A Sandbox for Industrial Control Systems (Security)

For CyberSecurity Research

MEGHADIO GONZEU Cedric Jordan,

NKOM TAKEU Guy Junior

Master thesis submitted under the supervision of
Ramin Sadre

the readers:
Gorby Kabasele, Mohsen Shirali

in order to be awarded the Degree of
Master in Cybersecurity
Corporate Strategies/System Design

Academic year
2024 – 2025

We hereby confirm that this thesis was written independently by ourselves without the use of any sources beyond those cited, and all passages and ideas taken from other sources are cited accordingly.

The authors give permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this master dissertation.

The authors transfer to the project owners any and all rights to this master dissertation, code and all contribution to the project without any limitation in time nor space.

01/06/2025

Title: A Sandbox for Industrial Control Systems (security)

Author: MEGHADIO GONZEU Cedric Jordan,

NKOM TAKEU Guy Junior
Master in Cybersecurity – Corporate Strategies / System Design
Academic year: 2024 – 2025

Abstract

Industrial Control Systems (ICS) are integral to the operation of essential infrastructure, yet they are increasingly facing growing cybersecurity challenges. The integration of operational technology with IT, the use of old systems, and growing connectivity all increased the attack surface, exposing the ICS environment to potential cyber threats. These vulnerabilities can result in major disruptions, safety threats, and economic loss. The work is an extension of a testbed for a virtual ICS that had been previously established, augmenting its functionality to simulate practical cyberattack scenarios in a controlled environment. The ultimate aim is to apply the testbed to discover and study potential security vulnerabilities of ICS configurations, thus enabling a better comprehension of threat vectors and system vulnerabilities. Through the simulation of different attack scenarios, the testbed is a useful tool for research and education, and one that can be used to experiment with countermeasures without endangering actual industrial operation. During the project, we will implement several attack simulations against various parts of the ICS architecture. Their performance under the testbed environment in regard to process disruption, detectability, as well as their potential use in education and research, will be evaluated. The testbed thus acts as a pragmatic, flexible instrument to aid cybersecurity research and training in the domain of ICS.

Keywords: ICS CYBERSECURITY TESTBED SIMULATION ATTACKS RESEARCH

Preface

This thesis represents more than just the completion of our academic journey; it is the result of many late nights, moments of doubt, and small victories that shaped not only our knowledge, but also who we are becoming as researchers. What lies in these pages is the continuation of a vision initiated by Yi Zhu, a former student whose foundational work laid the groundwork for this project and encouraged us to delve deeper into the world of Industrial Control Systems and their security.

When we first encountered the idea of simulating cyberattacks in a virtual ICS environment, it seemed both intimidating and fascinating. We stepped into a system built by someone else, a digital structure carrying hours of thought and dedication. Our task was to not only understand it, but to extend it, challenge it, and reshape it with new meaning. We didn't just inherit a project; we inherited responsibility.

Through this work, we hoped to answer a bigger question: how can we use technology, safely and ethically, to prepare others for the threats that exist in the industrial world? How can this testbed become a space for learning, for experimentation, and for preparing tomorrow's defenders?

This journey was not one we walked alone. We are deeply grateful to our supervisors, who offered both guidance and patience, and to the teachers who never failed to believe in our potential even when we questioned it ourselves. We thank Yi for the quality of his work and the openness it offered for exploration. And to our families and friends: thank you for standing behind us, especially when we disappeared into our code, our diagrams, and our silence.

This thesis is a chapter in a longer story one that will, we hope, continue in labs, classrooms, and conversations yet to come.

Acknowledgements

We would like to express our deepest gratitude to all those who supported us, directly or indirectly, throughout the writing and completion of this thesis.

First, we extend our sincere thanks to Pr. Jérôme Dossogne for providing the beautiful and professional template that shaped the structure and appearance of this document. His work greatly contributed to presenting our research in a clear and polished manner.

We are truly grateful to Mr. Gorby Kabasele, our supervising researcher, for his guidance, encouragement, and thoughtful insights throughout the development of this project. His availability and critical feedback played an essential role in shaping both our technical decisions and our academic mindset.

Our heartfelt thanks also go to Pr. Ramin Sadre, our academic supervisor, for overseeing this work and for his availability when needed during the key phases of the thesis.

We would also like to thank Mr. Mohsen Shirali, who kindly accepted to serve as a reader of this thesis. His time and attention are greatly appreciated.

Lastly, and most importantly, we thank our families and friends for their unwavering support, patience, and encouragement throughout this journey. Their presence, even in silence, reminded us that we were never alone in this effort.

To all of you, thank you for being a part of this chapter in our academic lives.

AI Usage Disclaimer

During the writing process of this thesis, we made limited use of OpenAI's ChatGPT to improve the clarity, coherence, and grammatical quality of certain passages. This tool was employed as a writing assistant, similar to grammar-checking software, and did not generate original scientific content, analysis, or conclusions.

All research, implementations, interpretations, and critical discussions presented in this thesis are entirely our own. We remain fully responsible for the academic integrity and originality of this work.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Industrial Control Systems (ICS) are the backbone of critical infrastructure and industrial operations. From energy production and water treatment to transportation and manufacturing, ICS orchestrate the automated control of physical processes through the integration of programmable logic controllers (PLC), supervisory control and data acquisition (SCADA) systems, human-machine interfaces (HMI) and a variety of sensors and actuators [4]. Although these systems were traditionally isolated and built for reliability and availability, their growing interconnectivity, especially with IT networks, has introduced significant security challenges [4][2].

During the past decade, the cybersecurity landscape of ICS has evolved dramatically. Attacks like Stuxnet in 2010, which targeted Iranian nuclear facilities through manipulation of PLC, demonstrated the devastating impact of cyber operations against physical infrastructure [62]. More recently, ransomware incidents such as the Colonial Pipeline attack in 2021 highlighted how vulnerabilities in ICS-adjacent networks can lead to national-level disruptions [33]. Despite increasing awareness, many ICS installations continue to rely on legacy systems with outdated protocols, hardcoded credentials, and insufficient authentication or encryption mechanisms [7][15][32].

Research in ICS security is hampered by several practical constraints. Due to the high cost of industrial equipment and the risks associated with experiments in real infrastructure, access to physical testbeds is limited. In addition, many organizations are reluctant to share data or system details because of confidentiality and liability concerns. These limitations slow the development of tools and techniques to better secure ICS environments [18].

To address these barriers, simulation-based testbeds have emerged as a valuable alternative for cybersecurity research and training. Virtual ICS platforms allow controlled experimentation, repeatability of attack scenarios, and safe exploration of vulnerabilities without the risk of physical damage or operational downtime [7]. These testbeds also support education, giving students and researchers the opportunity to gain hands-on experience with ICS components and attack simulations.

This thesis builds on the work of Yi Zhu [63], who developed a lightweight Docker-based ICS simulator capable of modeling networked PLCs, HMI systems, and physical processes using OpenPLC and Modbus/TCP. Although the initial focus of the project was on the creation of a functional and flexible ICS sandbox, our contribution extends its application to the field of cybersecurity research and education. In this work, we demonstrate how this simulator can be used to emulate realistic cyberattack scenarios in ICS environments, evaluate their impact, and propose strategies to detect or mitigate such threats.

The remainder of this thesis explores the theoretical background of ICS and cybersecurity frameworks, develops threat models, implements various attack scenarios, and discusses the implications of using a virtual ICS testbed for research and training purposes.

1.1 Motivations

Cybersecurity in ICS is no longer a theoretical concern, it is a real and growing threat. From ransomware attacks on pipelines and hospitals to targeted sabotage of physical equipment, modern ICS environments are being increasingly exposed to cyberattacks with the potential to cause catastrophic damage to public services, economies, and human safety [4][24]. However, as much as the risks are escalating, the tools and platforms available for ICS security research remain limited in scope, accessibility, and adaptability.

We chose to work on this subject because we believe in the urgent need for realistic, accessible, and reproducible test environments that can help researchers, students, and engineers understand the risks, behaviors, and potential mitigations of ICS-targeted cyberattacks. Our motivation stems from the observation that many existing ICS testbeds fall into one of two categories:

- **Hardware-based setups,** which are expensive, difficult to replicate, and limited in scale [19], or
- **Static software-based testbeds,** which lack flexibility, real-time physical process simulation, or the ability to integrate diverse attack scenarios [7][21].

Several studies have underlined the lack of low-cost, fully virtualized testbeds capable of both realistic threat modeling and adaptable scenario generation [52]. While notable efforts like GRFICS [29], SWaT [31], and ICSSIM [21] have paved the way for simulation-based research, these platforms often require substantial computing resources, are tied to specific processes, or lack extensibility.

Our work builds upon Yi Zhu's project, which laid the foundation for a lightweight, modular ICS simulator. By expanding on his architecture, we bring a new focus to security testing and attack emulation, enriching the educational and research potential of the platform. Our simulator is not tied to a single industrial process and supports a fully customizable configuration via YAML, enabling users to model different system architectures, physical behaviors, and sensor-actuator interactions.

The added value of our work lies in:

- Its **ease of deployment** using Docker and Vagrant.
- Its ability to **simulate complex physical processes** simulate complex physical processes in real-time using lightweight Python logic.
- The **integration of common attack vectors** and their visualization through SCADA interfaces.
- Its **reproducibility and openness**, allowing anyone to replicate and extend the testbed for their own research or training purposes.

In essence, this project provides a scalable and realistic ICS simulation platform tailored for cybersecurity training, attack emulation, and defense strategy development all without requiring costly hardware or restricted lab access. In doing so, we hope to lower the barrier of entry for ICS security research and contribute a versatile tool to the cybersecurity community.

1.2 Problem Statement

The increasing digitization of industrial systems has introduced a new set of risks to environments that were traditionally isolated and physically secured.ICS, which form the operational core of critical infrastructure, such as energy grids, water treatment facilities, and manufacturing lines were not originally designed to withstand cyberattacks [40]. Many ICS still rely on legacy hardware and protocols that lack fundamental security features such as encryption, authentication, or access control [4].

This lack of preparedness, combined with rising connectivity and convergence with IT systems, creates a dangerous vulnerability gap. Attacks like Stuxnet, Triton, and BlackEnergy have shown that ICS environments are not only targets but also susceptible to disruptions that can have physical, safety, and economic consequences [62] [47]. Recent reports, such as Dragos's "ICS/OT Cybersecurity Year in Review" [24], emphasize that both nation-state and financially motivated threat actors are increasingly targeting ICS sectors.

Despite the urgency, ICS security research is hindered by two main factors: the high cost and risk of working with real ICS infrastructure, and the scarcity of open, reproducible testbeds for hands-on experimentation and learning. Most existing platforms are either tied to proprietary equipment, lack flexibility, or are inaccessible to students and researchers due to their complexity or licensing restrictions.

There is a clear need for a lightweight, open, and modular ICS simulator that supports realistic process behavior and allows the implementation of cyberattacks in a safe and controlled environment. Our project addresses this gap by extending an existing virtual testbed and focusing on its usefulness in cybersecurity education, attack modeling, and defense testing.

1.3 Related Works

The field of ICS security testbeds has evolved significantly over the past decade, with contributions ranging from hardware-based physical testbeds to fully virtualized environments. To position our work within this landscape, we review key developments in ICS security testbeds across various implementation approaches.

1.3.1 Physical Testbeds

Physical testbeds offer high-fidelity environments that closely mirror real-world industrial systems. The Secure Water Treatment (SWaT) testbed developed by the iTrust lab at Singapore University of Technology and Design represents one of the most comprehensive physical ICS testbeds (Goh et al., 2017) [31]. This testbed replicates a full six-stage water treatment process with 51 sensors and actuators, generating valuable datasets under both normal operations and attack scenarios. Similarly, national-level initiatives like the U.S. Department of Energy's National SCADA Test Bed [49] and Japan's Control System Security Centre testbed provide extensive hardware environments for security research [38].

While these physical testbeds offer excellent realism, they come with significant drawbacks high acquisition and maintenance costs, limited accessibility, and restricted scalability. These limitations make them inaccessible to many researchers and educational institutions, creating a gap that virtual solutions aim to fill.

1.3.2 Hybrid Testbeds

Hybrid approaches combine physical hardware with simulation elements to balance fidelity and cost. LICSTER [28] stands out as a low-cost ICS security testbed that integrates OpenPLC running on Raspberry Pi with physical components like a Fischertechnik punching machine. Priced at approximately 600 euros, it aims to make ICS security education more accessible.

Similarly, Faramondi et al. (2021) [41] developed a hardware-in-the-loop Water Distribution Testbed that connects real hardware components with simulated subsystems implemented using the minicps framework. This hybrid approach enables researchers to validate machine learning-based intrusion detection systems using both network and physical datasets.

1.3.3 Virtualized Testbeds

Fully virtualized testbeds offer the greatest flexibility, portability, and cost-effectiveness, though often at the expense of some fidelity. Several notable implementations have influenced our work.

GRFICS (Formby et al., 2018) [29] provides an open-source graphical ICS simulation tool based on the Tennessee Eastman process, using OpenPLC for controller simulation and AdvancedHMI for the human-machine interface. VISCORT (Ekisa Conrad et al., 2022) [17], an extension of GRFICS, improves resource efficiency by using lightweight container hypervisors and adds attack scenario support.

ICSSIM (Dehlaghi-Ghadim, 2023) [21] offers a framework for building ICS security testbeds using Docker and GNS3, with ModbusTCP communication between virtual devices. It supports various connectors for Python scripting and third-party software like Matlab/Simulink.

Trungadi (2023) [57] implemented a virtual version of the SWaT process using MiniCPS and GNS3, demonstrating common attacks like ARP poisoning and DoS while analyzing their impacts. Similarly, Masset and Taburiaux (2018) [11] [35] created a fully virtualized testbed using Mininet for SDN network emulation, with a modified version of OpenPLC and custom visualization tools.

In the smart grid domain, Mosaik (Ofenloch et al., 2022) [9] stands out as an open-source co-simulation framework for integrating multiple simulators into unified scenarios with thousands of simulated entities, featuring a discrete time synchronization mechanism.

1.3.4 Emulation Approaches

Some projects focus specifically on emulating PLC behavior with high fidelity. Awlsim is an open-source Python project that emulates Siemens S7 CPU execution, capable of running Statement List (STL) code with support for a large subset of S7 instructions (Busch et al., 2017) [44]. Building on this foundation, Babu and Nicol (2016) [59] developed a network simulation using Awlsim PLCs controlled by TimeKeeper, a kernel modification that synchronizes the emulators and network simulator via a virtual clock. Their approach demonstrates how accurate emulation can enable precise network traffic patterns within a controlled environment, making it particularly valuable for intrusion detection system development.

1.3.5 Positioning Our Work

Our work builds upon these foundations while addressing several gaps in the current landscape. Unlike hardware-based approaches, our fully virtualized solution eliminates cost barriers and accessibility limitations. In contrast to testbeds tied to specific processes like the Tennessee Eastman implementation in GRFICS, our solution supports configurable process modeling through YAML configuration files.

We extend Yi Zhu's Docker-based ICS simulator by enhancing its security focus, adding support for attack scenarios, and improving its educational value. While Masset and Taburiaux (2018) used Mininet for network simulation, we leverage Docker networking for greater deployment flexibility. Unlike many existing testbeds that focus primarily on technical implementation, our workplaces equal emphasis on educational applications and research utility, providing a platform accessible to students, researchers, and security professionals alike.

Through this balanced approach, our testbed contributes to the growing ecosystem of ICS security research tools while making such research more accessible to a wider audience.

1.4 Scope and Limitations

This thesis builds upon the simulator developed by Yi Zhu [63], which emulates ICS components using containerized environments. Our contribution expands the platform's application by exploring its use in cybersecurity testing and training.

Our work focuses on designing and documenting a modular virtual ICS testbed suitable for simulating real-world industrial operations. We implement both network-based and logic-layer attacks such as spoofing, injection, and denial-of-service to demonstrate security vulnerabilities in ICS setups. Throughout the thesis, we highlight the testbed's potential applications in training, red teaming exercises, and educational labs.

Despite these contributions, we acknowledge several limitations in our approach. The physical process logic is based on simplified simulations rather than precise industrial physics models. Real-time performance and fidelity are constrained by the virtualized nature of the system. Currently, our implementation only supports certain ICS protocols, primarily Modbus/TCP. Additionally, the impact of attacks observed through simulation may not fully reflect safety-critical failure modes in actual systems.

Nevertheless, the testbed provides an accessible, adaptable, and reusable foundation for experimentation something still missing from many large-scale or proprietary testbeds.

1.5 Research Objectives and Questions

The main goal of this thesis is to extend and apply a virtual Industrial Control System (ICS) testbed to evaluate its use in cybersecurity research and education. While previous efforts have focused on simulating the behavior of industrial environments, our work aims to assess how such a testbed can be used to simulate cyberattacks, identify vulnerabilities, and train security professionals in a safe and reproducible environment.

Our research objectives encompass several interconnected aims. We seek to explore and document the limitations of existing ICS security testbeds, justifying the need for a lightweight, open, and modular simulation environment. By extending an existing virtual ICS testbed, we integrate realistic cyberattack scenarios affecting both network and process layers. We demonstrate the testbed's utility in identifying vulnerabilities and simulating

adversarial behavior through practical, ethical attack implementations. Through comparative analysis, we assess the added value of our testbed in cybersecurity education and research compared to existing platforms. Finally, we outline future use cases, including potential integration into intrusion detection systems, red-team exercises, and hands-on cybersecurity training.

The testbed provides a realistic and controlled environment that closely simulates an actual ICS, making it a vital tool for detecting security vulnerabilities. It allows researchers to safely carry out cyberattack simulations, such as denial-of-service (DoS), man-in-the-middle, and malware injections, without risking damage to real-world systems. By observing how the system reacts under these conditions, analysts can study its behavior and assess how well it resists attacks. The testbed also enables the evaluation of security tools like Intrusion Detection Systems (IDS), allowing them to be tested and fine-tuned using realistic traffic and attack patterns. In addition, it supports training activities by giving security teams a safe space to learn how to identify and respond to threats. Finally, the environment can generate valuable labeled datasets, which are essential for developing and validating machine learning-based methods for automatic threat detection.

These objectives give rise to several research questions that guide our work. We examine how a fully virtual ICS testbed can realistically simulate cyberattacks on industrial systems without requiring physical hardware. We investigate which types of attacks such as spoofing, injection, and denial of service can be effectively emulated in a virtual ICS environment. Our research identifies vulnerabilities commonly present in typical ICS setups and explores how a simulated testbed can help identify and demonstrate them. We compare our extended testbed against existing ICS simulation platforms in terms of flexibility, usability, and educational value. Finally, we evaluate the extent to which our testbed can support training and research on ICS cybersecurity in a safe, reproducible, and low-cost manner.

By addressing these questions, the thesis aims to contribute both a technical implementation and a reflective evaluation of how such environments can support the growing need for hands-on ICS security learning.

1.6 Organization of this document

To guide the reader through our research and implementation process, the thesis is organized as follows:

We begin with this introduction chapter, presenting the background, motivation, challenges, and scope of our work. Chapter 2 provides an overview of ICS and processes, describing the ICS architecture used in this work, the simulated physical process, and the communication protocols involved. In Chapter 3, we explore cybersecurity frameworks and security models, discussing industry standards, architectural models, and best practices relevant to ICS security.

The middle portion of the thesis focuses on practical security aspects. Chapter 4 outlines threat modeling techniques and identifies risks within the virtual ICS. Chapter 5 explores system weaknesses through reconnaissance, scanning, and protocol analysis, while Chapter 6 implements simulated cyberattacks and evaluates their effects on the system. Chapter 7 proposes both technical and organizational defense mechanisms inspired by real-world practices.

Chapter 2
Industrial Control Systems and Process Overview

2.1 Overview of the Target ICS Setup

This chapter presents the virtual testbed used throughout this work, including its architecture, technologies, and role in simulating ICS environments. Originally developed by Yi Zhu, the testbed was extended to support cybersecurity-focused experiments. It enables the simulation of industrial processes, network communication, and attack scenarios in a safe and reproducible manner.

2.2 Physical Process Description

The physical process simulated in this testbed is inspired by a simplified water treatment system, modeled entirely in Python. This design follows the foundational work of Yi Zhu [63], and has been further extended in this thesis to better support experimentation with cyber-physical attacks and anomaly analysis.

The simulated environment mimics real-world components such as water tanks, reservoirs, pumps, valves, filters, flow meters, and level sensors. Each of these devices is represented as a Python class, allowing for modular design and behavior customization. Their states (e.g. open/closed for valves, on/off for pumps) and measured values (e.g. fluid level, flow rate) evolve over time based on a looped simulation logic that emulates the behavior of physical processes.

Each simulation cycle (or time step) updates the process variables based on device interactions. For example, if a pump is turned on and its output valve is open, water will move from the reservoir into the next tank, and the flow rate sensor will reflect this change. If the receiving tank is full, the system will prevent overflow, mimicking realistic behavior constraints.

The entire process logic is driven by a central simulation loop that:

- Activates or deactivates devices based on control signals from the PLCs.
- Computes water flow and volume transfer between components.
- Updates sensor measurements and writes them to Modbus registers.

As you see in Figure 2.1, Devices labeled with the prefix T correspond to tanks or related components such as reservoirs, while those beginning with P indicate pumps. Elements marked MV represent motorized valves. In the testbed, UF-301 functions as an ultrafiltration membrane, and UV-401 serves as a UV-based dechlorination unit both of which are simulated using simple filtering mechanisms. Additionally, reverse osmosis units, identified as RO, are modeled as vessels within the system.

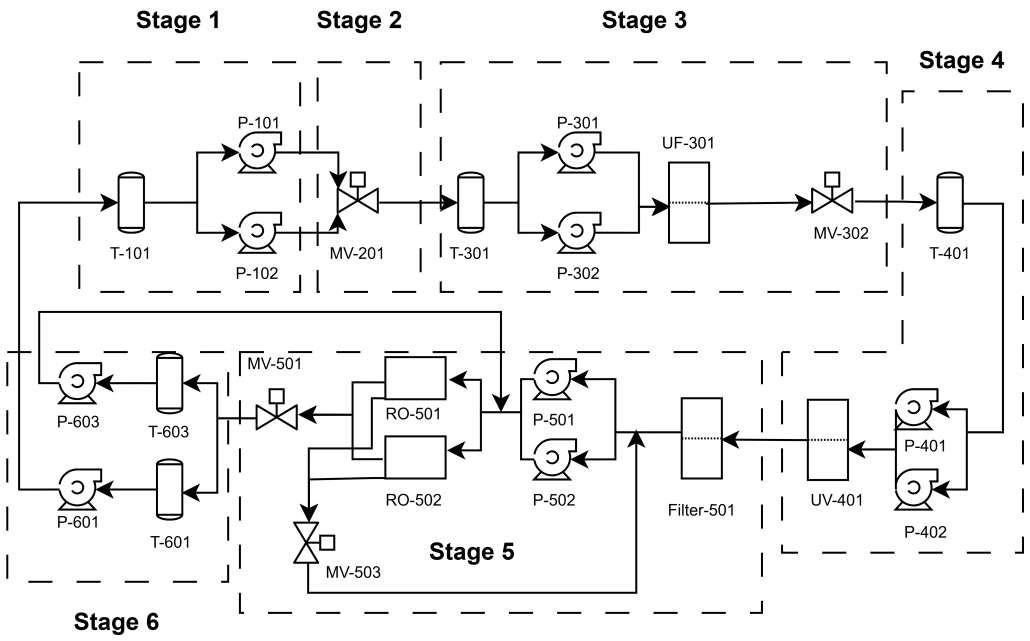

Figure 2.1: ICS-Controlled Water Treatment Process Diagram

This configuration supports white noise injection into sensor data to replicate measurement imperfections, making the system more realistic for anomaly detection and intrusion testing.
The modular architecture also enables rapid scenario reconfiguration using YAML files, allowing different process topologies or behaviors to be modeled without rewriting the simulation code.
Overall, the simulated physical process serves as a safe, observable, and flexible foundation for the cybersecurity experiments conducted in later chapters.

2.3 Logical Architecture of PLC-HMI-Sensors

The logical architecture of the testbed mirrors the layered structure typically found in real-world ICS environments, where data flows between sensors, controllers (PLCs), and operator interfaces (HMIs/SCADA systems). In our virtual setup, this communication and control logic is implemented through containerized components, using standardized protocols and modular simulation code.

