Author: Marco Lucchese

Design, implementation and evaluation of a physics-aware honeynet for Industrial Control Systems

DOCTORAL THESIS

Università degli Studi di Verona
Dipartimento di Informatica

Design, implementation and evaluation of a physics-aware honeynet for Industrial Control Systems.

Marco Lucchese

Ph.D. Thesis - 36th cycle

Advisor: Prof. Massimo Merro

Università degli Studi di Verona
Dipartimento di Informatica
Strada le Grazie 15, 37134 Verona
Italy

I, Marco Lucchese, declare that this thesis titled, “Design, implementation and evaluation of a physics-aware honeynet for Industrial Control Systems” and the work presented in it are my own. I confirm that: (i) This work was done wholly or mainly while in candidature for a research degree at this University. (ii) Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated. (iii) Where I have consulted the published work of others, this is always clearly attributed. (iv) Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work. (v) I have acknowledged all main sources of help. (vi) Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Verona, 25th July 2023

ACKNOWLEDGMENTS

Completing this doctoral thesis, has been an immensely enriching journey, made possible by the support, guidance, and encouragement of many.

First and foremost, I extend my deepest gratitude to my family, for their unwavering love, understanding, and encouragement. Their belief in my abilities and their emotional support during the highs and lows of this journey have been my anchor.

I am deeply grateful to my advisor, Massimo Merro, whose expertise, patience, and insightful guidance have been the cornerstone of this research. His unwavering support and constructive criticism have been invaluable, shaping not only this thesis but also my growth as a researcher.

Finally, I wish to express my gratitude to all those who, in one way or another, contributed to this project and supported me through this academic endeavor. Your collective wisdom, encouragement, and faith in my work have been a constant source of strength and inspiration.

This thesis is not only a reflection of my efforts but also a testament to the collaborative spirit and collective support of all those mentioned above and many others who have been part of this journey.

ABSTRACT

Industrial Control Systems (ICS) are critical infrastructure components, and successful cyberattacks can have devastating consequences, causing economic losses, disrupting vital services, and even endangering public safety. Honeypots offer a promising approach to study attacker behavior and improve ICS security. However, existing honeypots often suffer from limited interaction capabilities, low configurability, poor scalability, and inadequate physics-awareness, limiting their effectiveness. This thesis addresses these limitations by proposing HoneyICS, a high-interaction and physics-aware honeynet architecture specifically designed for ICS. HoneyICS provides a realistic environment for attackers while capturing valuable data on their behavior and intentions. Furthermore, the dissertation investigates the impact of different HoneyICS configurations on attacker behavior and the types of information captured. Additionally, a dedicated attack tool capable of executing various attack scenarios, including Man-in-the-Middle (MITM), Denial-of-Service (DoS), and actuator manipulation via Modbus, has been developed. This tool facilitates the evaluation of HoneyICS's effectiveness against real-world attack techniques.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

In the dynamically evolving landscape of cybersecurity, the threats to Industrial Control Systems (ICS) have emerged as a growing critical concern. The thesis titled "Design, implementation and evaluation of a physics-aware honeynet for Industrial Control Systems" aims to propose a framework, called HoneyICS, for understanding and capturing the dynamics of these attacks. This exploration is not just a theoretical excursion but is firmly grounded in practical investigations. Leveraging a realistic honeypot infrastructure as a pivotal tool for study, we adopt a hands-on approach to understanding potential cyber-physical attacks on ICS. Additionally, we implement an automated tool for executing the attacks and a dedicated pipeline for their thorough analysis.

Recent trends and incidents demonstrate the increasing prevalence of ICS attacks. For instance, in the second half of 2020, 33.4% of ICS computers globally were attacked, with significant increases across multiple sectors, including building automation, oil & gas, and engineering [54]. These attacks encompass a variety of threats such as backdoors, spyware, and other types of Trojans, indicating the evolving complexity of cyber threats to ICS. The year 2021 saw a significant increase in groups targeting ICS, with high-profile incidents like the Colonial Pipeline and JBS ransomware attacks gaining global attention. Dragos OT Cybersecurity INC.'s [26] 2021 report highlighted the emergence of three new threat groups: Kostovite, Erythrite and Petrovite, each with distinct tactics and targets, demonstrating the diversification in cyber adversaries' approaches [33].

In the first half of 2023, malicious activities were detected on 34% of ICS computers, marking the highest level of global threats since 2019 [4]. The diversity of malware families used in these attacks increased, with denied Internet resources and malicious scripts being the most prevalent threats. Remarkably, even regions traditionally considered safer, such as Australia, New Zealand, the

United States, Canada, and parts of Europe, witnessed increases in attacked ICS computers during this period.

The thesis also aims to leverage the proposed infrastructure to capture and analyze these attacks, providing valuable insights into the techniques and objectives of cyber adversaries. Through this approach, the research will contribute to understanding the current threat landscape in ICS, offering a unique perspective on how these systems are targeted and compromised. This study is not only timely but also critical in addressing the ever-evolving threats to critical infrastructure and industries globally.

1.1 Motivation

The motivation behind this research arises from the escalating sophistication and frequency of cyber-attacks targeting ICS. While traditional honeypots offer valuable insights into general cyber threats, they often struggle to mimic real-world ICS environments, particularly in simulating process control and diverse network protocols. Additionally, inaccurate modeling of physical processes can tip off attackers. This gap hinders our ability to effectively study attacker behavior and develop robust defense mechanisms for these critical systems. This thesis has three main goals:

1. *Create a Robust Honeypot Framework:* Develop a resilient and effective honeypot framework capable of emulating industrial environments to attract and analyze cyber threats.
2. *Study the Data Collected and Attack Patterns Received:* Conduct an in-depth analysis of the data collected through the honeypot framework, focusing on understanding the patterns and tactics employed by attackers targeting ICS.
3. *Provide an Automated Tool for Attacks:* To validate the honeypot's ability to capture real-world attacks, we will develop an automated tool for simulating a controlled set of ICS cyber-attacks. By analyzing attacker interactions with these simulated attacks, we can identify areas for improvement and refine the honeypot's effectiveness in capturing and mimicking real-world threats.

1.2 Contribution

This thesis makes three contributions to the field of cybersecurity, particularly in the context of ICS. The key contributions are as follows:

Development of HoneyICS: The design and implementation of HoneyICS, a physics-aware honeynet. This system is capable of simulating real-world industrial environments with a high degree of realism, incorporating both cyber and physical process elements. This enhances the ability to attract and analyze cyber threats but also improves understanding of how these threats interact with physical industrial processes.

Insightful Data Analysis: The deployment of HoneyICS provided a rich dataset of cyber attack attempts on ICS. The comprehensive analysis of this data has led to a deeper understanding of the tactics, techniques, and procedures used by adversaries in targeting ICS.

Creation of a Specialized Attack Tool: A contribution of this thesis is the development of an attack tool designed to test the authenticity and resilience of HoneyICS. This tool not only serves as a benchmark for the honeynet's effectiveness but also demonstrates the potential real-world applicability of such tools in cybersecurity.

The thesis bridges the gap between theoretical cybersecurity concepts and practical applications in industrial settings. It offers a framework that can be adapted and extended by future researchers and practitioners.

1.3 Thesis Outline

The thesis begins by describing the motivation behind the research, its contributions, and an overview of the topics.

Transitioning into Chapter 2, the focus shifts to ICS. This chapter explores the Operational Technology involved, dissecting the Purdue Enterprise Reference Architecture to provide a structured and granular understanding. It also provides a summary of the notable security issues plaguing ICS, drawing a comparative analysis between Operational Technology (OT) and Information Technology (IT), highlighting their distinct characteristics and implications for security.

In Chapter 3, an overview of the literature on existing honeypots is provided, honeypots specifically designed for Industrial Control Systems. The chapter provides a comprehensive overview of the current state of ICS honeypots, touching upon various implementations and advancements. It also addresses the challenges and desiderata in modern ICS honeypots, discussing also the legal implications entailed in running such honeypots.

Chapter 4 introduces the reader to the core of the thesis: a new physics-aware and high-interaction ICS honeynet called HoneyICS. Here, the thesis presents a detailed architecture of this honeynet, elaborating on its hybrid nature. The chapter further explores the attacker model, providing insightful perspectives on potential threats and vulnerabilities.

The thesis progresses to Chapter 5, where it illustrates the prototype implementation of the honeynet, from the plant: a Secure Water Treatment system, to the controllers. This chapter is a deep dive into the technical aspects, detailing the honeypot's technology stack and various components of the implementation process. It's a practical application of the theoretical concepts discussed earlier.

Chapter 6 of the thesis offers a comprehensive analysis of the data captured during months of exposing HoneyICS to the Internet. This chapter is methodical in its approach, discussing the research questions posed, methodologies employed, and the results produced from the data analysis. It's a chapter that ties the practical findings back to the theoretical underpinnings of the thesis.

Chapter 7 explores the types of cyber-attacks that the honeynet is equipped to handle. It also introduces an automated tool designed for executing these attacks. This chapter offers a critical evaluation of the honeynet's efficacy in a real-world scenario.

Finally, Chapter 8 wraps up the thesis with discussions, concluding remarks, and future work. It synthesizes the research findings, reflecting on their implications and charting a course for future exploration in the realm of cybersecurity for Industrial Control Systems.

Chapter 2

BACKGROUND ON ICS

Before starting with the main results of this thesis, this chapter, alongside the next, revisits some fundamental concepts. We start with a comprehensive exploration of Industrial Control Systems (ICS) and their elements in Section **??**, we proceed to examine Operational Technology (OT). This involves an in-depth analysis of the architecture of ICSs as per the Purdue Enterprise Reference Architecture (PERA) [142] and the industrial protocols utilized, as outlined in Section 2.1.

Industrial Control Systems (ICS) [128] represent a diverse set of control systems and instrumentation used across various industries to manage and automate industrial processes. These systems include a wide range of devices, networks, and controls that work together to manage and automate industrial tasks. Depending on the specific industry, each ICS is designed to operate uniquely and efficiently in electronic management of processes.

The devices and protocols employed in ICS have become pervasive in almost every industrial sector, playing crucial roles in critical infrastructure such as manufacturing, transportation, energy production, and water treatment industries. These systems are instrumental in ensuring smooth and reliable operations, enhancing productivity and performance of industrial processes.

As shown in Figure 2.1, there exist three primary types of architectures [115]: Standalone Control Systems, Distributed Control Systems (DCS), and Supervisory Control and Data Acquisition (SCADA) systems.

- The Standalone Control System, a simplistic design that efficiently manages processes such as escalators and elevators, primarily employing analog and digital inputs. However, with the advent of greater connectivity, these systems are now being connected to the Internet, arising new security challenges.

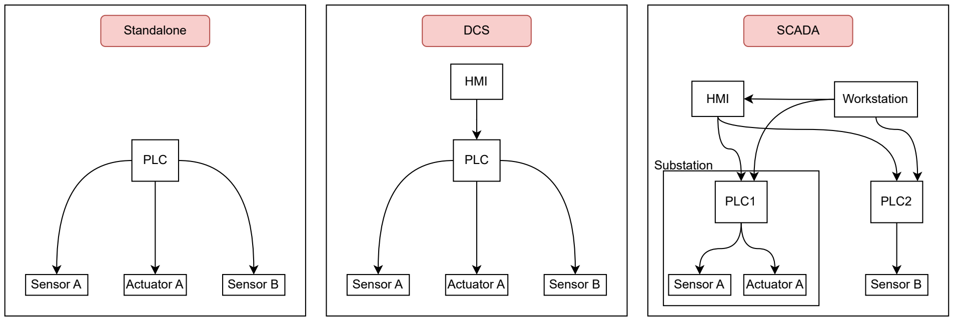


Figure 2.1: Difference between SCADA, DCS and Standalone systems

- The Distributed Control System (DCS) is more complex and commonly found in power plants, refineries, and manufacturing facilities. It comprises multiple components, such as PLCs, controlling various processes like turbines and pressure vessels. Redundant network connectivity to ensure maximum uptime. Moreover, multiple Human-Machine Interfaces (HMIs) could be used to allow operators to oversee and regulate processes.
- The SCADA system is prominent in power grids and water plants. It covers vast geographical areas with operators overseeing multiple regions through a centralized architecture. It usually consists of Remote Terminal Units (RTUs), Programmable Logic Controllers (PLCs), Human-Machine Interface (HMIs), communication infrastructures using industrial protocols and Supervisory Computers.

In the realm of organizational infrastructure, IT (Information Technology) and OT (Operational Technology) serve distinct purposes and have key differences in their functions and applications.

IT acts as the technological foundation of any organization, catering to front-end informational activities. Its primary role encompasses the monitoring, management, and security of core functions, including email, finance, human resources (HR), and various other applications housed in data centers and the cloud. Essentially, IT ensures the smooth functioning of administrative and communication processes, supporting the overall efficiency of an organization.

On the other hand, OT is specifically designed for the connectivity, monitoring, management, and security of an organization's industrial operations. Industries engaged in manufacturing, mining, oil and gas, utilities, transportation, and more heavily rely on OT. It operates behind the scenes, dealing with the production and operation of machinery and automation systems.

2.1 Overview

Operational Technology encompasses a wide range of technologies, including industrial control systems (ICS), Supervisory Control and Data Acquisition (SCADA) systems, Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs), Human-Machine Interfaces (HMIs), and various industrial sensors and actuators. These technologies work together to collect data from sensors, analyze it, and initiate actions to control and optimize industrial processes.

The convergence of IT and OT has become a prominent trend, leading to what is often referred to as the Industrial Internet of Things (IIoT). This integration enables enhanced data analysis, real-time monitoring, predictive maintenance, and overall improved efficiency and productivity in industrial settings. However, the increased connectivity also poses cybersecurity challenges, as securing OT systems becomes essential to protect critical infrastructure and industrial processes from potential cyber threats.

2.1.1 Purdue Enterprise Reference Architecture

The Purdue Enterprise Reference Architecture (PERA), also known as the Purdue model, is an influential enterprise architecture reference model from the 1990s. It was conceptualized and developed by Theodore J. Williams in collaboration with members of the Industry-Purdue University Consortium for Computer Integrated Manufacturing [142].

PERA provides a structured framework for designing and organizing OT systems, helping enterprises efficiently manage complex industrial processes. It facilitates seamless integration of various components, including industrial control systems (ICS), Supervisory Control and Data Acquisition (SCADA) systems, Programmable Logic Controllers (PLCs), Human-Machine Interfaces (HMIs), and industrial sensors and actuators.

The architecture consists of five layers [141]:

Level 0 — The physical process

This level defines the actual physical processes, it's the actual processes used to create or support the creation of the product the company sells. This could include various manufacturing or production processes [42] .

