. Furthermore, some requests are sent periodically and the corresponding responses are almost always the same, so they can easily be predicted.

The second approach models the communication pattern between components of the network. The model is made as network access policies. If a communication violates the policy then an alert is raised.

The last approach focuses on the availability of the servers and service. The system first learns about the service available in the network it monitors. Connection attempts on non-valid service, service unknown to the system, are considered as suspicious. This approach is more useful on traditional IP network which provides more services but it can be tweaked for SCADA network. Indeed, services on a server are the equivalent of a function codes on a PLC in that both of them define the use of the component.

Yang and al.[15] choose to build the model through statistical techniques. The authors first create multiple traffic profiles which correspond to "*sequence of measures over a specific period of time*"[27]. In this case measurement includes network features collected via SNMP and hardware features such as CPU usage, link utilization, etc.... Traffic profiles are then classified according to the period the measurement was carried out. The profiles are fed to an *AutoAssociative Kernel Regression* (AAKR) to build the model that will predict the valid behavior. During the monitoring phase, new profile will be created and given to the AAKR model. The model will compare the profile with what it expected and generates residuals. Residuals correspond to the difference between the two observations, actual and expected. These residuals are then submitted to a binary test called *Sequential probability ratio test*. This test will define whether the new observation is suspicious or not.

Gao and al.[16] described an IDS using a neural network to create the model. The neural network is trained with captured packet containing requests/ responses exchanged between the control server and the field devices. The neural network used as input the content of the message both sent by the control master and the field device, the frequency of the requests/responses transactions. This kind of IDS is able to detect requests/responses injection trying to provoke a denial of service.

Compared to the two previously cited works, Goldenberg and Wool[17] used deterministic finite automaton to model a channel between a field device and the control server or the HMI. The model goes deep into packet characteristics, it will not only cover function code but also the addresses of registers and coil that the packet wants to access. These characteristics will be used for creating the alphabet of the DFA and the transition function. The states are the position of a message exchanged on the channel for a period of time. For example, if every 30ms 2 queries are sent to a PLC, there will be 4 states for the DFA modeling this channel. Input that are unknown to the IDS or are not at the position they should be in the DFA triggers alert.

Usually, behavior based IDSs detects attack by observing anomalies in the traffic but Fovino and al.[18] took a different method where they described the critical state of the system via a language they have created. Critical state can be defined as state or configuration that can put the system at risk. The systems are divided into subsystems. Each subsystem is monitor by a local IDS which will periodically send request to the component of the subsystem to gain information about their status. Each local IDS is connected to a global IDS which will collect the critical state alert for local IDS and infer global critical states.

4.2 Our IDS

We implemented our own IDS which is a behavior based IDS. It is important to say that IDS has been for implemented with the simulator in mind. Therefore, it has been mostly designed for the MODBUS protocol as it is the protocol used in our simulation. The IDS can easily be extended to support other protocol though. We implemented this IDS to see if the trace we have generated could be used to evaluate an IDS. It can monitor live traffic but we will use it to read pcap file.

4.2.1 Approach

This IDS is a behavior based IDS using a whitelist approach. We went for a whitelist because SCADA networks are more stable than traditional IP networks. Components of the network are not removed or added as much as in an IP networks. Because it is a behavior IDS, it has two phases: a learning phase and the monitoring phase. During the learning phase, the IDS will build two whitelists. The first whitelist will be a flow whitelist[9] containing the legal flow in

the system. A flow is defined by the source IP address, destination IP address, source port, destination port and the IP protocol. All the packets with the same source/destination IP address, source/destination port and IP protocol belong to the same flow. The second whitelist details for each MODBUS server present in the system, the functions code and the corresponding length of the PDU, we will refer to it as the MODBUS whitelist. A MODBUS server is identified by its IP address, the port it uses and the IP protocol used. We decided to make a second whitelist because we thought that the first one would not be able to detect MODBUS oriented attack.

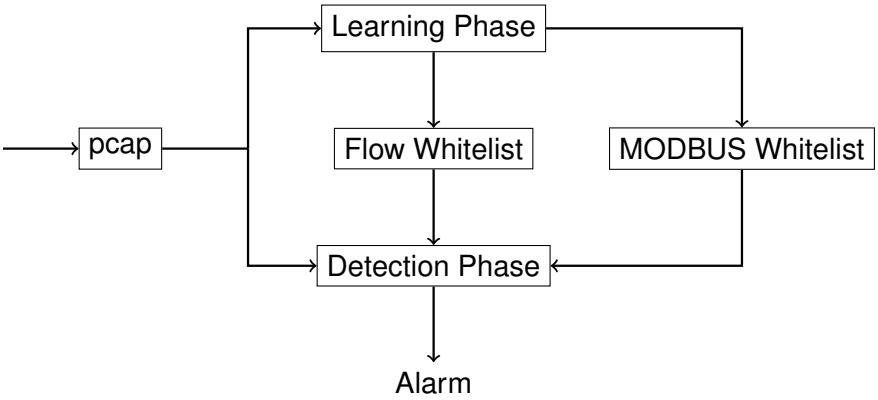

Figure 4.1: Phase of the IDS

Once the learning phase is over, the IDS enters in the monitoring phase. For each packet, it will first check if it belongs to a whitelisted flow. If it does, it will determine if it is a MODBUS packet sent to a MODBUS server, by analyzing the destination address/port and the IP protocol. If it is a MODBUS packet sent to a MODBUS server, it will perform a lookup in the MODBUS whitelist to see whether the function code and the length field in the MODBUS packet for the corresponding server are present. If not the packet is considered as malicious. If the packet belongs to a whitelisted flow but it is not malicious, no alert is raised. It is important to mention that the IDS will perform a lookup in the MODBUS whitelist only if the packet belongs to a legal flow. Also, the IDS is only able to raise alert, it cannot drop packet. This is due to the nature of IDS, which is behavior based and have a high false alarm rate. Unnecessary dropping packet in this kind of system can have massive impact.

In 3.3.2, we mentioned that a behavior based IDS might consider an attack as legal if it has been performed during the learning phase. To avoid this problem, the IDS can achieve the learning phase in offline mode, meaning it generates the whitelist from trace. If the trace is known to be safe and representative of what happens in the system, then the problem is avoided.

4.2.2 Improvements

A limitation that the IDS has, is that the whitelist cannot be edited to add or remove flow according to the changes in the topology of the network. Even though SCADA networks tends to be more stable, it does not mean that the topology is fixed. Furthermore, if the learning phase does not capture the whole behavior of the system, the IDS will generate false positive. Flow triggering those false positive should be integrated to the whitelist dynamically by an operator. Based on his/her knowledge and experience, he/she would decide whether to add the flow or not.

4.2.3 Implementation

The IDS has been implemented in C. We used the *libpcap* library to sniff packets. *Libpcap*[20] is widely used and provides a lot of documentation which make it easy to use. To implement the

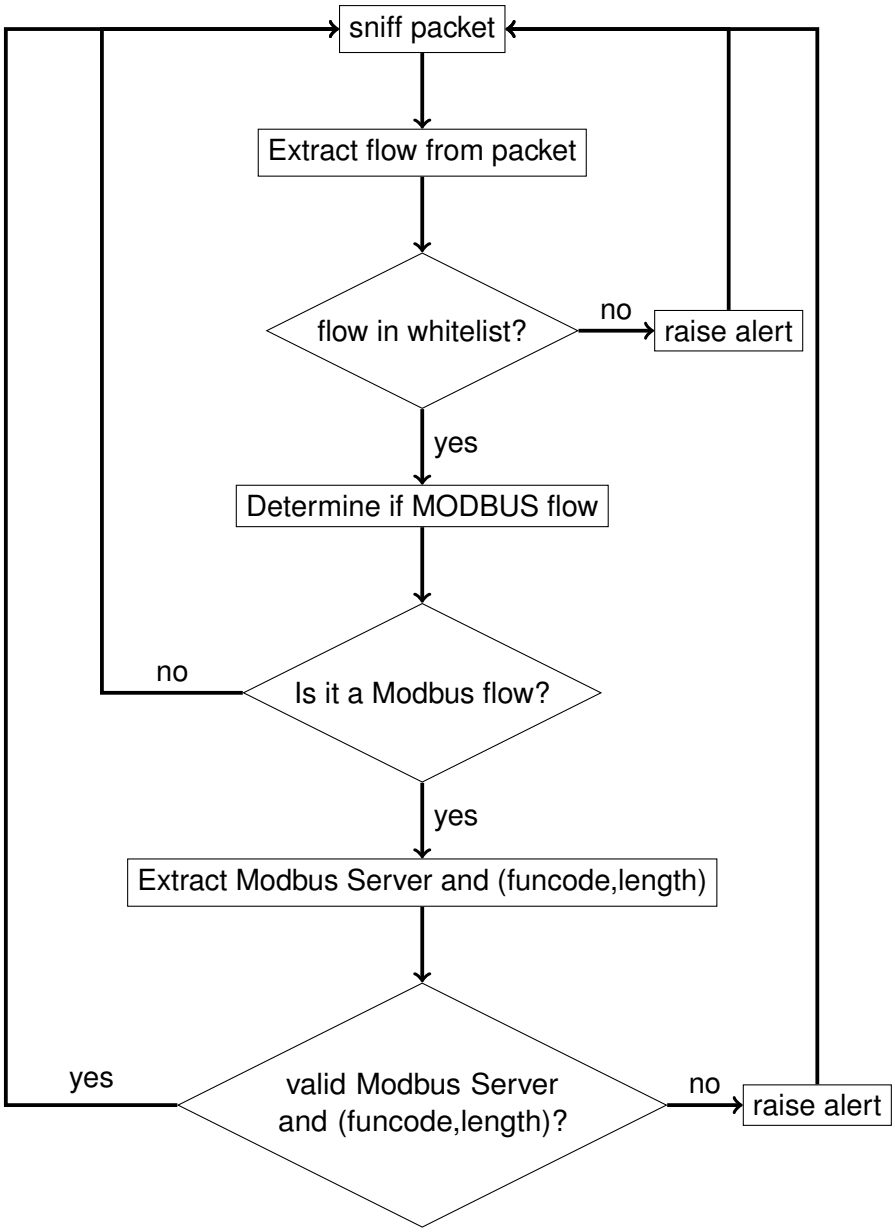

Figure 4.2: IDS Activity Diagram

whitelist, we decided to use a hash table[21]. Hash table are efficient in terms of lookup which is exactly what we need in order to make the IDS efficient. If the analysis of a packet is too long, the IDS may miss some packets. We had to choose which hash function provided by[21] would be the most efficient for the flows. We run tests in terms of speed and the number of collisions. The different hash function displayed approximately the same performance so we decided to keep the default one, which is the *Jenkins* hash function[22].