Programmable Logic Controllers (PLCs)

At the heart of the control logic are the OpenPLC instances, which act as the programmable controllers of the system. Each PLC is deployed in a separate Docker container and is responsible for:

- Reading real-time sensor data (e.g. tank levels, flow rates).
- Processing input signals using ladder logic.
- Sending actuation commands to pumps and valves.
- Interfacing with the SCADA layer for monitoring.

Each PLC receives input values through Modbus/TCP from the physical simulation and makes decisions based on preconfigured ladder logic, written in IEC 61131-3 compliant language (typically Ladder Diagram or Structured Text). These decisions influence the state of the actuators (turning pumps on/off, opening/closing valves, etc.). To ensure flexibility, the simulator includes a YAML-based system for defining PLC logic templates and mapping Modbus registers to virtual sensors and actuators. Additionally, a Python-based PSM (Python SubModule) is used for internal PLC drivers, making it easier to test logic or override default behavior during simulation experiments.

Sensors and Actuators

Sensors and actuators are modeled in the physical process module and are directly connected to the PLCs via simulated memory registers:

- **Analog Sensors** (e.g. LIT for level, FIT for flow) write their values to holding or input registers.
- **Discrete Sensors** (e.g. valve state indicators) use coils and discrete inputs.
- **Actuators** (e.g. pumps, valves) are driven by output coils or holding registers set by the PLC logic.

The simulator ensures that realistic sensor data (including white noise and rounding) is sent to the PLCs, which must then process that data and control the system accordingly.

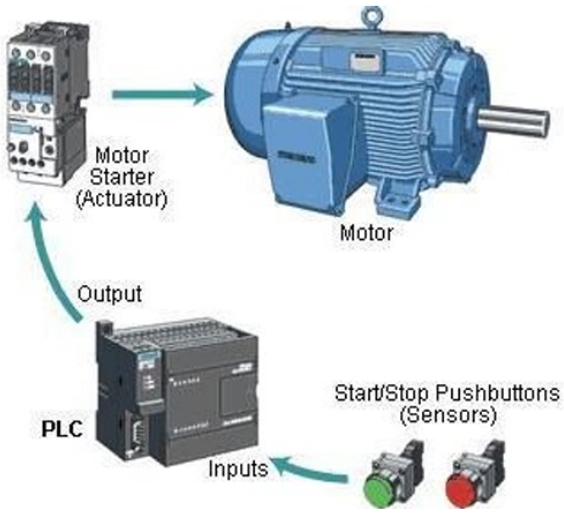

Figure 2.2: Sensor to PLC to Actuator to Motor Connection

SCADA / HMI System

The SCADA layer is provided by ScadaBR or ScadaLTS, which runs in its own container and connects to each PLC via Modbus. Its responsibilities include:

- Periodically polling PLCs for sensor values.
- Logging historical data in a SQL-based historian database.
- Providing a web-based HMI for visual monitoring and manual control.

- Graphing trends and issuing alarms for out-of-range values.

In practice, this HMI interface is what the **operator** of the system would use to oversee and manage the process making it crucial for both real-time awareness and forensic analysis after an incident.

Logical Flow Summary

- Sensors in the simulation push data into shared memory via Modbus.
- OpenPLC reads these values, processes control logic, and updates actuator states.
- Actuators adjust the physical simulation (e.g. moving water).
- SCADA polls the PLCs to display data and enable human interaction.

This layered structure forms the cyber-physical control loop, which mimics real ICS architectures. It provides a foundation not just for observing system behavior under normal operations, but also for injecting attacks and analyzing their consequences in Chapters 4–6.

2.4 Technologies Used

To implement a flexible, lightweight, and fully virtual ICS simulation environment, this project integrates a suite of open-source technologies and custom-built components. The primary objective in selecting these tools was to achieve portability, ease of deployment, modularity, and realism while remaining accessible to researchers, educators, and students.

Docker

At the core of the simulation infrastructure is Docker, a containerization platform used to create isolated environments for each component of the ICS system. Docker enables the deployment of multiple PLCs, the physical simulation node, SCADA software, and supporting services (e.g., database, network emulators) without conflict or system-wide installations.

- Each component (PLC, SCADA, simulation, etc.) runs in a dedicated Docker container.
- Docker Compose scripts orchestrate and configure these containers.
- Network settings and IP addresses are customizable to simulate realistic topologies.

This containerized approach enhances portability and reproducibility, allowing the entire testbed to run identically across systems with minimal setup.

OpenPLC

OpenPLC is used to simulate the behavior of real industrial Programmable Logic Controllers. It is an open-source, IEC 61131-3 compliant PLC runtime that supports common industrial languages such as Ladder Diagram (LD) and Structured Text (ST).
Key features:

- Logic programming is performed using OpenPLC Editor or generated directly from YAML configurations.
- The OpenPLC Runtime runs inside Docker and processes real-time sensor values, then outputs control signals to actuators.
- It supports Modbus/TCP, DNP3, and other protocols, though this work primarily uses Modbus/TCP.

A custom Python-based driver (PSM) was developed to interface OpenPLC with the simulated process, enhancing flexibility.

SCADA Systems: ScadaBR and ScadaLTS

ScadaBR and its fork ScadaLTS are open-source SCADA (Supervisory Control and Data Acquisition) platforms used in this project for system monitoring, data logging, and HMI (Human-Machine Interface) creation.

- Both platforms support Modbus polling, allowing real-time data visualization from the PLCs.
- A SQL-based historian logs sensor values for post-analysis and anomaly tracking.
- SCADA dashboards (graphical views) simulate what an operator would see in a real control room.

ScadaLTS was chosen as the primary SCADA layer due to its better long-term support and active development.

Python-Based Physical Simulation

The physical process representing tanks, pumps, valves, and fluid behavior is simulated using Python. The simulation includes:

- Custom Python classes for all devices.
- Centralized simulation loop updating system state every few milliseconds.
- Support for mathematical logic, flow balancing, and white noise injection.

All components and process topology are defined through a YAML configuration file, which allows users to easily change the system setup and behavior without modifying the core code.

Modbus/TCP

Modbus/TCP is the primary communication protocol used to connect PLCs, sensors, actuators, and SCADA systems. It was selected due to:

- Its simplicity and wide adoption in ICS environments.
- Built-in support in OpenPLC and ScadaBR.
- Ease of simulation using libraries such as pymodbus and pyModbusTCP.

Although Modbus lacks built-in security features, its vulnerabilities make it an ideal candidate for demonstrating typical ICS attack scenarios in a training or research setting.
Other Supporting Tools

- **Vagrant:** Used to automate VM creation with Docker pre-installed for users who prefer working in a virtual machine instead of directly on the host.
- **NetEm (Network Emulator):** Installed in Docker containers to simulate real-world network conditions (e.g. latency, packet loss, jitter).
- **SQLite/MySQL:** Used by the SCADA systems for persistent data storage and historical graphing.

With these technologies working together, the testbed offers a powerful yet accessible platform for exploring ICS behavior and testing security-related scenarios.

2.5 System Communication and Protocols

Communication within an Industrial Control System (ICS) is vital to ensure that all components from sensors and PLCs to SCADA systems and actuators work together to monitor and control physical processes. In our virtual testbed, communication is simulated through widely adopted ICS protocols, primarily Modbus/TCP, but the architecture is designed to accommodate other industrial protocols as well.

2.5.1 Internal Communication Flows

In the testbed, communication flows between the following layers:

- **Sensors and PLCs:** Sensor data (e.g. tank level, flow rate) is transmitted via Modbus registers to the PLC. Each sensor writes its measurement to a specific register that the PLC reads cyclically to make control decisions.
- **PLCs and Actuators:** Once a decision is made, the PLC writes control signals to registers that affect actuators (e.g. start a pump or open a valve). This data exchange is also handled using Modbus coils and holding registers.
- **PLCs and SCADA:** The SCADA system polls PLCs to retrieve live data and update operator dashboards. It can also write back to control registers if manual commands are issued through the Human-Machine Interface (HMI).

All communication occurs over a Docker-implemented virtual Ethernet network, simulating real ICS LAN conditions.

2.5.2 Protocol Overview: Modbus/TCP

Modbus/TCP is the primary communication protocol used in the testbed. Developed in the late 1970s and later adapted to TCP/IP networks, Modbus is simple, widely supported, and ideal for simulation environments.
Key features of Modbus/TCP:

- **Master-slave architecture:** The SCADA system (master) polls devices (slaves) like PLCs for information.
- **Deterministic polling:** Enables periodic updates with predictable timing.
- **Simplicity:** Easy to implement, monitor, and analyze.

However, **Modbus lacks native security features:**

- No encryption or authentication.
- Vulnerable to spoofing, tampering, and replay attacks.
- No user-based access control.

These weaknesses make Modbus an ideal candidate for demonstrating cybersecurity risks in ICS, as explored in later chapters.

2.5.3 Virtual Network Segmentation

To emulate realistic ICS network structures, the testbed uses Docker's virtual networking capabilities. Components are grouped into logically separated subnets:

- **Process subnet** (physical simulation, sensors/actuators).
- **Control subnet** (PLCs).
- **Monitoring subnet** (SCADA, HMI).

This segmentation simulates network zones inspired by the Purdue model (see Chapter 3), allowing experimentation with lateral movement, unauthorized access, and segmentation-based defenses.

2.5.4 Protocol Extensibility

While Modbus/TCP is the protocol implemented in this work, the testbed is designed to support additional ICS protocols such as: DNP3, S7Comm (siemens), OPC UA, MQTT (for IoT simulations).
This flexibility ensures that the platform can evolve with future work and accommodate more complex or modern ICS communication stacks.

Chapter 3
Industrial Cybersecurity Frameworks and Security Models

This chapter introduces the key models and standards used to understand, secure, and evaluate ICS. It provides the conceptual background for the work done in later chapters and places your testbed within the context of widely recognized security practices.
We'll break it down into these sections:

- 3.1 The Purdue Model for ICS Architecture.
- 3.2 ICS Cybersecurity Challenges and Attack Surfaces.
- 3.3 Overview of Cybersecurity Frameworks (e.g. NIST, IEC 62443).
- 3.4 ICS Security Goals (CIA, Availability-First, Safety-Driven).
- 3.5 STRIDE Threat Modeling for ICS.

3.1 The Purdue Model for ICS Architecture

The **Purdue Enterprise Reference Architecture (PERA)**, commonly referred to as the **Purdue Model** is one of the most widely adopted frameworks for understanding the structure and segmentation of ICS. It provides a layered model that separates enterprise IT systems from real-time industrial operations and is commonly used as a foundation for designing secure architectures [48][16].

Purdue Model Overview

The model is divided into six layers (numbered from 0 to 5), each representing a different part of an industrial system:

- **Level 0 – Physical Process:** This includes pumps, valves, tanks, motors, and sensors. It is the layer where physical changes happen.
- **Level 1 – Control Devices:** PLCs and RTUs reside here. They directly interact with sensors and actuators, executing logic to control the process.
- **Level 2 – Supervisory Control:** Includes SCADA systems and HMIs. Operators use this layer to monitor the system and send high-level commands.
- **Level 3 – Operations Management:** Handles production planning and quality monitoring. Often includes MES (Manufacturing Execution Systems).
- **Level 4 – Business Planning and Logistics:** Consists of enterprise IT systems (ERP, databases) that focus on strategic business decisions.

- **Level 5 – Cloud/External Services (modern addition):** Includes remote access, cloud analytics, and vendor connections.

This layered approach promotes network segmentation, which is essential for limiting the spread of cyber threats [36].

Why It Matters for Security

The Purdue Model highlights the importance of isolating operational technology (OT) from enterprise IT systems. Without proper segmentation, attacks that originate at the business level (e.g. phishing, ransomware) can propagate downward and disrupt real-time operations.
In our testbed, this concept is mirrored using virtual network zones to separate SCADA interfaces, PLCs, and process simulation. This allows us to explore different attack paths targeting each layer of the ICS stack.
The model also helps define access control policies, where devices in lower levels should never communicate directly with cloud or business systems without strict filtering, monitoring, and authentication.

3.2 ICS Cybersecurity Challenges and Attack Surfaces

ICS differ fundamentally from traditional IT systems in both design priorities and operational context. While IT systems typically prioritize confidentiality, ICS prioritize availability and safety any disruption to control systems can have direct physical consequences, including equipment damage, environmental harm, or even human injury [40].

The security challenges facing modern ICS environments emerge from several interconnected factors. Many systems still rely on legacy infrastructure developed decades ago when security was not a design consideration. These systems often include unpatched operating systems, hardcoded credentials, unauthenticated protocols like Modbus/TCP and S7Comm, and lack built-in encryption or access control mechanisms. These insecure-by-design characteristics persist because updating ICS components is difficult—downtime is costly, and patching risks interrupting critical operations.
These insecure-by-design characteristics persist because updating ICS components is difficult **downtime is costly**, and **patching risks interrupting critical operations** [61], [14].

Compounding these issues, ICS are increasingly integrated with IT systems for data analysis, remote management, and cloud services as part of the Industry 4.0 shift, dramatically expanding the attack surface. What was once air-gapped is now reachable via corporate VPNs, remote engineering stations, or internet-exposed interfaces. This convergence introduces exposure to common IT threats like ransomware and phishing, creates lateral movement opportunities from IT to OT networks, and often results in poorly configured segmentation between business and control layers.

Unlike IT environments, many ICS setups lack modern Security Information and Event Management (SIEM) systems or centralized logging tools. Events such as configuration changes, failed authentications, or traffic anomalies often go undetected due to proprietary

log formats, low-frequency polling that misses transient anomalies, and lack of visibility across segmented networks. This makes intrusion detection and incident response much more difficult.This makes intrusion detection and incident response much more difficult [2].

Human and organizational factors further complicate security efforts, with limited cybersecurity training for OT personnel, outdated security policies, and conflicts between IT and OT teams regarding patching, updates, and access control. Many operators still treat security as a secondary concern, focusing on uptime and process continuity above all. Many operators still treat security as a secondary concern, focusing on uptime and process continuity above all [24].

These vulnerabilities have been exploited in numerous high-profile attacks that highlight the diversity and impact of ICS threats. Examples include Stuxnet (2010) which sabotaged Iranian centrifuges by modifying PLC logic [62], Triton/Trisis (2017) that targeted safety instrumented systems in a Saudi petrochemical plant [47], Colonial Pipeline (2021) which disrupted fuel distribution in the U.S. due to ransomware affecting IT-OT systems [33], and CyberAv3ngers (2023) that attacked water facilities by exploiting exposed PLC interfaces [34]. These cases reveal how even basic vulnerabilities can be leveraged for sophisticated attacks, especially in environments with poor segmentation, legacy systems, and limited detection capabilities.

These cases reveal how even basic vulnerabilities can be leveraged for sophisticated attacks, especially in environments with poor segmentation, legacy systems, and limited detection capabilities.

3.3 Overview of Cybersecurity Frameworks for ICS

Securing ICS requires a specialized approach that goes beyond traditional IT security models. ICS environments have unique operational constraints like real-time control requirements, legacy systems, and high availability demands that make general-purpose security practices insufficient. To address these challenges, multiple organizations have developed **tailored cybersecurity frameworks** and standards that guide the design, assessment, and improvement of ICS security programs.
This section introduces three of the most influential frameworks: **NIST Cybersecurity Framework, IEC 62443, and NIS2 Directive** all of which contribute foundational principles relevant to this thesis.

3.3.1 NIST Cybersecurity Framework (CSF)

Developed by the U.S. National Institute of Standards and Technology (NIST), the **Cybersecurity Framework (CSF)** provides a flexible and risk-based approach to managing cybersecurity in critical infrastructure, including ICS [10].
The framework is structured around five core functions:

- **Identify :** Develop an understanding of risks and assets.
- **Protect :** Implement safeguards to ensure delivery of services.
- **Detect :** Develop activities to identify cybersecurity events.
- **Respond :** Take action regarding detected incidents.

- **Recover :** Maintain plans for resilience and restoration.

Each function includes categories and subcategories mapped to existing standards and best practices (e.g. ISO/IEC 27001). NIST SP 800-82 extends this framework specifically to ICS environments by offering implementation guidance tailored to OT systems [40].
In our project, the NIST framework is indirectly applied in defining how attack detection and recovery can be integrated into a simulated ICS workflow.

3.3.2 IEC 62443 Series

The **IEC 62443** is an international standard developed by the **International Electrotechnical Commission** and the **ISA99 committee**. It is widely considered the most comprehensive ICS cybersecurity standard and is adopted in both industrial and governmental settings [58][30].
The IEC 62443 standard is structured into four parts:

- **General** (concepts, models, terminology).
- **Policies and Procedures** (organizational practices).
- **System-Level Requirements** (architecture, network zones).
- **Component-Level Requirements** (device security).

One of the key principles of IEC 62443 is **defense-in-depth**, which promotes the segmentation of systems into zones and conduits to reduce the impact of intrusions. It also emphasizes **security levels (SL 1–4)** [54], which define the protection needed against different types of attackers (from casual to nation-state level).
Our testbed mirrors this idea by isolating SCADA, PLCs, and process simulation into **virtual zones** and evaluating attack paths across these zones.

3.4 Defense-in-Depth for ICS

Defense-in-depth implements multiple layers of protection, ensuring that the failure of any single security measure doesn't compromise the entire system. In ICS environments, this typically includes:

Physical security controls such as locked cabinets, access control systems, and surveillance, forming the outer layer of defense for critical infrastructure.

Network segmentation creates boundaries between different functional areas, with firewalls, data diodes, and DMZs controlling communication between zones. This aligns with both the Purdue Model hierarchy and IEC 62443's zone and conduit approach [25].

Endpoint protection includes device hardening, application whitelisting, and malware protection adapted for ICS environments. These measures are particularly important for systems that cannot be frequently patched or updated.

Authentication and access control mechanisms verify the identity of users and devices before granting system access, with multi-factor authentication increasingly adopted for critical functions.

Monitoring and detection systems continuously observe network traffic and system behaviors, identifying potential security incidents and enabling timely response.

Chapter 4
Threat Modeling and Risk Assessment

As the cyber-physical nature of ICS grows in complexity, so do the risks that threaten their integrity. These systems, foundational to critical infrastructure, are increasingly exposed to cyberattacks due to connectivity, legacy protocols, and insecure design practices. Although securing ICS is essential, it is impossible to mitigate all vulnerabilities. For this reason, **threat modeling** and **risk assessment** are crucial tools in understanding where to focus defensive efforts.

In this thesis, threat modeling is used not to teach general ICS theory, but to support and justify the attack scenarios implemented in our testbed. We aim to identify relevant threats in a structured, realistic way, using established methodologies to simulate adversarial behaviors and study system responses.

This chapter presents a targeted threat modeling approach applied to our virtual ICS setup. Using STRIDE, an industry-known threat categorization model, we highlight the specific risks that arise within PLCs, HMIs, SCADA servers, and network communications. We also apply a qualitative risk assessment to map these threats to possible impacts on process integrity, safety, and availability. Our contribution is not to reinvent the models, but to **demonstrate their application** in a testbed designed for research and education, thus validating the testbed's relevance for security studies. The methodology used here mirrors those applied in government and academic ICS security assessments, including NIST [10] and works by Upadhyay and Sampalli [20] and Kamal et al. [51]

ICS threat modeling differs from IT environments in several ways. While IT systems typically emphasize data confidentiality, ICS places **availability and safety** as top priorities [40]. Consequently, attack vectors targeting system uptime or physical control logic even without data leakage can be far more dangerous. Moreover, ICS components like PLCs are often designed with minimal security, making them soft targets for attackers. [14]

This chapter proceeds by first exploring how the STRIDE methodology applies to the virtual ICS testbed, with particular attention to how each threat category maps to specific components [37]. We then present the process of identifying system assets, vulnerabilities, and plausible attack paths, followed by the development of a simplified risk assessment matrix. This combination of modeling and evaluation supports our core contribution: demonstrating how threat-informed attack simulations can enhance both research and learning in the ICS security domain [64].

4.1 STRIDE Methodology Applied to ICS

The STRIDE model offers a structured lens for examining threats that may compromise an information or control system. In the context of our virtual ICS testbed, we applied this model to identify practical and observable risks across system components. Rather than treating STRIDE as a theoretical checklist, we focused on how each threat type could emerge through multiple real-world-inspired attack paths. The following analysis describes two representative scenarios for each STRIDE category, drawing directly from

our simulation environment and supporting the real world literature [5].

4.1.1 Spoofing

Threat S1:

- **Target:** Modbus/TCP communication; PLC and SCADA device identity.
- **Example 1 in testbed:** The attacker performs IP spoofing and ARP poisoning to impersonate the OpenPLC on the network.
- **Implication:** The SCADA system receives data and sends commands to a rogue device, unaware that it is no longer communicating with the real controller. This results in control over pumps and valves being silently redirected.
- **Justification:** Modbus/TCP lacks any identity verification, making spoofing feasible and highly dangerous in ICS environments where trust in device identity is essential.

Threat S2:

- **Example 2 in testbed:** A phishing page clones the ScadaBR login screen to harvest operator credentials (APT Phase 1).
- **Implication:** Once credentials are stolen, attackers access the SCADA dashboard and issue commands as legitimate users, potentially modifying process variables or deactivating alarms.
- **Justification:** Spoofed interfaces have been part of well-documented APT campaigns (e.g. MITRE ATT & CK T1598.001) and are an initial vector for deeper access.

4.1.2 Tampering

Threat T1:

- **Target:** PLC coil/register values, logic files, database integrity.
- **Example 1 in testbed:** An attacker modifies PLC register values via unauthorized Modbus write commands, forcing pumps to operate continuously regardless of tank level.
- **Implication:** Water overflows the tank, creating a physically hazardous condition. The logic inside the PLC is bypassed completely.
- **Justification:** Tampering with register states has been central in past real-world attacks like Stuxnet, and mirrors many ICS-CERT vulnerability advisories.

Threat T2:

- **Example 2 in testbed:** The attacker overwrites the ladder logic file in OpenPLC via the vulnerable upload process, injecting custom control behavior (e.g. disabling emergency stop conditions).

- **Implication:** Malicious logic executes continuously, potentially without being visible on the SCADA dashboard, as the PLC behavior no longer matches the operator's expectations.
- **Justification:** Direct logic injection attacks exploit weak validation in OpenPLC's file handling system, aligning with database and logic persistence flaws detailed in ICS testbed literature.

4.1.3 Repudiation

Threat R1:

- **Target:** SCADA interface logging, PLC configuration changes.
- **Example 1 in testbed:** Ladder logic is replaced on the PLC, but the SCADA or PLC interface logs do not store metadata or timestamps related to who made the change.
- **Implication:** After an incident, there is no way to determine whether it was caused by operator error, insider action, or cyber intrusion.
- **Justification:** OpenPLC's upload and configuration interfaces lack auditing, exposing systems to undetectable manipulation.

Threat R2:

- **Example 2 in testbed:** An attacker performs coil modifications through intercepted Modbus traffic, but there are no audit trails in the SCADA or PLC components.
- **Implication:** Unauthorized changes to actuator behavior cannot be traced to a specific source, weakening accountability and forensics.
- **Justification:** Repudiation is particularly dangerous in OT where system integrity affects safety; yet it is often overlooked due to poor event tracking.

4.1.4 Information Disclosure

Threat I1:

- **Target:** Modbus/TCP traffic, OpenPLC database, system architecture.
- **Example 1 in testbed:** Modbus traffic is passively sniffed using Wireshark to capture real-time process data, including tank levels and actuator states.
- **Implication:**Attackers can map system behavior patterns and identify critical timings, paving the way for more precise and harmful attacks.
- **Justification:** Modbus transmits data in plaintext, making it vulnerable to eavesdropping. These scenarios reflect common reconnaissance stages in ICS attacks.

Threat I2:

- **Example 2 in testbed:** The OpenPLC.db file is accessed and parsed, revealing stored credentials, current running programs, and user roles.

- **Implication:** With access to the internal PLC state and logic, an attacker can prepare logic injection or replay attacks with high precision.
- **Justification:** Exploitation of OpenPLC's unprotected SQLite database was shown to give deep visibility into the logic of the system and the user data.