For instance, in a SWAT (water treatment) system, Level 0 encompasses the tangible processes essential for water purification. These processes are the fundamental steps where raw water is treated and transformed into clean, potable water that meets regulatory standards. The operations at this level involve various treatment techniques such as filtration, chemical dosing, sedimentation, and

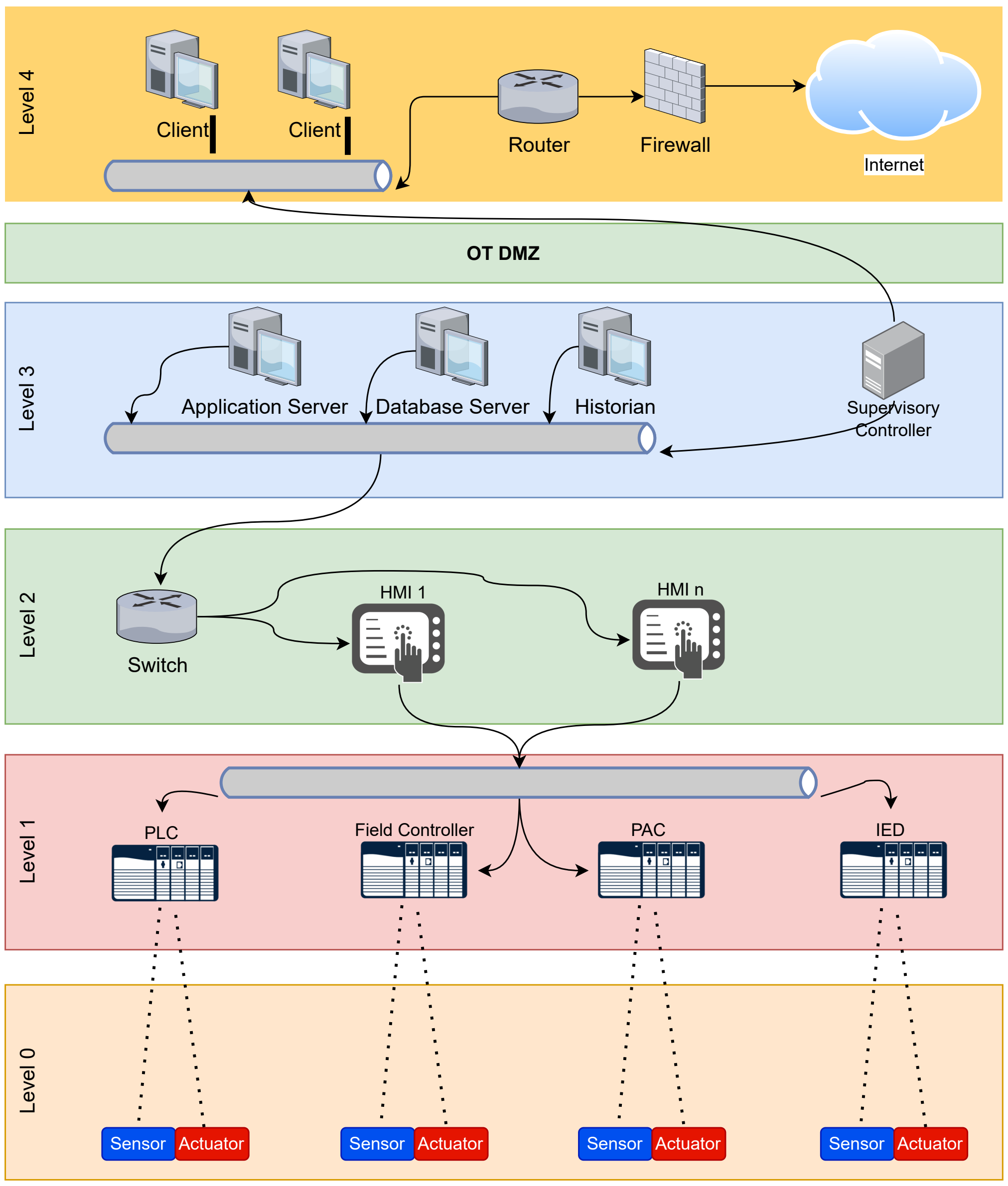


Figure 2.2: Purdue model

disinfection, all aimed at ensuring the quality and safety of the final water product. In this context, Level 0 serves as the foundation where the actual work occurs, turning source water into a finished product ready for distribution and consumption.

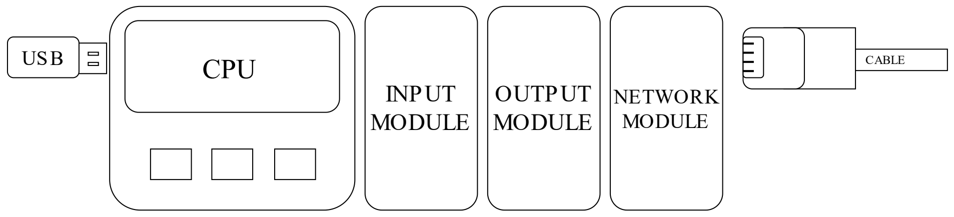


Figure 2.3: Programmable Logic Controller main components

Level 1 — Intelligent devices

This level involves sensing and manipulating the physical processes, employing various components such as process sensors, analyzers, actuators, and related instrumentation, including Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs), or Intelligent Electronic Devices (IEDs).

PLC

A Programmable Logic Controller (PLC) is a specialized industrial computer that possesses the ability to control complex industrial and manufacturing processes [128]. PLCs are optimized for control tasks in harsh industrial environments. They are ruggedly designed to withstand conditions such as dust, vibrations, humidity, and temperature, ensuring a higher level of reliability compared to commercial computers that are more susceptible to faults and crashes.

Moreover, PLCs have built-in I/O interfaces, simplifying the expansion process with additional I/O modules to manage more inputs and outputs without the need for hardware reconfiguration.

The control programs can be written using a simple and intuitive language called Ladder Diagram (LD) based on logic and switching operations, as opposed to requiring expertise in general-purpose programming languages like C or C++.

The PLC architecture comprises several essential elements [140]. The Processor Unit (CPU) serves as the core of the system, containing the microprocessor responsible for interpreting input signals from I/O modules, executing the control program stored in the Memory Unit, and sending output signals to the I/O Modules. It relies on two types of memory: RAM memory stores data from inputs, while ROM memory houses the operating system, firmware, and user program to be executed by the CPU.

The Power Supply Unit is responsible for converting AC voltage to DC voltage to power the system.

The I/O Modules play a crucial role in providing the interface between sensors and final control elements (actuators).

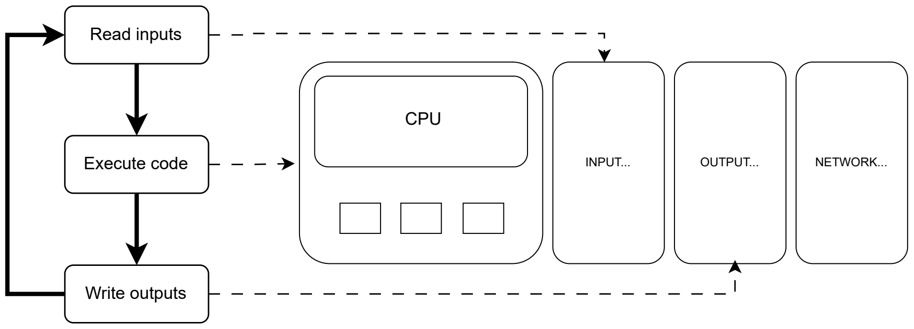


Figure 2.4: Scan cycle

Lastly, the Communications Interface enables the PLC to send and receive data on a network from/to other PLCs, facilitating interconnectivity and communication in industrial automation systems.

Within a PLC, two distinct programs are executed: the operating system and the user program. The operating system performs several essential tasks, including executing the user program and managing memory areas, as well as the process image table. This table acts as a collection of memory registers where inputs from sensors and outputs for actuators are stored. To run the user program, it must first be uploaded onto the PLC via the programming device. As shown in Figure 2.4 the program operates within scan cycles, each consisting of three phases [145]:

- Reading inputs from the process image table.
- Executing the control code and computing the physical process evolution.
- Writing output to the process image table to impact the physical process. The CPU refreshes the process image table at the end of each cycle.

IEC 61131-3 [25] standardized programming language for industrial automation. It defined the following programming languages for PLCs Instruction List (IL), Structured Text (ST), Ladder Diagrams (LD), Function Block Diagram (FBD), and Sequential Function Chart (SFC).

Level 2 — Control systems

This level involves the supervision, monitoring, and control of the physical processes. It involves the implementation of real-time controls and software, including Distributed Control Systems (DCS), Human-Machine Interface (HMI), and Supervisory Control and Data Acquisition (SCADA) software.

Human-Machine Interface

A Human-Machine Interface (HMI) is a user interface or dashboard that connects a person to a machine, system, or device. While the term can technically be applied to any screen that allows a user to interact with a device, HMI is most commonly used in the context of an industrial process [62].

In essence, an HMI is a specialized device or software that allows operators to communicate with and control machines or production systems. It achieves this by translating complex data into a user-friendly format, providing operators with the necessary information and tools to monitor and manage the production process effectively [61].

HMIs are the virtual access to the process data and configuration. They allow programmers to make interface changes on the fly and can restrict access to details based on user privileges.

HMIs essentially offer virtual access to process data and configuration. They empower programmers to make real-time interface adjustments and restrict access based on user privileges. In simpler terms, HMIs allow operators to monitor and control devices through a user-friendly interface.

Several open-source HMI applications are available, including ScadaBR and OpenSCADA:

- ScadaBR is a SCADA (Supervisory Control and Data Acquisition) system with applications in Process Control and Automation. It is being developed and distributed using the open source model [45]. ScadaBR has the capability to communicate with a variety of PLCs, including OpenPLC. It is based on a Tomcat web server, specifically version 6. Tomcat uses WAR (Web Application Archive) files to define a project. Within this file, which is a special type of Java archive, it contains the ScadaBR template, encompassing all the essential functionalities of ScadaBR.
- OpenSCADA: OpenSCADA is an open source HMI. It is platform independent and based on a modern system design that provides security and flexibility at the same time. But it could also serve for: acquisition, archiving (conduct history), visualisation of the information, issuing control actions, and also for other related operations, which are characteristic for full-featured SCADA or HMI systems [39].

Both of these systems are written in Java and are designed to be platform independent, providing flexibility and security for your SCADA needs.

Level 3 — Manufacturing operations systems

At this level, the focus is on efficiently managing the production workflow to achieve the desired products. This entails batch management, the utilization of manufacturing execution/operations management systems (MES/MOMS), and the integration of laboratory, maintenance, and plant performance management systems, alongside data historians and associated middleware.

Level 4 — Business logistics systems

This level is responsible for overseeing the business-related activities of the manufacturing operation. The primary system used for this purpose is Enterprise Resource Planning (ERP), which plays a key role in establishing the fundamental plant production schedule, material usage, shipping, and inventory levels.

2.1.2 Industrial communication protocols

The establishment of efficient and reliable communications plays a pivotal role in facilitating seamless data exchange among diverse devices and systems at the different levels of the PERA architecture. In this section, we embark on an exploration of the most prevalent and fundamental industrial network protocols that have become ubiquitous in various industrial sectors.

The most common industrial network protocols [125] used in industrial automation and control systems include:

- **Modbus**: Modbus [36] is a simple and widely adopted communication protocol used for connecting various devices, including programmable logic controllers (PLCs), sensors, and actuators. Originally it was a serial communication protocol, but in later updates it has been incapsulated into a TCP packet making it suitable for industrial Ethernet applications. The well-known TCP port for Modbus traffic is 502.
- **EtherNet/IP**: EtherNet/IP [17] is an industrial Ethernet protocol most commonly used in North America. It's based on Common Industrial Protocol (CIP), which is an object-oriented protocol where devices are viewed as a collection of objects. Ethernet/IP employs an open, standards-based approach, making it compatible with a wide range of industrial equipment. This protocol enables the exchange of data, control commands, and information among various components, facilitating efficient manufacturing processes and improving system interoperability. The well-known TCP port for EtherNet/IP traffic is 44818.
- **DNP3 (Distributed Network Protocol 3)**: DNP3 [40], was developed by Westronic, Inc. (now GE-Harris Canada) in the early 1990s, serves as the primary SCADA protocol in the electrical power grid domain. It enables control and data communication between SCADA system components using a master-slave architecture. Utility companies typically employ a central control station as the top-level DNP3 master, gathering data from substations,

displaying it, and making control decisions. DNP3 supports three communication modes: unicast transactions (e.g., read or write commands) between the master and a specific outstation device, broadcast transactions to all outstation devices, and unsolicited responses from outstation devices. The well-known TCP port for DNP3 traffic is 20000.

- **OPC UA (Open Platform Communications Unified Architecture)**: OPC UA [57] is an open and platform-independent protocol, facilitating secure data exchange between industrial devices and systems. It is the successor to the widely adopted OPC Classic and has established itself as a significant choice for adaptable communication in industrial contexts without stringent real-time demands. OPC UA functions as a client-server communication protocol, supporting a service-oriented architecture (SOA) for industrial use cases, spanning from devices on the factory floor to enterprise-level applications. It unifies the various iterations of previous OPC specifications into a coherent address space, which is accessible through a comprehensive array of standardized services. For each service, a pair of request-response data structures is clearly defined. Furthermore, OPC UA incorporates a range of robust security features. The well-known TCP ports are: 4840 for unencrypted communication and 4843 for TLS encrypted communications.
- **S7comm**: S7comm [41] is a proprietary communication protocol developed by Siemens for use with their PLCs. It is widely used in industrial automation and control systems, particularly in Europe. S7comm supports a wide range of I/O modules, making them suitable for a variety of applications. It is used for PLC programming, exchanging data between PLCs, accessing PLC data from SCADA (supervisory control and data acquisition) systems, and diagnostic purposes. The S7comm data comes as a payload of COTP data packets. The well-known TCP port for S7comm traffic is 102.

Among these protocols, one that stands out for its ubiquity and historical significance is Modbus. The next paragraph will explore the intricacies of the Modbus protocol and gain a deeper understanding of its key features, applications, and advantages.

Modbus

Modbus, a serial communications protocol, was conceived by Modicon, presently a subsidiary of Schneider Electric, in 1979. Its primary design objective was to facilitate an open, straightforward communication channel between programmable logic controllers (PLCs) and an array of devices, encompassing sensors, actuators, and supervisory control and data acquisition (SCADA) systems. [35]

Modbus is available in two distinct versions: the original Modbus RTU (Remote Terminal Unit), which initially constituted a straightforward serial communication protocol. Over time, the industry witnessed an escalating demand for a standardized framework capable of accommodating more intricate functionalities and seamless integration with prevalent transport protocols such as the Transmission Control Protocol/Internet Protocol (TCP/IP) and the User Datagram Protocol (UDP). To meet this evolving need, Modbus TCP was introduced in 1999. This variant was specifically designed to address the burgeoning requirements for extended capabilities and wider interoperability within the Modbus communication standard. Modbus RTU primarily utilizes RS-232 or RS-485 ports, depending on the specific implementation. RS-232 is suitable for short-distance connections, while RS-485 is favored for longer distances and multi-device networks. On the other hand, for Modbus TCP, which is designed for Ethernet-based communication, it relies on port 502 for communication [64]. MODBUS TCP is byte-oriented and has established itself as a de facto open standard. Polling communications adhere to the request-response mechanism, wherein a client initiates queries with the server to request specific data or execute commands within the server[1]. The server responds to client queries by transmitting a byte frame, which may contain either sensor measurement data or confirmation of command execution. Measurement values are stored in sixteen-bit data registers, while the status of ON and OFF switches is maintained in coils [101].

Modbus facilitates the mapping of temporary memory within a Programmable Logic Controller (PLC) program into four distinct categories of registers: (i) discrete output coils, (ii) discrete input contacts, (iii) analog input registers, and (iv) analog output holding registers. Notably, the latter registers also serve as general memory registers of varying sizes, including 16, 32, and 64 bits. Table 2.1 offers a comprehensive overview of the register information. The operations performed on these registers are executed through specific commands referred to as function codes, which are encapsulated within a Modbus frame. These function codes enable operations such as reading coils (FC01), querying discrete inputs (FC02), accessing multiple holding registers (FC03), retrieving input registers (FC04), writing to single coils (FC05), updating single holding registers (FC06), modifying multiple coils (FC15), and manipulating multiple holding registers (FC16) [1]. Table 2.2 offers a summary of the main function codes [81].

Figure 2.5 illustrates an example of a Modbus communication, using a function code (0x04) to read three continuous input registers in a remote device. The function can read from 1 to a manufacturer-defined maximum number of contiguous input registers. In this scenario, a client query requests a server to read the values of three continuous input registers: register address “14” (0x000E), register address “15” (0x000F), and register address “16” (0x0010).