Chapter 5

Simulating attacks

In this section we will describe the attacks we have implemented for the simulation. We first discuss the existing work we have come accross during our search. Then we will explain the attack and how they were implemented.

5.1 Existing work in the field

To experiment the attacks, we needed a simulator behaving like a SCADA system. It was important to simulate the communication between the different devices in the system so that we could see the impact of an attack on such system. We had the choice between implementing our own system or using an existing one.

At the beginning we have chosen to explore the idea of using an existing simulator. We found a project on github called **VirtuaPlant**[19]. VirtuaPlant is an Industrial Control Systems simulator written in Python. This simulator uses a 2d physics engine to show what is happening in real time in the plant. The goal of this project is to be a collection of several plants, achieving different tasks and using different protocols. For now, there is only one plant, running Modbus, which is a bottle factory with one task which is to fill container with bottles. The initial idea was to use this simulator, but the problem is that the current version as of right now is a bit too limited. It only uses two Modbus function codes and the behavior of the system is always the same which lacks of realism. Despite these limitations, the simulator was a huge asset for us to understand how the Modbus protocol works. We found other simulators but we set them aside because they were not open source or they lack of flexibility.

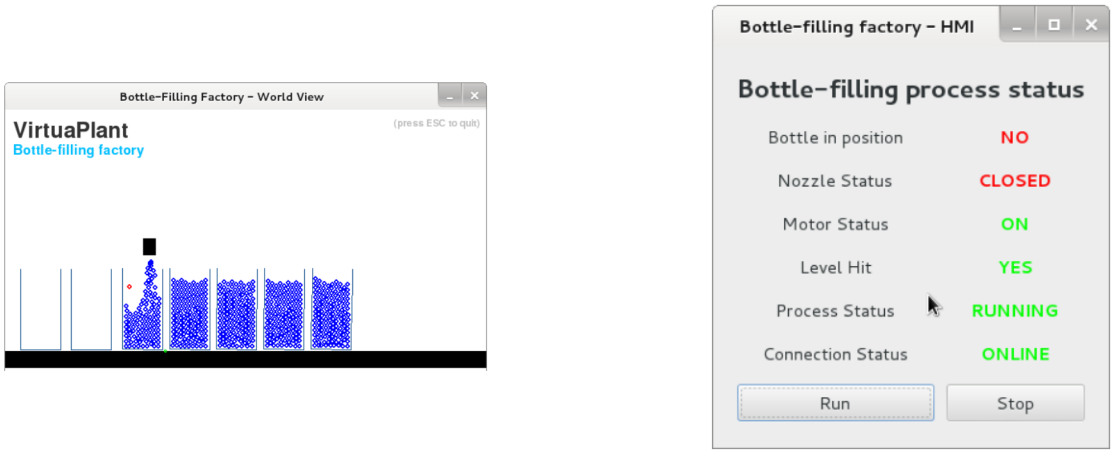

Figure 5.1: VirtuaPlant

We were also provided with an article [44] reviewing all existing methodology to use simulation

in the context of SCADA. This work highlighted three main ways of doing so: *High Level Architecture* (HLA), *Network simulators* and *SCADA simulator frameworks*.

High Level Architecture It is a standard method developed by the US Department of Defense to *ensure the interoperability and reusability of models and simulation components*[44]. These are rules you can comply to, to ensure that the simulation is working with other existing simulations.

Network simulators Network simulators are not only oriented at ICS, but they can simulate all kinds of network, trying to copy the behaviour of real network[44]. There are some that are used to test ICS security, such as *OMNet++*.

As we have to perform attacks on the network level, capturing traffic traces, it seemed like a perfect fit for our project. But we were interested by the idea of having a fully working SCADA environment, and even though this can be configured on a network simulator, it gives us less control than the creation of our own simulation.

SCADA simulator frameworks These frameworks are oriented at replicating the behaviour of SCADA system. The paper [**?**] gives us a SCADA simulator used to test attacks: *SCADASim*. Those SCADA simulator makes use of Network simulators and other different system.

This seemed like another good idea to use these kinds of simulation. It gives us a good advantage over network simulators, it is SCADA oriented. But *SCADASim* has a few disadvantages. The first one is the lack of documentation. The second is that it works upon the Omnet++ Network simulators, which is a system that covers way more network systems that we are really interested of. And finally, the system has been created 5 years ago, in 2011, and the creators stopped working on it 3 years ago, creating complication to use with the current system, as it is not up-to-date[37].

To conclude, we decided to develop our own simulation that would be really oriented towards our goal, performing attacks simulations on a SCADA system.

5.2 Our Simulation

We will present the different aspects of our simulation, we will begin with a tour of the choice we made and the objectives we set up for the simulation. We will also explain why we decided to implement our own simulation. Then we will present the architecture and the content of the simulation, finally we will get to the implementation itself and the result we had.

5.2.1 Choice and objectives

This simulation has been realised having in mind that we wanted to simulate attacks on all kind of SCADA systems. The main target was to devellop a simulation that had three main characteristics. The first characteristic is that the goal of the simulation is to get Modbus/TCP trace to perform our attacks, so the traces we can collect from it have to be close to what we could get from a real SCADA system. The second important point was to have a simulation that would be able to evolve according to our needs, if we choose to test a new attack on the system, if some SCADA system characteristics are missing, we have to be able to easily add the particular characteristic that is missing. Finally, it has to have a SCADA-like architecture.

One of the hypothesis of our work is that we do not take care of protection that occurs before what we might reffer as the trusted zone. The trusted zone is the inner area of the system where the PLC, the control server and the HMI communicate freely. So our attacker have bypassed already protection such as firewall and other elements.

5.2.2 Coal Powerplant Simulation

We first took a decision about what type of system we would create. We decided to implement a simple coal power plant system, because the basic concepts of a coal power plant are quite easy to understand, even for non scientific people. The system will work as follows. There will be 6 elements working together, an element will drop some coal on a rack, which will bring, by rotating, the coal into a furnace. This furnace will heat up water, which will create some steam to activate a turbine. Finally the turbine rotation will create electricity.

Now we are going to explain the few mathematical formulas we used to simulate the different reactions in the power plant. Those formulas are quite simple and we will not detail them much, considering that it is not the purpose of this work. But, still, to have a realistic simulation is important in order to have the correct relations between the different variables. Still, our mathematical model is a highly simplified one.

Heat in the furnace For the furnace we need to know four elements.

1. How long take 1kg of coal to burn,
2. What amount of energy it will produce,
3. What will be the temperature in the furnace,
4. What will be the energy loss of the furnace.

We solve all those problems by considering that a kilogram of coal is able to produce $30.000kJ$ of energy. We have tried different parameters, but to have something not too slow to be executed we chose that 0.2% of the coal available in the furnace will burn each second.

The temperature in the furnace, is equal to the energy in MJ in the furnace multiplied by $1.006K \cdot MJ^{-}1$. We consider that 0.2% of the power is lost each second and that there is 1.4% of energy transfer to the water cuve. So we have the maximum amount of energy in the furnace equal to $\frac{Production(kJ \cdot s^{-1})}{Loss}$. By default the temperature cannot go lower than the ambient temperature of $20°C$.

Coal (kg)	Burn coal ($g \cdot s^{-1}$)	Prod. ($kJ \cdot s^{-1}$)	Total (kJ)	$°C$
0	0	0	0	20
50	100	3.000	187.500	189
100	200	6.000	375.000	377
150	300	9.000	562.500	566
200	400	12.000	750.000	755
250	500	15.000	937.500	943
300	600	18.000	1.125.000	1.132
350	700	21.000	1.312.500	1.320
400	800	24.000	1.500.000	1.509
450	900	27.000	1.687.500	1.698
500	1.000	30.000	1.875.000	1.886

Figure 5.2: Furnace variables

Heat in the cuve The energy in the furnace will be slowly transferred to the water cuve, which, when reaching a certain temperature, will create some steam, that will output to the turbine in order to create electricity. The water cuve can contain up to 3000 liters of water. The maximum

flow of water in the cuve is about 400 liters per seconds. Let E be the energy in kJ and L the amount of water in the cuve in liters, we have the following temperature in the water cuve

$$\frac{E}{4.185L}$$

Once reached 100 degree Celsius, the energy needed to evaporate one liter of water is equal to $2264.78kJ$, once we have enough energy to reach $100°C$ (so this energy is $E = 418.5L$).