4.1.5 Denial of Service (DoS)

Threat D1:

- **Target:** OpenPLC controller stack, ScadaBR frontend, network interface.
- **Example 1 in testbed:** The attacker launches a Modbus flood attack using custom Python scripts, sending hundreds of requests per second to the OpenPLC device.
- **Implication:** The PLC becomes non-responsive, unable to respond to sensor data or operator commands, leading to unsafe shutdown of the process.
- **Justification:** DoS attacks via Modbus are simple yet effective, commonly used as diversionary tactics or in the early stages of APT.

Threat D2:

- **Example 2 in testbed:** A script repeatedly pings ScadaBR's Web interface while also collecting data at a high rate, eventually causing the interface to hang or crash.
- **Implication:** Operators lose visibility and control, especially during critical process moments, increasing the chance of accidents.
- **Justification:** Lightweight open-source SCADA tools are especially vulnerable to resource exhaustion due to minimal optimization.

4.1.6 Elevation of Privilege

Threat E1:

- **Target:** SCADA admin functions, simulation container, database logic.
- **Example 1 in testbed:** An attacker uses credentials stolen from a phishing campaign to log into the ScadaBR interface, escalating from viewer to administrator via the web UI.
- **Implication:** Once admin access is achieved, the attacker disables alarms, changes thresholds, and installs persistent logic for future use.
- **Justification:** Privilege escalation through weak account management is a top ICS threat; credential reuse and role misassignment are common in undersecure environments.

Threat E2:

- **Example 2 in testbed:** The attacker modifies OpenPLC's database directly to create a new admin user, bypassing login interfaces altogether.

- **Implication:** Hidden admin accounts allow long-term persistence and covert manipulation of logic programs.
- **Justification:** The OpenPLC.db file is unprotected and manipulable, allowing backend privilege escalation with no alerts or logs.

This revised STRIDE threat model captures the real nature of ICS security: not just isolated software weaknesses, but cyber intrusions that lead to physical consequences. By linking each attack scenario to observable impacts on simulated industrial processes, the threat model becomes a powerful foundation for the attack demonstrations in the next chapters.

However, it is important to emphasize that the capabilities of our testbed extend to the simulation of direct physical consequences resulting from cyber intrusions. These consequences are not theoretical. In practice, our virtual environment allows for realistic replication of physical process disruptions, enabling a full-spectrum analysis of attack impact from data layer to actuation.

For example, during the simulation of sensor spoofing attacks, we were able to artificially manipulate level readings in a tank control process. By delaying the reaction of the PLC to rising water levels, the system continued pumping beyond safety thresholds, leading to a simulated overflow condition. This mirrored what could occur in a real bottling or wastewater treatment plant. Similarly, by modifying the control logic of the PLC, we introduced a condition in which a pump continued to operate while the upstream reservoir was empty, an operation that in physical systems would result in cavitation, potential damage to the pump mechanism, and energy waste.

Another observed effect involved the injection of alternating sensor values to simulate an oscillating signal. The control loop entered an unstable regime, constantly toggling pumps and valves, highlighting how sensor tampering can destabilize a process without triggering traditional alarms. In another case, sudden, unauthorized valve closure during active pump operation produced a simulated pressure spike in the system, providing insight into how actuator commands - when issued out of sequence - can threaten mechanical safety.

These process-level effects serve to demonstrate that the testbed is not merely a protocol-level simulation, but a cyber-physical environment capable of capturing both the technical and operational impact of adversarial actions. This reinforces the practical value of our threat model and lays the foundation for the attack scenarios and mitigation strategies explored in the chapters that follow.

4.2 Risk Assessment Methodology

Having identified and analyzed a wide spectrum of threats using the STRIDE framework, the next step in our methodology involves evaluating their relevance and severity through a structured risk assessment process. In cybersecurity for ICS, not all vulnerabilities pose the same level of concern. Some may be relatively easy to exploit but carry low operational impact, while others though less accessible could lead to catastrophic consequences in physical systems. This dual dimension of likelihood and impact is central to the risk

assessment performed in this work.

The assessment approach adopted here aligns with widely accepted principles such as those outlined in NIST SP 800-30 and IEC 62443, but has been adapted to the specific context of our virtual testbed. Rather than applying a purely numerical model, we used a qualitative risk matrix to evaluate each threat based on empirical testing, feasibility of execution, and its resulting effect on the physical process [27].
Each identified threat was rated along two axes:

- **Likelihood:** the probability that the attack could occur in the current system setup, considering existing protections, configuration defaults, and attacker requirements.
- **Impact:** the extent of disruption to system integrity, availability, safety, and process continuity if the attack were successful.

For instance, attacks such as unauthenticated Modbus register modification or OpenPLC database exploitation were rated as high likelihood, given their ease of execution and lack of inherent defenses. The impact of such attacks when used to overflow a tank, disable logic, or modify actuator behavior was also assessed as high, due to the direct effect on physical safety and system availability. In contrast, privilege escalation via phishing was marked with a lower likelihood, as it required prior reconnaissance and user deception, but still carried high impact potential if successful.
The table below summarizes an illustrative sample of this matrix for selected attack scenarios:

Attack Scenario	Likelihood	Impact	Risk Level
Modbus write to actuator register	High	High	Critical
Logic overwrite via OpenPLC web UI	Medium	High	High
ARP spoofing between SCADA and PLC	Medium	Medium	Medium
Sniffing unencrypted Modbus traffic	High	Low	Medium
DDoS via Modbus flooding	High	Medium	High
SCADA credential theft via phishing	Low	High	Medium
Injecting instability via oscillating sensor spoofing	Medium	High	High
Modbus packet sniffing	High	Low	Medium
Unauthorized coil manipulation via Modbus write	High	High	Critical

Table 4.1: Risk Matrix

While the matrix provides a structured view, the context of the testbed plays a crucial role in these evaluations. Our environment lacks endpoint detection, protocol filtering, or user authentication beyond basic default setups, thus realistically reflecting many underprotected ICS deployments still in use today. The impact scores also consider **physical outcomes** such as tank overflow, pressure instability, or system-wide lockups rather than just technical disruption.

This tailored risk assessment served two critical purposes: it helped prioritize which attacks to simulate in-depth and guided the selection of realistic mitigation techniques discussed in later chapters. It also reinforced the educational value of the testbed by

showing students and researchers how to connect**threat modeling** with **consequence-driven evaluation** a mindset that is essential in operational technology environments.

4.3 Mapping Threats to System Zones (Based on the Purdue Model)

Although the original testbed inherited from Yi Zhu was designed as a fully virtualized ICS simulation, it lacked architectural segmentation between control, supervisory, and interface components. All elements including PLCs, SCADA systems, HMIs, and physical process logic were deployed within the same flat network. This configuration made it highly functional for simulation and experimentation, but also extremely vulnerable to lateral movement and direct attacks across system layers.

In this section, the Purdue Model is not used to describe an existing architecture, but rather as a conceptual framework to analyze the threats we observed and to illustrate how a lack of segmentation allows for rapid threat propagation. By mapping threats to their corresponding Purdue levels, we highlight where attacks landed, and which architectural zones were most exposed in the absence of any isolation or filtering.

Many attacks in our testbed began at what would correspond to Level 3 or 2 of the Purdue Model where SCADA interfaces and HMI dashboards reside. Examples include credential theft through phishing, brute-forcing login interfaces, and exploiting weak authentication on the ScadaBR system. Because there was no enforcement of role-based access or isolation between components, attackers could immediately pivot from the interface layer into control logic.

Once at Level 1, threats had direct access to PLC devices. Using Modbus protocol tools or database exploits, attackers were able to tamper with actuator register values, overwrite control logic, or introduce malicious programs into OpenPLC. Without network segmentation or protocol-level filtering, these attacks were delivered without resistance and had immediate impact on the simulated physical process.

The absence of any boundary between Levels 2 and 1 meant that compromised SCADA systems were effectively treated as trusted peers to PLCs. This undermined one of the core principles of the Purdue Model—defense by separation of responsibilities and allowed simple attack chains to evolve into high-impact process disruptions. In particular, the APT scenario developed in this thesis demonstrated how a single phishing message could lead to full control over pumps and valves.

In reality, the Purdue Model would serve as a foundational defense mechanism in such systems, enforcing strict separation between enterprise IT (Levels 4–5), control layers (Levels 2–3), and physical devices (Levels 0–1). Our analysis therefore uses Purdue not to describe the testbed as it exists, but to highlight where segmentation could have stopped or slowed down the attacks, and to support the later security recommendations in Chapter 8.

This mapping exercise confirmed that flat ICS architectures remain highly vulnerable, even when protocols and components appear operational. The threats identified in this thesis would have been significantly harder to execute or at least easier to detect if Purdue-inspired zone-based segmentation had been implemented from the beginning.

4.4 Summary of High-Risk Scenarios

The analysis conducted throughout this chapter has revealed a range of vulnerabilities and threat vectors affecting the different layers of our simulated ICS environment. By combining structured threat modeling (via STRIDE), a tailored risk assessment matrix, and conceptual mapping onto the Purdue Model, we identified the scenarios most likely to result in significant process disruption or compromise of control integrity.

Across all categories, attacks targeting Level 1 (PLCs) and Level 2 (HMI/SCADA) consistently emerged as the most impactful. This was largely due to the lack of segmentation, weak authentication, and protocol-level vulnerabilities such as unencrypted Modbus communication and unauthenticated PLC access. These weaknesses created conditions where attackers could, with minimal effort, directly manipulate actuator commands, disable control logic, or influence operator decisions through falsified sensor values or spoofed interfaces.
Among the most critical scenarios observed were:

- **Direct Modbus register manipulation** to force actuators into unsafe states, such as continuous pumping or locked valve positions.
- **Ladder logic injection and overwriting,** exploiting the unsecured OpenPLC upload process to disable safety conditions or automate malicious behavior.
- **Credential harvesting and session hijacking,** which enabled unauthorized access to SCADA systems and complete control over operator interfaces.
- **Database exploitation,** particularly the manipulation of OpenPLC.db to escalate privileges or extract sensitive logic files and user data.
- **Denial-of-service conditions,** including Modbus flooding and SCADA polling saturation, which disrupted process visibility and responsiveness.

What makes these scenarios especially concerning is not just their technical feasibility, but their ability to lead to physical consequences tank overflows, pump cavitation, pressure imbalances, and delayed operator response. In many cases, these outcomes were achieved using only publicly available tools, minimal system knowledge, and no privilege escalation beyond default access.

These findings confirm that even in a small scale virtual environment, the absence of foundational security controls exposes ICS systems to severe threats. The insights gained here set the stage for the next phase of this thesis: a detailed vulnerability assessment that examines how these risks can be systematically identified through reconnaissance, scanning, and direct probing of the simulated environment.

Chapter 5
Vulnerability Assessment

Before delving into the detailed vulnerability assessment, it is important to present the architecture of the SCADA testbed on which all attack scenarios were implemented. The environment was designed to emulate a realistic industrial control system, while remaining fully virtualized, flexible, and safe for experimental intrusion testing. Its structure reflects a flat network where all components interact without any enforced segmentation or protocol filtering, a condition that intentionally mirrors many insecure real-world industrial deployments [3][42].

The testbed builds upon the foundational work developed by Yi Zhu, which provided the initial container-based simulation environment. In this thesis, we extend that framework by incorporating security assessments, structured threat modeling, attack simulations, and targeted exploit development. The goal is to demonstrate how even a small, simplified SCADA system can become a highly vulnerable target when lacking foundational protections such as access control, authentication, and network isolation.

At the core of the system, a virtual PLC is implemented using OpenPLC, which executes ladder logic programs that determine how actuators respond to sensor inputs. This PLC communicates via the Modbus/TCP protocol with both the SCADA interface and the physical process simulation. The SCADA layer is handled by ScadaBR, which functions as both the Human-Machine Interface (HMI) and data acquisition system. Through its web interface, users can visualize process states, control devices manually, and monitor sensor readings in real time.

The physical process itself is simulated using a lightweight Python-based engine. This engine emulates common industrial behaviors such as fluid level changes, valve actuation, and pump activation, and exchanges data with the PLC using Modbus registers. The design of the simulator ensures that every digital instruction whether from the PLC or the SCADA system results in observable physical changes, providing a valuable platform for testing cyber physical attacks [13].

The environment also includes a dedicated attacker machine, configured using Kali Linux, which serves as the platform for reconnaissance, scanning, and exploitation activities. This node is deployed as a container within the same virtual network and is granted unrestricted access to all other components, reflecting a worst-case scenario where the attacker has already gained a foothold inside the ICS environment. Throughout this work, we consider several possible attacker positions within the network, including internal compromise (such as an infected engineering workstation), lateral movement from a breached IT zone, and direct access through exposed services. These varied scenarios help evaluate how different levels of network exposure affect the attacker's ability to reach and manipulate control assets.

All elements of the testbed are deployed using Docker containers and connected via a shared virtual bridge. This infrastructure not only simplifies orchestration and repeatability but also allows for quick reconfiguration of attack paths and system states.

As the next sections describe each component in more detail, the reader will see how the testbed captures the essential dynamics of a real SCADA network while remaining

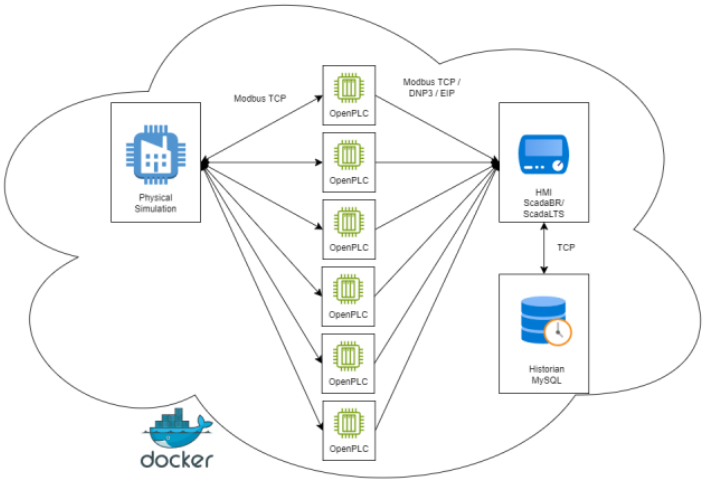

Figure 5.1: Flat Network Docker Architecture

agile and accessible for structured cybersecurity experimentation. Its vulnerabilities are not hypothetical they are observable, reproducible, and consistent with those reported in field studies and industrial advisories.

5.1 OpenPLC: Architecture and Vulnerabilities

Within the SCADA testbed presented in this thesis, the OpenPLC Runtime plays the role of the Programmable Logic Controller (PLC). Located functionally in the architecture control layer, it acts as the real-time decision making unit of the system, receiving sensor input from the physical process simulation and issuing actuator commands in response. All communication between the PLC and the other components such as the SCADA interface (ScadaBR) and the Python-based physical simulation takes place over the Modbus/TCP protocol, which, in its default configuration, provides no encryption or authentication.

The OpenPLC project was originally developed as an open-source and vendor neutral PLC solution, offering academic and industrial communities a cost-effective and flexible alternative to traditional hardware PLCs. Over the years, it has gained considerable popularity due to its IEC 61131-3 language compliance, modular architecture, and ability to run on inexpensive hardware such as Raspberry Pi or Arduino boards. However, as highlighted in the recent work by Alsabbagh et al. [60], this flexibility has come at the cost of a fragile security posture.

The architecture of OpenPLC includes several critical elements: the Editor, where ladder logic or structured text programs are developed and compiled into ST files; the Webserver, through which users upload programs and manage runtime configurations; and the Runtime Engine, which interprets these compiled programs and controls process behavior. The platform also maintains a lightweight SQLite database (openplc.db) that stores user credentials, uploaded program metadata, and execution logs.

Despite its functionality and ease of use, this architecture introduces multiple severe vulnerabilities, many of which are actively observable within our testbed. Communication between users and the OpenPLC web interface is transmitted entirely in plaintext over HTTP, exposing login credentials and program data to passive sniffing.
The openplc.db file is not access-restricted and can be directly read or modified through

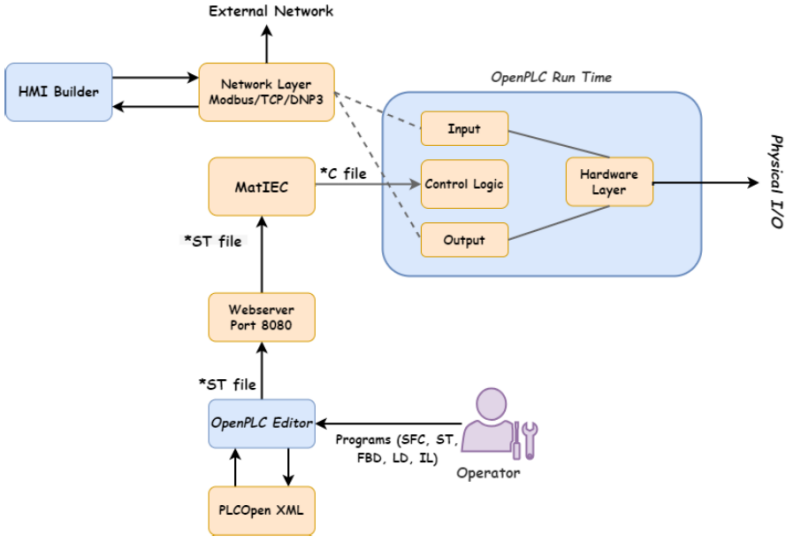

Figure 5.2: OpenPLC architecture

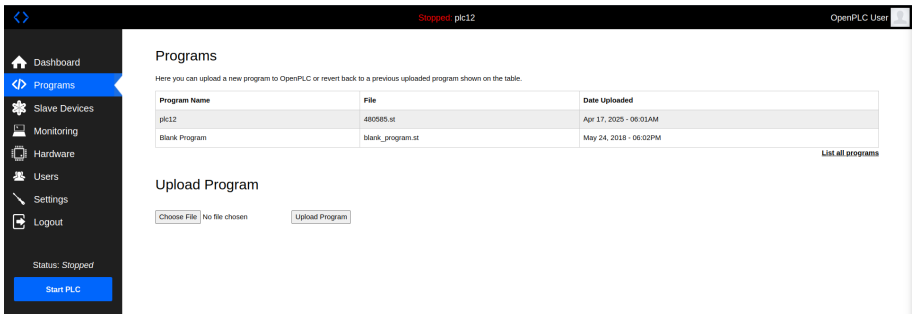

Figure 5.3: OpenPLC ladder logic program list

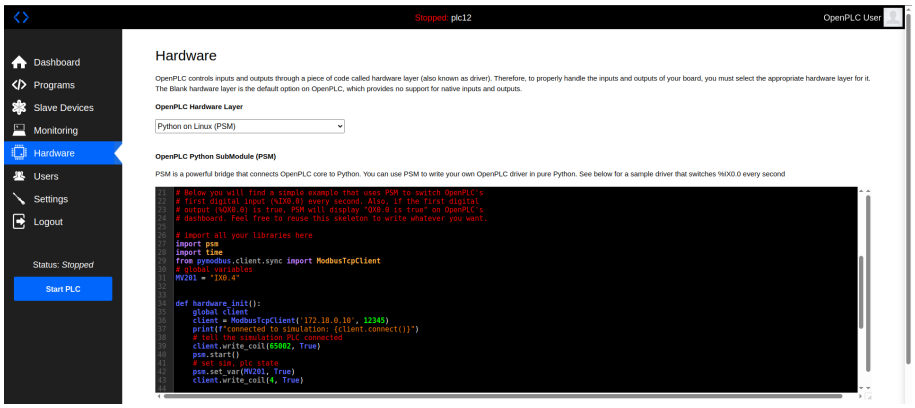

Figure 5.4: OpenPLC PSM configurations

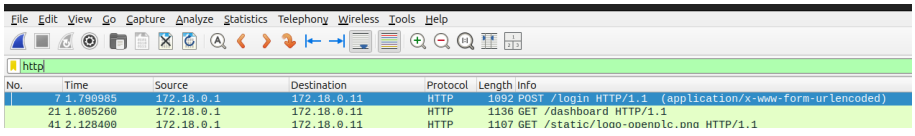

Figure 5.5: Login pattern in wireshark

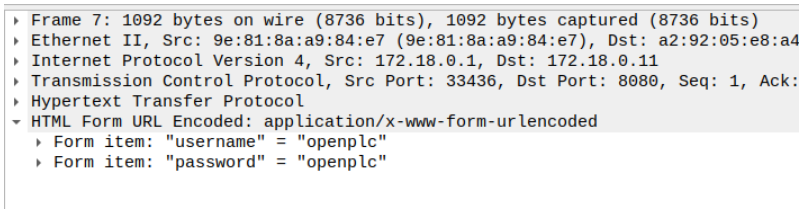

Figure 5.6: Credentials in plaintext

scripts running on the same host or container. This file contains not only user credentials but also program identifiers and status flags that an attacker can use to impersonate users, hijack sessions, or overwrite logic silently.

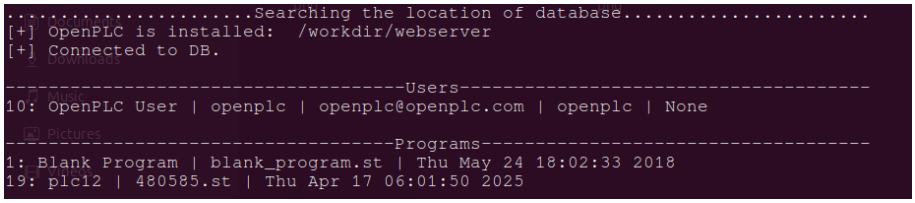

Figure 5.7: Vulnerable OpenPLC database

Further, the process of uploading and executing control logic programs is itself exposed to manipulation. When a user uploads a new program, OpenPLC stores copies of the ST file in persistent directories that are **not cleaned after program deletion.** As demonstrated in Alsabbagh et al.'s [60] analysis, these leftover files can be collected, altered, and re-uploaded stealthily using a modified compilation flow that bypasses the standard interface. The system does not perform any integrity checks, nor does it validate program provenance, allowing malicious logic to be injected into the process without triggering alarms or audit trails.

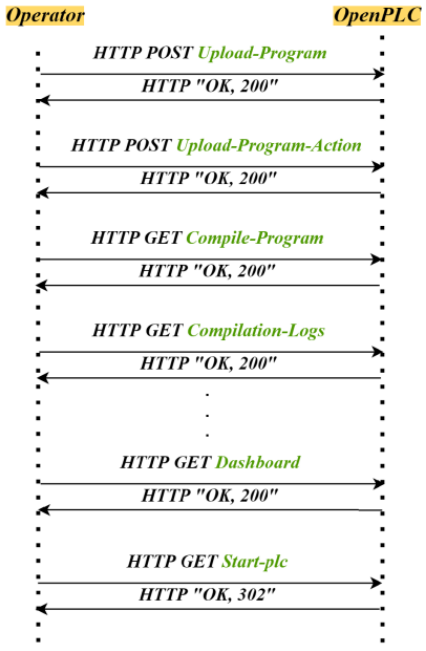

Figure 5.8: Ladder logic program upload process

At this stage in the thesis, our objective is to **observe and document these vulnerabilities using scanning and fingerprinting techniques.** We will not yet exploit them in full; instead, we focus on **gathering intelligence** about the exposed surfaces and architectural weaknesses. In subsequent sections, these same vulnerabilities will serve as the basis for demonstrating targeted attacks including stealthy logic injection, authentication bypass, and Modbus register manipulation each of which highlights the practical risks associated with running critical process control on insecure-by-design platforms like OpenPLC.

5.2 ScadaBR: Architecture and Vulnerabilities

ScadaBR plays a central role in our SCADA testbed, acting as the Human-Machine Interface (HMI) that allows operators to monitor process values, send control commands, and manage alarm states. It sits between the PLC and the user, polling field data through Modbus/TCP and rendering it visually through customizable dashboards. In a typical industrial environment, this kind of system would be the operator's primary point of interaction with the underlying process.