[1] In accordance with the Modbus press release dated July 9, 2020 [3], and following the recommendation by ACM.org’s “Words Matter” initiative [65], the terminology ”client-server” has been employed in lieu of ”master-slave.”

Consequently, the client sends a single message, and the server responds by sending one frame containing the values of the three continuous registers1 [101].

The Modbus protocol, by design, lacks inherent security features, making it vulnerable to a range of specific threats as stated by [123]. These vulnerabilities include the potential for unauthorized parties to disclose confidential information through unauthorized read coil or read register requests, jeopardizing the confidentiality of sensitive data. Moreover, the integrity of data can be compromised when unauthorized write register or write coil requests are executed, allowing malicious actors to tamper with critical information. Additionally, the protocol is susceptible to availability compromise through denial-of-service (DOS) attacks, particularly those involving multiple write requests that overload the system. Furthermore, there are risks of authentication bypass through scan UID or scan discover requests, enabling unauthorized access to the network. Another significant finding, as highlighted by the research conducted by [139], is that certain vulnerabilities in Modbus systems may be attributed to the specific implementations rather than inherent flaws in the protocol itself. In their study, they developed a Modbus/TCP Fuzzer to assess eight different Modbus protocol implementations. Their investigations uncovered a range of bugs and vulnerabilities within these implementations, some of which had the potential to crash the system's execution, effectively leading to a denial-of-service scenario.

Table 2.1: Modbus Register Information

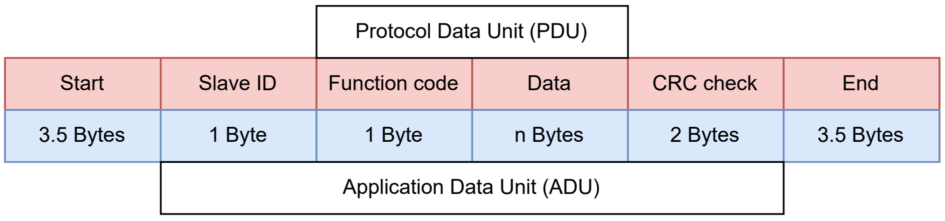
Data Type	Usage	PLC Addr.	Data Size	Access
Discrete Out. Coils	Digital Outputs	%QX0.0 – %QX99.7	1 bit	RW
Discrete Input	Digital Inputs	%IX0.0 – %IX99.7	1 bit	R
Analog Input Reg.	Analog Input	%IW0 – %IW1023	16 bits	R
Holding Reg.	Analog Outputs	%QW0 – %QW1023	16 bits	RW
Holding Reg.	Memory (16-bits)	%MW0 – %MW1023	16 bits	RW
Holding Reg.	Memory (32-bits)	%MD0 – %MD1023	32 bits	RW
Holding Reg.	Memory (64-bits)	%ML0 – %ML1023	64 bits	RW

2.2 ICS security

Since Industrial Control Systems (ICS) represent specialized industrial computers that play a pivotal role in managing critical infrastructure and process automation systems that find application in diverse sectors such as the power grid, water and wastewater management, transportation, and natural gas, as well as in process-intensive industries like nuclear power plants, oil refineries, steel mills, and factories. The potential consequences of a cyberattack on indus-

Table 2.2: Data Access Type and Function Codes in Modbus

Data Access Type	Function Code	Meaning
1 bit physical discrete input	0x02	Read discrete inputs
1 bit internal bits, physical coils	0x01	Read coils
1 bit internal bits, physical coils	0x05	Write single coil
1 bit internal bits, physical coils	0x0F	Write multiple coils
16 bit physical input registers	0x04	Read input registers
16 bit internal and physical output reg.	0x03	Read holding registers
16 bit internal and physical output reg.	0x06	Write single register
16 bit internal and physical output reg.	0x10	Write multiple registers
16 bit internal and physical output reg.	0x16	Mask write register
16 bit internal and physical output reg.	0x17	Read/write registers



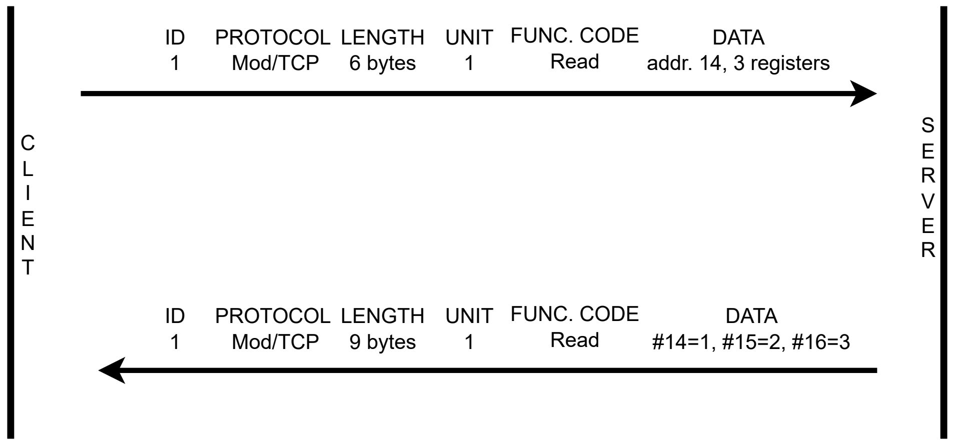


Figure 2.5: Modbus frame

trial control systems are far-reaching and catastrophic. A compromised power grid could lead to widespread outages and jeopardize critical services such as healthcare facilities, while contamination of water supplies could result in mass illnesses. Similarly, disruptions in transportation systems could cause chaos and pose grave risks to public safety. Consequently, the significance of cybersecurity in protecting these systems from exploitation and malicious intent cannot be overstated.

2.2.1 Cyber-physical attacks: a threat model based on STRIDE

In this section, we leverage existing literature to define a comprehensive threat model related to cyber-physical attacks. This exploration enables us to delineate the key distinctions and vulnerabilities that characterize the threat model, providing a solid foundation for comprehending the intricacies of these attacks and their potential ramifications on OT systems. According to Xiong et al. [144] "Threat modelling is proposed as a solution for secure application development and system security evaluations. Its aim is to be more proactive and make it more difficult for attackers to accomplish their malicious intents.". After analyzing the available literature they discovered that threat modelling often involves the following steps:

1. *System Representation*: This involves creating a detailed representation of the system, often using diagrams or other visual aids. This can include data flow diagrams, architectural diagrams, and component diagrams.
2. *Identification of Threats*: Using the system representation, potential threats are identified. Common methodologies for this include STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) and PASTA (Process for Attack Simulation and Threat Analysis).
3. *Vulnerability Analysis*: Once threats are identified, the system is analyzed to determine potential vulnerabilities that could be exploited by these threats.
4. *Risk Assessment*: Each identified threat is then evaluated based on its potential impact and the likelihood of it being exploited. This often involves creating a risk matrix.
5. *Mitigation Strategies*: For each identified threat, countermeasures are proposed to mitigate the risk. This could involve changes in the system architecture, implementing additional security controls, or other protective measures.
6. *Documentation*: The findings from the threat modelling process, including identified threats, vulnerabilities, risks, and proposed countermeasures, are documented. This documentation serves as a guide for developers, security professionals, and other stakeholders.
7. *Review and Iteration*: As systems evolve and new threats emerge, the threat model should be periodically reviewed and updated.

The STRIDE framework offers a well-established approach for identifying and categorizing potential security threats in systems. Its strengths include:

- Systematic analysis: STRIDE provides a structured way to analyze vulnerabilities at the component level.

- Effectiveness in identifying threats: STRIDE has proven effective in uncovering various threats in software systems.

However, it's important to acknowledge that STRIDE was originally developed for IT systems. While Jelacic et al. [103] and Khan et al. [105] advocate for its use in CPS, limitations exist:

- *Software-centric approach*: STRIDE's focus on software elements might not fully capture the unique vulnerabilities of physical processes integrated with CPS
- *Limited physical domain consideration*: The framework might not adequately address threats that directly manipulate or disrupt physical components within a CPS.

The STRIDE framework was developed by Microsoft to identify and categorize potential security threats in a system. The acronym STRIDE stands for:

- *Spoofing*: Masquerading of a legitimate user, process, or system element.
- *Tampering*: Modification or editing of legitimate information.
- *Repudiation*: Denying or disowning a certain action executed in the system.
- *Information* Disclosure: Data breach or unauthorized access to confidential information.
- *Denial of Service (DoS)*: Disruption of service for legitimate users.
- *Elevation of Privilege*: Gaining higher privilege access to a system element by a user with restricted authority.

When applying the STRIDE methodology to Cyber-Physical Systems (CPS), [105] proposes the following steps:

- Decompose System into Components: Break down the system into its logical or structural components. Components can be internal processes/elements communicating internally within the system or external elements communicating with the system.
- Plot Data Flow Diagram (DFD) for System Components: Visualize the functionalities of each system component within or external to the system using DFD. The DFD uses four standard symbols:
  - External Entity (EE): End-points of the system.
  - Process (P): Units of functionality.
  - Data Flow (DF): Communication data.
  - Data Store (DS): Database.
- Analyze Threats in the DFD: Identify potential threats in the data flow diagram. It was observed that certain STRIDE threats impact a group of DFD elements. Spoofing and tampering are especially critical and they impact the

operations of other elements, particularly in the physical domain, resulting in more severe consequences for the system.

- Identify Vulnerabilities: Recognize vulnerabilities against each system component which could be exploited by an attacker to compromise the entire system. Due to inter-dependencies between system components, the entire system security can only be ensured by addressing vulnerabilities of each system component.
- Plan Mitigation Strategies: Develop strategies to counter the identified threats and vulnerabilities.

However, it's important to note that threat modelling is an ongoing process, and as CPS evolve and new threats emerge, it is imperative to periodically review and update the threat model. This iterative approach ensures that the security of CPS remains robust and adaptive to the ever-changing threat landscape.

2.2.2 Vulnerabilities of ICSs

In this section, we investigate a critical aspect of modern industrial systems. We will examine how attackers gain access to these environments, gather critical information, exert control over industrial processes, inflict damage, and attempt to conceal their activities. Additionally, we will elucidate the multitude of attack vectors that adversaries may employ to compromise ICSs, ranging from exploiting vulnerabilities within Programmable Logic Controllers (PLCs) and Human Machine Interfaces (HMIs) to infiltrating Engineering workstations and leveraging weaknesses in network security.

Anatomy of a cyber-physical attack targeting an Industrial Control System

A cyber-physical attack is composed of different stages [94], each playing a crucial role in achieving the attacker's objectives:

1. *Access Stage:* At the Access Stage, the attack initially resembles traditional IT hacking. The attacker's goal is to execute code within the victim's network, gaining a foothold to manipulate critical processes. This stage serves as the entry point into the targeted system. There are two possible ways to achieve this: either through compromising the IT network, possibly exploiting a vulnerable device or an unprotected VPN, or by infiltrating an Internet-facing device that might be vulnerable due to network misconfiguration (e.g., UPnP [63] activation), or simply out of necessity to access a remote device that lacks built-in VPN support because of its age.

2. *Discovery Stage:* The Discovery Stage revolves around gathering information about the industrial plant from available documentation. Without comprehensive knowledge, an attacker is limited to causing nuisance rather than significant damage. This phase underscores the importance of understanding the system's intricacies. It can be executed using a variety of tools. Initially, the attacker must gain an understanding of the devices within the network, including PLCs, HMIs, and other components. To achieve this, they may choose to employ network discovery tools such as Nmap [37]. Nmap is also valuable for identifying known vulnerabilities in the discovered devices. Subsequently, the attacker can transition to the analysis of the physical system's behavior. This can be accomplished using tools like the one introduced by Ceccato et al. [81], which is designed to capture register values and to collect them into a dataset for acquiring additional insights.
3. *Control Stage:* Within the Control Stage, intricate knowledge of the industrial system's dynamics is essential. In dynamic systems like cyber-physical systems, process variables evolve over time according to physical laws. Here, the attacker studies the functions of each actuator and assesses potential side effects. For instance, disabling a pump may lead to rapid pressure buildup in an upstream pipe. Timing is also a critical aspect of this stage, as precise coordination may be required to manipulate the system effectively.
4. *Damage Stage:* The Damage Stage, while less familiar to traditional IT hackers, poses profound challenges. It often necessitates the input of subject matter experts to comprehend the full spectrum of possibilities. This phase is where the attacker inflicts substantive harm to the industrial process, potentially resulting in equipment damage or financial losses.
5. *Clean-up Stage:* In contrast to traditional IT hacking, where stealth is paramount, the Clean-up Stage in process control scenarios does not permit going undetected. Any equipment damage or sudden profit reduction prompts investigation. Therefore, the Clean-up Stage focuses on creating a forensic footprint that misleads investigators. This involves manipulating the process and log data to lead analysts to incorrect conclusions. For example, showing the operator an out-of-control process and coercing specific actions can be part of the cleanup phase. The goal here is to obfuscate the true nature and origin of the attack, diverting attention from the actual perpetrator.

Attack vectors leading to an ICS compromise

ICS are susceptible to attacks through various vectors. These attack vectors can involve multiple elements and allow for diverse combinations, making them complex to defend against. The attacker can potentially exploit various points of entry [129], including PLCs, Remote Terminal Units (RTUs), Intelligent Electronic Devices (IEDs), Engineering workstations, HMIs, and the IT network. Vulnerabilities within PLCs could include buffer overflows, backdoors, weak authentication and encryption, which could allow attackers to take control of the device and interfere with or halt the process it controls.It could also occur that vulnerabilities do not result from the manufacturer introducing bugs but from the client's actions, such as failing to set up a password, as highlighted by a study conducted by Claroty [8]. The study revealed that 62.5% of TBox RTUs exposed to the Internet did not require authentication. Additionally, the potential for remote code execution via HMIs introduces an additional layer of complexity to the threat landscape, facilitating lateral movement within operational technology networks.

2.2.3 Goals of the attackers

Cyberattacks on ICS have increased over time, with notable incidents such as the infamous Stuxnet attack in 2010, followed by attacks on steel mills [110], power grids [80], petrochemical plants, gas pipelines, and water treatment facilities. The motivations behind such attacks vary, ranging from political gain by nation-states [55, 89] to financial gains by organized cybercrime groups [95].

In the context of politically motivated attackers, the likelihood of observing zero-day exploits increases significantly due to the strategic nature of their actions. These attackers, often nation-state actors or highly organized groups, invest considerable time and resources into their campaigns, driven by the need to achieve specific political objectives or disrupt critical infrastructures. Zero-day vulnerabilities are weaknesses in software or hardware that are unknown to the vendor and for which no patch or other fix is yet available. Because these vulnerabilities are undetected, attackers can exploit them to gain unauthorized access to systems. Zero-day exploits are malicious programs specifically designed to take advantage of these unknown vulnerabilities.Given the precision required to infiltrate well-protected systems associated with government agencies or politically significant entities, these attackers are more inclined to utilize previously unknown vulnerabilities (zero-day exploits). These exploits offer a distinct advantage by bypassing existing security measures, enhancing the efficacy of their targeted attacks, and allowing them to maintain a covert and persistent presence within the compromised systems.

These targeted attacks can be divided into three classes, as proposed by Gollmann et al. [94].

Firstly, the class of *Equipment Damage* seeks to inflict physical harm on vital equipment and infrastructure components, such as pipes and valves. *Overstress of Equipment*, a subcategory within this class, can accelerate the wear and tear of machinery, posing a severe threat to industrial processes. Notably, historical instances like the wear-off attacks on valves in the second version of the Stuxnet worm underscore the destructive potential of this approach. Additionally, Violation of Safety Limits can lead to equipment damage, as demonstrated by the remote destruction of a power generator at Idaho National Labs [5].