We will then consider that half of the steam produced each second will output to the turbine.

Figure 5.3: Cuve variables

Turbine Rotation and Battery The formula for the rotation and the battery are quite straightforward. Half of the steam will output from the cuve and depending on the amount of steam inputting the turbine, the rotation will increase, trying to reach a certain level. The steam amount will be divided by 4 to have the rotation speed.

The rotation varies according to the steam produced, it is not supposed to suddenly change speed.

The turbine rotation speed will be the amount, in Watt, of energy added to the battery. The output of the battery will be a constant value of 10. A full battery would contain 50000 Watts[1].

5.2.3 Architecture

For the architecture, we chose to implement a basic version of the classical control network (Figure 6).This kind of control network is working with 3 main components:

1. PLC: The Modbus servers
2. Control Server: The master server running the plant
3. HMI: A monitoring system

On the PLC side we have 6 different PLC, each of them holding a various amount of different types of register to hold sensors values and other variables. The PLCs are:

1. Coal Dropper PLC: this plc is responsible of the action of dropping coal to a rack, which by rotation will bring it to the furnace.
2. Coal Bringer PLC: this plc takes care of the rotations of the rack, there are 8 racks rotating. They are charged with coal by the coal dropper and they bring the coal into the furnace.
3. Furnace PLC: this plc keeps in memory the amount of coal available in the furnace and the temperature of the furnace. It is also responsible for the command of burning the coal.
4. Water Cuve PLC: this plc keeps data about the amount of steam, water and the temperature inside the water cuve. It can act on valves to bring in and out water inside the cuve.
5. Turbine PLC: this plc will be able to act on the turbine by enabling or disabling its rotation and it will monitor the rotational speed of the turbine.
6. Battery PLC: this plc will keep track of the stored energy inside the battery. You can also limit this amount and you can decide to link or unlink the battery from the global electrical system.

[1] this value is randomly defined and does not represent any real value

The HMI is a terminal which is accessible to the user. You can check all commands and possibility in the appendix that explains how to use the simulation. The only real important aspect of the HMI is the constraint system. You can create constraint for the system to act on its own. These constraints are on the coal amount in the furnace, the turbine rotation speed, the temperature of the water, etc. Concretly, you specify a constraint on the system on the HMI and the control server will try to validate it. We will explain a bit more in the implementation part below, how we implemented this system.

This architecture is simple and fits our needs completely. The other elements of SCADA system were, in our case, not needed to generate the traces for our attacks.

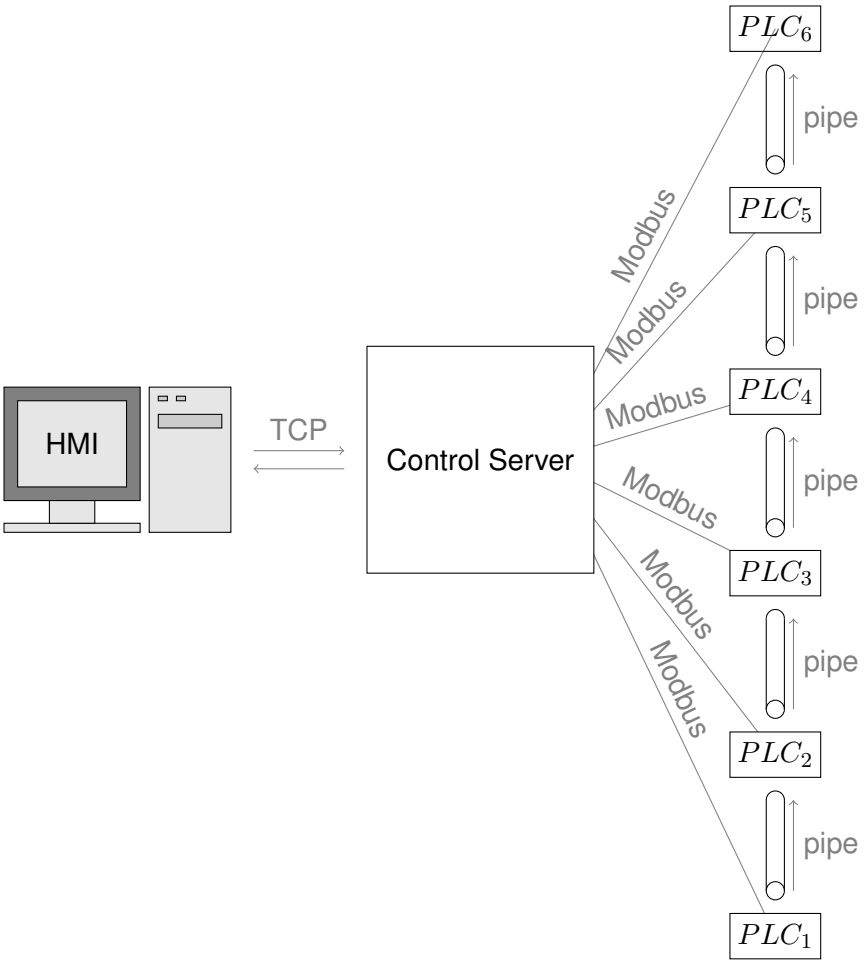

Figure 5.4: Our simulation overview

5.2.4 Implementation

Here we will focus on the implementation part of the simulation. We shall separate in three chunks, one for each element of the system. We will start with the *control server* which centralize all information, we will then explain the implementation of the *HMI*, which will operate the system and then we will express how we managed *PLCs*.

Control Server The control server will connect itself to all the PLC using ModbusTCP connections. So he can query PLC with the modbus protocol. The library used is the *LibModbus*[4], which is a C implementation of the Modbus protocol.

The control server will hold two threads, the first one will be used to communicate with the HMI and the second one will manage all operations on the simulation. To coordinate these two

threads, two shared linked lists have been created. The first one will contain constraints on the system and the second one will contain actions. Constraints are sent by the HMI and are then stored in this list, such that the main thread will go through the constraint and try to satisfy them. Some constraint will resolve in the creation of other constraints on which they depend, this is a system of recursive constraint. For instance, a constraint on the cuve temperature will create a constraint on the furnace temperature, since the second one will be a dependency for the first one. When it will see the need, the system will create new actions and add them at the end of the linked list. At each iteration of the system, the main thread will check the action list and query the good PLC to execute the selected action.

This constraint satisfaction system will work periodically depending on the constraint type. The system will verify if one constraint is satisfied only at given intervals.

HMI The HMI will create a TCP communication with the control server. On this connection we will send either constraint or action using define commands. An action can be the request of a modification to the system or simply requesting the display of information. So a terminal is available to the user, and the commands are parsed in a simple way to perform the proper action.

PLC For the PLC, we chose a particular structure to launch them. As we needed for them to be independent but still to be able to communicate between one another, we decided to create one process by PLC. Each process can communicate, for the purpose of the simulation, with the one that is following it in the order of PLC using pipes.

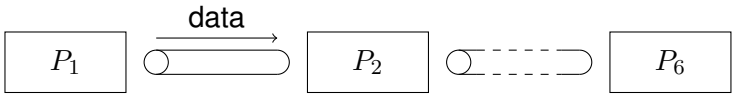

Figure 5.5: PLC processes using pipes

Those processes create a ModbusTCP server for the control server to connect to. PLC uses alarm each second to update the simulation and pushes on the pipes, for the next plc to read, the information related to the state of their simulation. So each PLC is blocked waiting for ModbusTCP message incoming and each second they are stopped to update their information and warn the next PLC.

5.2.5 Simulation Genericity

We now have to talk a bit about the ability to modify our simulation. We can quite easily build the simulation and modify it to work for another kind of simulation. For two reasons:

1. The PLC have been implemented using a function to collect and answer to the ModbusTCP messages, this function is almost the same for all PLC. Some of them make different initialization but mostly it is the same function for all PLC. And each PLC has an update function that has to be programmed with the logic of the PLC.

2. The second thing to be done to modify the system is to add in the control server all possible functions to interact with the newly defined PLCs. Also commands have to be added to the HMI.

Even though the simulation can be quite easily modified, it has not completely been built thinking of this possibility, as the simulation itself was not the main purpose of this work. It would require a bit of more work to have a completely generic simulation. The only part that

is particular to our simulation are the physics calculations, but the rest of the structure can be modified.

Also our structure gives the ability to easily change constraints and actions that are used by our system to define the interactions between the control server and the PLCs.