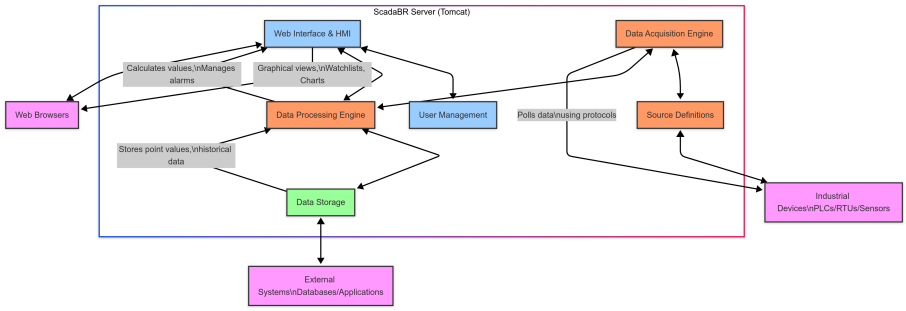

Figure 5.9: ScadaBR core architecture

Figure 5.10: ScadaBR main interface(WatchList)

The platform is built on a Java-based architecture using the Mango M2M core and typically runs on Apache Tomcat. It includes components for acquiring data from devices, rendering graphical views, recording historical values, and defining alarms or automation scripts. In our setup, ScadaBR communicates directly with the OpenPLC over Modbus, continuously polling for sensor and actuator values, and updating the graphical interface with real-time feedback. This interaction occurs entirely over unencrypted channels, which makes the system observable and highly vulnerable to manipulation or impersonation.

The application exposes a web-based interface that allows users to log in, configure data points, upload logic scripts, and change system settings. This surface provides multiple entry points for attackers. Our own fingerprints and the attack scenarios shared by the

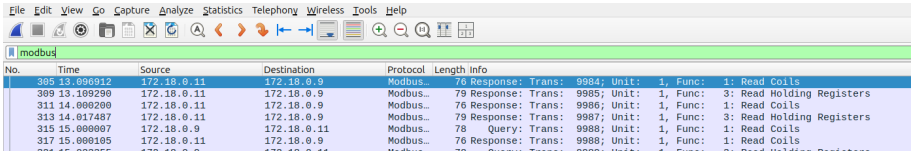

Figure 5.11: Traffic between ScadaBR and PLCs

OT-ICS project [46] confirm that ScadaBR is vulnerable to a variety of web-based threats. For example, we observe that all login attempts are conducted over HTTP, with no HTTPS redirection or encryption. This makes it trivial to capture operator credentials using passive sniffing tools such as Wireshark or tcpdump.

Our investigation confirmed the presence of several publicly reported CVEs, including both stored and reflected cross-site scripting vulnerabilities. One notable issue lies in the system settings panel, where it is possible to inject persistent scripts that execute whenever the page is accessed. In our testbed, we validated this behavior by crafting a JavaScript payload that triggered an alert box each time the affected page loaded. Another flaw is present in the login page itself, where unsanitized URL parameters allow attackers to execute malicious scripts by crafting deceptive links. These issues expose operators to credential theft, session hijacking, and interface tampering [50][45].

A more critical vulnerability allows an authenticated user to upload a **.jsp** file and execute it on the server. This effectively turns ScadaBR into a remote code execution platform, enabling the attacker to issue system-level commands or drop persistent payloads. This behavior was successfully replicated in our environment, demonstrating the potential for complete SCADA compromise once access is gained.

In addition to these core issues, we observed a lack of input validation in form fields and script editors. The platform accepts and executes JavaScript without sandboxing or user confirmation, which opens the door to privilege escalation and unauthorized action chaining through the GUI. From an architectural standpoint, there is no enforcement of user roles beyond basic username-password credentials, and the interface lacks meaningful session timeout or intrusion detection mechanisms.

The vulnerabilities identified in ScadaBR are not isolated bugs; they reflect a broader design pattern of insecure defaults and poor access control. For an HMI system that sits between critical process logic and human operators, these flaws represent significant attack surfaces. In this thesis, we will not exploit them fully at this stage. Instead, we focus on identifying, documenting, and verifying them through reconnaissance, traffic analysis, and controlled testing. Their exploitation will be addressed in later chapters, where we demonstrate their real-world consequences in a controlled, research-driven context.

5.3 Physical Process Simulator: Architecture and Exploitation Potential

In our SCADA testbed, the physical process is emulated through a custom-built Python script that acts as the Master Terminal Unit (MTU). While traditional ICS environments use specialized hardware or PLCs for process simulation and feedback, this approach allows for greater control, observability, and modularity. The simulator responds to control instructions sent by the OpenPLC and generates sensor values that are polled by the SCADA system. It represents dynamic elements such as tank levels, pump states, and flow rates

translating control logic into observable process behavior.

The simulator operates continuously within its own Docker container and interacts with the rest of the testbed through Modbus/TCP, just like any physical industrial device would. It exposes Modbus registers for sensor readings and receives coil commands or holding register values that represent actuator changes. Its architecture is built around a simple event loop that processes state updates, performs logic calculations, and logs outputs. This design allows real-time interaction between the SCADA interface, the control logic running on OpenPLC, and the simulated process state.

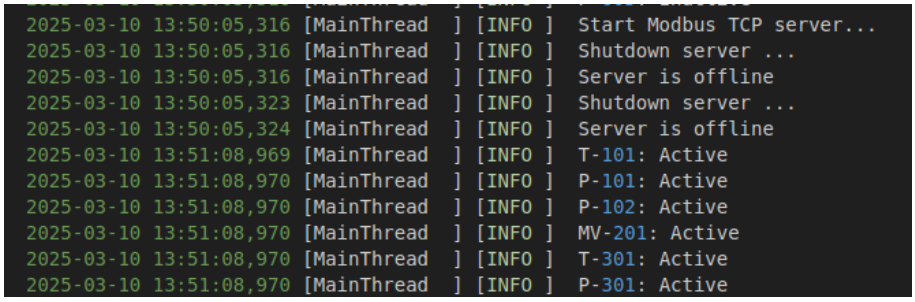

Figure 5.12: Simulator or MTU

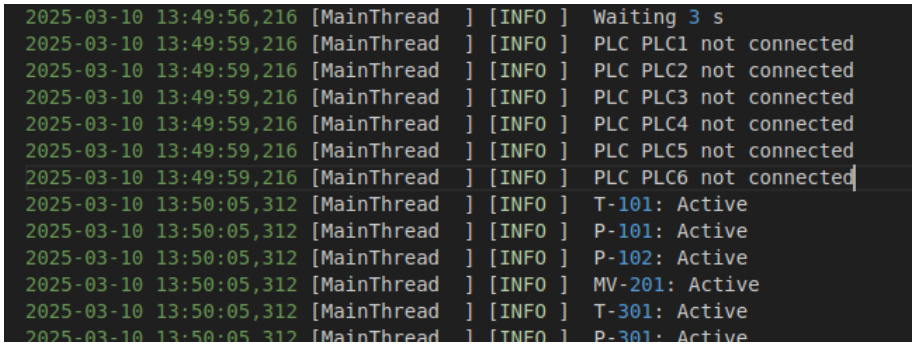

Figure 5.13: PLC status in MTU logs

Because the simulator is written entirely in Python and runs as an open process, it introduces a unique attack surface that would not typically exist in embedded industrial hardware. In a standard ICS setting, MTUs are often hardened and monitored. Here, however, the simulator is as vulnerable as any user-space Python application exposed on a network.

During our preliminary investigation, we identified several characteristics that make this component particularly easy to exploit. The simulator does not perform input validation, and all Modbus requests are processed without verification. This means an attacker who can access the Modbus port can send arbitrary commands that force the process into unstable states activating pumps at the wrong time, causing tank overflows, or rapidly toggling valves. These attacks are not hypothetical; we will later demonstrate how false sensor values and forced actuator states directly alter the simulated process in real time.

Another avenue of exploitation lies within the container's own file system and process access. Since the simulator runs as a Python script, any attacker who gains access to the container (e.g. through a misconfigured Docker socket or a pivot from another compromised component) can modify the script itself. This enables stealthy manipulation of the simulation logic, allowing persistent backdoors or invisible sabotage. For instance, by

inserting a few lines of code, an attacker could log all incoming control instructions, delay certain actuator responses, or inject logic that gradually destabilizes the system.

In our testbed, no integrity checks are applied to the simulator's runtime code or configuration files. There is no watchdog process, no logging mechanism that could alert the operator to code-level changes, and no authentication on internal APIs or register access. This reflects the reality of many low-budget or prototype ICS deployments, where physical simulation and control are decoupled for convenience, but security is not taken into account.

Although the simulator is not directly exposed via a web interface, it becomes an attractive lateral movement target once other components like the SCADA or PLC have been breached. Because it executes the final "real-world" effect of control logic decisions, an attacker who compromises it effectively seizes control of the process layer even without altering the logic on the PLC itself.

In this chapter, our focus is on documenting the vulnerabilities and behavioral weaknesses of the simulator through basic inspection and observation. Full exploitation will be handled later, where we demonstrate how minimal changes to Python logic or Modbus register handling can result in process instability, loss of control, and untraceable sabotage within a research-safe virtual environment.

5.4 Modbus/TCP: Communication Backbone and Vulnerability Vector

The Modbus protocol plays a foundational role in our SCADA testbed, serving as the primary channel of communication between the SCADA system (ScadaBR), the control logic (OpenPLC), and the simulated physical process. Originally introduced in 1979 by Modicon for use with programmable logic controllers, Modbus has evolved into one of the most widely adopted protocols in industrial control environments. Despite its popularity and simplicity, Modbus was designed with no security in mind, which makes it highly vulnerable in modern, network-connected deployments.

In our testbed, Modbus/TCP is used for both polling and control operations. ScadaBR uses it to request current sensor values and to send operator-driven or automated commands to the PLC. The OpenPLC also uses Modbus to read from and write to simulated process states. This bidirectional flow between control, visualization, and process emulation creates a dynamic environment where data is constantly exchanged in real time.

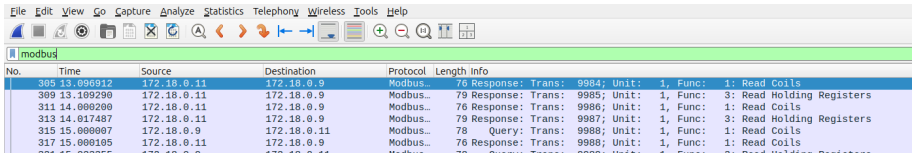

Figure 5.14: Clair modbus functions and code in traffic

However, Modbus communication in this setup is entirely unprotected [22]. All packets travel in plaintext, without encryption, authentication, or integrity checks. Any node on the network that can intercept traffic can read, modify, or replay these messages. For instance, an attacker with access to the network can observe register addresses, data values, and function codes. This allows them to build a complete map of the system's structure what coils control which actuators, which registers correspond to level sensors, and so on.

Because of its deterministic and verbose design, Modbus is also highly predictable. The protocol's simplicity makes it ideal for legitimate industrial use, but it also allows attackers to craft malicious payloads without needing deep technical knowledge. In our reconnaissance phase, tools such as Wireshark, Modpoll, and Scapy allowed us to inspect and manually construct Modbus packets that could manipulate actuator states, simulate sensor failures, or cause SCADA visualization to fall out of sync with real process behavior.

The lack of built-in protections makes Modbus an ideal vector for stealthy attacks, especially when paired with MITM techniques such as ARP spoofing. In a typical attack, the adversary could intercept a Modbus read request from ScadaBR, respond with falsified sensor values while simultaneously sending true data to the PLC, creating a split-view where the operator sees one version of the process and the controller reacts to another. This exact method is documented in multiple studies, including Alsabbagh et al.'s work on stealthy Modbus false command injection [60].

Our testbed recreates these vulnerabilities faithfully. Because there are no firewalls, authentication layers, or application-level gateways, the Modbus service on each component responds to any valid query regardless of origin. During traffic capture, we were also able to observe control logic uploads triggering register updates, making the upload process itself traceable and, potentially, interruptible or spoofable.

In industrial environments, Modbus is still widely used precisely because it is simple, lightweight, and compatible with legacy systems. However, this convenience comes at a cost. When deployed without additional security layers, as in our simulated environment, it becomes one of the weakest links in the system and a **primary enabler of both direct and lateral attacks.**

Understanding Modbus is essential not only for interpreting the reconnaissance results in this thesis but also for appreciating the nature of the attacks that follow. Whether through traffic manipulation, packet injection, or replay, the lack of safeguards in the protocol allows almost any threat actor to move freely through the SCADA environment once they are on the network.

5.5 Summary and Transition to Exploitation

The vulnerability assessment carried out in this chapter reveals just how exposed the testbed is to a wide range of cyber threats. Through architectural inspection, component-level analysis, and documentation of publicly reported flaws, we have identified critical weaknesses across every layer of the SCADA environment. From unauthenticated Modbus access on OpenPLC, to web-based code execution and scripting vulnerabilities in ScadaBR, to the direct manipulability of the Python-based physical simulator, each element of the system presents an entry point for attackers.

These vulnerabilities are not isolated technical misconfigurations, they are systemic design weaknesses that reflect the real-world challenges faced by many operational technology (OT) environments, especially those relying on legacy protocols and open source control platforms. The flat, unsegmented architecture of the testbed only amplifies the risks, making it possible for low-complexity attacks to escalate into full control of the physical process.

Importantly, the assessment performed here does not yet involve active exploitation. Instead, it focuses on identifying what is possible what services are exposed, which components lack controls, and where trust boundaries are missing. These findings will serve

as the foundation for the next chapter, where we transition from observation to action.

In the next chapter, we present a structured series of **attack scenarios** developed to emulate both basic and advanced adversarial behavior. Beginning with a short overview of the attack lab and adversary models, we will move into **reconnaissance and fingerprinting** , and then demonstrate how the vulnerabilities identified here can be exploited to affect both system behavior and physical outcomes. Each scenario will be grounded in the threats modeled earlier and validated within the testbed.

Chapter 6
Exploitation and Attack Scenarios

The exploitation scenarios presented in this chapter are executed within the same SCADA testbed introduced earlier. Unlike the vulnerability assessment phase, which focused on identifying and documenting weaknesses, this chapter moves into **actively leveraging those weaknesses** to compromise the system altering logic, misrepresenting data, and triggering undesired physical outcomes. All actions are performed in a controlled, ethical, and fully virtual environment designed to simulate real-world industrial conditions[26][39].

Testbed Flexibility and Usage

The testbed was designed to be modular and adaptable for a wide range of research and educational applications. While this document focuses on selected scenarios, users can modify key aspects such as PLC logic, process components, and network segmentation. For example, network isolation can be reconfigured or removed by adjusting Docker Compose files or relevant network settings.

To support reuse and customization, a public GitHub repository accompanies this work. It contains the full source code and configuration files. Although a complete manual is not included in this manuscript, the repository is progressively updated with documentation, usage instructions, and sample scenarios. This enables researchers and practitioners to quickly deploy, extend, and adapt the testbed to their own needs. The repository is available at: The repository is available at: `https://github.com/Max-planck-Jr/ICS_Water_testbed_cyber`

6.1 Attack Lab Overview and Threat Context

The attack lab consists of the previously described Docker-based infrastructure, including the OpenPLC controller, ScadaBR HMI, Python-based physical simulator, and a Kali Linux attacker container. All components reside on a shared virtual bridge network with no internal segmentation or intrusion detection, reflecting a worst-case security posture. Communication continues to occur over Modbus/TCP and HTTP, both of which are unencrypted and unauthenticated by design[55].

To maintain relevance and structure, we adopt **multiple attacker profiles** within the same lab: an internal malicious operator with login credentials, a remote attacker who gains access through phishing and reconnaissance, and an unprivileged guest user able to reach exposed ports. Each profile targets different layers of the SCADA system, from web-based interfaces to control logic and sensor-actuator loops.

The exploitation scenarios are designed to **mirror real-world adversarial behavior,** progressing through stages of reconnaissance, credential harvesting, control injection, and persistent manipulation. Each scenario draws directly from the vulnerabilities previously identified and reflects common tactics documented in industrial attack reports and academic research. Rather than enumerating every possible attack, we focus on those with the greatest process impact or educational value.

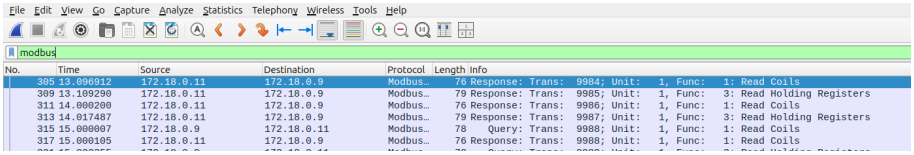

Figure 6.1: Wireshark capturing Modbus/HTTP traffic

In the following sections, we begin with reconnaissance and fingerprinting, not as a passive exercise, but as the first stage of each attack chain. From there, we escalate into direct command injection, logic tampering, stealthy Modbus manipulation, and denial-of-service demonstrations all within a safe yet realistic SCADA simulation environment. All scripts, payloads, and configuration files used in the attack scenarios such as Metasploit handlers, Modbus scanner parameters, and injected logic are provided in the GitHub repository for reproducibility[56].

6.2 Attacker Node: Role, Setup, and Penetration Capabilities

To realistically simulate adversarial behavior within the SCADA testbed, a dedicated attacker machine was integrated into the environment. This node, configured using **Kali Linux,** functions as the offensive toolset base for scanning, fingerprinting, and controlled exploitation. While Kali is often associated with penetration testing in IT networks, it is equally capable in operational technology (OT) scenarios, particularly when adapted with ICS-specific tools and configurations.

In the context of our experiment, the attacker machine is deployed as a Docker container within the same virtual bridge network that connects the PLC, SCADA, and physical process simulation. This placement mirrors the kind of access an attacker would have after breaching the perimeter of a poorly segmented ICS environment. However, throughout this work, we do not restrict ourselves to a single attacker profile. Multiple scenarios are considered, including internal threats originating from a compromised engineering workstation, lateral movement from breached IT assets, or even a third-party service point with temporary access to the OT network. This flexible approach ensures that the analysis reflects both realistic entry points and worst-case assumptions.

The Kali node was pre-loaded with a wide range of reconnaissance and exploitation tools. During the passive information gathering phase, Wireshark, tcpdump, and SET (Social Engineer Toolkit) were used to intercept traffic and simulate credential harvesting through phishing. These tools revealed unencrypted Modbus commands, HTTP-based ladder logic uploads, and cleartext authentication exchanges, confirming the absence of even the most basic security controls across all layers.

During active probing, tools like Nmap, PLCScan, and modpoll enabled rapid enumeration of exposed services and Modbus registers. With no access control or rate-limiting in place, the attacker was able to identify register maps, discover default login portals, and verify that service banners disclosed version details providing everything needed for follow-up payload development.

In addition to standard tools, custom Python scripts and crafted packets were developed to test low-level behaviors of the physical process simulator and to evaluate the resilience of the Modbus protocol. These utilities, executed directly from the Kali container, allowed a

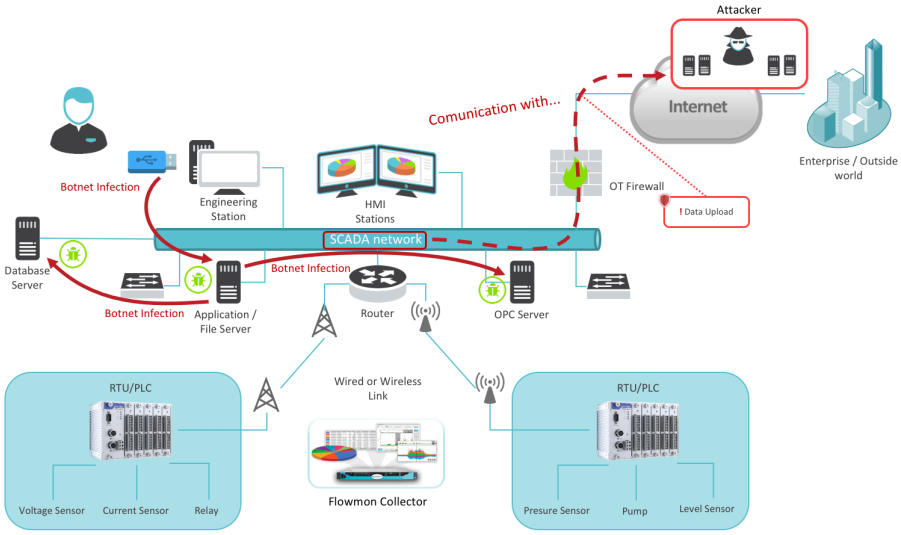

Figure 6.2: Attacker patterns in Scada network

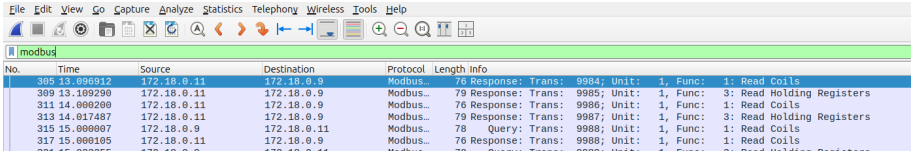

Figure 6.3: Scada Network traffic

detailed analysis of how each system component reacted to malformed or malicious inputs, laying the groundwork for attack chains that will be implemented and evaluated in later sections.

The presence of this attacker node does more than simulate intrusion. It also represents a tool for educating and training students, engineers, and researchers in the nature of ICS vulnerabilities. Through this environment, users can experience the attacker's perspective without the legal, ethical, or operational risks associated with real system testing.

As the following sections will show, the attacker node is capable of affecting all parts of the SCADA system. From intercepting SCADA-to-PLC communication to injecting false logic into OpenPLC, and from hijacking HMI views to disrupting the simulated physical process, the node enables a full-spectrum analysis of the threats an unprotected ICS environment may face in the wild.

6.3 Reconnaissance and Fingerprinting

In the early phase of the simulated attack, the adversary is assumed to have already gained access to the internal network. This could reflect an insider threat a successful lateral movement from the IT layer. From this initial foothold, the attacker begins with passive and active reconnaissance to understand the structure, services, and devices that make up the SCADA environment.

6.3.1 Network Sniffing

The first step involved confirming the attacker's own network configuration. The attacker, already present in the network, began by identifying their own IP address using the **ip**

addr command. The interface revealed an IP in the 172.18.0.0/24 subnet.

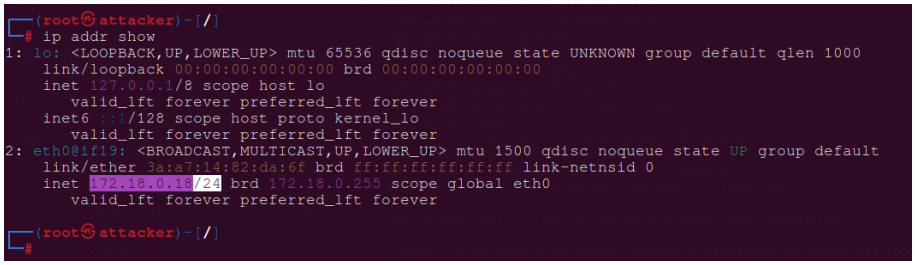

Figure 6.4: Attacker discovers its IP address in the SCADA network

Next, **arp-scan** was used to detect active hosts. Six additional machines were identified. These responses confirmed the presence of multiple SCADA components on the network.

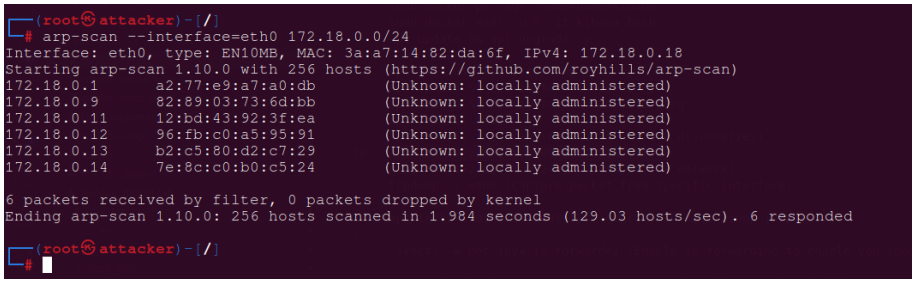

Figure 6.5: Discovery of SCADA components using arp-scan

A network discovery tool was also used to verify ARP activity and visualize host responses. It captured packets from six unknown hosts, matching the ARP scan results.