Secondly, *Production Damage* shifts the focus from equipment destruction to disrupting the production process itself, aiming to spoil products or increase production costs. Within this category, sub-groups encompass attacks on Product Quality and Production Rate, Operating Costs, and Maintenance Efforts, all of which can significantly impact industrial efficiency and economic outcomes.

Lastly, *Compliance Violation*, can have significant consequences for industries that are heavily regulated. These violations are categorized into three categories:

- *(i)* "Safety," focuses on attacks that endanger occupational and environmental safety, potentially leading to catastrophic accidents and long-lasting environmental harm;
- *(ii)* "Environmental Pollution," encompasses attacks breaching regulatory pollution thresholds, encompassing concerns related to emissions, water and soil contamination, with consequences including financial penalties, plant shutdowns, and reputation damage for non-compliance with environmental regulations;
- *(iii)* "Contractual Agreements," with a specific focus on production schedules and commitments.

On the other hand, as stated by Chen et al. in [82] "financially motivated attacks target IT systems more than industrial systems, poor network architecture management can result in the spread of malware from IT systems to industrial systems.". The Air Canada attack of 2003 is a compelling demonstration of how malware initially targeting IT systems can infiltrate industrial systems due to inadequate network architecture management. This incident was caused by the Welchia worm which was designed to remedy the havoc caused by the Blaster worm by downloading patches directly from Microsoft. However, its infiltration into Air Canada's network systems disrupted the airline's passenger processing operations at reservation and call centers, contrary to its intended purpose of neutralising the Blaster worm's threat [58, 77].

2.2.4 Notable cyber-physical attacks

This section lists a series of notable cyberattacks that have struck at the heart of industrial control systems, serving as stark reminders of the vulnerabilities inherent in these complex environments. These attacks are an image of the evolving tactics employed by malicious actors and the growing importance of safeguarding ICS against digital threats. Miller et al. [82] present a comprehensive chronological overview of ICS malware incidents. In Figure 2.6, you can observe

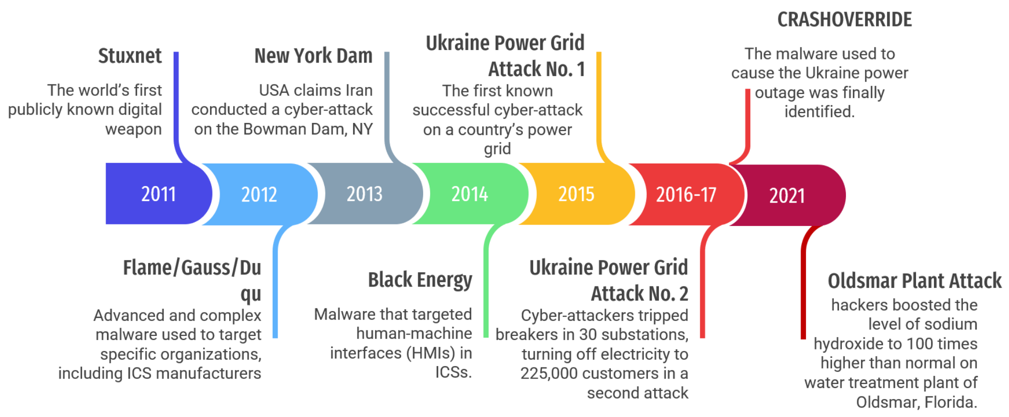

Stuxnet
The world's first publicly known digital weapon
New York Dam
USA claims Iran conducted a cyber-attack on the Bowman Dam, NY
Ukraine Power Grid Attack No. 1
The first known successful cyber-attack on a country's power grid
CRASHOVERRIDE
The malware used to cause the Ukraine power outage was finally identified.
2011
2012
2013
2014
2015
2016-17
2021
Flame/Gauss/Duqu
Advanced and complex malware used to target specific organizations, including ICS manufacturers
Black Energy
Malware that targeted human-machine interfaces (HMIs) in ICSs.
Ukraine Power Grid Attack No. 2
Cyber-attackers tripped breakers in 30 substations, turning off electricity to 225,000 customers in a second attack
Oldsmar Plant Attack
hackers boosted the level of sodium hydroxide to 100 times higher than normal on water treatment plant of Oldsmar, Florida.

Figure 2.6: Thread landscape

the chronological order of many significant incidents that will be discussed in the following pages.

- Stuxnet (2011) [89]: A highly sophisticated worm that marked a turning point in ICS security, targeting Iran's nuclear facilities and exploiting vulnerabilities in Siemens industrial control systems.
- Flame/Gauss/Duqu (2012) [78]: A trio of malware campaigns linked to nation-state actors, with Flame being one of the most complex espionage tools ever discovered. Through binary analysis, it was found that Gauss shares some features with Stuxnet and Flame, such as the use of object-oriented structures, which are utilized due to the complex logic of these threats. However, there are distinctions in their implementations, like different injection techniques and configuration information storage. Despite some similarities, Gauss is a standalone malware, possibly developed by the same individuals behind Stuxnet and Flame, but it doesn't directly build upon either of the two frameworks.
- New York Dam Attack (2013) [28]: An attempt to gain control over a small dam's systems in Rye, New York, carried out by Iranian hackers in 2013. According to the U.S. Justice Department, this cyber attack involved the infiltration of the computerized controls of the dam, and was part of a coordinated effort by seven Iranian hackers who not only targeted the dam but also conducted cyber attacks on dozens of U.S. banks, causing substantial financial loss.
- BlackEnergy (2014) [22]: Cyberattack on western Ukraine's Prykarpattyaoblenergo utility, which led to a power outage for 80,000 customers, marking it as the first known instance of a power outage induced by a cyberattack. Ukraine's state security service attributed this attack to state-sponsored hackers from Russia.

- Ukraine Power Grid Attacks (2015, 2016-2017)
  - Ukraine Power Grid First Attack (2015) [10]: A cyberattack that resulted in widespread power outages, showcasing the real-world impact of ICS breaches.
  - Ukraine Power Grid Second Attack (2016-2017) [10]: A follow-up attack that targeted the Ukrainian power grid once again, highlighting the persistence and evolving tactics of threat actors.
- Crashoverride (2016-2017) [9]: The first known malware framework designed specifically to disrupt electric grid operations. It is a highly capable platform designed to target Industrial Control Systems (ICS), notably used in a cyberattack against Ukraine's critical infrastructure in 2016, attributed to Russian nation-state cyber actors. CrashOverride malware exploits a known issue in Siemens Siprotec digital relay to manipulate circuit breakers.
- Oldsmar Plant Attack (2021) [11]: Hackers allegedly infiltrated the water treatment system of Oldsmar, Florida, and attempted to poison the local water supply by significantly increasing the levels of sodium hydroxide (lye) in the water. A recent controversy [21] suggests was, in fact, the result of an employee error.

These incidents serve as case studies, offering valuable insights into the methodologies, motivations, and consequences of attacks on industrial control systems. It is now crucial to better explore one that marked a significant turning point in cyber-attacks on industrial systems: Stuxnet and as stated by [78] its variants: Duqu, Flame and Gauss.

Stuxnet

Stuxnet had two main technological components: a computer worm that spread the malware on Windows networks and a digital payload that infected Windows hosts and controlled Siemens PLCs. The worm component used several 0-days to propagate the virus: CVE-2010–2568 and CVE-2008–4250.

Firstly, it spread through USB devices by crafting specific .LNK files. These files leveraged a vulnerability in Microsoft Windows LNK File Execution Shortcuts (CVE-2010–2568), allowing the virus to execute automatically when the USB device's contents were accessed, even if AutoRun and AutoPlay were disabled. This method was believed to breach "air gap" network defences, possibly through the unwitting actions of an Iranian facility operator or a double agent within the facility. Secondly, the virus could execute and exploit certain requests, enabling remote code execution on the local host. It exploited vulnerabilities in Microsoft Windows Print Spooler Server (CVE-2010–2779) and Microsoft Windows RPC Server (CVE-2008–4250) for this purpose. Moreover, it had the capability to copy itself to network folders shared on the local computer, using local users found on the computer, within the Windows domain, or through WMI

Explorer impersonation. Lastly, the virus searched for Siemens SIMATIC Step7 projects (identified by .s7p file extensions) on the infected system. When found, it infiltrated these project folders, modifying the main index files.

Remarkably, Stuxnet's rootkit employed a method to load dynamic link libraries (DLLs) that evaded behavior blocking and intrusion-detection technologies. It manipulated Windows kernel's NTDLL.DLL, intercepting commands and injecting code into trusted OS processes, like svchosts.exe and security programs like avp.exe (Kaspersky) and mcshield.exe (McAfee).

Once established as a service, Stuxnet sought Siemens SIMATIC Step7 software, altering the original S7OTBXDX.DLL with a modified version that retained the same exports but contained critical code changes affecting PLC functions. It acted as a Man-in-the-Middle, intercepting and modifying data between the compromised host and the PLC, facilitating control manipulation. Importantly, if Siemens Step7 software was absent, Stuxnet refrained from further malicious actions.

To maintain communication with a remote server, Stuxnet conducted connectivity tests using non-malicious URLs such as windowsupdate.com or msn.com. Upon successful testing, it connected to remote servers (mypremierfutbol.com, todaysfutbol.com) to send and receive commands, employing XOR-encrypted data containing system information and installation status. The server could respond with updated malware versions or data exfiltration commands.

Lastly, if Siemens software was detected, Stuxnet accessed connected PLCs through S7OTBXDX.DLL and WinCC's default MSSQL database credentials. It identified the CPU type and monitored connected field devices, specifically Vacon or Fararo Paya frequency driver converters operating within a specific frequency range. At periodic intervals, Stuxnet altered the output frequency of these converters, causing mechanical stress to centrifuges and increasing the likelihood of failure and reduced uranium processing quality. Stuxnet also manipulated visual SCADA components to conceal its actions from operators.

Duqu, Flame and Gauss

"The Cousins of Stuxnet: Duqu, Flame, and Gauss" [78] is a technical report that provides an in-depth analysis of the malware samples that belong to the Stuxnet family. The report discusses the similarities and differences between Duqu, Flame, and Gauss.

Duqu is an information-collecting malware that was first detected in 2011. It is believed to have been used in state-sponsored cyber espionage operations mainly in the Middle East. Duqu is designed to collect sensitive information from infected systems, including keystrokes, network traffic, and system information. It is also capable of communicating with command and control servers to receive additional instructions. Duqu is similar to Stuxnet in that it uses a modular design and exploits zero-day vulnerabilities to infect systems.

Flame is an advanced information-gathering malware that was discovered in 2012. It is also believed to have been used in state-sponsored cyber espionage operations. Flame is unique in the sense that it used advanced cryptographic techniques to masquerade as a legitimate proxy for the Windows Update service . It is designed to collect a wide range of sensitive information from infected systems, including keystrokes, screenshots, and audio recordings. Flame is also capable of communicating with command and control servers to receive additional instructions.

Gauss is a malware sample that was discovered in 2012. It is also believed to have been used in state-sponsored cyber espionage operations. Gauss is unique in the sense that one of its modules is encrypted such that it can only be decrypted on its target system. This makes it difficult for researchers to analyze the module and understand its functionality. Gauss is designed to collect sensitive information from infected systems, including browser history, cookies, and passwords. It is also capable of communicating with command and control servers to receive additional instructions .

Chapter 3

RELATED WORK ON ICS HONEYPOTS

In this section, we explore the world of ICS honeypots, examining their fundamental concepts and the motivations behind their deployment. We analyze the current state of ICS honeypots, their inherent limitations, and the contemporary challenges they face. Finally, we address the legal implications and ethical considerations associated with the operation of honeypots in the pursuit of cybersecurity.

3.1 An introduction to honeypots

Honeypots are a unique type of cybersecurity tool designed to deliberately attract and deceive attackers. They act as strategically placed, vulnerable systems that appear legitimate to intruders [132]. This intentional deception allows honeypots to observe attacker behavior, gather valuable intel, and ultimately improve an organization's overall security posture. Honeypots can be categorized based on their level of interaction with attackers:

- *High-Interaction Honeypots*: These honeypots mimic real systems and services in great detail, providing attackers with an authentic environment to interact with. This type allows for extensive observation of attacker tactics, techniques, and procedures (TTPs) but requires significant resources and poses a higher risk if compromised.
- *Medium-Interaction Honeypots*: These offer a balance between realism and safety. They simulate certain aspects of a real system, providing enough

interaction to engage attackers without the complexity and risk associated with high-interaction honeypots.
- *Low-Interaction Honeypots*: These are simpler and simulate specific services or ports. They are easier to deploy and maintain, offering a basic level of engagement with attackers but providing limited information about their behavior.
- *Physics-Aware Honeypots*: These are specialized honeypots designed for environments where physical processes are involved, such as industrial control systems (ICS) or Internet of Things (IoT) devices. They simulate the physical behaviors and responses of these systems, enabling the detection and study of attacks that target physical processes.

The core value of a honeypot lies in its ability to:

- *Facilitate Attack Observation*: By mimicking real systems, honeypots lure attackers into attempting exploits or malicious actions. This controlled environment allows security teams to observe attacker tactics, techniques, and procedures (TTPs) without risking damage to production systems.
- *Enhance Intrusion Detection*: Honeypots can function similarly to Intrusion Detection Systems (IDS) by detecting and logging suspicious activity within the honeypot environment. Thus, providing valuable early warnings of potential threats targeting the network.

While both honeypots and traditional IDS serve intrusion detection purposes, a key distinction exists. Traditional IDS typically monitor network traffic for patterns indicative of malicious activity. Honeypots, on the other hand, take a more proactive approach. They actively "bait" attackers by presenting a seemingly attractive target, allowing for deeper analysis of attacker behavior and the collection of in-depth threat intelligence.

3.2 State of the art of ICS honeypots

In this section, we explore into the 'State of the Art of ICS Honeypots,' since the literature is pretty rich we use a classification method based on the underlying technology employed by these honeypots [113]. Our primary focus lies on two significant frameworks that have shaped the landscape of ICS honeypots: Honeyd [127] and Conpot.

These two foundational honeypot architectures, Honeyd and Conpot, have contributed profoundly to the study and defense of ICS environments. We structure our analysis around these key distinctions: honeypots based on Honeyd, those based on Conpot, and those that are not based on either of them. This categorization allows us to comprehensively examine the development and utilization of ICS honeypots, shedding light on the unique characteristics and strategies employed by each group.

Furthermore, our exploration extends beyond the surface, exploring the methodologies employed for the evaluation and analysis of these honeypots. We scrutinize the assessment criteria, data analysis techniques, and the insights generated by each work. By doing so, we aim to provide a better understanding of the effectiveness and limitations of ICS honeypots.

3.2.1 ICS honeypots based on Honeyd

Honeyd is a virtual honeypot framework that was developed by Niels Provos [127], which aim was to provide a flexible and scalable solution for creating honeypots. The initial release of the project was made on February 15, 2003, while the most recent update was made on May 27, 2007 [23], despite its age Honeyd is still used in recent and prominent works such as [112]. This framework allows for the creation of thousands of virtual honeypots (up to 65536).

Honeyd's architecture is based around various components, as illustrated in Figure 3.1. These components include a configuration file where the user sets the personality, ports and services to use, a personality database file called *nmap-os-db* with all the available personalities, a central packet dispatcher, protocol handlers, a personality engine, and an optional routing element. Honeyd's architecture was designed to emulate various routing topologies to confuse potential adversaries and network mapping tools. This purpose differs from the discrete event network (NS-based) simulators [88], which aim for an accurate representation of network behavior. In Honeyd's approach, the simulation is designed to deceive adversaries rather than precisely replicate network behavior. The virtual routing topology resembles a tree structure, with the root at the point of packet entry. Each interior node represents a router, and each edge is a link with a defined latency and packet loss number. Terminal nodes correspond to the emulated devices or network.