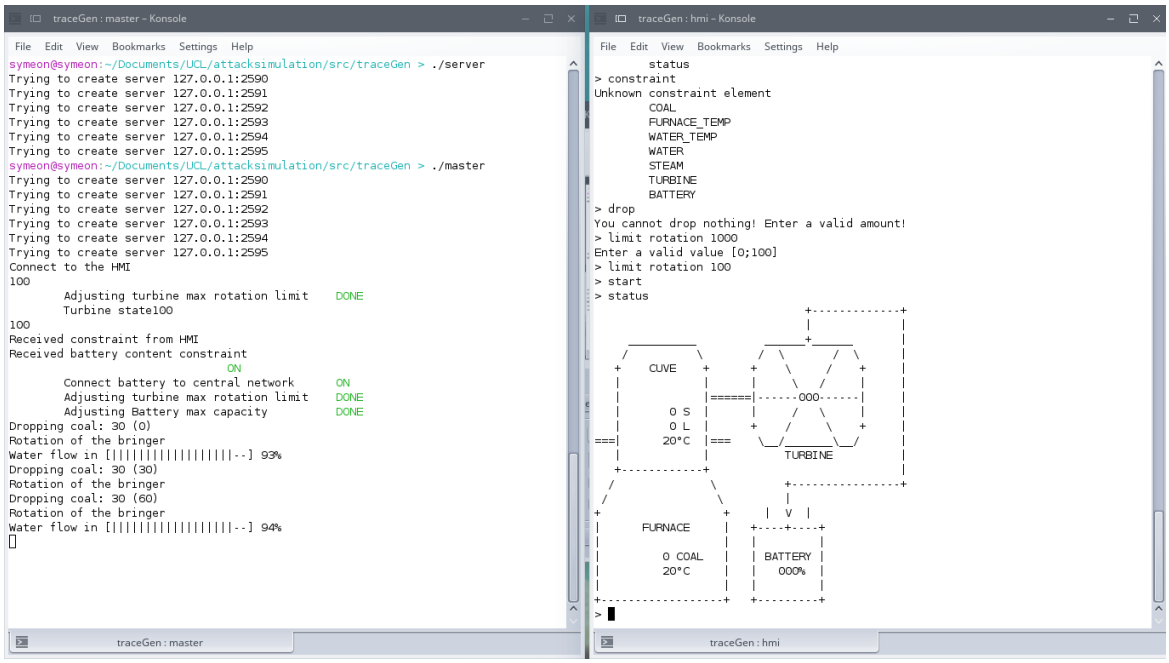

Figure 5.6: Our simulation

5.2.6 Result and Limitations

In the end, the simulation fills the goal we had fixed. You can easily launch it, manage it as a true ICS and you can perform the attack on the system. We will explain how we used the simulation in further parts when we shall explain the combination with the different attacks we created.

The limitation we see in the simulation for now is the fact that it is completely periodic. The traces produced will globally be constantly the same from a point in time. To add noise and variations in the traces we could set up scripts to define fake engineer tweaking the systems or we could modify the environment of the power plant such that events could impact the plant.

Another amelioration would be to have a fully generic simulation. We could create a GUI interface to be able to create blueprints of PLC and to connect them together using a node interface. And for each PLC we could define rules for their sensors.

5.3 Our Attacks

The attacks we have implemented focus on the communication protocol used in SCADA network, MODBUS in this particular case. We mentioned earlier that the fact that the initial design of these protocols was not planned for running on TCP constitutes a serious security flaw. Some of the attacks that we will describe could be avoided with basic security mechanisms.

5.3.1 Assumption

We have made an assumption about what access the attacker has in order to perform his/her attacks. We consider that the attacker can access a trusted part of the network in the system. Although this assumption eases the possibility for an attacker to carry out an attack, it is

not completely absurd. Indeed, there have been cases regarding ICSs showing that internal employee can be the cause of an attack[33].

5.3.2 Network scan

Even though this attack is more about the transport layer than the application layer we thought we had to implement it as every attacker will first begin by scanning the network to discover running host and port. There exist different scan techniques, but we choose to use `TCP SYN` scan. This technique has the advantage of being really fast. The principle behind is quite simple. We send a `SYN` packet to an host on several port,if the port is open, it will respond with an `SYN/ACK` otherwise it will respond with a `RST` packet.

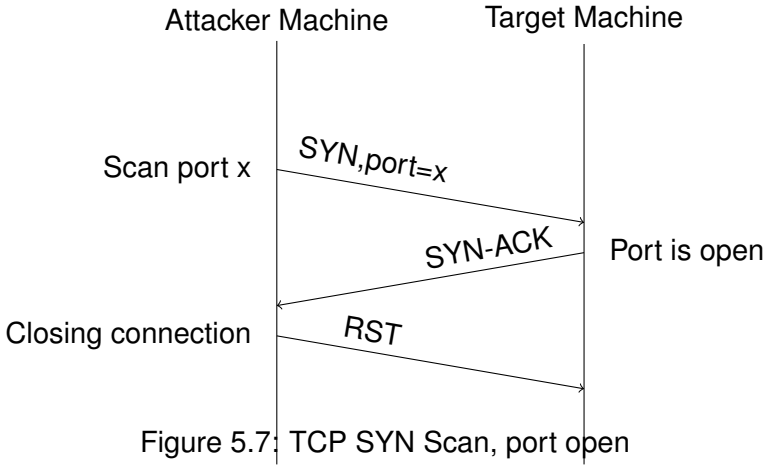

Figure 5.7: TCP SYN Scan, port open

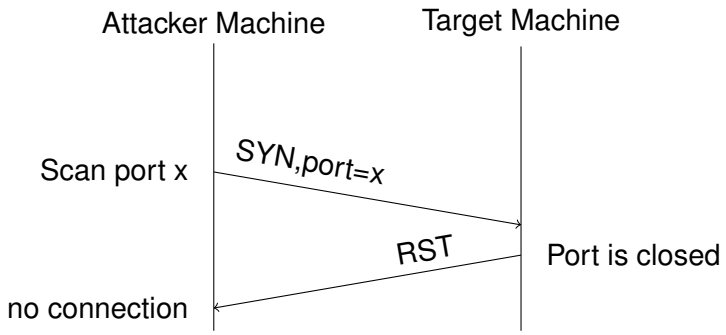

Figure 5.8: TCP SYN Scan, port is closed

5.3.3 Function Code Scan

As we mentioned in the explanation about the MODBUS protocol, the client defines the task it wants the server to perform by specifying a function code. When no error occurs, the server will echo this function code in its response. Conversely, if something is wrong, the server responds with the function code incremented by 128 alongside an exception code specifying the nature of the error.

The idea of this attack is to detect an exception and look for a specific exception code. This exception code is **01**, which corresponds to *illegal function*. This code is used by a MODBUS server to inform the client that it does not support this function code. Therefore, the attack will consist of trying all function code from 1 to 127 on a MODBUS server and checking in response whether or not the server supports the function. Reader might wondering why testing all the

function code knowing that some of them are public and defined in the specification of the protocol. Actually, even if those function codes are defined in the specification of the protocol, it is not mandatory for the server to support those functions[6].

5.3.4 Denial of Service

Listen Mode Only The function code in MODBUS can be used to access data but also for monitoring purposes. Indeed, MODBUS provides functions to analyze the communication between the server and the client, we refer to them as diagnostic function. Compared to other functions code which use 1 byte, diagnostic function uses 3 bytes. The first byte is set to 8 (function code for a diagnostic) and the two other bytes are used to specify what diagnostic to perform (sub-function code for a diagnostic).

Among these diagnostics, there is one that can lead to a denial of service. The sub-function is 4 and what this function does is to force the server to enter in *Listen Only Mode*. During this mode, the server will monitor any MODBUS message for which it is the destination or broadcast messages but it will be in an isolated state. It means that, it will no more respond or perform action upon the reception of a request from a MODBUS server. The only function it will react to is an other diagnostic function with sub-function 1. This function is the *Restart Communication* function which will get the server out of the *Listen Only Mode*.

Packet Size MDOBUS limits the size of its PDU to 253 bytes. This limitation exists so that the packet could be sent on a specific serial line. A PDU will be encapsulated further by the 7 bytes header and then by TCP so by limiting the size, there is an upper bound on the total size of the packet[32]. In this attack, we forged a packet where the PDU has a size of 300 bytes. This attack can cause of a buffer-overflow or denial of server on the client.

5.4 Implementation

We have implemented a Command-Line Interface in Python to run the attacks. Each attack is implemented using a script and has several parameters, such as the IP address of the target and its port. The CLI allows setting these parameters according to the user wishes. We used the library *Scapy*[23] in order to forge packet. We focused on the design of the tool so that new attacks can be added effortlessly.

Chapter 6

Experiments

In this section we will present to you the results we have obtained. First we are going to explain our methodology, then we are going to present the different results and finally conclude with the answer to our initial thesis. Finally we will tell you what could be done to push this work further ahead.

6.1 Goal of the work

It seems important to perform a quick reminder of the purpose of this work before going deeper into the analysis of our results. The main purpose of this work was to find a way to simulate attacks against IDS in the context of ICS. Because it can be difficult to use IDS in real situation considering the risk it can create on the actual system on which it might be running on.

With this idea in mind, we selected three attacks to be performed on different IDS (a whitelist based and the famous Snort IDS) in order to see if it was possible to easily perform attacks on those systems and find a way to improve them. So the main line of our analyze will be to see if we succeeded in the objective of simulating these attacks, and if it gives us interesting information about IDS usefulness and improvements that could be possible. And if this manner of simulation can be used on more complex IDS with more complex attacks.

6.2 Methodology

The methodology will be the following. We will take all the attacks and perform them on the simulation we have created. During the attack we will capture the traffic we are creating. This traffic will be stored in an attack file.

We will take the two types of IDS we have to be tested. The two IDS will read attack files to see whether or not they are able to detect attack. But for the whitelist IDS we will also vary the size of the training traces, to see its impact on the results we obtained.

Once we have gathered all of these elements we will classify all packets in four categories of packets for each IDS.

1. **True Positive**: The packet has been considered by the IDS as malicious and was indeed malicious.
2. **True Negative**: The packet has been considered as legitimate by the IDS and was indeed legitimate.
3. **False Positive**: The packet has been considered by the IDS as malicious but was legitimate.