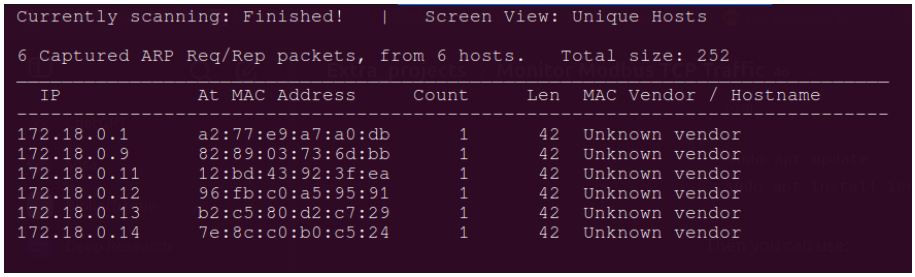

Figure 6.6: ARP fingerprinting using discovery tool

The attacker then performed an **nmap -sn** scan to verify host availability. Hostnames such as **plc11.swat and HMI-HIS.swat** were revealed, clearly indicating system roles.

At this stage, without sending any exploit, the attacker mapped the entire SCADA network, including PLCs, HMI, and the process simulator. This confirmed the lack of host discovery defenses or network segmentation.

6.3.2 Capturing Modbus/TCP Traffic via ARP Spoofing

After identifying the SCADA components and confirming IP reachability, the attacker attempted to monitor communication between the HMI and the PLC. Direct observation using **tcpdump** initially failed to capture any Modbus traffic, suggesting the attacker was outside the direct communication path [8][43].

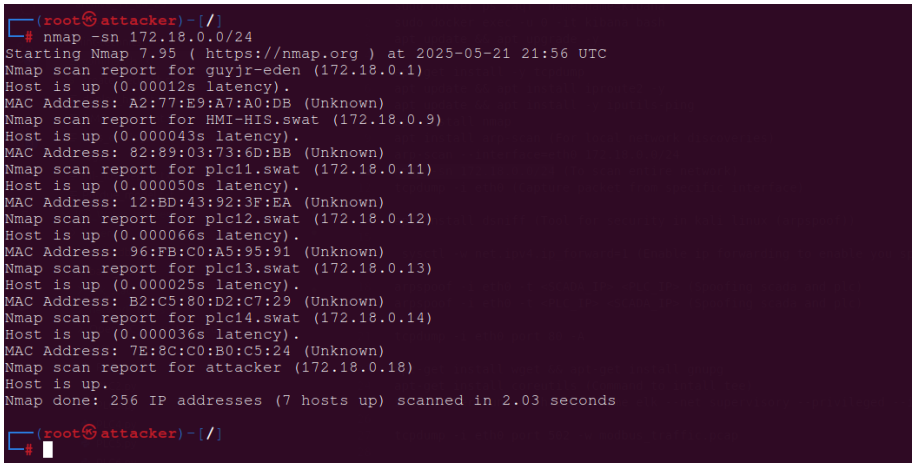

Figure 6.7: Host discovery and name resolution with nmap

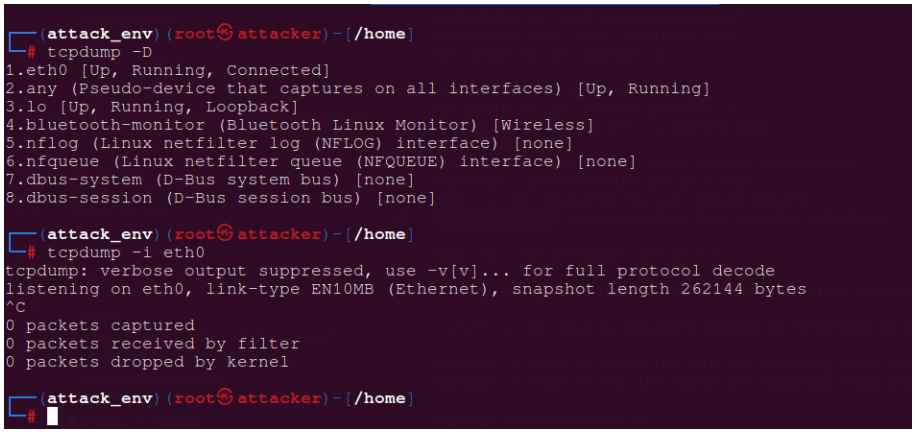

Figure 6.8: Attacker fails to capture Modbus traffic directly

To overcome this limitation, the attacker executed a **man-in-the-middle (MITM)** attack using ARP spoofing. The goal was to intercept all packets exchanged between the SCADA interface (at 172.18.0.9) and the PLC (172.18.0.11). The arpspoof utility was used to poison ARP caches and redirect traffic through the attacker's node.

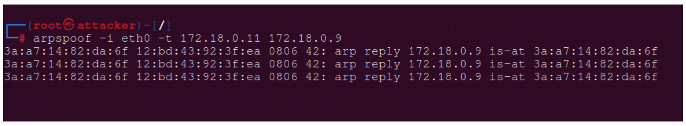

Figure 6.9: ARP spoofing PLC to SCADA

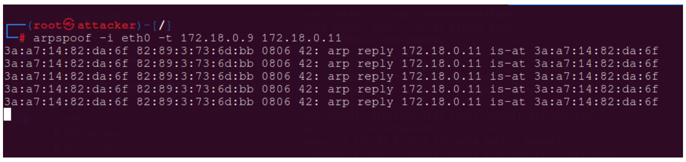

Figure 6.10: ARP spoofing SCADA to PLC

Once both ends were spoofed, the attacker re-ran **tcpdump**, this time successfully capturing the Modbus TCP communication in transit. The terminal output showed packets with predictable SCADA-to-PLC read/write behavior, confirming that the attacker was now effectively intercepting control-layer traffic.

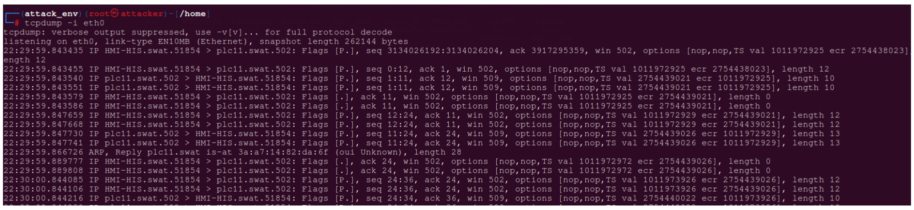

Figure 6.11: Intercepted Modbus/TCP packets during ARP spoofing

To preserve this data for later analysis, the attacker exported the traffic into a **.pcap** file using **tcpdump -w** , enabling further inspection in Wireshark. This file would later help inspect modbus packets, map register addresses, analyze function codes, and detect patterns in actuator control sequences.

Figure 6.12: Saving intercepted SCADA-PLC traffic for offline analysis

This technique confirmed a core vulnerability of Modbus/TCP: its complete lack of traffic integrity or source authentication. The attacker required no credentials and triggered no alarms while placing themselves invisibly between two critical ICS components. This interception technique is not only viable in controlled environments,it mirrors methods used in real-world attacks on critical infrastructure.

6.3.3 Intercepting Operator-to-OpenPLC Runtime Communication

After successfully performing ARP spoofing to capture traffic between the SCADA interface and the PLC, the attacker turned their attention to another critical interaction: the communication between the OpenPLC operator interface and the OpenPLC runtime. This path is particularly valuable because it carries sensitive actions such as program uploads, compilation requests, and operator login credentials.

The attacker used the same technique as before spoofing ARP replies between the operator's machine and the PLC at **172.18.0.12** . This enabled them to insert themselves in the communication flow and observe all traffic using **tcpdump**.

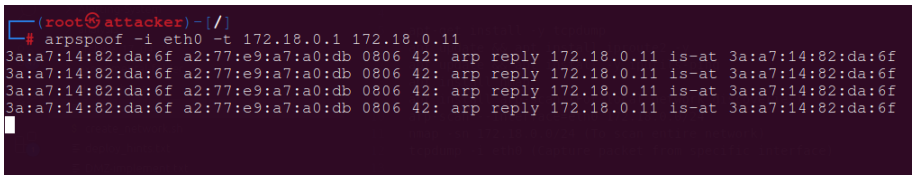

Figure 6.13: ARP spoofing between OpenPLC operator and runtime

Traffic was then captured and written to a PCAP file for detailed analysis. The tool collected over a thousand packets in just a short session, demonstrating the volume of HTTP-based communication between the two components.

Figure 6.14: Saving intercepted OpenPLC traffic to PCAP file

Opening this capture in Wireshark revealed that the entire OpenPLC interaction occurs over plaintext HTTP. During the login process, both the **username and password** were visible directly in the packet payload. The attacker was able to extract these credentials with no effort.

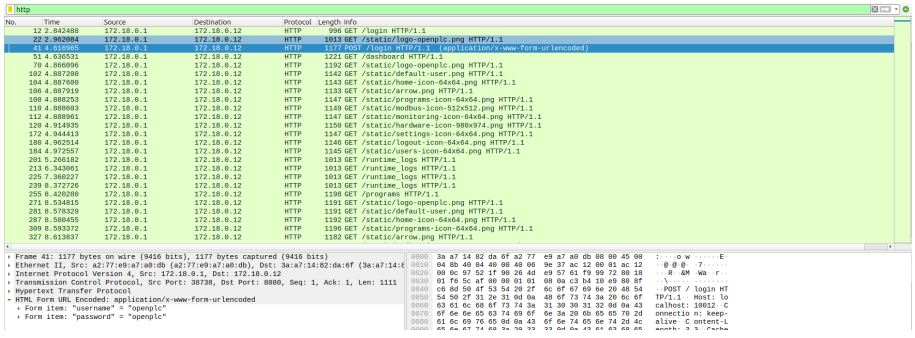

Figure 6.15: OpenPLC login credentials visible in plaintext

Further inspection showed the full ladder logic program upload sequence. This included HTTP POST requests to endpoints such as **/upload-program, /compile-program,**

and /start-plc, /upload-program, /compile-program, and /start-plc, confirming the step-by-step control process used by the operator.

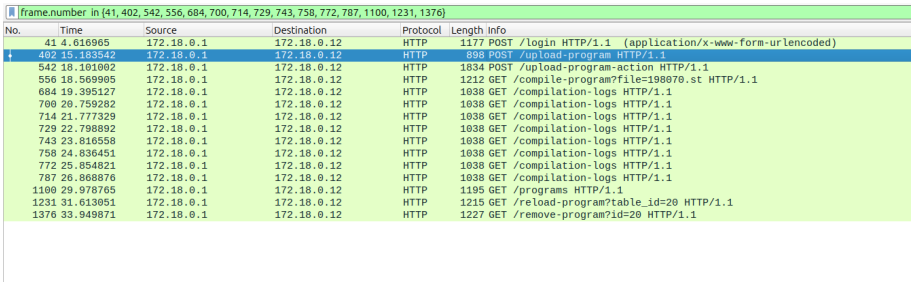

Figure 6.16: Intercepted OpenPLC logic upload process

The attacker was now able to reverse engineer the operator's interaction workflow. They knew when logic was uploaded, when it was compiled, and when the controller was activated. This insight provides all the context needed to later develop replay attacks, logic manipulation payloads, or even timed disruptions that coincide with upload operations.

This analysis underscores a major security gap in OpenPLC: the lack of encrypted communication channels or any authentication validation at the traffic level. With only ARP spoofing and passive sniffing, a man-in-the-middle attacker can observe or hijack the most critical parts of the ICS control chain.

6.3.4 Fingerprinting and Vulnerability Scanning on ScadaBR

Once the attacker identified the HMI component at **172.18.0.9** , further investigation was performed to fingerprint the web server and discover potential vulnerabilities. The goal was to map the technology stack supporting ScadaBR and find ways to either bypass authentication or exploit known flaws.

The attacker began with **WhatWeb** , a reconnaissance tool used to analyze HTTP headers and infer technologies used by the web server. It confirmed that the service was running on **Apache Tomcat**, powered by the **Mango M2M platform** , with active session cookies and basic HTML login forms. These details hinted at default configurations and potential weaknesses in web input handling.

Figure 6.17: Web fingerprinting on ScadaBR using WhatWeb

Next, Nikto was used to scan for known web vulnerabilities. The results confirmed that the server was using **Apache-Coyote/1.1** and had , such as X-Frame-Options or X-Content-Type-Options. More critically, Nikto discovered that the **Tomcat Manager interface** was exposed at **/manager/html** and **/host-manager/html** , which are commonly targeted by attackers for code execution and admin access.

After confirming that the Tomcat manager interface was reachable, the attacker attempted a **brute-force login** using Medusa. A small password dictionary was used against the default "admin" username. Within seconds, the attacker successfully authenticated using the credentials **admin:password** , gaining access to the Tomcat administration panel.

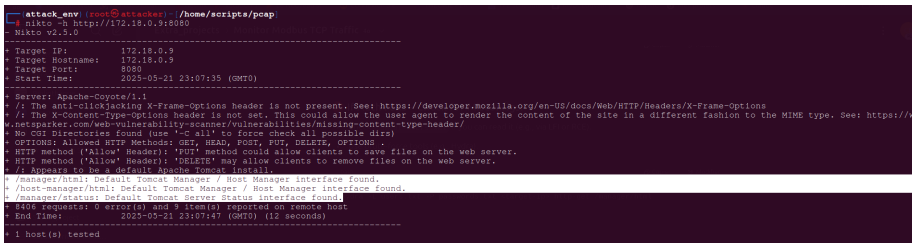

Figure 6.18: Vulnerability scanning on ScadaBR web server using Nikto

Figure 6.19: Admin access to Tomcat manager via password brute-force

This combination of fingerprinting, vulnerability detection, and brute-forcing represents a common attack path in SCADA environments where default configurations are left exposed. Without rate-limiting, account lockout, or intrusion detection, the attacker was able to access a critical control interface using automated tools and known defaults.

With access to the Tomcat manager, it becomes possible to upload custom JSP shells, execute arbitrary commands on the server, or hijack session tokens all without directly attacking the ScadaBR application logic. This opens a high-privilege backdoor into the HMI and allows for long-term persistence or manipulation of operator views.

6.4 Modbus Attack Scenarios

6.4.1 Modbus Exploitation - Unauthorized Read Operation (Data Exfiltration)

The first attack scenario explores the risk posed by unauthorized Modbus reads a common yet often overlooked threat in ICS networks. While the absence of authentication in Modbus/TCP is widely known, the ease with which an attacker can **extract sensitive process data** simply by connecting to a device and issuing read commands remains alarming.

To demonstrate this vulnerability, we used the **Matrix attack tool**, developed and released by **Karl Biron** and featured by **Trustwave SpiderLabs** in their publication *"Lights Out and Stalled Factories"* [12]. The tool was designed specifically for learning and demonstrating the security shortcomings of Modbus-based SCADA environments.

In our testbed, the attacker used Matrix to connect to the PLC11 Modbus server over port 502. Without any credentials or prior access, a full read operation was executed across coil statuses, discrete inputs, input registers, and holding registers. The device responded with all data values including actuator states and water level readings as shown in the Python terminal output.

During the process, the process simulator was active, updating LIT101 (the tank level indicator) based on the execution of normal logic. These values were captured and visualized through the ScadaBR dashboard and the OpenPLC web interface, both reflecting consistent, live data at the moment the attack was performed.

The attacker, however, was able to **silently intercept these values** from outside either interface. Using Matrix's read operation **-a read**, all process telemetry was retrieved in real time, without alerting either the SCADA operator or the PLC logic engine. Among

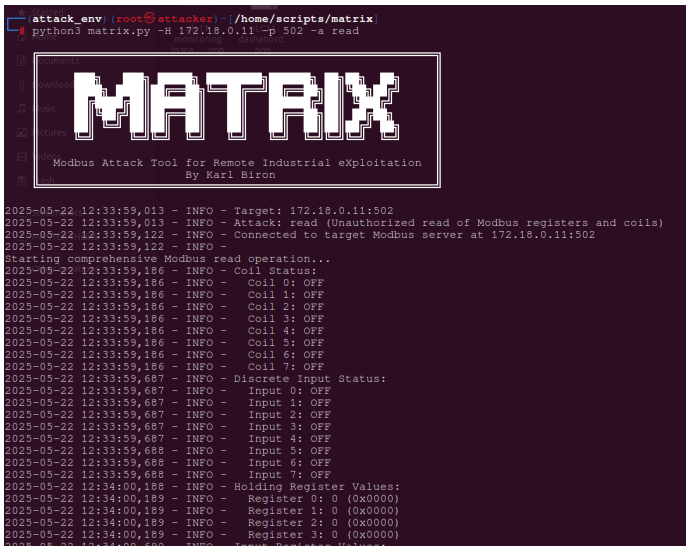

Figure 6.20: Unauthorized Modbus data read using Matrix

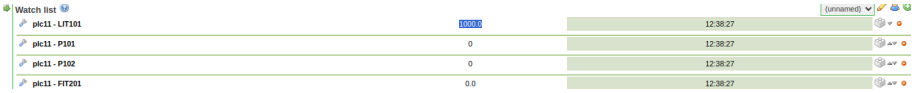

Figure 6.21: Normal process values visible during read

the extracted values was a holding register with a value of 1000, corresponding to a high tank level state actively managed by the ladder logic.

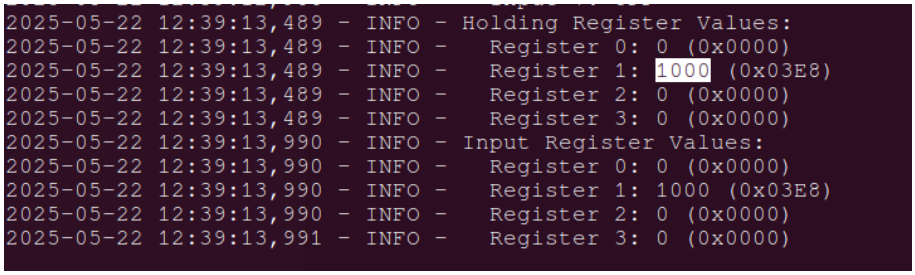

Figure 6.22: Water level exfiltrated via unauthorized read

This attack proves that even without modifying logic or sending any write commands, an attacker can collect critical real-time telemetry from ICS systems. In an industrial setting, this kind of data could be used to plan timed physical sabotage, develop more sophisticated spoofing payloads, or enable ransomware operations targeting process stability.

What makes this vulnerability especially dangerous is its stealth. Unlike write-based attacks that might immediately disrupt the system, unauthorized reads are quiet, non-disruptive, and easy to overlook yet they reveal everything. In our simulation, no alert was triggered, no log was generated, and no user was notified.

This scenario confirms the urgent need for network-level protections, such as Modbus-aware firewalls, role-based access control, and encrypted SCADA communication layers. Without these, even a basic unprivileged attacker can observe and harvest industrial secrets using open-source tools.

6.4.2 Modbus Exploitation - Coil Manipulation Attack (Turning Devices On/Off)

The second attack scenario demonstrates how an unauthorized attacker can manipulate coil states to control critical actuators such as pumps, valves, or alarms by simply sending write commands to the Modbus server. In an unprotected ICS network, where Modbus/TCP traffic is unauthenticated, this type of attack can be launched with minimal effort and devastating impact.

This scenario was executed using the same tool developed by **Karl Biron**, Matrix, which is designed to expose security flaws in industrial systems using Modbus protocol. Trustwave's SpiderLabs blog has previously showcased how this tool can be used to illustrate real-world ICS risks [12]. Before the attack, the status of PLC11's coils was verified from both the OpenPLC web interface and the Matrix tool's read function. All actuators were set to their default state **(OFF)**, and no unauthorized operations had yet been executed.

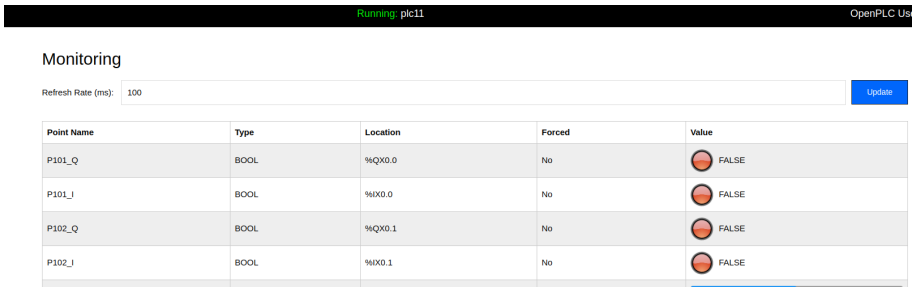

Figure 6.23: Coil status before attack (OpenPLC view)

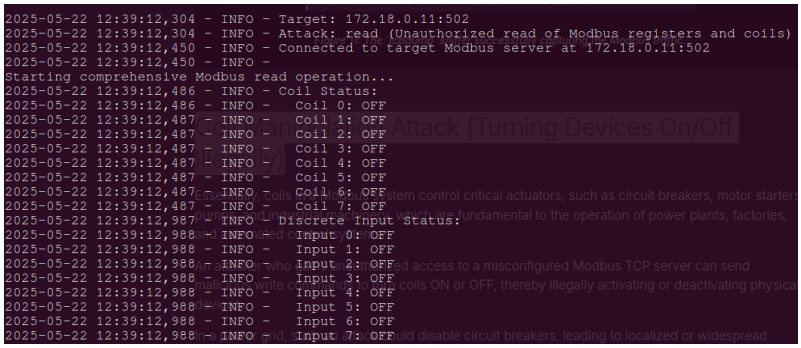

Figure 6.24: Initial coil states as seen by Matrix

The attacker then launched a coil write operation using the Matrix tool's **-a coil** action. This sent raw Modbus write commands to coil addresses on the PLC at **172.18.0.11**. The device accepted the requests without challenge and acknowledged the changes. A follow-up read confirmed that coils had been modified: actuators previously OFF were now ON.

To enhance the persistence of this attack, the attacker created a bash script named **coil_overwrite_loop.sh**. This script issued write commands in a continuous loop, ensuring that even if the system logic tried to restore the original coil states, the attacker's values would be reapplied every second.

This scenario illustrates how **basic write access** to a Modbus server can turn into a complete **actuator hijack.** Without authentication, command validation or monitoring, the attacker was able to force process changes in real time. In a real ICS environment, this

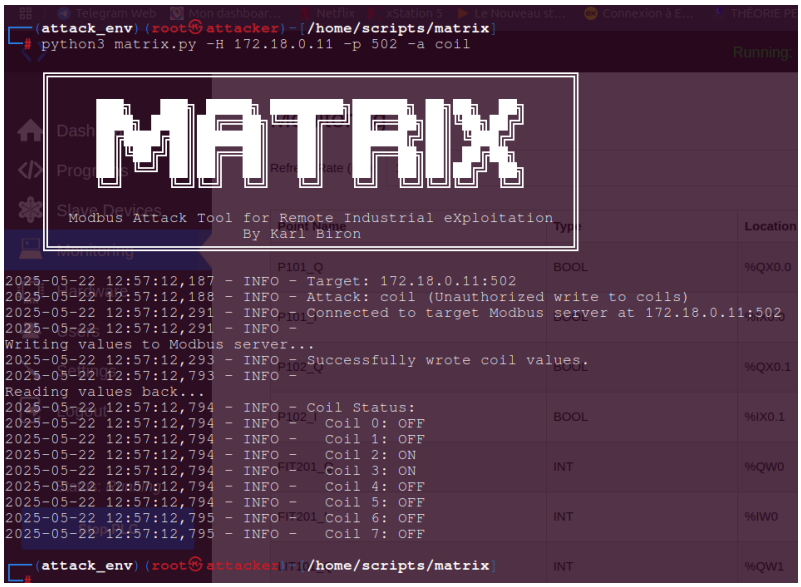

Figure 6.25: Coil values successfully manipulated by unauthorized attacker

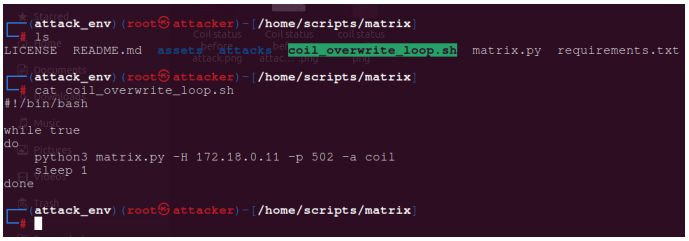

Figure 6.26: Looping coil attack to maintain control

could lead to unsafe pump cycles, tank overflows, or interrupted process flow.

As with the unauthorized read scenario, this attack was not detected by the SCADA or PLC system. There were no logs, alerts, or visual warnings to the operator. The system simply accepted the instructions, even though they came from an unauthorized and potentially malicious source. This reinforces the critical need for Modbus-aware intrusion detection, secure write protections, and access control enforcement at the network and protocol layers.