Listing 3.1: Sample config file for Honeyd

```
create windows
set windows personality "Microsoft Windows XP Professional SP1"
set windows uptime 1728650
set windows maxfds 35
add windows tcp port 80 "scripts/web.sh"
add windows tcp port 135 open
add windows tcp port 139 open
add windows tcp port 445 open
set windows ethernet "dell"
set windows default tcp action closed

create avaya
```

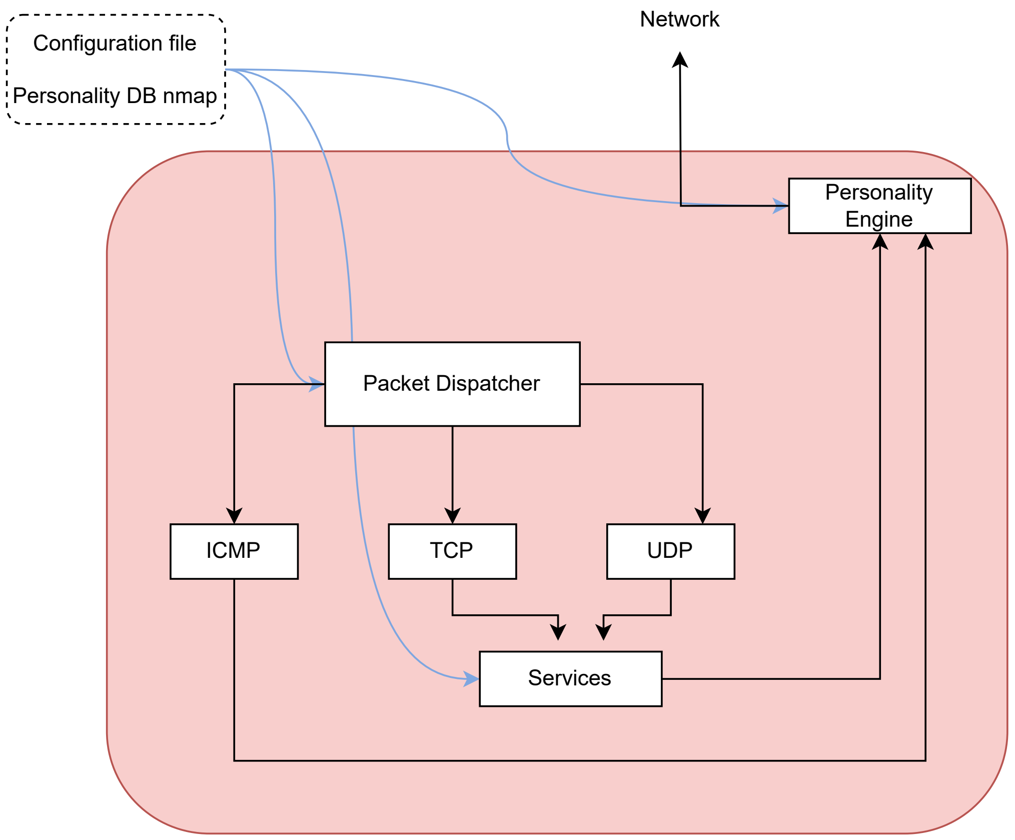


Figure 3.1: Honeyd's architectural overview

```
set avaya personality "Avaya G3 PBX version 8.3"
set avaya default tcp action reset
add avaya tcp port 4445 open
add avaya tcp port 5038 open
```

In listing 3.1, a sample configuration file for Honeyd is shown. The configuration file showcases how Honeyd can be written to mimic specific devices, operating systems, and network services. In this scenario, two distinct honeypots are configured with unique characteristics, a network diagram is shown in Figure 3.1. The first honeypot is configured to mimic a system with the personality of Microsoft Windows XP Professional SP1. This simulated system opens ports 80, 135, 139, and 445, and when an attacker accesses port 80, it triggers the execution of the web.sh script. Furthermore, this honeypot is configured to emulate a network interface with a MAC address associated with the manufacturer Dell. The second honeypot, on the other hand, is designed to replicate an

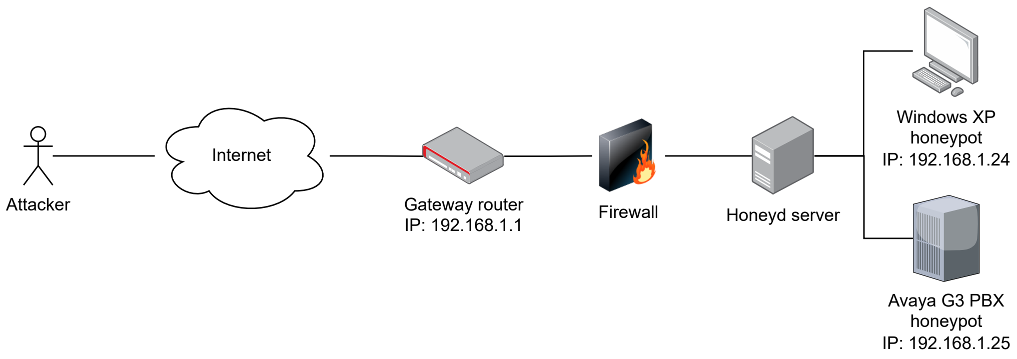


Figure 3.2: Network Diagram for Honeyd's config file in Listing 3.1

Avaya G3 PBX system with version 8.3, typically employed in private telephone networks within corporate or organizational settings.

One of the key features of Honeyd is its ability to mimic the network stack behavior of operating systems to deceive fingerprinting tools like Nmap [37]. Nmap's fingerprinting technique allows to identify the operating system and other characteristics of a remote host by analyzing its responses to specially crafted packets. Nmap fingerprinting works by sending different types of probes to the target and comparing the results with a database of known fingerprints, namely *nmap-os-db*. The probes can vary in protocol, port, flags, options, payload, and timing. As shown in Listing 3.2, a fingerprint entry associates a name with specific behavior. This information helps identify the type of device being scanned, such as *Allen Bradley MicroLogix 1100 PLC*. It includes information on open and closed TCP ports (OT and CT), closed UDP port (CU), private IP space (PV), network distance (DS), distance calculation method (DC) and target MAC prefix (M). OT and CT are printed in decimal format, while CU is the same as CT but for UDP.

Listing 3.2: Sample Entry from the *nmap-os-db* Database

```
# Allen Bradley MicroLogix 1100 PLC
Fingerprint Allen Bradley MicroLogix 1100 PLC
Class Allen-Bradley | embedded || specialized
CPE cpe:/h:allen-bradley:micrologix_1100
SEQ(SP=82-8C%GCD=1-6%ISR=99-A3%TI=I%CI=I%II=I%SS=S%TS=U)
OPS(O1=M4000NNS%O2=M4000NNS%O3=M4000NNS%O4=M4000NNS%O5=M4000NNS%O6=M4000NNS)
WIN(W1=800%W2=800%W3=800%W4=800%W5=800%W6=800)
ECN(R=Y%DF=N%T=7B-85%TG=80%W=800%O=M4000NNS%CC=N%Q=)
T1(R=Y%DF=N%T=7B-85%TG=80%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=Y%DF=N%T=7B-85%TG=80%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T3(R=Y%DF=N%T=7B-85%TG=80%W=800%S=O%A=S+%F=AS%O=M4000NNS%RD=0%Q=)
T4(R=Y%DF=N%T=7B-85%TG=80%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T5(R=Y%DF=N%T=7B-85%TG=80%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
```

```
T6(R=Y%DF=N%T=7B-85%TG=80%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=N%T=7B-85%TG=80%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
U1(R=N)
IE(DFI=N%T=7B-85%TG=80%CD=Z)
```

Nmap fingerprinting can be performed in two modes: active and passive. Active mode requires sending probes to the target, while passive mode only observes the traffic between the target and other hosts. Using the personality engine Honeyd can modify packets to match the fingerprints of other operating systems. This makes it difficult for attackers to identify the honeypot as a fake system, as it appears to be a legitimate system running a specific operating system.

In addition to its ability to mimic operating systems, Honeyd can also be set to selectively proxy connections to services in the backend. This feature is called subsystem virtualisation and leverages dynamic library preloading [30] to effectively substitute the original networking functions of OpenPLC with Honeyd code. This feature complements its capacity to selectively proxy connections to backend services. This allows for the creation of more complex honeypot environments that can emulate entire networks. Moreover, Honeyd has been shown to support 30 MBit/s aggregate bandwidth and sustain over two thousand TCP transactions per second. This makes it a scalable solution that can be used in large-scale deployments.

Cisco Systems' SCADA HoneyNet Project [135]

The first honeynet relying on Honeyd was Cisco Systems' SCADA HoneyNet Project [135], in 2004. It can emulate routers, wireless access points, serial interfaces, and SCADA protocols. The project is based around Honeyd to emulate diverse applications on SCADA devices, including web servers and management applications. It has an open-source Python module that facilitates serial interface programming, which can be used to emulate serial communication between a PC and a SCADA device or serial network, even emulating serial protocols like MODBUS and DNP3. Additionally, the project utilizes the HostAP Driver, that converts a client adapter into an access point, allowing for the emulation of an access point within a SCADA network. It can respond to 802.11b management packets and use proprietary wireless protocols. In addition to the tools described above, the project also has keystroke loggers to capture the keystrokes of attackers who access the web interfaces of emulated devices, employs Java applets to communicate with the attacker's web browser, replicates Remote Desktop Access (RDP) and HMIs, that typically provide remote access through methods like VNC or RDP and finally, the tool also emulated a network access server that permits dial-in access to the network using PPP and authentication via a PPP password.Where, once authenticated, it provides direct access to the industrial device or network. To achieve this, the authors describe potential deployment scenarios for the honeypot and scripts, including the placement in a subnet near an actual industrial network, assigning a phone number associated with a SCADA plant, or connecting to a remote access server linked to industrial devices.

Winn et al. [143]

Winn et al. proposed *Honeyd+* [143], which is able to build up to 75 high-interaction honeypots using a proxying technique with a single physical PLC device. The technique allows multiple high-interaction honeypots to be created using a single programmable logic controller (PLC), which reduces the hardware and maintenance costs. The technique also enhances the authenticity and targetability of the honeypots by using search and replace functions to modify the network identifiers and protocol responses of the PLC. The main features of the project are:

- *Proxy functionality*: Honeyd+ uses Honeyd's built-in proxy capability to forward incoming connections to a physical PLC. This enables Honeyd+ to simulate realistic behavior.
- *Search and replace functionality*: Honeyd+ modifies the Honeyd source code to add search and replace functions that can dynamically change the network identifiers and protocol responses of a PLC. This makes each honeypot host unique and consistent. The search and replace terms are specified in the Honeyd+ configuration file using a custom plug-in module called icsproxy.
- *Protocol independence*: Honeyd+ supports any protocol that uses length-based error checking, such as EtherNet/IP. However, Honeyd+ cannot support encrypted or compressed protocols.
- *Scalability*: Honeyd+ can advertise multiple honeypot hosts that share a single underlying PLC, creating a large surface to attract attackers. The number of hosts is limited only by the IP address space and the performance of the PLC. The cost of deploying Honeyd+ is slightly more than the cost of a single PLC, making it an economical solution. However, according to the authors, Honeyd+ suffers performance drops in the presence of more than five simultaneous attacks.
- *Flexibility*: Honeyd+ can also be configured to emulate different types of PLCs by changing the search and replace terms. Other industrial control system components, such as human-machine interfaces, sensors and actuators, could be added.

The authors also present the results of functional and performance testing of Honeyd+, using two types of PLCs (Omron CP1L and Allen-Bradley L61) and two types of platforms (Raspberry Pi and laptop). The functional testing shows that Honeyd+ can successfully represent 75 authentic PLCs on both platforms, with five search terms each. The performance testing shows that Honeyd+ can handle multiple simultaneous connections with reasonable error rates for EtherNet/IP protocol, but not for HTTP protocol. Honeyd+ is a feasible and reduced-cost technique for deploying multiple physical honeypots of the same size, and that it has applications as a research or production honeypot.

Buza et al. [86]

Buza et al. proposed CryPLH [86], a medium interaction ICS honeypot simulating Siemens Simatic 300 PLC devices. It simulates the behavior of real PLCs without any consistent physical process simulation. CryPLH is implemented as a virtual machine that emulates a real PLC device. The virtual machine runs on a host computer and is connected to the network through a virtual network interface card (NIC). The virtual machine is configured to respond to Simple Network Management Protocol (SNMP) requests and other requests that are typical for PLC devices.

Morales et al. [112]

Morales et al. presented HoneyPLC [112], an extensible honeypot able to emulate a broad spectrum of PLCs models. All requests coming from attackers are handled by Honeyd, running a profile generated through the HoneyPLC Profiler Tool. The authors also added an S7comm server based on the Snap7 [51] project to accept connection and commands from the Siemens Step7 Manager software, they used PLCinject to inject a sample program into HoneyPLC and verify that the program is correctly saved in the honeypot file system. It is worth noting that no specification of the S7comm protocol has been officially released from Siemens, therefore all the information available has been inferred through reverse engineering techniques. The tool to create PLC profiles for different models of PLCs, helped the authors to create three profiles for Siemens S7-300, Siemens S7-1200, Allen-Bradley Micrologix 1100 and ABB PLCs. The profiles contain information about the web pages of the real PLCs. HoneyPLC was also tested with Nmap and PLCScan, two reconnaissance tools commonly used by attackers. The results show that HoneyPLC achieves a confidence level similar to those of the real PLCs, even Shodan Honeyscore identified the honeypot as a real system. The authors also evaluate the compatibility of HoneyPLC with Step7 Manager, a proprietary software by Siemens, showing that the honeypot can establish a stable connection and interact with Step7 Manager without errors. HoneyPLC was exposed to the Internet for a period of 5 months and recorded more than 5 GB of data. The data show that they received various S7comm functions, HTTP conversations, login attempts and SNMP requests.

Conti et al. [83]

ICSpot [83] is a Honeyd based honeypot that simulates a realistic and interactive physical process of a water tank. It integrates a physical process simulation using MiniCPS [76], a toolkit that uses Mininet [34] to emulate network communications and physical layer interactions in cyber-physical systems. It allows researchers to create, investigate, and exchange realistic and reproducible CPS network topologies, and to test attacks and defenses that are applicable to real systems. MiniCPS also supports software-defined networking and industrial protocols such as EtherNet/IP and Modbus/TCP.. ICSpot simulation is based on the IHS project [7], which mimics a simplified water treatment process. The physical process can be controlled and monitored by the attacker through the S7comm protocol, which allows reading and writing PLC memory blocks. ICSpot also provides a web interface (HMI) that shows the status and evolution of the physical process in real-time. ICSpot exposes various industrial services and protocols, such as HTTP, SNMP, Modbus, and S7comm. These services are implemented using different open-source tools, such as Honeyd, snap7, and SCADA Honeynet. The services are designed to emulate the behavior and features of a Siemens Simatic S7-300 PLC, one of the most widely used PLCs in ICSs. ICSpot also enables the capture and analysis of the programs injected by the attackers through the S7comm protocol. It records and stores all the interactions that occur with the honeypot in a log file. The log data can be imported into a MySQL database using Honeyd2MySQL, and then visualized using Honeyd-viz, a web interface that shows useful statistics and graphs about the collected data, such as the number of connections, the IP addresses of the attackers, and their origin.

3.2.2 ICS honeypots based on Conpot

Conpot [108] is an open-source low-interaction honeypot developed under the Honeynet Project [24], and is still being maintained nowadays. Conpot supports several industrial protocols including IEC 60870-5-104, Modbus, S7comm, EtherNet/IP, HTTP, FTP, and BACnet. It provides a user-friendly web-based interface for real-time monitoring of its activity, analysis of captured traffic, and the generation of alerts in response to suspicious behavior. Conpot is built upon the Twisted networking framework [56] and utilizes Python for the protocol handlers. The authors of ”Industrial Control Systems Honeypot: A Formal Analysis of Conpot” proposed a test of Conpot behaviour in the presence of deadlock and livelock states using Coloured Petri Net (CPN). The analysis uncovered that Conpot has the potential to induce a deadlock state when attackers reach specific points, impeding the generation of attack trails. Conpot can engage attackers in infinite loops, effectively ensnaring them within the honeypot.