4. **False Negative**: The packet has been considered by the IDS as legitimate but was malicious.

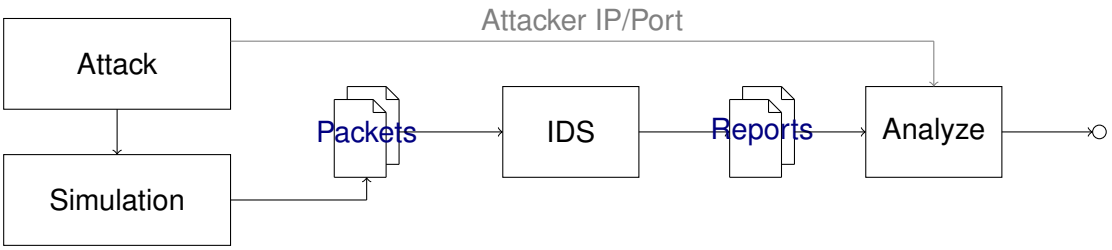

Figure 6.1: Methodology

We will then see how each IDS responded and we will conclude if this is a good way to test these attacks. We will also analyze how the different kinds of IDS performed against each type of attack we tested.

6.2.1 Other methodology

Before going further we have to give a small explanation about the choice of our methodology. At one point during this work we thought of another methodology. In which the simulation generates packets and we separate completely the generation of the attacks. Then we would merge both packets. The advantage of such technique is that you can merge the attack packets with any simulation trace. But we chose to discard this methodology for one reason, it was too static. We could not consider in the trace file the behavior of the system, when this one was confronted with attacks, it seemed like a non-natural way to perform our test. This is why we preferred to attack our simulation in real time and to create attack traces based on this method, such that the answer to the attack could be seen in the attack traces. It seems a more dynamic way to proceed, even though it requires a bit more work to create the trace files.

6.2.2 Attack Generation

We will quickly go through all the attacks and define in which condition they have been created. For each attack we have launched a software to capture all traffic traces going on the localhost connection, *Wireshark*. Then we have launched the simulation to run for approximately 2000 packets. Why 2000 packets in the traces? Because at this point, all PLCs in the simulation is communicating as they are all getting busy working in the simulation logic. Once reached this point we launched the attack with specific conditions for every attack. Then we filtered the trace file to keep only TCP traces, and we changed all the ports and IP so that every MDOBUS/TCP port would on port 502 (the official MODBUS/TCP port)[1]. Once all of these operations are performed we end up with a trace file for each attack, ready to be tested on an IDS.

Scan Attack For the scan attack, we set up the system to scan 1000 ports, containing our MODBUS/TCP port in the range. Once reached 2000 packets we launched the scan attack generating about 2000+ TCP packets with SYN and SYN+RST flags raised for most of them. Our attack properly discovered the MODBUS/TCP port that was available.

In the final trace file, as port have been modified, it will look like a scan attack except for the valid ModbusTCP port which will, out of the blue, be tested with port 502 and their particular IP.

[1]This is for readability reasons, so when we open Wireshark, MODBUS/TCP trace can be properly recognized, and it is also useful for using Snort with the proper MODBUS/TCP port

DoS Attack For the denial of service attack we used two different techniques. One sends a message to force listen mode and the other sends a packet too big such that the MODBUS server will not communicate anymore. We performed the two attacks on two different PLCs . In our simulation, probably due to the limited library we used the attack had no particular effect. We tested on another simulation on which the attack did exactly what it was supposed to do. But as the important part is the final trace and not the effect it had on our simulation, this won't really matter for our tests.

Function Code Scan The function code scan attack looks like the DoS, we also performed two attacks on different MODBUS/TCP server to attack them. We got the same result as for the DoS attack, we saw no particular consequence on our simulation, probably for the same reason as the DoS attack. So we followed the same path and tested it on another simulation to verify that it worked properly.

6.2.3 Testing the attacks

Once we have generated traces for all of the attacks, we still had to use the IDS to notify the attacks that are considered to be dangerous. We used Snort and our whitelist IDS to generate the list of the packets that are considered to be malicious packets. Using the IP and the port of the attacker (that are gathered when attacking the system), we can categorize the packets in the above four categories (true negative, true positive,. . .).

To test the attack against the whitelist IDS, we used different levels of training set size to create the different rules for categorizing the packets.

To test the attack against the Snort IDS, we first changed the default configuration in order for Snort to detect which entity is what. We defined the IP address of the PLCs, the HMI and the control server, we did the same for the port of each component of network infrastructure. We find some rules published by the Digital Bond Labs[24]. This lab focus on testing ICS security. These rules identify unauthorized requests, malformed protocol requests and responses or dangerous commands that can possibly be malicious in the MODBUS/TCP protocol.

6.3 Results

Using the methodology defined above, we gathered results that we can now analyze. This part will be divided into two subsections. In the first part we will develop the different results we obtained, and then we will use the results to speak about IDS in the context of ICS.

6.3.1 Analysis of the Whitelist IDS

In this first part we will see the different results of our attacks on our whitelist IDS. The whitelist IDS will use a training set with different packets in order to create a database of acceptable packets and the other packets will be considered to be malicious packets.

The first attack we performed was the DoS attack (figure 6.2). On the graph we can see on the x axis the size, in terms of number of packets, of the training set and on the y axis the percentage of the total packets. The first thing we can see on the plot (and on all other graphs) is that we do not have any false negative packets. This is due to the fact that packets used for the training of the whitelist IDS do not contain any attack packets, since the attack occurred around 2000 packets. We tested different amount of packets for the training set and at one point it stabilized itself so we did not go further.

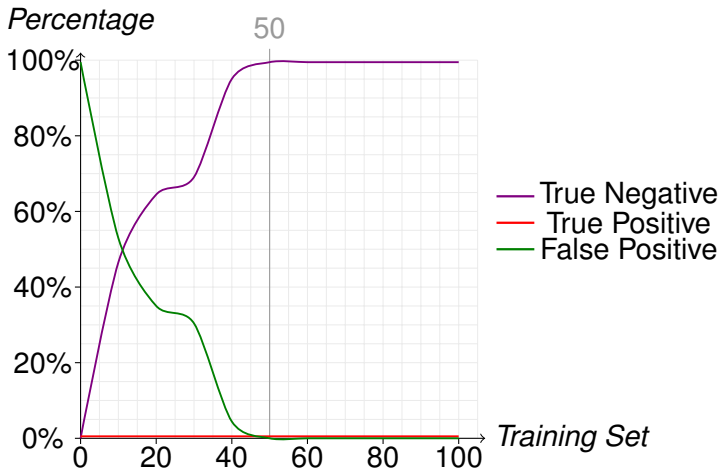

Figure 6.2: Denial of Service (DoS) attack

We can see that around 20 packets, the number of false positives is equal to the number of true negatives, because at this point half of the PLCs in the simulation have already sent packets to the control server. The two curves of the true negative and the false positive category are symmetrical because little by little all packets seen in the training set will be considered as legitimate packets as they should be. They will not be considered as malicious packets as they are seen, so they will be transferred from false positive category to the true negative category.

For the DoS attack, as it is only composed of a few packets trying to crash the PLC with a big packet or trying to call a function to stop MODBUS/TCP servers, we have very few packets used for the attack. All of them, even with no training packets, are recognized as attacks. This is for the same reason as explained above, we will never put attack packets in the training set with such small training set size. Our whitelist IDS will understand that it is an attack based on the unseen source IP address. But we could understand that it is an attack based on the special function used or on the packet size.

With no training packets, all packets will be considered as an attack. We have to reach 50 packets for the IDS to have no false positive packets. Because at this point the initialization phase of the simulation is over and the whitelist contains all the flows in the simulation. For a whitelist IDS it seems important to have no attack packets in the training set and a global view of all possible packets that might be used in the future for it to work as designed.

A DoS attack (when using special features as size or special function) is easily detected by a whitelist IDS. But an attacker can perform a DoS sending millions of packets to overflow the system with packets. If the attacker would use valid packets with possibly spoofed IP addresses you could perform a DoS attack that the IDS would not notice. In this kind of situation the IDS would be of no use.

The second attack we performed is the function code scan attack (figure 6.3). the first thing we noticed looking at the graph is the parallelism we can make with the DOS attack. The two graphs look very similar. The reason is that the function code scan attack, as for the DOS attack, will use a little number of packets very specific to perform an attack.

These two attacks tell us that any attack using specific function calls to act on the system in a way it was not supposed to do will not efficiently work against any whitelist based IDS. The only condition for them to work would be that during the training period the whitelist would see these function calls. But even if you cannot use special functions to attack the system you can still steal confidential informations, reading the plc registers for instance, as these functions are considered legitimate. If you can create attack scheme using only legitimate function you could theoretically bypass our whitelist IDS very easily.