6.4.3 Modbus Exploitation - Denial of Service Attack (Resource Exhaustion)

In this attack scenario, we simulate a Denial of Service (DoS) attack on Modbus devices by overwhelming their Modbus TCP ports with concurrent connection threads. This attack does not require credentials, exploits no software vulnerability directly, and yet can effectively disrupt the availability of the system's control logic. It is a textbook case of resource exhaustion in exposed ICS environments.

Before launching the attack, we used the **htop** utility to capture baseline CPU usage on both PLC11 and PLC12. Both devices showed normal load averages and minimal thread activity.

The attack was then launched from the attacker node using 1000 concurrent threads against the Modbus service on PLC11:

```
python3 matrix.py -H 172.18.0.11 -p 502 -a dos -t 1000
```

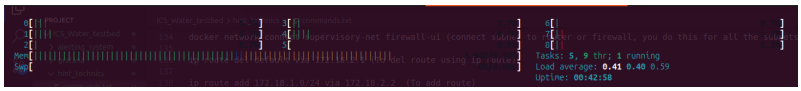

Figure 6.27: PLC11 operating normally before DoS attack

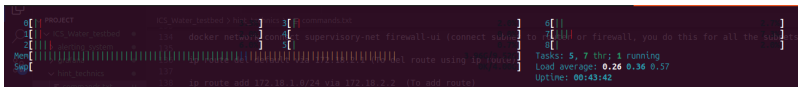

Figure 6.28: PLC12 operating normally before DoS attack

The matrix immediately began to create threads and send aggressive connection requests, flooding the target with traffic. The attack was sustained over a short period, enough to simulate a real-world overload condition.

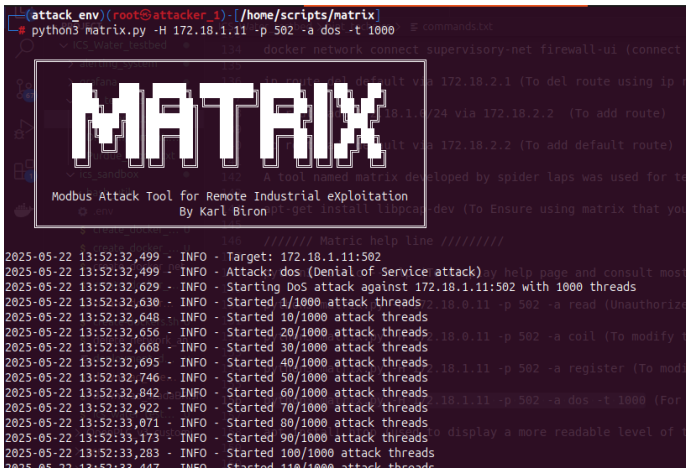

Figure 6.29: Denial of Service attack initiated using Matrix

Following the attack, we again inspected the CPU metrics. This time, both PLC11 and PLC12 showed significantly higher load averages and increased thread counts. In particular, PLC11, the primary target, displayed nearly saturated CPU cores and multiple queued or active Modbus threads. These effects degraded the system's responsiveness and could lead to logic execution delays or complete runtime failure if sustained.

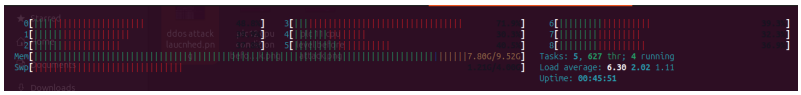

Figure 6.30: Resource exhaustion observed post-DoS attack on plc11

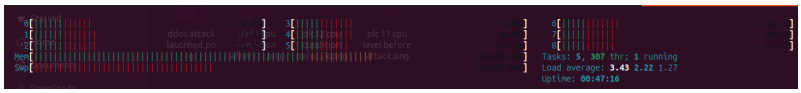

Figure 6.31: Resource exhaustion observed post-DoS attack on plc12

This attack highlights a key flaw in many ICS deployments: a lack of rate limiting or connection throttling on Modbus services. Without such protections, even a low-skills attacker can degrade system performance and interfere with control loop timing, potentially leading to unsafe states or system lockups.

During the attack, no alerts were triggered, no firewall attempted to filter the traffic,

and the SCADA interface continued to poll normally completely unaware of the underlying performance degradation. This demonstrates the **invisibility and effectiveness** of protocol-level DoS in systems that rely on Modbus over TCP/IP without modern security mechanisms.
The Modbus-based attacks demonstrated in this section highlight the critical security flaws that continue to exist in many industrial control systems. Through the use of the Matrix tool developed by Karl Biron and inspired by the practical work presented by Trustwave's SpiderLabs team, we were able to replicate multiple real-world scenarios in our testbed, each of which requires no authentication and causes significant impact.

We showed how a simple unauthorized read operation can silently exfiltrate process values such as tank levels and actuator states. We followed that with an unauthorized coil manipulation attack, allowing the attacker to take direct control of actuators like pumps and valves, bypassing all operator or logic-layer safeguards. Finally, we demonstrate a denial-of-service attack that overwhelmed Modbus services using concurrent thread floods, resulting in CPU saturation and performance degradation on targeted PLCs.

All of these attacks required **minimal setup, no specialized hardware, and no insider knowledge.** With open-source tools and default configurations, any attacker present on the same network as an unprotected Modbus device can perform the exact same techniques read, manipulate, or disable critical process operations.

For researchers, educators, and security analysts, this work provides a reproducible, controlled environment to understand how Modbus vulnerabilities can be observed, exploited, and eventually mitigated. The modular nature of our testbed makes it easy to simulate additional use cases, test intrusion detection rules, segmentation etc.

The lessons from this section make it clear: **Modbus should not be deployed in isolation without layered defenses.** When used without authentication or encryption, as is still common in legacy and budget-constrained systems, it becomes one of the weakest links in industrial networks.

Having explored the core vulnerabilities at the Modbus protocol layer, we now turn our attention to a different attack surface **the OpenPLC controller itself.** In the next section, we investigate attacks that exploit the web interface, runtime behavior, and storage architecture of the OpenPLC platform to modify logic, escalate privileges, and compromise control integrity.

6.5 OpenPLC Attack Scenarios

6.5.1 Remote Access and Database Extraction via Payload Injection

In this scenario, we move beyond the limitations of protocol-level attacks and demonstrate how an attacker can gain direct access to the underlying file system and database of the OpenPLC controller. This kind of attack enables full manipulation of the PLC's configuration, stored logic programs, user credentials, and runtime metadata. It simulates a targeted compromise where the attacker uploads a malicious payload, establishes a reverse shell, and executes commands to steal or modify critical information.

The attack begins with the generation of a payload using **Metasploit's msfvenom tool**, configured to initiate a reverse TCP connection from the target PLC to the attacker's host. The payload is crafted in **ELF format,**masquerading as a firmware update file to blend in with legitimate operations. It's configured with the attacker's IP and a listening port to receive the incoming connection once the payload is executed.

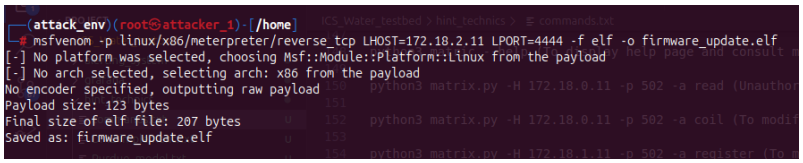

Figure 6.32: Payload generation for remote access

The attacker then launches **Metasploit's multi/handler module,** listening on the specified port and awaiting the connection. Once the payload is executed on the target machine (PLC11), a **reverse shell is established,** giving the attacker full access to the device's terminal through Metasploit's Meterpreter interface.

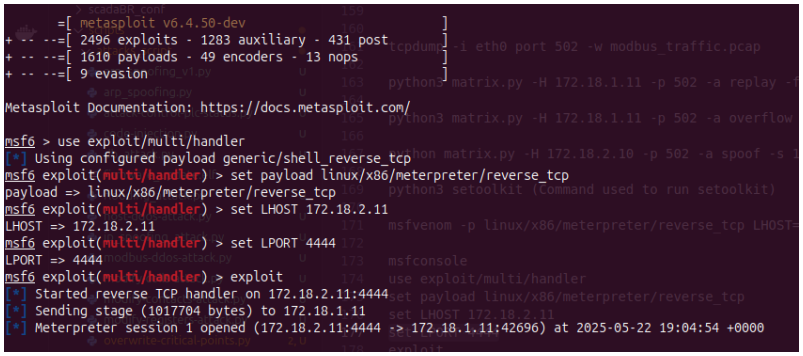

Figure 6.33: Reverse shell established on PLC11

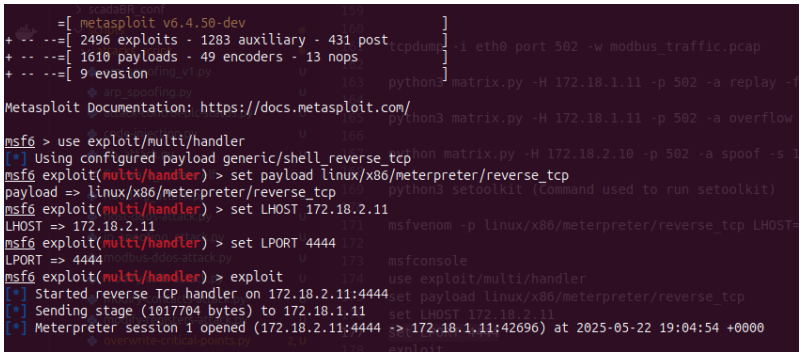

Figure 6.34: Reverse shell established on PLC11

Once inside the system, the attacker locates the OpenPLC installation directory and verifies that the SQLite database (openplc.db) is present in the expected path (/workdir/webserver). This database stores critical information such as the list of authorized users, active ladder logic programs, and metadata for the controller's runtime.

To automate the database access and extraction, a custom script named db_attack.py is uploaded into the PLC using the Meterpreter upload command. This script connects to the SQLite backend and dumps all user records and logic program details from the OpenPLC database.

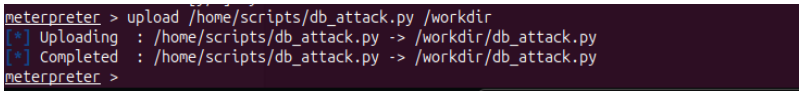

Figure 6.35: Script transferred to compromised PLC

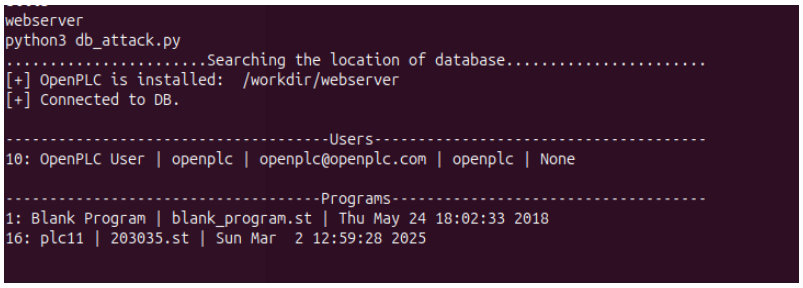

Figure 6.36: Database contents dumped post-compromise

The result is a complete list of all logic programs installed on the PLC, their file paths, and associated users. In our testbed, this included blank_program.st and plc11_203035.st, alongside the OpenPLC default user credentials.

This attack showcases the **impact of gaining file-level access** to a PLC controller. Beyond Modbus interaction, the attacker now has the ability to:

- Replace or modify ladder logic files.
- Alter user credentials or role permissions.
- Extract sensitive system metadata and process configurations.

None of these actions triggered alerts on the SCADA interface or within the OpenPLC runtime. The attacker operated silently through the shell, exploiting a platform that lacked basic endpoint protection, file integrity checks, or shell access restrictions.

This scenario emphasizes the need for hardening the PLC environment, not just securing its network services. Protection against payload injection, shell access control, and database encryption must be considered for any deployment of OpenPLC in a production-like environment.

6.5.2 Code Injection and Control Logic Manipulation

In this attack, the objective was to tamper with the control logic of PLC11 by injecting malicious Python code capable of modifying live process values and actuator states. Unlike Modbus-level manipulation, this scenario targets the internal behavior of the PLC runtime, creating persistent and deceptive alterations at the data source[53].

Access to the target was gained via a reverse shell using the same payload delivery technique demonstrated in the previous OpenPLC attack scenario. Once the shell was active, the attacker uploaded a custom script named **code-injection.py**, which was specifically developed to manipulate runtime values used by both OpenPLC and ScadaBR.

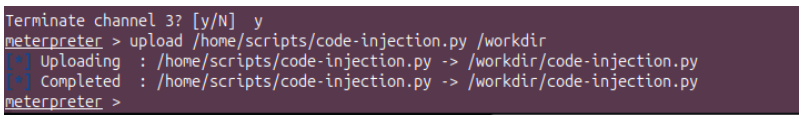

Figure 6.37: Injection script transferred to compromised device

Before execution, the attacker retrieved credentials from the OpenPLC database and verified the currently running ladder logic file. This confirmed that the target was actively executing 203035.st, and that valid user credentials (e.g. openplc:openplc) were still accepted for authenticated operations.

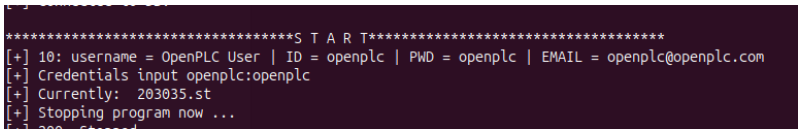

Figure 6.38: Malicious code-injection.py available on PLC11

After resolving some missing dependencies (such as the requests module), the attacker executed the code injection script. The payload introduced random or falsified values into critical registers such as tank levels, flow rates, and pump states which were immediately reflected in the OpenPLC monitoring interface and the ScadaBR dashboard.

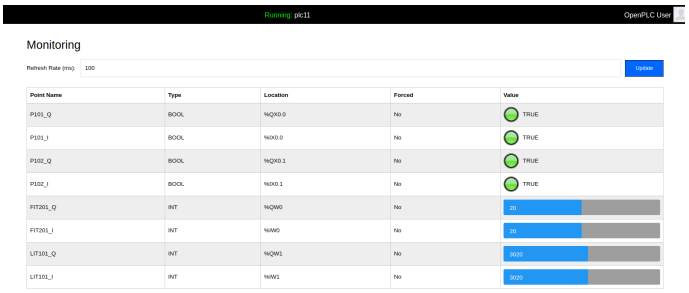

Figure 6.39: Tampered values seen in PLC interface before injection attack

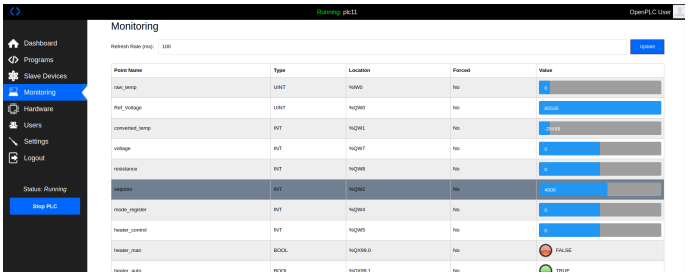

Figure 6.40: Tampered values seen in PLC interface after injection attack

Figure 6.41: Tampered values seen in SCADA interface

Post-attack, the logic continued to appear functional. However, the data feeding that logic was no longer trustworthy. The attacker had effectively **injected a runtime distortion layer,** hijacking the view of the physical process and masking unauthorized control over system behavior.

This attack illustrates the **high-risk nature of code-level vulnerabilities** in PLC platforms like OpenPLC. When access controls are weak or missing, attackers can bypass the logic upload process and modify values directly at runtime. Without detection mechanisms, the effects can go unnoticed while causing real operational damage.

6.6 ScadaBR Attack Scenarios

6.6.1 Information Gathering on target

The attacker began by scanning the IP address of the HMI server (172.18.0.9) using nmap. The results revealed two key services running on the target:

- Apache Tomcat/Coyote JSP engine (port 8080).
- MySQL (MariaDB) database (port 3306).

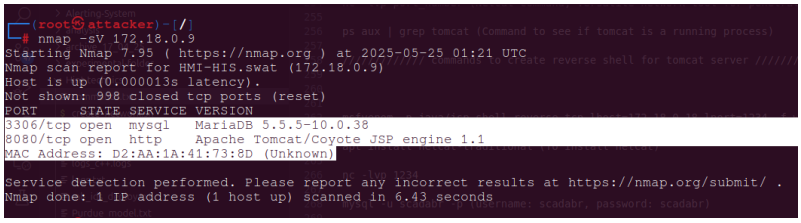

Figure 6.42: Service discovery on ScadaBR host

This confirmed that ScadaBR is running on top of an Apache Tomcat server and is backed by a SQL database, opening two clear exploitation paths: web service injection and database manipulation.

6.6.2 Historian(MySQL) - Credential Brute-Force and SQL Tampering

Using **Metasploit's mysql_login module**, the attacker performed a brute-force attack against the historian, successfully gaining access with default credentials **(scadabr:scadabr)**. An interactive session was launched from within Metasploit to explore and manipulate the database.

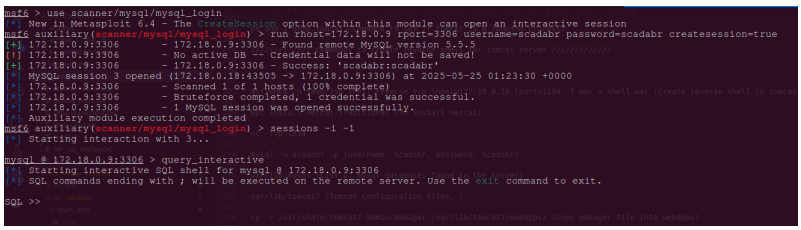

Figure 6.43: Gained access to ScadaBR's MySQL database

Once connected, the attacker explored the **events** table, which logs operator actions, alerts, and system status. A query revealed an active login alarm **(event.login|admin)** stored with ID 403.

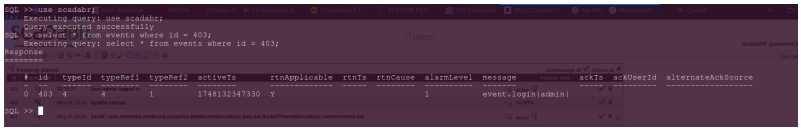

Figure 6.44: Alarm log located inside MySQL database

To simulate alarm tampering, the attacker issued a delete query to remove this event from the system:

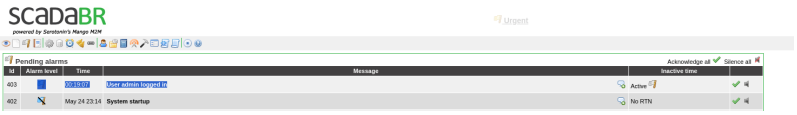

Figure 6.45: Alarm in Scada interface before deletion

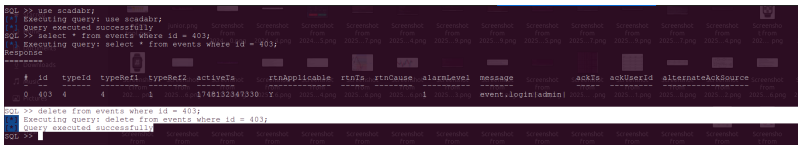

Figure 6.46: Deletion of alarm record via SQL injection

Upon refreshing the ScadaBR interface, the login alarm was gone silently removed from the dashboard. This scenario demonstrates how a malicious user with database access can erase forensic traces and interfere with operator visibility.

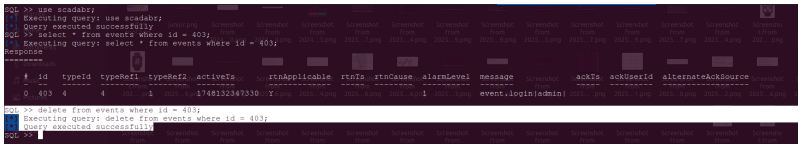

Figure 6.47: Deletion of alarm record via SQL injection

6.6.3 Reverse Shell Access via Apache Tomcat

Having discovered the Tomcat server, the attacker proceeded to deploy a reverse shell via the Tomcat Manager's **/manager/html**, interface. A WAR payload was crafted using Metasploit:

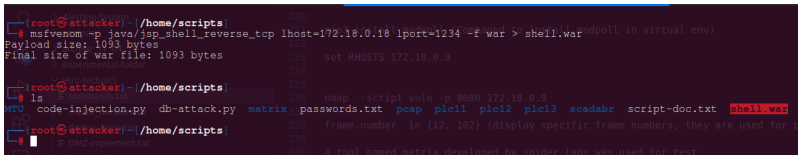

Figure 6.48: Reverse shell WAR file created for Tomcat exploitation

The attacker accessed the Tomcat web manager, authenticated using weak or default credentials, and uploaded the shell.war file. Once deployed, this payload allowed the attacker to receive a reverse shell using netcat, gaining full control of the underlying OS.

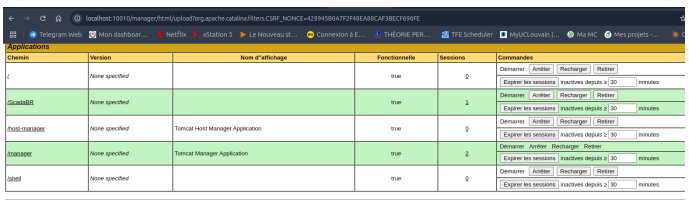

Figure 6.49: Shell application uploaded to Apache Tomcat

With this access, the attacker could browse directories, modify application files, or drop additional scripts for persistent access. This method bypassed all web-based authentication and went undetected by the system.

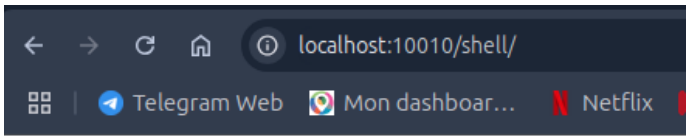

Figure 6.50: Shell application launched on browser

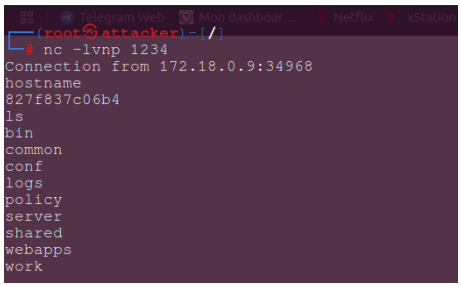

Figure 6.51: Interactive shell session established with ScadaBR server

6.7 Physical Process Attack Scenarios in ICS

While previous chapters demonstrated how vulnerabilities in ICS components can be exploited for unauthorized access, manipulation, and disruption, this chapter shifts the focus to a more impactful perspective: the **consequences of such attacks on the actual physical process** being monitored and controlled.

These scenarios simulate real-world damage, where the attacker's intent goes beyond reconnaissance or data theft and targets operational functionality. The goal here is to disrupt the engineering team's ability to monitor and react appropriately to changes in the system **either by falsifying data, blocking visibility, or completely corrupting the SCADA interface.**

We begin this section with a case where an attacker compromises the ScadaBR database to **silently disable process monitoring** for critical components, without triggering any alarms or fault indicators. This not only erodes operator situational awareness but also opens the door to more destructive follow-up attacks.

6.7.1 Tampering with ScadaBR's Data Source Integrity (Visibility Disruption Attack)

In the initial state of the simulation, all PLCs are operating normally. The tank represented by sensor LIT101 is filling with water, indicated by a value of 1000.0, while pumps P101 and P102 remain inactive (0). The flow indicator FIT201 reads 0.0, and valve MV201 from PLC12 is open (1). This stable view represents the operator's understanding of system conditions through ScadaBR.