These findings confirmed the effectiveness of Conpot in entrapping and engaging attackers, preventing them from infiltrating real-time active systems and services.

Zhao and Qin [146] improved Conpot honeypots by introducing additional Siemens S7comm protocol functions and a dynamic Human Machine Interface (HMI) in order to better evaluate threats. The authors state that their study improved the interaction level of Conpot and provided better support for the simulation of Siemens S7 class PLCs. During their extensive 43-day deployment, their honeypots interacted with traffic originating from 244 valid IP addresses from 34 different countries.

Abe et al. [70]

Abe et al. [70] proposed an ICS honeypot system that is based on both Honeyd and Conpot, it also adds a traceback capability, that performs a counter-scan to the source of scan in order to gain more information about attackers (OS, open ports, etc). The proposed system emulates ICS protocols and devices by using Conpot framework, and performs basic honeypot functions by means of Honeyd (webserver interfaces, FTP, TELNET, etc.). The authors implemented Nmap in the Honeyd to perform a reverse scan to the attackers and obtain useful information regarding the attack. The authors developed a data analysis system that collects and processes logs and payloads acquired from both the traceback system and the deception network system. They later assess the effectiveness of the deception network system through two distinct attack scenarios: Havex RAT, a malware targeting OPC servers in ICS networks, and Modbus Stager, an exploit that embeds malware in Modbus-enabled PLCs. The project demonstrates that the honeypot successfully detects and gathers information about these attacks while prolonging attackers' engagement with the honeypots.

Cao et al. [100]

Dipot is a distributed Conpot based honeypot system engineered to monitor and analyse Internet-based scanning and attack activities directed at industrial control systems (ICS). What sets DiPot apart from existing honeypot systems is its array of advanced functionalities, including attack clustering, visualization services, high-fidelity simulation, and in-depth data analysis. Dipot is based around three key components: Honeypot Node (HN), Data Processing Node (DPN): this element assumes the role of clustering and analyzing data gathered by HNs, utilizing algorithms such as k-center clustering to differentiate various attack types, and Management Node (MN). Over the course of six months, DiPot accumulated 317,484 access sequences and identified 4,827 suspicious IPs.

Lau et al. [130]

Xpot simulates Siemens S7-300 series PLCs in which the attacker may upload code in MC7 format under some constraints (the authors did reverse engineering of a set of MC7 instructions). To trick OS fingerprinting attempts,the proposed solution adopts a unique approach, it emulates the network stack associated with the Siemens S7-300 series PLCs. This strategy ensures that the honeypot remains indistinguishable from a genuine PLC when subjected to such fingerprinting techniques. Xpot supports the execution of PLC programs that adversaries may attempt to load onto the system, it interprets the bytecode until the compilation process is complete, leveraging the capabilities of LLVM [31] to ensure the accurate execution of these programs. During the exposure, the authors observed several full S7comm handshakes and queries. However, no significant suspicious activity was detected. This suggests that the honeypot has the potential to blend in effectively when deployed in real-world scenarios. However, the proposed honeypot is physics-less, and no physics feedback is provided to the attacker.

Pliatsios et al. [85]

Pliatsios et al. [85] present a low-interaction proof-of-concept honeypot, which is configured to emulate a hydro power plant's Saitel Remote Terminal Unit (RTU) device, facilitating interactive communication with the Human-Machine Interface (HMI) located at the plant's control center. To increase the level of emulation and thus, the realism of the honeypot, they simulate the interactions between the HMI and the Conpot honeypot based on the traffic data from the real RTU. Conpot uses the traffic data from the real RTU (by feeding pcap files from the real RTU), and the virtual HMI generates requests for the Conpot honeypot. Architecturally, the honeypot is rooted in the Conpot framework, offering support for various ICS communication protocols, and it boasts a modular design with components encompassing the ICS system module, simulation system, and a monitoring system. While it excels in emulating complex systems under a constant load, as stated by the authors, the honeypot's performance in diverse operational scenarios may exhibit variability.

Kuman et al. [107]

Kuman et al. [107] present a honeynet that is designed to be easy to use and customizable, allowing users to create honeypot networks that are tailored to their specific needs. In order to achieve this goal, the authors use IMUNES, a network simulator that allows users to define and simulate networks of almost arbitrary complexity on a single physical or virtual machine. The nodes of the emulated/simulated network are made of virtual nodes on Docker, which can be customized to emulate a wide range of devices and protocols. The Conpot honeypot is used to emulate PLCs in the simulated network, and the OSSEC tool is used to monitor all activities on the honeypots and alert the owner of the honeynet when something interesting is happening. The architecture of the honeynet system integrates three core components: IMUNES for network emulation, Conpot for ICS device simulation, and OSSEC for intrusion detection. IMUNES is utilized to create a virtual network topology that closely resembles a typical ICS environment, specifically emulating Siemens S7 PLCs. The Conpot honeypot is employed to simulate the ICS devices, including the emulation of a known vulnerability in the Siemens S7-300 PLCs to attract potential attackers. OSSEC HIDS is configured to monitor and alert on changes within the honeynet, providing real-time detection of intrusion attempts. The architecture is designed to be lightweight, using virtual nodes within IMUNES to minimize resource usage while maintaining a high degree of realism in the emulation. The Conpot honeypots are strategically configured to log all interactions, which are then monitored by OSSEC to detect any unauthorized access or attack patterns.

Ferretti et al. [90]

Ferretti et al. [90] deployed a set of Conpot-based honeypots, emulating the behavior of different types of ICS devices and ICS protocol. They improved the implementation of some of the ICS protocols supported by Conpot such as BACnet, EtherNet/IP, IEC-104, and Siemens S7. They also created accurate profiles of ICS devices based on data collected from real devices. Each honeypot instance was deployed in a dedicated Docker container and was configured with a different device profile and ICS protocol. All instances were deployed behind a remote endpoint and connected through it over a VPN. The honeypots attracted around 5000 connections but the majority of the connections were from public scanners such as Shodan.

3.2.3 Other relevant ICS honeypots

In addition to Conpot and Honeyd, there exists a diverse array of ICS honeypots and honeynets that contribute significantly to the literature of industrial control system honeypots. These noteworthy solutions stand out by addressing critical concerns, including the integration of physical awareness. This section introduces such ICS honeypots that offer unique capabilities, exploring their distinct approaches.

Vasilomanolakis et al. [137]

HosTaGe [137] is a low-interaction honeypot capable of generating signatures for Intrusion Detection Systems (IDSs) to recognise future similar attacks. Its architecture is centered around its ability to emulate various ICS-specific protocols, such as Modbus, S7, SNMP, HTTP, Telnet, SMB, and SMTP. The detection mechanism is defined using an Extended Finite State Machine (EFSM) [19], which allows for the transition between states based on specific conditions. This EFSM model enables HosTaGe to detect attacks by transitioning from a normal behavior state to an attack state upon protocol communication detection. The honeypot can identify multi-stage attacks by observing attacks originating from different protocols but the same host. One of the key limitations of HosTaGe is its low-interaction nature, which might limit the depth of interaction with attackers, potentially affecting the richness of collected data. Additionally, while it can generate signatures for detected attacks, these signatures are primarily useful for misuse analysis and may not be as effective for anomaly-based detection systems.

Litchfield et al. [111]

Litchfield et al. introduced HoneyPhy [111], which is a physic-aware honeypot designed to improve security in networked control systems. The architecture of HoneyPhy includes three main modules:

- Internet Interface(s) Module: This module manages the network interfaces, allowing for communication between the CPS and external networks.
- Process Model(s) Module: It simulates the physical processes of the CPS, using empirical data to create accurate models based on Newton's Law of Cooling.
- Device Model(s) Module: This module contains models of the physical devices interacting with the CPS, which can be created using either black box (empirical) or white box (theoretical) modeling techniques.

The configuration of the honeypot is managed through a XML file, which defines the contents, permissions, interfaces, controllable variables, and metadata for each module. The authors also discuss the importance of accurately modelling the physical processes and device interactions to avoid detection by attackers. They developed a proof-of-concept implementation which involves a heating system simulation to provide a realistic simulation for potential attackers.

Wilhoit et al. [104]

GasPot is a honeypot designed to simulate a gas-tank monitoring system. It is a Python script that logs connections and attempts at compromise, with each instance being unique to prevent attackers from easily identifying and fingerprinting it. GasPot supports six different commands, including those that provide tank information and status reports, and can simulate attack vectors observed in actual systems. Its architecture is straightforward, consisting of a single Python script that functions as a honeypot, logging connections and compromise attempts. It is unique in that each instance of GasPot is distinct, making it more challenging for attackers to fingerprint. The system logs activities locally on the device running it, without requiring additional services, thus maintaining the appearance of an authentic gas-tank monitoring system. The logs are timestamped in Coordinated Universal Time (UTC) to facilitate synchronization across multiple instances. It was deployed globally, with instances in the United States, Brazil, the United Kingdom, Jordan, Germany, the United Arab Emirates, and Russia.

Hilt et al. [99]

The honeypot created by Trend Micro Research was designed to mimic a small fictitious company whose business was to serve clients in critical industries but had inadequate security defenses. This setup was intended to attract cybercriminals and allow researchers to monitor and analyze the attacks. The infrastructure comprised a Raspberry Pi 3, USB Ethernet adapters, SharkTap Ethernet taps, and a large external drive. Ethernet taps were inserted at specific network points to capture data traffic. The honeypot also featured a realistic Human-Machine Interface (HMI) and used various tactics to lure threat actors, including posting information on Pastebin to attract attackers and creating a believable company backstory with employee names, working phone numbers, and email addresses. The system included various components such as industrial and consumer cellular routers, Omron PLCs, proxy routers, protocol gateways, a Dell Precision M4800 for the Human-Machine Interface (HMI), and virtual machines (VMs) for different services and controllers like Siemens S7-1200 and Allen-Bradley MicroLogix 1100. The honeypot aimed to mimic a realistic factory setting. The honeypot was successful in attracting several attacks, including a malicious cryptocurrency mining campaign, two ransomware attacks, and various scanning attempts.

Antonioli et al. [84]

Antonioli et al. proposed a high-interaction, server-based ICS honeypot that uses the MiniCPS [76] framework to simulate a water treatment testbed. Its architecture consists of the following components:

- Vulnerable VPN endpoint: A device that runs an OpenConnect VPN server with weak credentials, allowing the attacker to access the internal network.
- Vulnerable gateway device: A device that runs ssh and telnet servers with weak credentials, allowing the attacker to get a command shell on the device.
- Network emulation: A virtual network that reproduces the same topology, addresses, and link characteristics as the real ICS network, using Mininet [34] and Linux network namespaces.

- Physical process and devices simulation: A collection of python scripts that simulate the hydraulic part of the water treatment process, the control logic of four PLCs, and an HMI, using MiniCPS API and Ethernet/IP protocol.
- Data collection: A subsystem that logs the attacker's activities, such as keystrokes, network traffic, and device commands, using a keylogger, tcpdump, and MiniCPS API.

A preliminary evaluation of the honeypot was done in the context of a Capture-The-Flag (CTF) competition. The competition was a part of a broader ICS security event called SWaT Security Showdown (S3) [75], hosted by Singapore University of Technology and Design (SUTD) in July 2016. The honeypot was also evaluated using qualitative metrics that include:

- Physical layer interaction
- Network layer interaction
- Data collection
- Realistic system configuration
- Realistic system behavior
- Attack detection and recording

According to the authors, the proposed ICS honeypot provides all of the features listed, showing that it is effective in detecting and recording attacks while also providing a realistic representation of an ICS system.

Murillo et al. [69]

The honeynet proposed is a virtualized environment that mimics a large electrical substation network, with realistic devices, protocols, and traffic. The honeynet uses Mininet [34], a lightweight virtualization tool, to create a network topology with multiple nodes that represent intelligent electronic devices (IEDs) in a substation. Each node runs a traffic mirroring service that redirects the packets to a separate machine running SoftGrid [97], a software platform for IED emulation. PowerWorld power system simulation: The honeynet also uses PowerWorld, a commercial software for power system analysis, to simulate the physical behavior of the electrical grid. PowerWorld communicates with SoftGrid through a TCP/IP interface, and provides feedback to the IEDs based on the network state and control actions. Attack scenarios and data collection: The honeynet implements two attack scenarios: a denial-of-service attack that disrupts the communication between IEDs, and a false data injection attack that alters the measurements and control commands of the IEDs. The honeynet collects network traffic data, power system data, and attacker behavior data for further analysis and evaluation.

Bernieri et al. [93]

MimePot [93] aims to detect complex cyber-physical attacks in industrial networks by simulating the physical processes in order to mislead attacks and study their behavior. The authors chose a model-based (mathematical and computational) approach to simulate the physical process. They use Software Defined Networking (SDN) technology to provide:

- Traffic Redirection: SDN allows for dynamic redirection of network traffic. Malicious traffic can be sent to the honeypot, while legitimate traffic continues to the real network.
- Network Address Camouflaging: SDN can mask the network addresses of real devices, making it appear to the attacker that they are targeting the actual control systems when they are actually interacting with the honeypot.
- Granular Control: SDN provides a high level of detail in traffic management. It can apply specific rules to manage how traffic is handled, making the honeypot more effective at engaging attackers.
- Scalability and Flexibility: SDN's centralized control plane makes it easier to scale and adapt the network architecture as needed.

MimePot has the capability to capture behaviors of complex cyber-physical attacks, such as Zero Dynamics Attacks [134].This kind of attack refer to the behavior of a system's internal states when its output is artificially constrained to zero, which can be exploited in sophisticated attacks to alter the system's state without changing the observable outputs, thereby evading detection. The architecture of MimePot comprises two main modules: Mime Plant that is responsible for simulating physical plant processes and can be considered as a PLC interacting with physical plants and Mime E&C (Estimation and Control), designed for control routines computation, similarly to a SCADA workstation. MimePot's physical components simulate a subset of physical processes using Linear Time Invariant (LTI) models. The cyber components are implemented in virtualized environments where Mime Plant and Mime E&C communicate using industrial protocols.

Navarro et al. [124]

iHoney [124] honeypot replicates the operations of a water treatment plant. It integrates a concealed monitoring infrastructure that remains nearly imperceptible to attackers. This covert surveillance employs both Network Intrusion Detection Systems (NIDS) and Host-based Intrusion Detection Systems (HIDS) to scrutinize activities, amassing invaluable intelligence in the process. The core of iHoney is composed of a SCADA server/HMI, a control network featuring PLCs, and various industrial communication protocols. In real-time, the simulation system evaluates the status variables of the physical processes, dynamically interacting with ICS inputs and generating corresponding outputs. This ensures a coherent and convincing simulation. To effectively capture and analyze network traffic, iHoney deploys TAP devices as in [99] for passive transmission to a monitoring system. This surveillance infrastructure incorporates SNORT NIDS and protocol dissecting agents for Modbus and S7Comm. Additionally, a custom-built HIDS is deployed on the SCADA server to enhance threat detection capabilities. iHoney garnered immediate attention upon exposure to the Internet, with continuous and regular attacks highlighting its potential to allure attackers. For the most part, attacks targeted the IT components of the SCADA system and were characterized by automation, underscoring the persistent threat posed by automated cyber-attacks.

3.3 Desiderata and limitations in ICS Honeypots

From our previous exploration of related work 3.2 and a thorough review of the existing literature, we have identified a set of desiderata for the development of an effective ICS honeypot. In this section, we will outline and discuss these key requirements that guide the design and implementation of honeypots tailored to industrial control systems (ICS). These desiderata are important in order to make sure that the honeypot can convincingly mimic a real system, attracting potential attackers while providing valuable insights into their tactics and strategies.