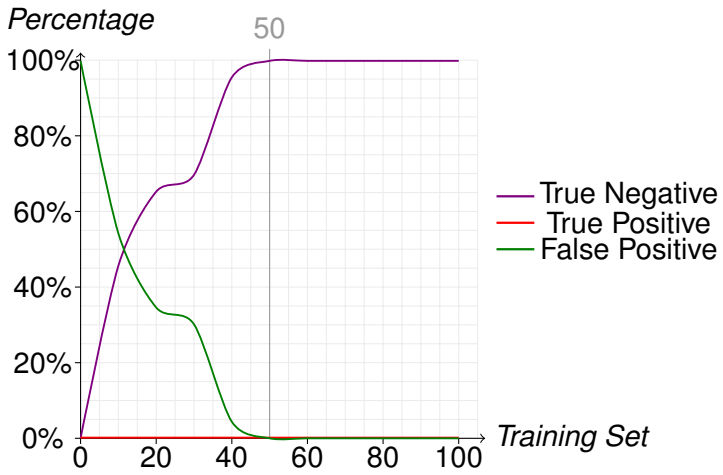

Figure 6.3: Function code scan attack

The last attack we performed, very different from the two other ones, is the scan attack (figure 6.4). We find the same features we could find in the two other attacks (symmetric curves, the system is completely trained around 50 packets, and curves crossing around 20 packets). With a single difference, as the scan sends way more packets than the two other attacks, it has detected almost 45% packets of attack that are valid. The interpretation on the plot are the same as on the other two attacks.

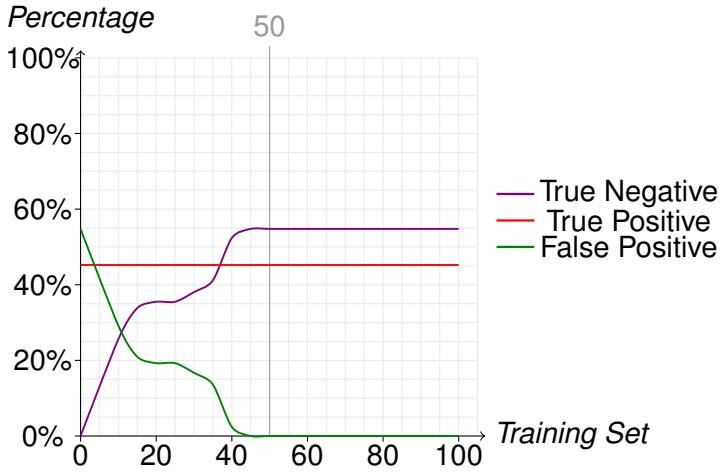

Figure 6.4: Scan attack

A scan attack will probably be always easily detected by a whitelist IDS, for the simple reason that the main principle of the scan attack is to send message on all possible ports to detect services provided by an host in the network. And the different port will never be in the whitelist training set as they are completely unused in the system.

Based on these attacks and on the whitelist IDS, what can we say about the feasibility of simulating attacks against IDS. It seems that it grants us good results that can provide us an interpretation about known attacks and give us the behaviour of the IDS against these attacks. If an engineer would consider using that IDS to protect its own industrial control system, what could he conclude with the results we obtained? He could take measures to protect the system with the packets that pass the IDS and that are not valid packets. When training whitelist IDS, it is almost impossible to have 100% of the packets in the valid rules (or you might add attacks in the set). Our simulation does not give us cases where a valid packet might be rejected when using the correct size training set. It is the only limitation that we can oppose to our technique.

But even with this limitation, the information gathered can be really useful to protect against malicious packets.

A last question to ask before moving to Snort IDS, is why we have not included attack packets in our training sets. The reason is based on the types of attack we selected. For a scan attack, it would not change much to include malicious packets in the training set as every attack packets are different. And for the two other attacks, include very few packets would mean to include enough packets to permit the attack to occur. Indeed, there are only 14 and 5 malicious packets for the DoS and function code scan attack.

6.3.2 Analysis of Snort IDS

Now we are going to test our 3 attacks against another IDS: Snort. To recall, Snort is a signature based IDS, meaning that we have to specify the signature of different existing attacks in order for the IDS to recognize them. We also had to configure it such that it could recognize the control server among the different TCP connections.

In the figure 6.5 we can see the result we obtained using Snort IDS. There is not much difference with our own IDS. The only difference we can see is the presence of false negative during te DOS and function code scan attack. It is probably that one packet in the connection establishment was said to be a valid packet by the IDS. Also, the rule used in Snort to detect function code scan counts the number of exception code it has observed during a period. If Snort counts three packets containing a exception code equals to 1 within one minute, it considers that a function code scan is occurring. We set up our attacks so that this rules can be bypassed by increasing the time between probes.

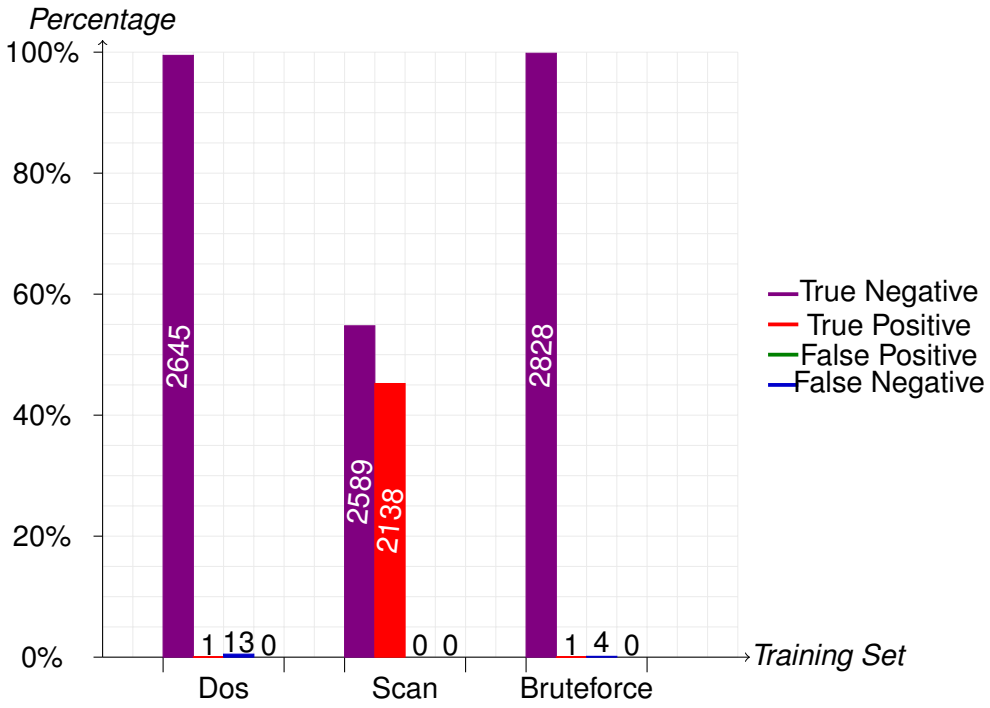

Figure 6.5: Snort intrusion detection

The fact that there is no real difference, for the chosen attacks, between a signature and a behaviour based IDS (our whitelist IDS), is easily explainable by the nature of those attacks. The DOS attack and function code scan attack can be easily stopped as Snort can be configured to stop particular function calls and/or packet size. And the connection establishment done during

the scan attack can also be stopped as we configured the port that are used by the different connection between the PLC and the control server.

6.3.3 IDS in the context of ICS

Based on the results we have obtained, we can tell that performing attacks on IDS is a success, but what information does it provide us on the IDS itself? And what implication does it have in the context of ICS?

For the first question we can really see that performing simulation in order to test IDS is a good way to behave when trying to protect ICS. Using this method we can, in a safe environment easily predict the behavior of the IDS against known attacks.

For the second question, an idea, following this path, would be to create an international database of attack traces, such that industrial system designers could test their IDS before adding them to their system. But there is one big issue with this idea. For very generic attack (such as the one we analyzed) the traces can be easily generated, but each system can be very different, such that each system can suffer from very different vulnerabilities. It would seem hardly realizable to have all the attacks possibilities covered using a database. So we have to conclude that for ICS, we have to test IDS locally on every different ICS.

But is it realizable considering the amount of work and effort which are required to create simulation for the system. The solution would be to have an easy to use and completely generic simulation to work with in order to test the system.

It should be a classical methodology to simulate attacks and try to constantly identify attacks scheme or to discover them, in order to predict intrusion before they happen. The simulation in this view is seen as an excellent tool to prepare against these intrusions.

6.4 Further Work

After the analysis of the different attacks against IDS, we do have a clear view of the limitations of our methodology and our system and the further works that could be started based on the observations we made.

6.4.1 Limitations

We could see 3 main limitations, important enough to be pointed out:

1. Environment: A big limitation of our methodology, is the system that has been used. Even though our simulation is a great way to perform attacks on a small level. If we want to simulate more complex attack construction on ICS, it would be necessary to have a fully supported MODBUS/TCP library, working upon a complex simulation of ICS, with real monitoring system and control server.

2. Network: A second limitation, which is tightly linked to the first one is the network. Our whole construction is based on local network, such that all PLCs are simply placed on different port on the same IP. A real simulation should have server working with different PLC that could be connected either with different machines or over a VLAN. This would give us possibility to perform man-in-the-middle attack or to perform other kind of network attacks.

3. Methodology: The methodology we used, having to modify the IP address and the port of the PLC and having to recognize the attacker based on its IP address and port is also a big limitation. We cannot perform any attack where the attacker would spoof its IP address because we couldn't interpret the results of the IDS when categorizing the packets in one

of the four categories (true negative, true positive,... A solution would be to have a real computer for the attacker and keep track of the packets coming from this user, that would certainly be an improvement.

6.4.2 Going further

We have also different elements that we did not have the time or need to do, because they were out of the field of this work, that could be improvement of our work.