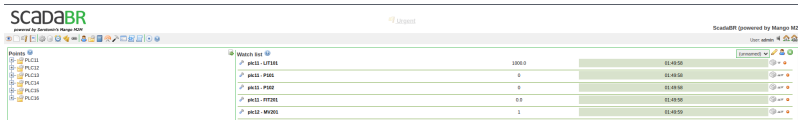

Figure 6.52: Normal simulation with accurate process monitoring

In this attack, the goal is to **break that visibility** by deleting the database entries that define how ScadaBR retrieves and presents values from PLC11 and PLC12. Rather than injecting false data, the attacker aims to **erase the monitoring links altogether,** leaving the operator blind to any change in the physical process. To do this, the attacker accessed the **ScadaBR MySQL database** i.e the Historian using previously gathered credentials (scadabr:scadabr) and retrieved records from key tables:

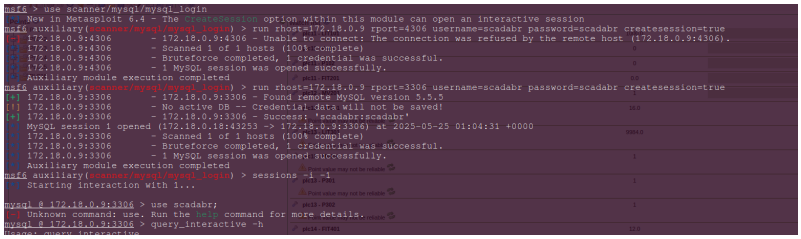

Figure 6.53: Historian accessed successfully

- **dataSources:** which contains Modbus IP definitions for each PLC.
- **dataPoints:** which maps those sources to individual sensors and actuators.
- **watchListPoints:** which links those points to the user-facing Watch List dashboard.

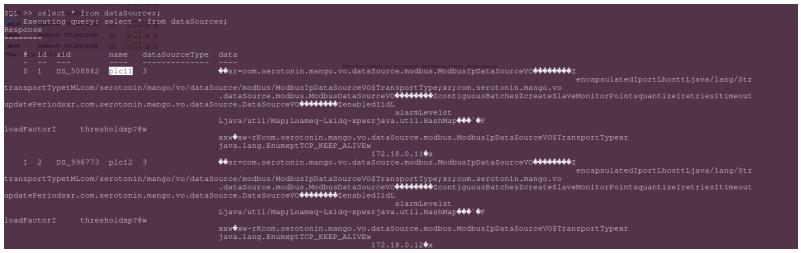

Figure 6.54: Extracted configuration data from dataSources

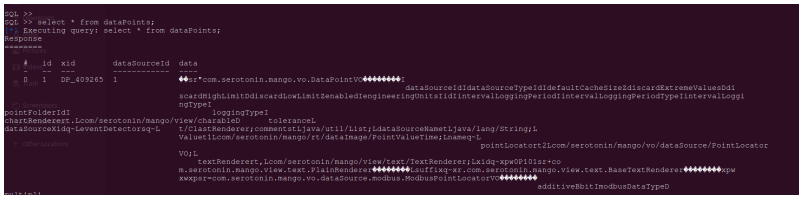

Figure 6.55: Extracted configuration data from dataPoints

The attacker discovered that each **watchListPoint** entry includes a reference to a **dataPointId**, which in turn is tied to a **dataSourceId** from the dataSources table. For instance, the Watch List referencing PLC11's tank and pump values relies on these chain-linked entries to remain active.

By starting with the watchListPoints table, the attacker planned to remove all visibility connections to PLC11 and PLC12. This action alone would **wipe all related signals from the operator dashboard,** without removing the actual sensor inputs or affecting Modbus communication at the protocol level making this a subtle but effective attack.

In the next step, the attacker will issue DELETE operations against this table to sever user visibility, followed by deeper tampering of dataPoints and dataSources for persistence.

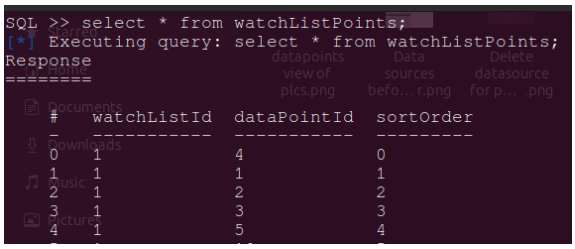

Figure 6.56: Extracted configuration data from watchListPoints

After deleting the entries in the watchListPoints table, the attacker took the next step in fully compromising operator visibility by targeting the dataPoints table in ScadaBR's database.

What are Data Points?

In ScadaBR, a data point is a virtual link between a SCADA component (e.g. a tag like **LIT101** or **P101**) and its Modbus address on the physical PLC. Each point defines:

- The **Modbus register or coil** it maps to (e.g., Holding Register 0/1 or Coil 0)
- Its **data type** (e.g. Numeric or Binary)
- Its **position in the interface** (Offset, Slave ID, Status)

These data points are the building blocks that make real-time monitoring possible. Without them, ScadaBR has no way of retrieving values from the PLC making the point "invisible" to the interface.

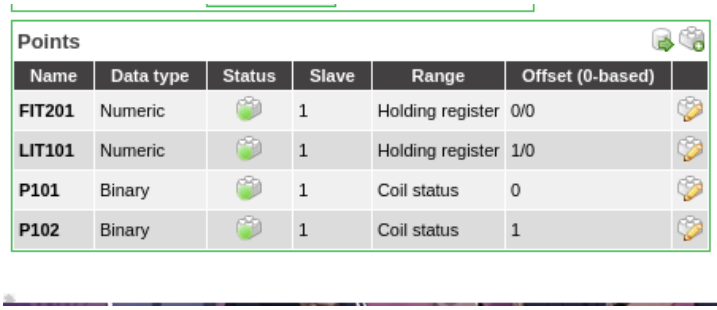

Figure 6.57: Overview of PLC11 data point bindings in SCADA system

Figure 6.58: Overview of PLC12 data point bindings in SCADA system

With the watchlist entries removed, the attacker executed a second SQL command to **completely erase the data points** from the backend:

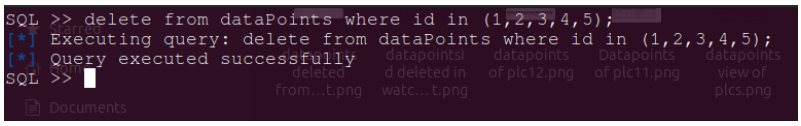

Figure 6.59: Destruction of datapoint records from ScadaBR backend

The command was executed successfully, and the change was immediate. On the ScadaBR interface, all previously visible process variables water levels, pump states, flow indicators were gone. The operator no longer had access to live monitoring for PLC11 or PLC12.

Figure 6.60: All datapoints deleted with success from scadaBR

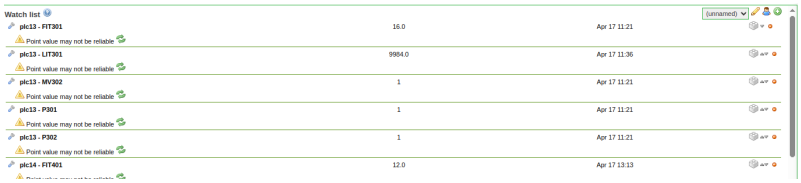

Figure 6.61: Operator interface no longer shows critical process data

This attack didn't require logic modification, malware, or command injection. It simply broke the internal structure that ScadaBR depends on to visualize the process. And because the underlying Modbus devices are still running and responding, **no alarms or Modbus faults are triggered.** The result is a system that looks "normal" from the network layer but is fully blind from the operator's view.

After removing all relevant entries from the watchListPoints and dataPoints tables, the attacker carried out the final stage of the disruption: deleting the dataSources themselves. These records define the **Modbus connection parameters** for each PLC, such as IP address, port, and communication settings. Without them, ScadaBR no longer knows where to poll for data even if the PLCs remain operational.

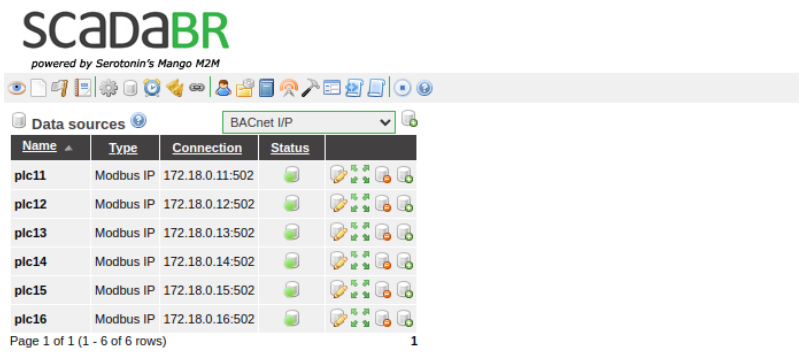

Figure 6.62: Initial configuration of PLC Modbus data sources in ScadaBR

The attacker executed the following SQL command to delete the data source configurations for PLC11 and PLC12:

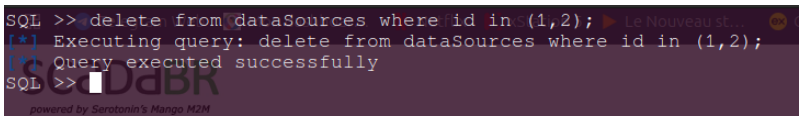

Figure 6.63: Modbus data source entries removed from ScadaBR database

Back in the ScadaBR interface, the change was immediate. Only PLCs 13 through 16 remained visible in the system. PLC11 and PLC12 were completely removed from the configuration as if they never existed.

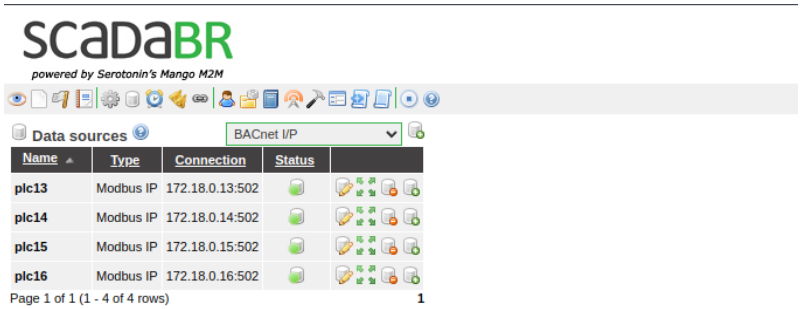

Figure 6.64: ScadaBR no longer shows PLC11 and PLC12 after tampering

To confirm that this change had real communication-level consequences, the attacker used tcpdump to inspect Modbus traffic targeting ScadaBR (172.18.0.9). The result showed that **no Modbus packets were being received,** confirming that the polling for PLC11 and PLC12 had ceased entirely.

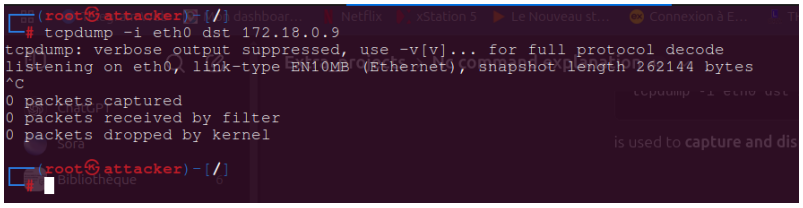

Figure 6.65: Verification that Modbus communication has been broken

Impact and Reflection

This attack shows that **a database-level compromise alone is enough to sabotage real-time visibility** in ICS environments. The PLCs themselves were not modified. The Modbus protocol was untouched. Yet, because the attacker surgically deleted the database records that define how values are visualized and polled, **the SCADA operator became blind to an active system.**

In a real-world setting, this could allow an attacker to:

- Launch unsafe physical operations undetected.
- Prepare for logic manipulation attacks without notice.
- Falsify system availability and integrity from the inside out.

This scenario highlights the need for:

- Backup validation of SCADA configurations.
- Logging and alerting on backend database changes.
- Segmentation and access control for SCADA storage systems.

6.7.2 Disrupting the Physical Process through Code Injection in PLC

In a final critical scenario, the attacker targets **PLC11** with a logic injection attack. Since the principles behind code injection were already detailed in a previous chapter, we now focus directly on how such an action can alter the behavior of the physical process itself.

Initial Simulation State

Before any attack was launched, the SCADA interface displayed a well-synchronized system. The **water level sensor LIT101** indicated a value of 3020.0, reflecting a near-full tank. The **flow indicator FIT201** was functioning properly, reporting an active value of 20.0. Both pumps, P101 and P102, were in the ON state (represented by 1), contributing to the flow cycle. Finally, the valve MV201 was shown as open (1), completing the water circulation process at this stage.

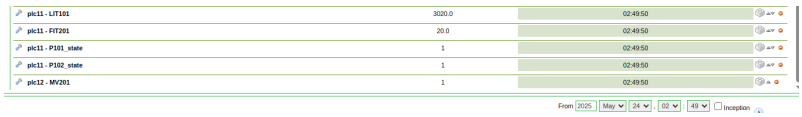

Figure 6.66: Normal operating conditions shown before launching code injection.

From the operator's perspective, the system was in a stable and active operational phase. All control points and sensor feedback were consistent with expected fluid dynamics in the tank-pump-valve loop. This made the target ideal for sabotage: the process was fully active, meaning any injected logic could instantly produce impactful results. With the attack path already tested and prepared, the attacker initiated the exploit targeting PLC11.

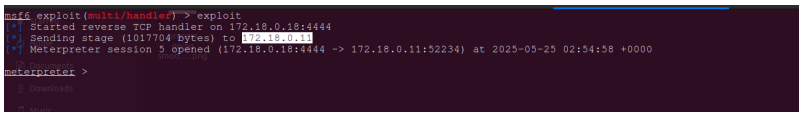

Figure 6.67: Remote shell successfully opened to PLC11

Once the attacker successfully opened a reverse shell to PLC11, they quickly navigated the file system to deploy the preconfigured injection script. Without disrupting visibility to the SCADA interface, the attacker silently executed **code-injection.py**, targeting port 502 the default Modbus communication channel.

The consequences were immediate. Upon checking the monitoring interface, the values reported by the pumps, valves, and sensors were radically changed:

- Setpoints were altered.
- Logic blocks were overwritten to produce incorrect control sequences.
- Variables like temperature, voltage, and flow began displaying impossible values.

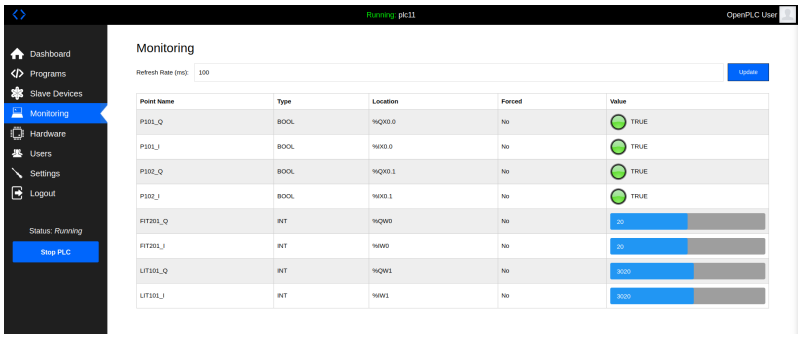

Figure 6.68: The original stable state of PLC11 before logic manipulation

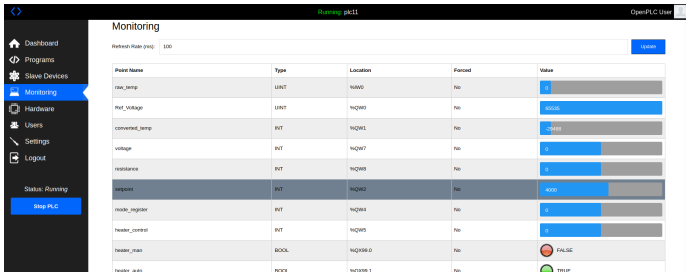

Figure 6.69: Control logic corrupted—visible anomalies in monitored data.

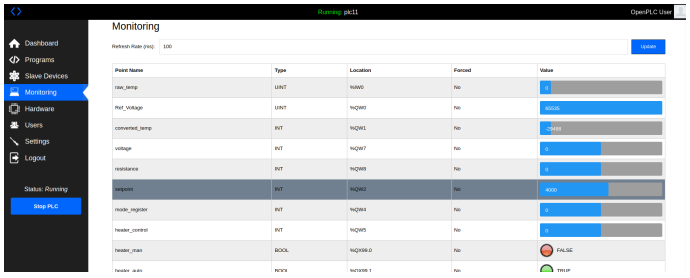

Figure 6.70: Control logic corrupted—visible anomalies in scadaBR interface

This injection did not simply distort displayed values **it altered how the PLC processed its environment,** triggering inappropriate actuator responses. For instance, pumps were activated despite already full tanks, and flow control values were overridden with erratic figures. Ultimately, the PLC was no longer performing its intended automation role it was **driven by injected parameters,** making decisions based on entirely **forged sensor states.** In a real-world scenario, this could lead to severe consequences ranging from system failure to physical hazards.

Chapter 7
Mitigation Strategies Based on Standards

While the previous chapters demonstrated how easily an unprotected SCADA system can be compromised, this chapter shifts focus toward defense. The goal is to show that even in our virtual testbed, it is not only possible but practical to implement **realistic, standards-based mitigation strategies.** By applying principles from frameworks like the Purdue Model and IEC 62443 [23], this testbed was hardened using segmentation, firewalls, encrypted communication, and real-time monitoring. These solutions were adapted to fit the virtual Docker-based environment and serve as a blueprint for researchers or engineers seeking to evaluate or teach ICS security in a hands-on setting.

Testbed Usability and Customization

While this chapter focuses on implementing selected mitigation strategies based on industry standards, the testbed itself was designed to be highly modular and adaptable. Users are not limited to the defense configurations demonstrated here. All components such as the network layout, firewall rules, monitoring tools, and PLC logic can be modified or replaced to simulate alternative protection techniques or system architectures. For example, if network segmentation is not required, users can simply adjust or remove the router container and associated firewall rules directly through the Docker Compose and configuration files.

To support such flexibility, the GitHub repository provides full access to the testbed's mitigation setups, including files for firewall rules, detection logic, and visual monitoring dashboards. Although a comprehensive user manual is not embedded within this manuscript, the repository is actively being updated with documentation, usage instructions, and examples to guide users in adapting the testbed for their own research, training, or experimental purposes.You can access the mitigation configurations at: `https://github.com/Max-planck-Jr/ICS_Water_testbed_cyber/tree/main/Mitigations_scenarios`

7.1 Purdue Model-Based Network Segmentation

To enforce strong network isolation and reduce the attack surface in the SCADA testbed, a segmentation strategy based on the **Purdue Model** was implemented. This strategy separates system components by function and criticality into distinct **zones and conduits**, minimizing exposure and preventing unauthorized lateral movement [1]. The architecture was divided into four core zones:

- **Device Zone:** Contains PLCs, RTUs, and the physical simulator.
- **Supervisory Zone:** Hosts SCADA systems such as ScadaBR and OpenPLC operator interfaces.

- **Demilitarized Zone (DMZ):** Acts as a buffer between OT and IT environments.
- **IT Department Zone:** Represents external or administrative access points (e.g. engineering workstations or update servers).

Each zone was implemented as an isolated **Docker subnet**, with a dedicated **router container** positioned between them to manage communication. Traffic between zones was strictly regulated using **Linux iptables firewall rules,** allowing only predefined protocols and IP ranges to pass. For example, only the Supervisory Zone was allowed to query the Device Zone via Modbus/TCP, while the IT Department could only access the DMZ under controlled conditions.

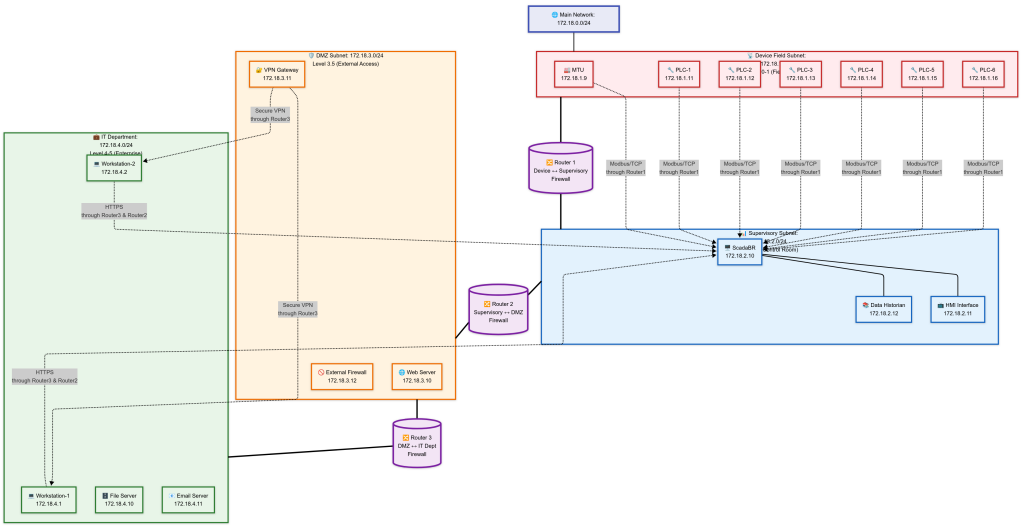

Figure 7.1: Zone-based segmentation of SCADA testbed following Purdue model

The firewall container was configured to act both as a router and as a filter. **IP forwarding** was enabled, but granular **iptables** rules blocked unsolicited requests, restricted port access, and enforced directionality between zones. These rules ensured that:

- Devices in the Field Zone could not be reached directly from the IT Zone.
- SCADA services in the Supervisory Zone could only talk to PLCs through allowed conduits.
- The DMZ served as the only bridge to external access, hardened with limited privileges.

This implementation aligns with IEC 62443's defense-in-depth principle, creating multiple layers of security. It prevents attackers from exploiting one compromised service to reach more sensitive systems deeper in the architecture, forming the backbone of all subsequent protections.

All configuration files used to implement these zones including Docker network setups, IP routing rules, and the **iptables** firewall scripts applied on the router container are included in the GitHub repository. This provides readers with a reproducible reference for applying similar segmentation strategies in their own SCADA testbeds or lab environments.

7.2 Real-Time Monitoring with Grafana, Loki, and Promtail

To reinforce the security posture of the SCADA testbed, a centralized monitoring system was implemented using **Grafana, Loki, and Promtail.** A custom logging plugin named **loki-driver** was added to the Docker configuration **(via daemon.json)** to collect logs from all containers across the segmented network. Grafana's Explore panel allows log filtering per container. Engineers can inspect logs from key assets such as the MTU, HMI-HIS, PLC11, PLC12, or even the attacker if compromised.

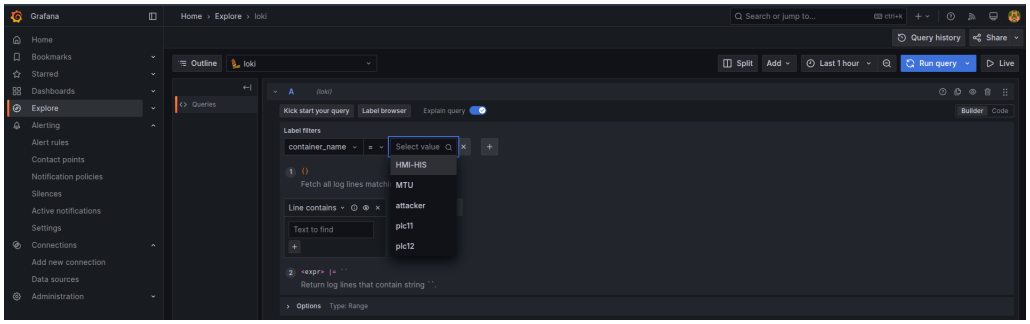

Figure 7.2: Selecting logs based on container names in Grafana Explore

We also see that the Loki data source is correctly connected and used as the default for log ingestion.

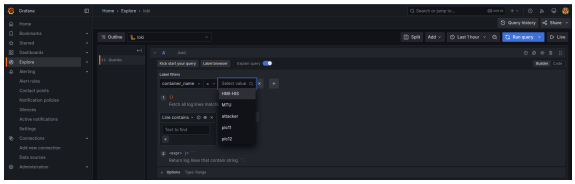

Figure 7.3: Loki data source connected to Grafana

Live logs from physical simulation activity including fluid movement, actuator changes, and sensor reads can also be visualized. This provides valuable visibility into Modbus-based behavior for any device.

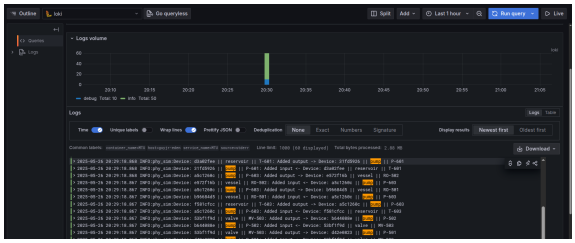

Figure 7.4: Real-time logs from the MTU showing sensor and actuator activities

Additionally, logs from suspicious sources like the attacker container are also captured, enabling retrospective analysis in case of a breach.