3.3.1 Level of interaction

Honeypots and honeynets are typically categorized based on the level of interaction they offer to attackers. Low-interaction honeypots replicate basic services with limited functions, while high-interaction ones accurately mimic real devices, enabling comprehensive data collection about an attacker's actions. For an ICS honeypot to be effective, it must emulate an industrial network connecting multiple PLCs, potentially supervised via HMI interfaces, and facilitate observable, accessible network traffic involving PLCs and HMIs. Therefore, an ICS honeypot should not only provide precise fingerprints of the involved devices and ICS networks, as seen in low-level interaction ICS honeypots, but also grant attackers the ability to interact with the honeypot extensively. This includes inspecting and modifying PLC registers, uploading malicious PLC code, examining and exploiting HMI interfaces, effectively gaining full control over the OT network. Furthermore, as highlighted in studies like [111] and [106], physics-awareness is a vital component in creating convincing and deceptive ICS honeypots. This means attackers should receive consistent feedback from a possibly simulated but manipulated physical process, enhancing the honeypot's authenticity and effectiveness.

3.3.2 Configurability

The honeypot should provide the flexibility to modify the attack surface exposed to attackers, enabling it to adapt to the ever-evolving exploit tools and techniques employed by attackers. This adaptability is essential to suit the specific ICS network it is intended to protect. Thus, the honeypot should have the capability to support various industrial network protocols, depending on the operational context. Additionally, it should be extensible, allowing for the emulation of PLCs from different manufacturers and models, ensuring comprehensive coverage.

3.3.3 Scalability

To emulate real-world ICSs, the honeypot must have the capability to scale effectively, accommodating middle-sized ICS environments with hundreds of diverse PLCs and HMIs without compromising performance. This scalability is crucial to accurately emulate real-world ICS networks and to ensure that the honeypot can handle the complexities of such systems. Utilizing virtual resources, rather than physical devices, is a fundamental prerequisite for achieving scalability in ICS honeypots. However, it is important to note that the mere adoption of virtual resources is not sufficient, practical testing is essential to assess the honeypot's response time at varying numbers of PLCs and HMIs, thereby validating its true scalability.

3.3.4 Entry Point

A robust ICS honeypot should be designed to withstand attacks that target the availability and integrity of the target system. When an attacker gains access to the honeypot via the Internet, they may attempt to manipulate the exposed PLCs and HMI interfaces, potentially after a brute-force attack on their authentication. Conversely, if the attacker manages to compromise the VPN underpinning the honeypot, they can use ARP poisoning to perform a Man-in-the-Middle (MITM) attack, intercepting network traffic on the supervisory control network and launching such attacks between various PLCs or between a PLC and its associated HMI.

3.3.5 Attack Monitoring

The honeypot should be capable of gathering extensive data on the behavior of the attackers. This data should include network interaction logs to detect scanning attacks as well as logs of system events that capture the methods attackers employ when interacting with the OS of the targeted ICS devices. Furthermore,

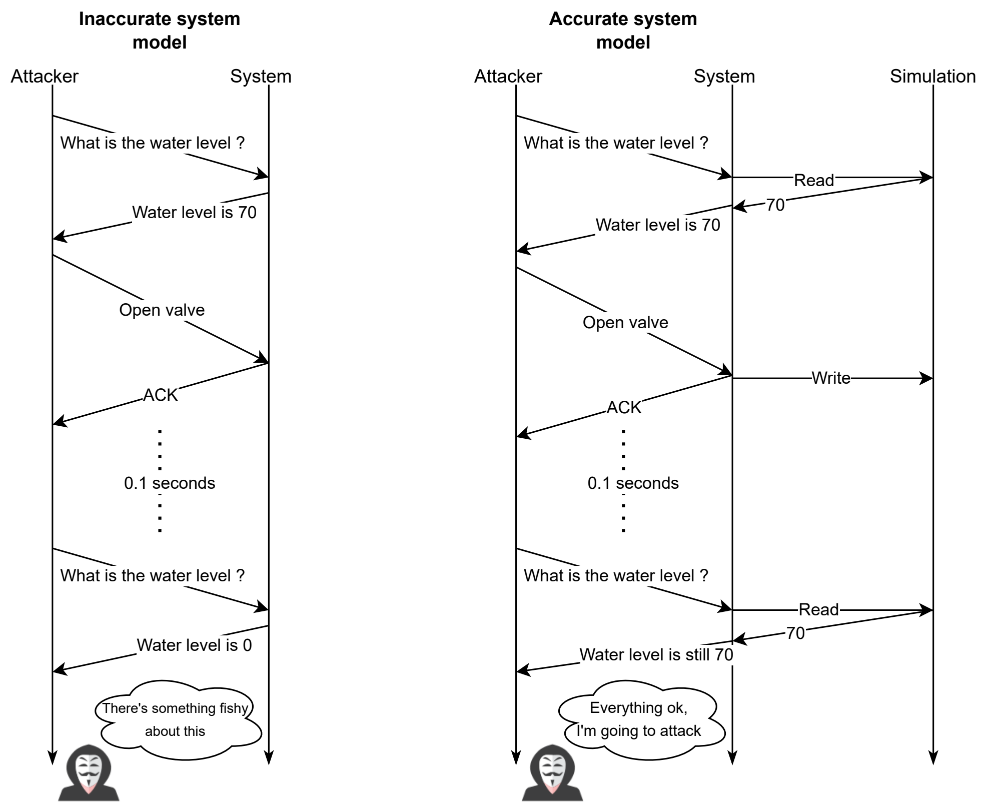


Figure 3.3: Attacker outcomes on a well-modeled and on a poorly modeled system

the honeypot should support real-time data analysis and the visualization of attack-related information as it unfolds. It should also support retrospective data analysis of the logs to identify attack patterns. These patterns can be utilized, for example, to generate attack signatures for configuring network and host intrusion detection systems.

3.3.6 Limitations of current ICS honeypots

In Table 3.1, we present a comprehensive comparison of related works, which will serve as a basis for evaluating our own honeypot, HoneyICS. We will introduce and discuss HoneyICS in Chapter 4, aligning its features with the desiderata outlined in this section. This comparative analysis will help in assessing the effectiveness of our ICS honeypot.

Level of interaction.

Current approaches mostly provide limited functionality when it comes to TCP/IP stack simulations, as well as native ICS network protocols. This poses serious limitations in the actions an attacker can perform within the honeypot and, thus, in the understanding of adversarial interactions and malware. Most of the reviewed honeypots and honeynets support only some of the network protocols and services specific to ICSs. This requires an attacker capable of accessing the *supervisory control network*, possibly protected via VPN. In this respect, Antonioli et al. [84] provide the possibility to build up a communication network between PLCs and/or HMIs, while HoneyPhy [111] only propose an ideal architecture where such communication is possible. As a consequence, only Antonioli et al. [84] support non-trivial MITM attacks between PLCs and/or HMIs; more limited forms of MITM attacks, between PLCs and their plant, can be simulated in [69,85,93,111]. In the previous section, we have seen a number of ICS honeypots providing different levels of emulation/abstraction of the underlying physical processes. However, all of them, except for HoneyPhy [111], Gaspot [104], iHoney [124] and MimePot [93], fail to perform an accurate emulation of the underlying physical process. Among the reviewed honeypots, only [70,85,93,137] explicitly support some form of register manipulation, and only [84–86,111] explicitly support some form of HMI manipulation. As regards *physics-awareness*, only the works in [69, 84, 93, 111] provide some form of simulation of the underlying physical industrial processes. As shown in Figure 3.3, it's important to note that a simulation should be realistic, as any inconsistencies could potentially confuse the attacker. On the left, the scenario portrays an attacker interacting with a CPS honeypot that doesn't simulate process behaviors and device delays. The absence of these delays and deviations from expected process behavior serves as an alert to the attacker. In the right scenario, an attacker interacts with a CPS honeypot capable of modeling both process behavior and device delay, resulting in realistic responses, leading the attacker to perform an attack. Moreover, only HoneyPLC [112] is able to simulate the upload of malicious user programs, although the injected code is only stored by the honeypot but not executed. While capturing the code is a first important step to support PLC malware analysis, the execution of the injected code together with consistent physical feedback is crucial to deceive the attacker.

Configurability.

All honeypots discussed in Section 3.2, except for [70, 108, 112], have limited extensibility, as they can only mimic one or two PLC models and provide the associated fingerprints when scanned using tools like Nmap. Likewise, the majority of existing research works, with, of course, some notable exceptions like [100,108,137,143], tend to provide support for only a limited subset of Industrial Control System (ICS) network protocols. This constraint poses a significant limitation, particularly in the face of sophisticated attacks such as Stuxnet [89], which may specifically target diverse types of Programmable Logic Controllers (PLCs). Consequently, this limitation has far-reaching implications, significantly restricting the scope of ICS networks these honeypots can authentically emulate and, consequently, the diverse contexts in which they can be effectively deployed.

Scalability.

The adoption of virtual resources over physical devices is an essential condition for achieving scalability. However, it is imperative to recognize that the mere utilization of virtualization does not universally guarantee scalability; rather, its efficacy is bound to the specific implementation approach employed. While many of the reviewed ICS honeypots incorporate scalable designs through the adoption of virtual resources and lightweight virtualization techniques like Docker containers [118], only Honeyd+ [143] stands out by explicitly evaluating the proposed honeypot's scalability, particularly in terms of the number of supported virtual PLCs.

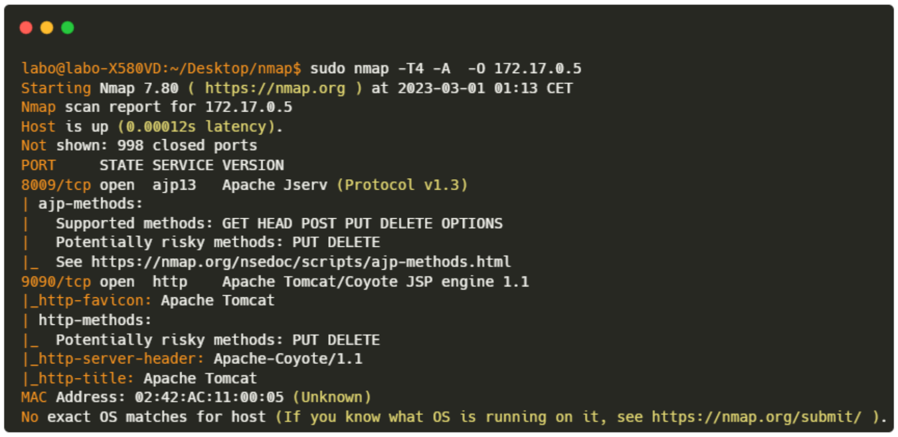
Attacker Entry Point.

Our review of the literature reveals that existing honeypots have followed two predominant approaches: they have either been openly exposed on the Internet [70, 86, 100, 108, 135, 137, 143], or they have been protected through a VPN, as demonstrated by [69, 84, 85, 93]. The decision to expose a honeypot on the Internet offers attackers a more accessible target but constrains the range of interactions they can initiate with it, as discussed in Section 3.3. Conversely, employing a VPN not only provides valuable insights into potential attacker strategies for compromising ICS networks but may also act as a deterrent by introducing an additional layer of defense. We advocate for the implementation of honeypots that support both entry points, thus enabling a broader spectrum of adversarial interactions. In this regard, HoneyPhy [111] stands out as the only honeypot designed to accommodate both entry modes.

By addressing these limitations, future research can pave the way for the development of next-generation ICS honeypots that offer a more comprehensive and realistic deception environment for attackers.

Unveiling some of the issues of Honeyd

It is worth noting that Honeyd has not been updated since 2013, lacking any official support from the original authors or DataSoft, although the company could have integrated Honeyd into its product *NOVA* and continued the development privately. This lack of updates and support raises questions about



```
labo@labo-X580VD:~/Desktop/nmap$ sudo nmap -T4 -A  -O 172.17.0.5
Starting Nmap 7.80 ( https://nmap.org ) at 2023-03-01 01:13 CET
Nmap scan report for 172.17.0.5
Host is up (0.00012s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE VERSION
8009/tcp open  ajp13   Apache Jserv (Protocol v1.3)
| ajp-methods:
|   Supported methods: GET HEAD POST PUT DELETE OPTIONS
|   Potentially risky methods: PUT DELETE
|_  See https://nmap.org/nsedoc/scripts/ajp-methods.html
9090/tcp open  http    Apache Tomcat/Coyote JSP engine 1.1
|_http-favicon: Apache Tomcat
| http-methods:
|_  Potentially risky methods: PUT DELETE
|_http-server-header: Apache-Coyote/1.1
|_http-title: Apache Tomcat
MAC Address: 02:42:AC:11:00:05 (Unknown)
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
```

Figure 3.4: Nmap scan of a Honeyd honeypot

the software's compatibility with modern operating systems, particularly newer versions of Ubuntu and various Linux distributions. Users have reported issues, such as difficulties obtaining DHCP for emulated hosts, which can hinder its functionality. Additionally, Honeyd relies on outdated fingerprint files that no longer align with the latest nmap algorithms [29]. This discrepancy in fingerprint data can impact the accuracy and realism of the honeypots it creates, since it is relying on this file to send appropriate TCP responses during an nmap scan. Therefore this functionality may not always work seamlessly, as new versions of nmap are continually released. The effectiveness of Honeyd's responses, as well as its reliance on nmap.prints, can vary depending on the specific nmap version used for scanning as shown in Figure 3.4. To address this, a practical strategy is to open ports commonly associated with specific device types. For instance, devices like Linux and Solaris typically have port 22 open, whereas routers and switches are likely to have port 161 open for SNMP.

3.4 Legal implications of operating honeypots

As stated in [133], operating honeypots entails a range of legal and ethical considerations that are important in ensuring that honeypot deployment complies with the law, respects privacy, and adheres to ethical principles. Here, we address key legal and ethical aspects associated with honeypot operations.

- Legal Right to Monitor: It is important to acknowledge that ownership of a computer network does not automatically confer a legal entitlement to monitor all activities of system users. The field of monitoring activities is complex and is subject to diverse legal constraints, encompassing statutes, privacy regulations, employment policies, terms-of-service agreements, and contractual obligations. Deviation from these legal parameters may potentially entail legal repercussions, encompassing civil liability and the prospect of criminal sanctions.
- Risk of Misuse: Neglected or unmonitored honeypots carry the risk of being exploited by malicious actors for unlawful activities. These activities can include storing stolen credit card data or distributing prohibited content. Honeypot operators have the responsibility of actively monitoring and controlling their honeypots to prevent inadvertent involvement in unlawful undertakings.
- Entrapment Concerns: Within the honeypot community, the concept of entrapment is occasionally contemplated. Entrapment, in criminal defense law, is a situation where the government induces an individual to commit a crime they would not have otherwise committed. Private honeypot operators are less likely to encounter entrapment concerns, as this law primarily applies to government actions.
- Fourth Amendment: Honeypot operators associated with government agencies must be aware of potential Fourth Amendment constraints on their monitoring activities. The Fourth Amendment restricts government agents from conducting searches or seizing evidence without a proper warrant. Evidence obtained in violation of this constitutional provision can be legally challenged.
- Patriot Act: The USA Patriot Act features a ”computer trespasser” exception, permitting government agencies to conduct warrantless monitoring of hackers under specific circumstances. This exception typically applies when the network owner has authorized interception, and the monitoring aligns with a lawful investigation.
- Consent to Monitoring: It can be argued that hackers accessing a system with a clear banner indicating monitoring consent have implicitly provided consent to be monitored. However, this implied consent may not extend to cases where hackers access the system through ports that do not offer a banner.
- Moral and Ethical Considerations: In addition to complying with legal requirements, honeypot operators must address moral and ethical concerns. These ethical considerations include the potential invasion of privacy, the responsibility to prevent honeypots from being exploited for malicious purposes, and the ethical use of collected information to enhance security rather than for malicious ends.