- Attacks: There are a lot of possible attacks that can be added to our test from crashing the system to retrieve periodically the data from the registers of the PLCs.
- IDS: We have only tested two IDSs with our system, but a lot of other approaches might be implemented. However, as the purpose of the work is to test IDS of ICS, it could be interesting to have a real IDS, currently operating in an ICS, being tested to see the capability of our method.
- Simulation: The simulation can be easily improved, a big improvement we have thought of (and that could be the subject of a complete other work) would be to have a generic simulation with a user interface to easily modify the simulation and define the logic behind the PLCs and the environment it is running on.

Chapter 7

Conclusion

The goal of our work was to generate Industrial Control System network traces containing attack traffic in order to evaluate Intrusion Detection System in these kinds of network. These traces are needed because obtaining trace from real ICS operators or companies is hard as they do not want to publish them because of security concerns. Our work covers two different subjects: IDSs in ICSs and simulation of ICSs.

For the simulation, we mainly focus on the MODBUS/TCP protocol as it is an open standard protocol that is widely used in SCADA network. The simulator represents coal power plant and we used it to generate the traffic. We implemented some attacks that we run on the simulator in order to get malicious traffic in our traces.

We implemented an IDS based on whitelists. The IDS uses two whitelists, one for the flow in the network and one for the function code used. The IDS built the two whitelists thanks to a training set. It takes advantage of the fact that ICS networks are much more stable compared to corporate network. Meaning that, changes in the topology of the network are less frequent. We also used Snort, an open source signature based IDS. Snort and the IDS we have implemented used different approaches to detect attack so we availed of this difference to see whether or not our traces can be used to evaluate IDSs.

Results show that attack simulation to evaluate how an IDS will behave against known and unknown attack. They also demonstrate that the training set size for our IDS has major impact regarding the accuracy of this IDS. They also showcase the pros and the cons of the approach used by each IDS. Our IDS is highly dependent of the training set size and Snort is not able to detect attack which it is unaware of.

As for future work regarding IDS evaluation in ICS, the simulation should be made more generic. It should encompass more protocol so that more attacks exploiting vulnerability of these protocols could be implemented.

We want to convince the reader that protecting ICSs is a hard task because of its constraint and properties. But because of the consequence a failure can cause, efficient IDS are needed to detect attack and possibly prevent it.

Bibliography

[1] European Network and Information Security Agency (ENISA), *Protecting Industrial Control Systems*, December 2011

[2] E. Hayden, M. Assante, T. Conway, *An Abbreviated History of Automation & Industrial Controls Systems and Cybersecurity*, A Sans Analyst Whitepaper, August 2014

[3] National Institute of Standards and Technology (NIST), *Guide to Industrial Control Systems*, June 2011

[4] http://libmodbus.org/, *Libmodbus*

[5] A. Scott, *Key Differences between DCS and SCADA Systems* http://blog.cimation.com/blog/dcs-vs-scada-systems, April 2012

[6] Modicon, Inc., Industrial Automation Systems, *Modicon Modbus Protocol Reference Guide*, June 1996

[7] H. Debar, *An Introduction to Intrusion-Detection Systems*, Proceedings of Connect 2000, May 2000

[8] B. Galloway, Gerhard P.Hancke, *Introduction to Industrial Control Network*, IEEE Communications Surveys and Tutorial, Volume 15, Issue 2, July 2012

[9] R.R.R. Barbosa, A. Pras, R. Sadre, *Flow Whitelisting in SCADA Networks*, Springer, March 2013

[10] R.R.R. Barbosa, A. Pras, R. Sadre, *A First Look into SCADA Network Traffic*, 2012 IEEE Network Operations and Management Symposium, April 2012

[11] R.R.R. Barbosa, A. Pras, R. Sadre, *Difficulties in Modeling SCADA Traffic: Comparative Analysis*, Springer Berlin Heidelberg, March 2012

[12] E.J. Byres, M. Franz, D. Miller, *The use of attack trees in assessing vulnerabilities in SCADA systems*, International Infrastructure Survivability Workshop (IISW'04), Institute of Electrical and Electronics Engineers, December 2004

[13] Cheung S., Dutertre B, Fong M., Lindqvist., Skinner K. Valdes A., *Using Model-based Intrusion Detection for SCADA Networks*, Proceedings of the SCADA Security Scientific Symposium, January 2007

[14] R.U. Rehman, *Intrusion Detection Systems with Snort Advanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID*, Prentice Hall; 1 edition, May 2003

[15] D. Yang, A. Usynin, J. Wesley Hines, *Anomaly-Based Intrusion Detection for SCADA Systems*, 5th intl. topical meeting on nuclear plant instrumentation, control and human machine interface technologies (npic&hmit 05), November 2006

[16] W. Gao, T. Morris, B. Reaves, D. Richey, *On SCADA control system command and response injection and intrusion detection.*, eCrime Researchers Summit (eCrime) 2010, IEEE, October 2011

[17] N. Goldenberg, A. Wool, *Accurate Modeling of Modbus/TCP for Intrusion Detection System*, International Journal of Critical Infrastructure Protection, Volume 6, Issue 2, June 2013

[18] I.N. Fovino, M. Masera, M. Guglielmi, A. Carcano, A. Trombetta, *Distributed Intrusion Detection System For SCADA Protocols*, Critical Infrastructure Protection IV, Springer Berlin Heidelberg, March 2010

[19] J. Seidl, *VirtuaPlant*, https://github.com/jseidl/virtuaplant

[20] T. Carstens, G. Harris, *PCAP*, http://www.tcpdump.org/pcap.html

[21] T.D. Hanson, *uthash*, https://troydhanson.github.io/uthash/userguide.html

[22] Eternally Confuzzled, *Hashing*, http://eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx

[23] SecDev, *Scapy*, http://www.secdev.org/projects/scapy/

[24] Digital Bond, *Quickdraw SCADA IDS* http://www.digitalbond.com/tools/quickdraw/

[25] *Snort*, https://www.snort.org/

[26] *The Bro Network Security Monitor*, https://www.bro.org/

[27] IGI Global Disseminator Of Knowledge, *What is Traffic Profile*, http://www.igi-global.com/dictionary/traffic-profile/30373

[28] *Tripwire*, http://www.tripwire.com/

[29] OSSEC Project Team, *Open Source HIDS SECurity*, http://ossec.github.io/

[30] Dell Software, *Dell Security Annual Threat Report*, 2015

[31] ICS-CERT, *Incident Response/vulnerability coordination*, September 2014 - February 2015

[32] Symantec Corporation, *TCP MODBUS - Illegal Packet Size*, https://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=20676

[33] E. Byres, J. Carter, A. Elramly, D. Hoffman, *Worlds in Collisions Ethernet and The Factory Floor*, Emerging Technologies 2002, 2002

[34] B. Zhu, A. Joseph, S. Sastry, *A Taxonomy of Cyber Attacks on SCADA Systems*, Internet of things (iThings/CPSCom), 2011 international conference on and 4th international conference on cyber, physical and social computing IEEE, October 2011

[35] Igure V.M., Laugther S.A., Williams R.D, *Security issues in SCADA networks*, Computers & Security, Elsevier Advanced Technology, October 2006

[36] K. Barnes, B. Johnson, R. Nickelson, *Review of Supervisory Control and Data Acquisition Systems*, Idaho National Engineering and Environmental Laboratory, January 2004

[37] *Scadasim: issue#1* https://github.com/caxqueiroz/scadasim/issues/1

[38] J.-C. Baptiste, *Introduction à la sécurité SCADA*, Korben, http://korben.info/securite-scada.html, 10 June 2015

[39] XMCO, *Stuxnet: Analyse, Mythes et Réalités*, ActuSécu no.27, February 2011

[40] R. Langner, *To Kill a Centrifuge: A technical Analysis of What Stuxnet's Creators Tried to Achieve* Langner, November 2013

[41] M.J. Assante, R.M. Lee, *The Industrial Control System Cyber Kill Chain*, Sans Institute - InfoSec Reading Room, October 2015

[42] McAfee, *Global Energy Cyberattacks: "Night Dragon"*, McAfee Foundstone Professional Services and McAfee Labs, February 2011

[43] The Engineering ToolBox, *Standard Grade Coal - Heating Values*, http://www.engineeringtoolbox.com/coal-heating-values-d_1675.html

[44] K. Mathioudakis, N. Frangiadakis, A. Merentitis, V. Gazis, *Towards generic SCADA simulators: A survey of existing multi-purpose co-simulation platforms, best practices and use-cases*, AGT Group (R&D), 2013

Appendix A

Simulation

This appendix will contain all instructions that are used to manage the simulation. The first part will tell you how to run the simulation and the second one will give you all commands that are used with the simulation. Do not forget to go through the installation appendix before trying to use the simulation.

A.1 Installation

For the simulation you will need to install the Libmodbus library, as it is the library we use to have ModbusTCP support.

To install libmodbus, you can either go to the website and follow the instructions to configure it, or you can go into your debian package manager and download libmodbus-3.0.5 (the version we used).

Once you have downloaded libmodbus, in order to be able to compile our code, you will possibly have to modify the linkage in the make file. Run the following command to have the path of the libmodbus library on your system.

```
$sudo apt-get install pkg-config
$pkg-config --cflags -libs libmodbus
```

Then go in *src/tracegen/Makefile* and change the **INCLUDE** variable with the proper path.