Figure 7.5: Logs from attacker container showing shell access attempt

This solution also demonstrates alerting rules defined in Grafana to automatically detect anomalies such as repeated coil state changes or reservoir overflow risk. These alerts serve as an early warning mechanism.

Figure 7.6: Alert rules configured in Grafana based on log patterns

Beyond just visualizing and querying logs, the monitoring system was extended to include **automated alert notifications.** Grafana's Alerting feature was configured to trigger alarms when abnormal patterns were detected for example, valves being toggled excessively or suspicious Modbus coil activity. As shown below, a webhook contact point was created in Grafana to integrate with Discord, where all alerts are immediately pushed to a designated server channel:

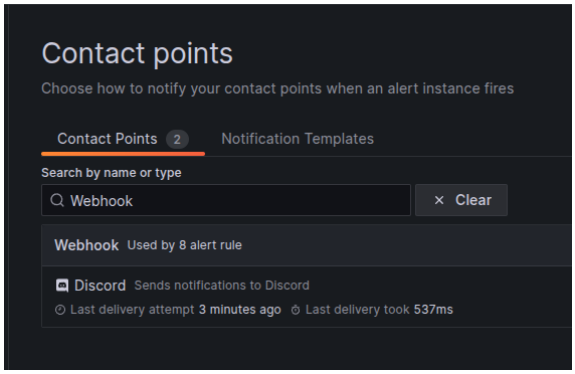

Figure 7.7: Grafana webhook contact point configuration for Discord notifications.

When rules fire (such as when a valve is repeatedly disabled or Modbus data is missing), they are forwarded to Discord in real time:

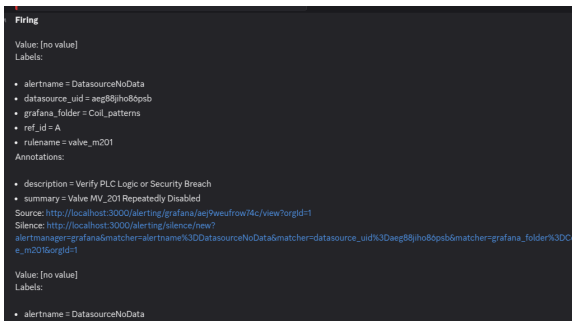

Figure 7.8: Example of a triggered Grafana alert showing detailed event context.

These alerts provide critical labels such as the container affected, the alert name, and contextual details describing the incident. This enables rapid assessment by operators or incident response teams.

Note: All configuration files for webhook setup, Loki logging, Promtail pipelines, and alert rules will be made available in a dedicated GitHub repository for reproducibility and transparency.

7.3 Machine Learning-based Anomaly Detection in SCADA

To further enhance the visibility and responsiveness to malicious behaviors within the SCADA testbed, a machine learning (ML) model was trained using logs collected from the MTU. The model is deployed in a dedicated container named **ml-ids**, which monitors MTU logs in real-time. Grafana is configured to query logs specifically from the ml-ids container.

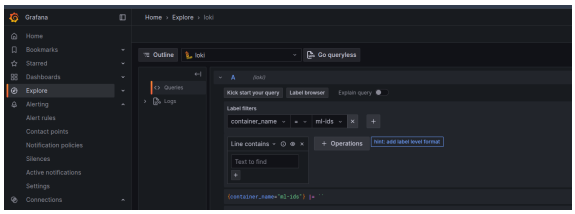

Figure 7.9: Grafana querying logs from the machine learning detection container.

Once deployed, the model flags any deviation from expected behavior as an **Anomaly**. These detections, as illustrated in the figure below, appear in Grafana's log view, allowing engineers to visualize exactly when and what triggered the alert such as a suspicious register value or control output. Further details about each anomaly such as timestamps, affected sensors, and locations can be seen in the logs themselves, offering immediate insight into potential intrusions or failures.

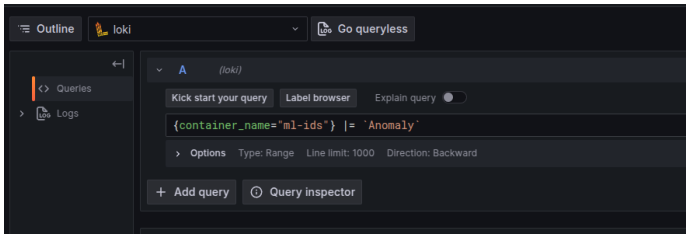

Figure 7.10: Grafana Query to display only malicious activities detected by ML

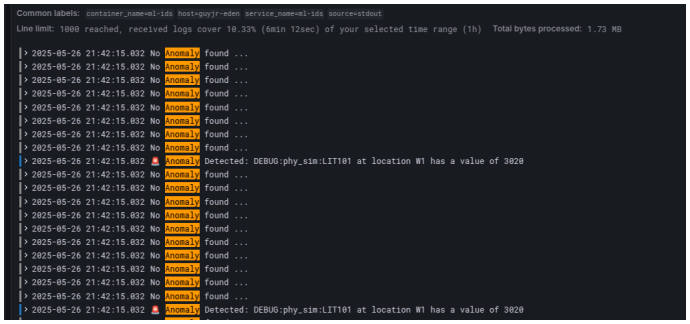

Figure 7.11: Annotated log entries showing anomalies and their exact locations.

To make detection actionable, Grafana was configured to generate alerts when an anomaly is detected. These alerts are routed to a Discord channel using webhook integration. This allows security teams to be notified in real-time, even outside the monitoring dashboard. All configuration details ranging from model training, container setup, to alert rules and webhook connections are provided in the GitHub repository to ensure full reproducibility and flexibility for similar testbeds.

7.4 Complementary Mitigation Strategies and Final Recommendations

While this work focused on showcasing real-world attacks against SCADA environments, it also demonstrated viable mitigation strategies. Among them, network segmentation using the Purdue Model, firewall enforcement via Docker routers and iptables, encrypted communication with WireGuard, NGINX, and Stunnel, as well as centralized monitoring through Grafana, Loki, and Promtail were successfully deployed. Additionally, an ML-based anomaly detection container was integrated for real-time alerting.

Other best practices including tunneling Modbus over secure channels, restricting write access, deploying IDS solutions, and implementing traffic rate limiting can further harden industrial control systems. All configuration files and commands are documented and available in the GitHub repository for reproducibility and adaptation in similar testbeds.

Chapter 8
Discussion

8.1 Comparison with Related Works

Our SCADA security testbed was designed to simulate realistic attack scenarios while offering layered mitigation strategies and live monitoring bridging academic experimentation and real-world application. Compared to many existing works that focus narrowly on isolated threats or single-component evaluations, this implementation offers a broader, system-wide perspective. Where some platforms simulate Modbus behavior or test attack tools in simplified environments, our testbed integrates real services like OpenPLC, ScadaBR, and a custom Python-based MTU, operating within a fully segmented Dockerized network reflecting the Purdue model.

What sets our work apart is the end-to-end pipeline it provides from reconnaissance and exploitation to detection and mitigation combined with the ability to visualize attacks in real time and react to them through a centralized monitoring and alerting system. While similar research often stops at protocol-level manipulations, we extended ours to application logic, database tampering, and physical process distortion. In doing so, we not only replicated known vulnerabilities but also built a space where researchers and engineers can implement both offensive and defensive strategies within a reproducible environment.

8.2 Modular Design and Research Potential

One of the key strengths of this testbed lies in its high degree of modularity and customization. While this thesis presents specific attack scenarios, monitoring strategies, and mitigation techniques, these implementations are only suggestions to demonstrate what is possible. Users are encouraged to modify or extend the testbed according to their own research objectives or industrial requirements. The architecture is designed with Docker containers, YAML-based configurations, and Python scripts making it easy to replace, extend, or rewire any component, whether it be the physical process simulator, PLC logic, or SCADA interface.
For instance, users can:

- **Define new physical processes by editing or replacing the simulation script and associated configuration files.**

- **Implement different attack paths or integrate custom detection tools.**

The testbed does not impose a fixed structure; instead, it provides a foundation upon which researchers, educators, and practitioners can build their own scenarios. All the configuration files, container setup scripts, and deployment guides are made publicly available in the project repository, reinforcing its reproducibility and extensibility.

8.3 Lessons Learned

Building and attacking this system revealed several core lessons. First, default configurations in ICS components are dangerously permissive. With minimal effort, we were able to inject code into PLCs, extract credentials, and manipulate SCADA databases all without triggering any alarms in the system's native interfaces. This underlined the importance of defense-in-depth: segmentation alone is not enough, and neither is logging if no one is watching the logs.

The second insight came from building out our observability pipeline. Integrating Loki, Promtail, and Grafana not only made activity transparent, but allowed us to correlate attacker movement with control system behavior. When combined with alerting and Discord integration, even subtle anomalies became visible and actionable. We also found that adapting DevOps tools like these to ICS monitoring is both practical and powerful when paired with well-structured log streams.

Training and deploying a machine learning model based on simulator logs further emphasized the value of behavioral baselining. The model proved effective in catching deviations that rule-based alerts might miss, showing that hybrid detection, combining **logs, human-readable events, and AI** offers a stronger line of defense than any individual measure.

8.4 Limitations of the Testbed

Despite its flexibility and depth, the testbed does have limits. It remains a virtual environment; Docker-based PLCs, SCADA services, and routers don't fully replicate the hardware behavior or timing nuances of real industrial systems. Physical-layer anomalies such as signal noise or device failure are not reflected, nor are operator reactions or decision-making under attack conditions.

Our ML detection system, while promising, was trained entirely on simulator logs. Deploying it in a real-world ICS would require retraining on production data, ideally incorporating device-specific behaviors. The use of WireGuard, Stunnel, and NGINX for encryption and isolation was successfully demonstrated, but not every component was fully integrated into a unified VPN or PKI trust framework leaving room for architectural improvements.

Lastly, while the testbed supports complex attack chains, it was not built to simulate long-term advanced persistent threats (APTs). Most payloads were manually injected into targets using Docker-level access, typically through commands like **docker exec cp <host_path> <container_name>:/<target_path>.** This method provided direct insertion of scripts and binaries for controlled testing. In a real-world APT scenario, such payloads would be delivered through phishing, supply chain compromise, or lateral movement techniques that involve multi-layered evasion and persistence. However, the purpose of this testbed was not to replicate network-level APT behavior but to **demonstrate how ICS-specific attacks could be launched, observed, and mitigated** in a realistic yet controlled environment. Future iterations may expand on this by incorporating stealthy delivery mechanisms and multi-phase attack campaigns for more advanced simulation[6].

Chapter 9
Futur Work

Despite the depth and flexibility of the testbed developed in this thesis, several improvements and expansions can be made to extend its realism, scalability, and research potential.

One major area for future work is the **integration of more realistic payload delivery mechanisms,** moving beyond direct Docker-level injection. This would include simulating phishing-based attacks, USB drop scenarios, or remote exploitation through vulnerable web interfaces bringing the attack process closer to the behavior of actual advanced persistent threats (APTs). A full network simulation including email clients, external access points, and privilege escalation paths could support this direction.

The monitoring stack could also be extended by integrating tools like Elastic Stack (ELK) or Wazuh, or by enhancing anomaly detection models using deep learning architectures such as LSTM for temporal prediction. A hybrid IDS combining rule-based and ML-based detection, possibly trained with both normal and attack traffic in a labeled dataset, would improve detection robustness.

In terms of simulation complexity, the MTU could be extended to support **multi-process interaction,** such as gas and temperature dynamics, cascading failures, or interlocked safety logic. This would allow researchers to simulate more dangerous or complex failure scenarios and observe how attacks impact chained physical outcomes.

Finally, future work could focus on scalability and deployment automation. Using orchestration tools like Docker Compose or Kubernetes, the testbed could be deployed in the cloud or replicated easily in academic labs. A GUI-based configuration interface could also lower the entry barrier for students and early-stage researchers.

Chapter 10
Conclusions

10.1 Conclusions

This thesis set out to explore the vulnerabilities and defense strategies of ICS by developing a realistic, modular, and extensible SCADA testbed. Through this environment, we were able to simulate full-spectrum attacks from reconnaissance to logic injection targeting Modbus-based communication, SCADA interfaces, and physical process emulation. Each attack was executed in a layered system modeled after the Purdue architecture, providing a real-world context for observing both the weaknesses of industrial protocols and the consequences of insecure configurations.

Beyond merely demonstrating these vulnerabilities, this work also implemented and validated several mitigation strategies aligned with industrial standards such as IEC 62443. These included network segmentation, encrypted communication tunnels, protocol hardening, and centralized log monitoring. We also showcased an ML-based anomaly detection system, integrated directly into a real-time alerting pipeline using Grafana and Discord. This combination of detection and response reinforces the testbed's ability not only to simulate attacks but also to support active defense research.

What makes this testbed particularly impactful is its adaptability. It supports not only security demonstrations but also educational exercises, monitoring experimentation, and potential integration with physical hardware. It is designed to be replicated, scaled, and extended whether for teaching, further research, or industrial proof-of-concept.

While certain limitations exist, including the use of virtual devices and Docker-level payload injection, the work presented here offers a meaningful and practical contribution to the field of ICS cybersecurity. It bridges theory and practice and lays a foundation for future research on complex intrusion scenarios, AI-driven defense, and hybrid cyber-physical attack modeling.

Ultimately, this thesis demonstrates that with the right design, even a virtual testbed can reflect the real challenges faced by critical infrastructure and provide a safe, flexible platform for exploring how those challenges can be met.

10.2 Source code and Documentation

The source code is available at: `https://github.com/Max-planck-Jr/ICS_Water_testbed_cyber.git`. For Matrix tools, the source code is available at: `https://github.com/SpiderLabs/MATRIX.git`. Examples of attacks could also be found at: `https://github.com/montimage-projects/OT-ICS-attacks.git`. Important documentaions on metasploit framework could be found at: `https://github.com/rapid7/metasploit-framework.git`.

Bibliography

[1] 8balla. Ics network segmentation, 2021. [Cited on page 62.]

[2] S. Dyer T. Patel H. Janicke A. Nicholson, S. Webber. Scada security in the light of cyber-warfare. *ScienceDirect*, 2012. [Cited on pages 1 and 16.]

[3] Mohammed Alanazi, Abdun Mahmood, and Mohammad J. M. Chowdhury. Scada vulnerabilities and attacks: A review. *Computers & Security*, 125:103028, 2023. [Cited on page 26.]

[4] Chowdhury M. J. M. Alanazi M., Mahmood A. SCADA vulnerabilities and attacks. *Computers Security*, 125(6):12–25, 2023. [Cited on pages 1, 2, and 3.]

[5] Claire Allen-Addy. Threat modeling methodology: Stride. *IriusRisk*, September 2023. [Cited on page 19.]

[6] Shankar Sastry Alvaro A. Cardenas, Saurabh Amin. Research challenges for the security of control systems. *ACM Digital Library*, 2008. [Cited on page 69.]

[7] Thiago Alves and Thomas Morris. Openplc: An iec 61131-3 compliant open source industrial controller for cyber security research. *Computers & Security*, 78:364–379, 2018. [Cited on pages 1 and 2.]

[8] Nicole Anne. Packet modification attack on plc with arp spoofing (mitm attack). *Medium*, 2020. [Cited on page 39.]

[9] Deborah Tolk Tobias Brandt Reef Eilers Rebeca Ramirez Annika Ofenloch, Jan Sören Schwarz. Mosaik 3.0: Combining time-stepped and discrete event simulation. *Politecnico di torino*, 2022. [Cited on page 4.]

[10] Matthew P. Barrett. Framework for improving critical infrastructure cybersecurity, 2018. [Cited on pages 16 and 18.]

[11] Olivier TABURIAUX Bertrand MASSET. Simulating industrial control systems using mininet. Master's thesis, École Polytechnique de Louvain, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 2018. Accessed: 2025-05-04. [Cited on page 4.]

[12] Karl Biron. Lights out and stalled factories: Using m.a.t.r.i.x to learn about modbus vulnerabilities, 2025. [Cited on pages 44 and 46.]

[13] Trend Business. One flaw too many: Vulnerabilities in scada systems, 2019. [Cited on page 26.]

[14] Chen-Ching Liu Chee-Wooi Ten, Govindarasu Manimaran. Cybersecurity for critical infrastructures: Attack and defense modeling. *Science Direct*, 2010. [Cited on pages 15 and 18.]

[15] Govindarasu Manimaran Chee-Wooi Ten, Chen-Ching Liu. Vulnerability assessment of cybersecurityfor scada systems. *Research Gate*, 4:2–3, 2008. [Cited on page 1.]

[16] Claroty Team. Ics security: The purdue model, March 2023. Accessed: 2025-05-20. [Cited on page 14.]

[17] Yvonne Kavanagh Conrad Eriksa, Diarmuid O Briain. Vicsort - a virtualised ics open-source research testbed, 2022. [Cited on page 4.]

[18] Donadel D. Turrin F. Conti, M. A survey on industrial control system testbeds and datasets for security research. *IEEE Communications Surveys Tutorials*, (6):1121–1134, 2021. [Cited on page 1.]

[19] Mauro Conti, Denis Donadel, and Federico Turrin. A survey on industrial control system testbeds and datasets for security research. *IEEE Communications Surveys & Tutorials*, 23(4):2248–2294, 2021. [Cited on page 2.]

[20] Srinivas Sampalli Darshana Upadhyay. Scada (supervisory control and data acquisition) systems:vulnerability assessment and security recommendations. *Research Gate*, 19:5–13, 2020. [Cited on page 18.]

[21] Amir Dehlaghi-Ghadim and et al. Icssim — a framework for building industrial control systems security testbeds. *Computers in Industry*, 148:103906, 2023. [Cited on pages 2 and 4.]

[22] Deric. |scada| openplc + raspberry pi + scadabr: Part 1, 2020. [Cited on page 33.]

[23] Amandeep Dhillon. Strengthening ics resilience with isa/iec 62443 standards and configuration management. *Industrial Cyber*, pages 123–130, 2024. [Cited on page 62.]

[24] Dragos, Inc. Ics/ot cybersecurity year in review. `https://www.dragos.com/year-in-review/`, 2023. [Cited on pages 2, 3, and 16.]

[25] Dragos, Inc. Key concepts of isa/iec 62443: Zones & security levels, 2023. Accessed: 2025-05-20. [Cited on page 17.]

[26] Mert Melih ÖZÇELİK Erdal IRMAK, Ismail ERKEK. Experimental analysis of the internal attacks on scada systems. *Journal of Science*, 15, 2017. [Cited on page 36.]

[27] Justin Lowe Eric Byres. The myths and facts behind cyber security risks for industrial control systems. *Research Gate*, 7, 2004. [Cited on page 23.]

[28] Susanne Kießling Felix Sauer, Matthias Niedermaier. Licster – a low-cost ics security testbed for education and research. *Research Gate*, 2019. [Cited on page 4.]

[29] Daniel Formby, Mansour Rad, and Raheem Beyah. Grfics: Lowering the barriers to ics security research. In *Proceedings of the USENIX Annual Security Symposium (ASE) Workshop*. USENIX Association, 2018. [Cited on pages 2 and 4.]

[30] Fortinet. Iec 62443 standard: Enhancing cybersecurity for industrial automation and control systems, 2023. [Cited on page 17.]

[31] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. A dataset to support research in the design of secure water treatment systems. In *Critical Information Infrastructures Security (CRITIS 2016)*, volume 10242 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 2017. [Cited on pages 2 and 3.]

[32] Andy Greenberg. A Hacker Tried to Poison a Florida City's Water Supply, Officials Say, 2021. [Cited on page 1.]

[33] Andy Greenberg. The colonial pipeline hack is a new extreme for ransomware. *Wired*, 2021. [Cited on pages 1 and 16.]

[34] Andy Greenberg. Cyberav3ngers, iran, and the growing threat to industrial systems. *Wired*, 2024. [Cited on page 16.]

[35] Arne Vidstrom Erik Westring Hannes Holm, Martin Karresand. Virtual industrial control system testbed. *Research Gate*, 84, 2015. [Cited on page 4.]

[36] Industrial Defender. Industrial defender's ot security & compliance platform, 2023. [Cited on page 15.]

[37] IriusRisk. Threat Modeling Methodology: STRIDE, 2025. [Cited on page 18.]

[38] IsaSecure. Japan's control system security center joins isa security compliance institute, 2022. [Cited on page 3.]

[39] ISPL. Scada attacks scenarios, 2016. [Cited on page 36.]

[40] Karen A. Scarfone Keith A. Stouffer, Joseph A. Falco. Guide to industrial control systems (ics) security. Technical report, National Institute of Standards and Technology (NIST), May 2015. [Cited on pages 3, 15, 17, and 18.]

[41] F. Flammini S. Guarino R. Setola L. Faramondi. A hardware-in-the-loop water distribution testbed dataset for cyber-physical security testing. *Research Gate*, 2017. [Cited on page 4.]

[42] Ravie Lakshmanan. Industrial control systems vulnerabilities soar: Over one-third unpatched in 2023. *The Hacker News*, August 2023. [Cited on page 26.]

[43] Biero llagas. Setup and exploit a ot lab: Modbus part 2 black blox , vlan and stuff. *Medium*, 2023. [Cited on page 39.]

[44] Rahix Michael Büsch. Awlsim open-source python siemens s7 cpu emulator, 2017. [Cited on page 4.]

[45] MITRE. National vulnerability database, 2019. [Cited on page 31.]

[46] montimage projects. Ot/ics attacks- simulation of an operational disruption in an ot/ics network., 2025. [Cited on page 31.]

[47] Alexandre Mundo. Triton malware spearheads latest attacks on industrial systems, 2020. [Cited on pages 3 and 16.]

[48] Palo Alto Networks. What Is the Purdue Model for ICS Security?, 2025. [Cited on page 14.]

[49] US Department of Energy. National scada test bed, 2023. [Cited on page 3.]

[50] OpenCVE. Vulnerabilities (scadabr), 2019. [Cited on page 31.]

[51] Sajjan Shiva Parves Kamal, Abdullah Abuhussein. Identifying and scoring vulnerability in scada environments. *Research Gate*, 14:3–11, 2017. [Cited on page 18.]

[52] Cássio Queiroz, Akhtar Mahmood, and Zahir Tari. Scadasim – a framework for building scada simulations. *IEEE Transactions on Smart Grid*, 2(4):589–597, 2011. [Cited on page 2.]

[53] Tim Conway Robert M. Lee, Michael J. Assante. A stealthy false command injection attack on modbus based scada systems. *SANS*, 10, 2016. [Cited on page 51.]

[54] Brett Sayles. Five steps to build an ics cybersecurity program with iec 62443 standards, 2022. [Cited on page 17.]

[55] Picus Security. What is an attack simulation?, 2025. [Cited on page 36.]

[56] Swetha. Hacking emulated ics/scada devices. *Medium*, 2023. [Cited on page 37.]

[57] Paolo Trungadi. Exploiting virtual networks for cps security analysis – the smart home environment. *Politecnico di torino*, 2023. [Cited on page 4.]

[58] Verve Industrial Protection. The ultimate guide to protecting ot systems with iec 62443. *Rockwell Automation*, February 2024. Accessed: 2025-05-20. [Cited on page 17.]

[59] David M. Nicol Vignesh Babu. Emulation/simulation of plc networks with the s3f network simulator. *Illinois Experts*, 2016. [Cited on page 4.]

[60] Diego Urrego Peter Langendörfer Wael Alsabbagh, Samuel Amogbonjaye. A stealthy false command injection attack on modbus based scada systems. *Research Gate*, 10, 2023. [Cited on pages 27, 29, and 34.]

[61] David Hutchison Jules Ferdinand Pagna Disso Kevin Jones William Knowles, Daniel Prince. A survey of cyber security management in industrial control systems. *Science Direct*, 2015. [Cited on page 15.]

[62] Kim Zetter. *Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon*. Crown Publishing Group, New York, 2014. [Cited on pages 1, 3, and 16.]

[63] Yi Zhu. A sandbox for industrial control systems (security). Master's thesis, École Polytechnique de Louvain, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 2024. Accessed: 2025-05-20. [Cited on pages 1, 5, and 7.]

[64] Can Özkan and Dave Singelée. Evidence-based threat modeling for ics. *arXiv preprint arXiv:2411.19759*, November 2024. [Cited on page 18.]