A.2 Build and Run

To build the simulation simply use the Makefile. You can clean all files using the same Makefile.

```
$ make
$ make clean
```

Then you will need to launch two instances of the terminal. In the first instance you will first launch all the processes of the PLC with the *server* program and then you will launch the instances of the control server with the *master* program. If you stop the control server, it will automatically stop the PLC processes.

```
$ ./server
$ ./master
```

At this point, the simulation has not started yet. You need to launch the HMI for it to begin. To launch the HMI, simply use on the other terminal instance the *hmi* program.

```
$ ./hmi
>
```

If you look at the terminal instance running the control server, you can see informations about the running simulation. These are debugging information, that a normal user would not see. On the HMI terminal, you will get a character '>' telling you that you can now enter instructions.

A.3 Commands

Now that the simulation is running you will need instructions to manage it, in this part we will give you a quick explanation of all instructions. There is the list of all instructions.

help display the help, concretely, it displays all the available commands

start start the simulation and set all variables for the power plant to produce 20% of electricity

stop stop the power plant

quit quit the simulation

get display the content of one register, you will have to specify a register

- dropped amount
- current rack
- furnace temperature
- furnace coal
- cuve water
- cuve steam
- cuve temperature
- turbine rotation
- battery charge

status display the status of the whole power plant

drop drop the amount of coal specified

rotate rotate the coal bringer

limit rotation limit the rotation speed of the turbine (0 to 100)

limit battery limit the battery content (0 to 100)

flow water flow water in or out the cuve

constraint create a constraint on one parameter of the power plant, you will have to define first the parameter and then the integer value

- COAL
- FURNACE_TEMP
- WATER_TEMP

- WATER
- STEAM
- TURBINE
- BATTERY

A.4 Modifications

In this section we are going to explain the different parts of the program that have to be changed in order to modify the simulation.

A.4.1 PLC

We have a structure that is fixed for the PLC. They are two functions. One that constitute the operation of the process running the PLC and one function used to update the logic of the PLC. The function for the process is often the same, it listens and answer to ModbusTCP query and sometimes initializes state variables. To add PLC, create your own functions.

Below we can see the prototypes of these functions, both the operations and the update.

```
// powerplant.h
...
// Operations
void op_coaldoorplc(modbus_t* ctx, modbus_mapping_t* map, int rfd, int wfd);
void op_coalbringerplc(modbus_t* ctx, modbus_mapping_t* map, int rfd, int wfd);
void op_furnaceplc(modbus_t* ctx, modbus_mapping_t* map, int rfd, int wfd);
void op_waterplc(modbus_t* ctx, modbus_mapping_t* map, int rfd, int wfd);
void op_turbineplc(modbus_t* ctx, modbus_mapping_t* map, int rfd, int wfd);
void op_batteryplc(modbus_t* ctx, modbus_mapping_t* map, int rfd, int wfd);

// Update functions
void rotation_over(int signal);
void updateFurnace(int signal);
void updateWater(int signal);
void updateTurbine(int signal);
void updateBattery(int signal);
...
```

Below you can see the structure of the operation functions. Almost all of the functions looks like this one. It listens to the Modbus query and answer to them. And it defines an alarm for the update function.

```
//powerplant.c
...
void op_batteryplc(modbus_t* ctx, modbus_mapping_t* map, int rfd, int wfd) {
  // Alarm variables
  _map = map;
  _rfd = rfd;
  _wfd = wfd;
  _lastUpdate = time(NULL);
  _stateA = 0;
  // Set alarm
  signal(SIGALRM, updateBattery);
  alarm(DEFAULT_UPDATE_TIME);
  while(TRUE) {
    uint8_t request[64];
    int request_len = modbus_receive(ctx, request);
    if(request_len == -1)
```

```
17          exit(-1);
        modbus_reply(ctx, request, request_len, map);
19    }
  }
21 ...
```

When the functions of the PLC have been defined you can also modify the number of registers for each type of register and initialize their content with the functions that are below.

```
1 // powerplant.h
  void getOp(int constant, modbus_t* ctx, modbus_mapping_t* map, int rfd, int wfd);
3 int getCoil(int constant);
  int getHolding(int constant);
5 int getInput(int constant);
  int getDiscrete(int constant);
7 void init_map(modbus_mapping_t* map, int constant);
```

The PLC process are launched in the *server.c* file, they are launch using a *plc()* function which is defined below. You will have to add a constant for your PLC to be launched, because the constant is used to increment the port on which the Modbus server will be opened.

```
1 // Create a new plc
  // @param in_pipe input pipe couple
3 // @param out_pipe output pipe couple
  // @param constant the constant that will be added to the PORT to make it unique
5 // @param reg the parameters for the modbus_mapping_t, the amount of register of the
      4 types
  // @param op the function for operations;
7 void plc(int in_pipe[2], int out_pipe[2], int constant);
```

Another modification will occur on the control server, you will have to connect the control server to the PLC. Finally, you have to send the proper pipe to the proper PLC to be able to give to the PLC the possibility to communicate between each others.

A.4.2 Control Server and HMI

The control server demands not much modifications. You will have to connect the server to every new PLC you have to rule. The control server will handle the logic of the system, you can modify the constraint system and the action system to work with brand new elements that have been defined for your requirements.

The HMI will have to be modified according to the modifications in the control server. By comparing string you can easily add commands in the HMI, you will find a way easily in the code.

Finally, there are controller functions that have been defined in two files, one for the HMI to communicate with the control server (*hmi.c*) and one for the control server to communicate with the PLC (*controller.c*). You can add and remove functions easily following the different models that are implemented.

Appendix B

Attacks - Command Line Interface

B.1 Requirements

To run the Command Line Interface to launch the attack, you will need Python and the following modules:

- python-nmap 0.4.4
- scapy

B.2 Commands

To run the CLI, you must use the following command:

```
#sudo python interpreter.py
```

The command will change and display the text Modbus attack tool. Now we explain command that you can use to launch an attack:

- To print the attacks implemented.

```
>> show exploits
```

- To select an exploit

```
>> select <exploit-name>
```

- Each exploit has a set of parameter used to specify the kind of packet to forge. You can see the parameters and set them:

```
(exploit-name)>> show parameters
(exploit-name)>> set <parameter> <value>
```

- Then finally to run the exploit:

```
(exploit-name)>> run
```

B.3 Attack Addition

To add a new attack, you must create a python file in the *script* directory. There is a template file called *simple.py* that you can use as a reference.

First, you must import needed module:

```
from _exploit import Exploit
from _constants import CONST
```

To add your attack, you must create a subclass of `Exploit`. The name of the class must finish by "Exploit" and have the same name of the file with the first later in uppercase. For example, if your file has the name *simple.py*, the name of class will be `SimpleExploit`.

The each exploits have a parameters stored in a dictionnary. The key of this dictionnary are in the `_constants.py` file. We use this file, to have coherence naming betwenn the exploits.

```
# in constants.py
class CONST(object):
    SRCIP = 'SRCIP'
    SRCPORT = 'SRCPORT'
    DESTIP = 'DESTIP'
    DESTPORT = 'DESTPORT'
    PROTO = 'PROTO'
    ...
```

Then you can then specify the paramaters of the exploit in initializing function of the class.

```
class SimpleExploit(Exploit):
    CONST = CONST()

    def __init__(self):
        self.params = {CONST.SRCIP: None,
                       CONST.SRCPORT: None,
                       CONST.DESTIP: None,
                       CONST.DESTPORT: None,
                       CONST.PROTO: None}
```

There are two functions, that you must implement:

- run(self): This will be like the main function of the exploit. This function will be called when using the command `run` in the CLI.
- __str__(self): This function is used to print the name of the exploit in the CLI.

To ease the forgery of MODBUS packets, we provide the file `_modbus.py` that contains basic sturcture of payload for MODBUS packets.

```
# in _modbus.py
class ModbusADU(Packet):
    name="ModbusADU"
    fields_desc=[
        ShortField("transactionID",0),
        ShortField("protocolID",0),
        ShortField("length",6),
        ByteField("unitID",0)
    ]
```

```
class ModbusSinglePDU(Packet):
    name="ModbusSinglePDU"
    fields_desc=[
        ByteField("funcode",1),
        ShortField("address",0),
        ShortField("numreg",1),
    ]
```

To forge the packet:

```
#example to use function code 3
payload = (ModbusADU()/ModbusSinglePDU(funcode=3))
```

Appendix C

Intrusion Detection System

C.1 MODBUS IDS

The IDS requires:

- slog
- Libpcap
- uthash

It uses whitelists and need a training trace to build these whitelists. Traces must be pcap files. To run the IDS:

```
\$ ./sniffer -f <training_trace> -m <monitoring_trace> -p <modbus_port> -s <
    number_plc>
```

- <training_trace>: The trace used to build the two whitelists.
- <monitoring_trace>: The trace that the IDS will monitor to detect attacks.
- <number_plc>: Number of PLC. We used this number because the simulation run on a local host and the PLCs are represented by different processes using consecutive ports. If you set up your simulation such that PLCs use the same port, set it up to 0.
- <modbus_port>: smallest port used by PCLs if they used consecutive ports. If they use the same port number, then its that port.

C.2 Snort

In order to use Snort with our traces, you will need to do some modifications. You must overwrite the `/etc/snort/snort.conf` and `/etc/snort/rules/scan.rules` by the file present in the repository. Then you must change the following variable according to your network configuration:

- MODBUS_SERVER: IP(s) used by the PLCs
- MODBUS_CLIENT: IP used by the control server
- MODBUS_PORTS: Port(s) used by the PLCs
- CONTROL_PORTS: Port(s) used by the control server

Finally you must add the file `modbus_1_2.rules` in the `/etc/snort/rules/` directory.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**