- Europe regulations: The GDPR aims to protect personal data and privacy of individuals within the European Union. According to Recital 30 of the GDPR [44], IP addresses are considered personal data if they can be used to identify an individual. This includes scenarios where IP addresses are logged by honeypots, as these logs could potentially link back to specific individuals, especially if combined with other data. Under GDPR, the processing of personal data must adhere to principles of lawfulness, fairness, and transparency. There must be a clear legal basis for data processing, such as consent, necessity for contract performance, compliance with legal obligations, protection of vital interests, tasks carried out in the public interest, or legitimate interests pursued by the data controller. Setting up a honeypot and logging malicious attempts could fall under the "legitimate interests" clause, as it is crucial for ensuring network and information security. This includes preventing unauthorized access and cyberattacks, which aligns with the GDPR's provisions for protecting public security and preventing criminal activities. However, publishing the collected IP addresses is more contentious. Even if the collection is justified under legitimate interests, disseminating the data publicly without further analysis might not meet GDPR requirements. Factors such as the necessity of publication, its effectiveness in achieving security goals, and the potential adverse impacts on individuals whose IP addresses are published must be carefully weighed.

Understanding these legal and ethical dimensions is fundamental for a responsible and effective honeypot deployment, ensuring that it serves its intended purpose while respecting legal and ethical boundaries.

Table 3.1: Comparison with other ICS honeypots

✗= Not supported ◒= Partially supported ✓= Fully supported

Honeypot	**Level of Interaction**					**Configurability**		**Honeypot Entry Point**	**Attack monitoring**	
	ICS Network simulation	**Physics aware**	**PLC registers**	**HMI**	**MITM**	**ICS protocols**	**Extens.**		**Data Collection**	**Data Analysis**
SCADA Honeynet [135]	◒	✗	✗	✗	✗	Modbus	◒	Internet	–	–
Conpot [108]	◒	✗	✗	✗	✗	Modbus, S7comm, BACnet, EtherNet/IP	◒	Internet	network traffic	–
Dipot [100]	◒	✗	✗	✗	✗	Modbus, S7comm, BACnet	◒	Internet	network traffic	real-time, posteriori
Kuman et al. [107]	◒	✗	✗	✗	✗	Modbus, S7comm	◒	Internet	network traffic	–
Ferretti et al. [90]	◒	✗	✗	✗	◒	Modbus, S7comm, BACnet, EtherNet/IP	◒	VPN	network traffic	posteriori
HosTaGe [137]	◒	✗	◒	✗	✗	Modbus, S7comm	✗	Internet	network traffic	posteriori
Pliatsios et al. [85]	◒	✗	◒	✓	◒	Modbus	◒	VPN	network traffic	–
Honeyd+ [143]	◒	✗	✗	✗	✗	EtherNet/IP	◒	Internet	network traffic	–
Abe et al. [70]	◒	✗	◒	✗	✗	Modbus, S7comm, BACNet	◒	Internet	network traffic	–
CryPLH [86]	◒	✗	✗	◒	✗	S7comm	✗	Internet	network traffic	–
HoneyPhy [111]	◒	✓	✗	◒	◒	DNP3	✗	Internet, VPN	–	–
GasPot [104]	✗	✗	✗	✗	✗	–	✗	Internet	network traffic	–
Hilt et al. [99]	✓	✓	✓	✓	✓	–	✗	Internet	network traf., system, malware	posteriori
ICSpot [83]	◒	✓	✓	✓	✗	Modbus, S7comm	◒	Internet	network traffic	posteriori
Antonioli et al. [84]	✓	✓	✗	✓	✓	EtherNet/IP	✗	VPN	network traf., system, malware	–
Murillo et al. [69]	◒	✓	✗	✗	◒	EtherNet/IP	✗	VPN	–	–
MimePot [93]	◒	✓	✓	✗	◒	Modbus	✗	VPN	–	–
iHoney [124]	◒	✓	✓	✓	◒	Modbus, S7comm	◒	VPN	network traffic	real-time
HoneyPLC [112]	◒	✗	✗	✗	✗	S7comm	✓	Internet	network traf., ladder logic capture	–
HoneyICS	✓	✓	✓	✓	✓	Modbus, DNP3	✓	Internet, VPN	network traf., system, malware	real-time, posteriori

Chapter 4

A PHYSICS-AWARE AND HIGH-INTERACTION ICS HONEYNET

4.1 A hybrid architecture

Our honeynet architecture consists of various components, ensuring the fulfillment of the features outlined in Section 3.3. HoneyICS is capable to emulate to key components within Operational Technology (OT) network, including Programmable Logic Controllers (PLCs), Human-Machine Interfaces (HMIs), communication networks, and even a simulated physical plant. Implementation flexibility is a key feature of our project, as each component can be implemented either using actual physical devices or emulation and simulation software.

As illustrated in Figure 4.4, our honeynet's architecture seamlessly integrates these diverse components. As shown, the inbound connections are able to contact directly the HMIs and PLCs, while the field communication network and the plant are not accessible from the outside. Through the supervisory control network, PLCs communicate with each other and with the HMIs using Industrial Network Protocols such as Modbus. "The probes, responsible for collecting network and system data, are deployed on every device or network directly accessible to the attacker. It's worth noting that the honeynet configuration allows accessibility through a compromised Virtual Private Network (VPN) or by exposing specific devices such as PLCs or HMIs directly to the Internet. In the former scenario, an attacker gains comprehensive control over the supervisory control network interconnecting PLCs and HMIs, albeit without direct access to the field communications network linking PLCs to sensors and actuators. Conversely, in the latter case, the attacker's capabilities hinge on the vulnerabilities specific to the exposed physical devices (refer to, for instance, the vulnerability in the HMI interface detailed in our subsequent implementation section 5.3).

Lastly, the entire honeynet framework is orchestrated and overseen through a dedicated management dashboard, providing a centralized interface for efficient control and monitoring. This comprehensive architecture ensures a versatile and robust platform for the emulation and analysis of potential cyber threats in OT environments.

In summary, our honeynet architecture aligns with the Purdue Enterprise Reference Architecture (PERA) up to the second level. On Level 0 we have the plant with sensor producing values based on the physical process, that are then sent to level 1 where the PLCs interpret them and their logic sends back commands to the actuators on level 0. On level 2 we have the HMI overseeing the operation of the PLCs.

4.1.1 Discussion on honeynet architectural design

Before arriving at the design depicted in Figure 4.4, many iterations have been made, each representing a valuable exploration of different architectural possibilities, considerations, and lessons learned. The evolution of HoneyPLC's architecture is summarized in Table 4.1.

First iteration

The first attempt is illustrated in Figure 4.1. The *frontend responder* functioned similarly to a load balancer, directing traffic either to the PLC scan responder, designed to respond to nmap OS fingerprinting attempts, or to the software PLC. If the query was directed to the HMIs, the *frontend responder* would allow the query to pass through directly. Thus, the front-end responder's role also involved unpacking traffic to determine whether the query directed to the PLC was a scan or a legitimate request. Having a *frontend responder* meant we could switch the PLCs as we wanted adhering to our *configurability* requirement (cf. Section 3.3.2). However, it constrained us to a single entry point for the attacker (cf. Section 3.3.4). The option of using a VPN was ruled out since the attacker could never be in the same network as the PLCs. Without the *frontend responder* managing requests, the honeynet would not function properly. Allowing the attacker into the same network as the PLCs posed a risk of them discovering additional elements, such as the *scan responder*, which were not intended to be revealed. Moreover, the *frontend responder* would not align with our *scalability* requirement (cf. Section 3.3.3), as it would present a significant bottleneck with the increasing number of PLCs and HMIs.

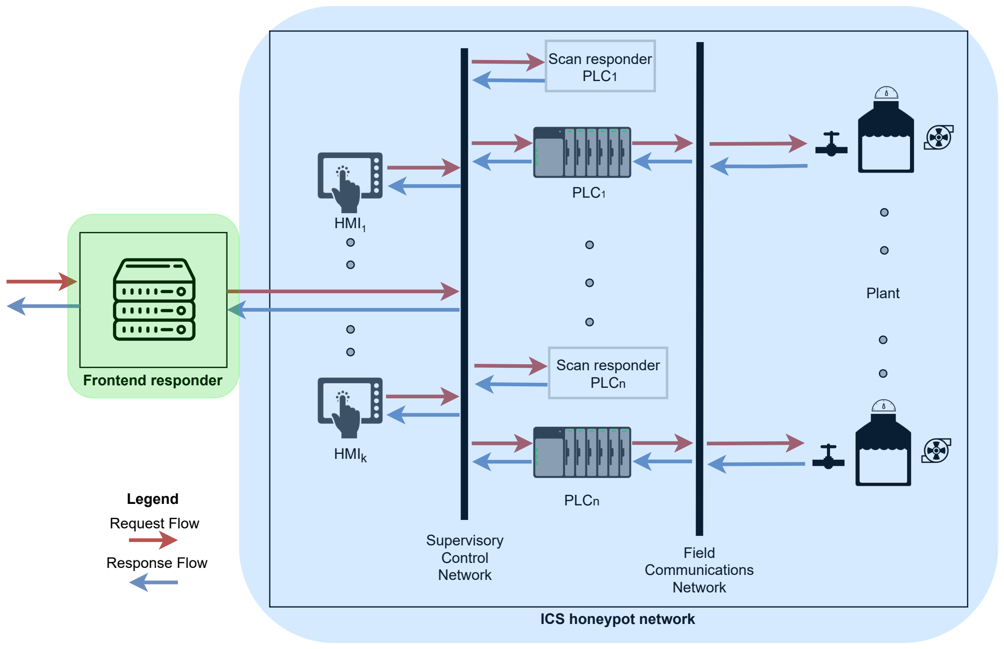


Figure 4.1: HoneyICS First architecture

Second iteration

Wanting to address also the *scalability* requirement we thought of the architecture illustrated in Figure 4.2. In this design, we replaced the *frontend responder* with a block of code within each container, incorporating the software PLC, the Scan responder, and a smaller handler. This handler efficiently redirected queries to either the Scan responder or the PLC based on the nature of the request. This adjustment significantly enhanced scalability by resolving the bottleneck issue associated with the *frontend responder*. While it might seem like a compromise on the *configurability* requirement, the consolidated scan responder and software PLC proved to be more straightforward to configure and manage. Moreover, this setup retained the flexibility to be replaced by a real PLC at any time.

Third iteration

Subsequently, we introduced a "Broker Container" to allow communication between software PLCs, a capability not initially supported and between the PLCs and the plant. While this addition was a step in the right direction for enhancing the *configurability* of the honeynet, it also introduced a new bottleneck, thereby somewhat reducing the scalability potential of HoneyICS. Additionally, we implemented a management dashboard capable of displaying collected logs, addressing the *attack monitoring* requirement.

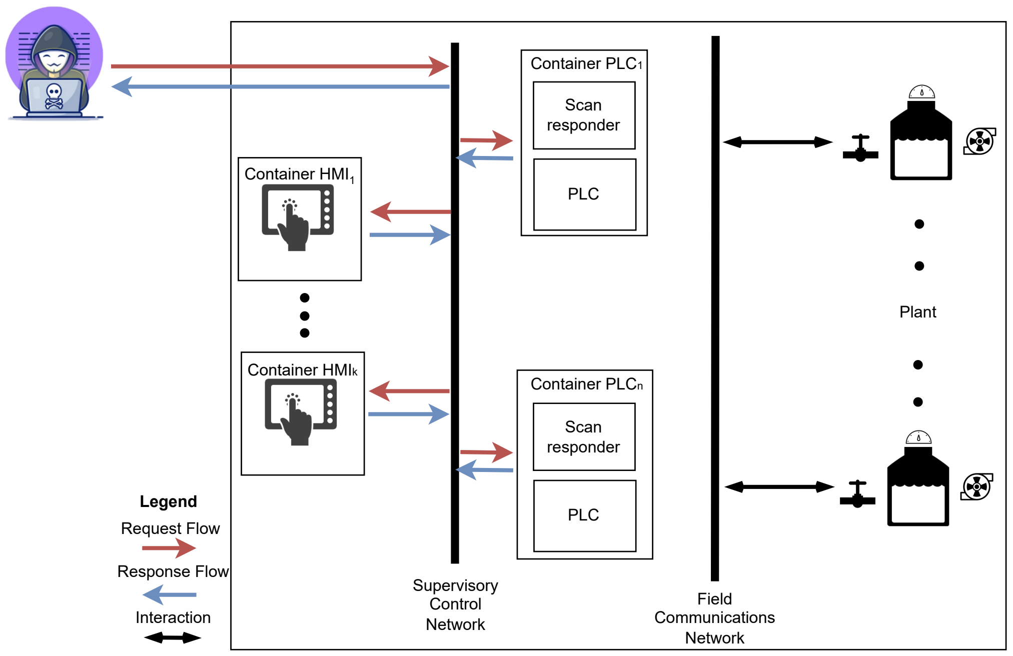


Figure 4.2: HoneyICS second architecture

Final iteration

The final iteration of the design of the architecture is the latest HoneyICS version (Figure 4.4). To tackle the scalability concern, we resolved it by embedding the communication code for the PLCs within their respective containers. Additionally, for communication with the plant, we containerized the plant and directly inserted the code into the plant container, further improving the scalability of the whole honeynet. By adding a firewall, we also manage to seamlessly integrate VPN access into our honeynet. Furthermore, to facilitate the seamless integration of new physical or virtual devices, therefore further addressing the *Configurability* desiderata, we crafted a reverse proxy designed to shield each device effectively. This not only gives the possibility of adding new devices but also enhances the security and thwarts potential lateral movements by attackers.

4.2 Attacker model

As illustrated in Fig. 4.4, our honeynet offers two primary entry points:

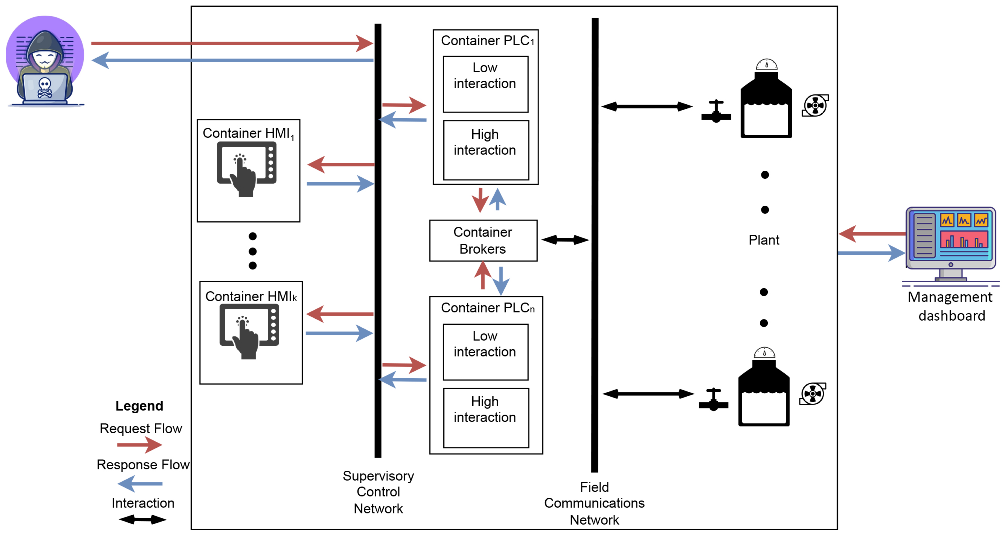


Figure 4.3: HoneyICS third architecture

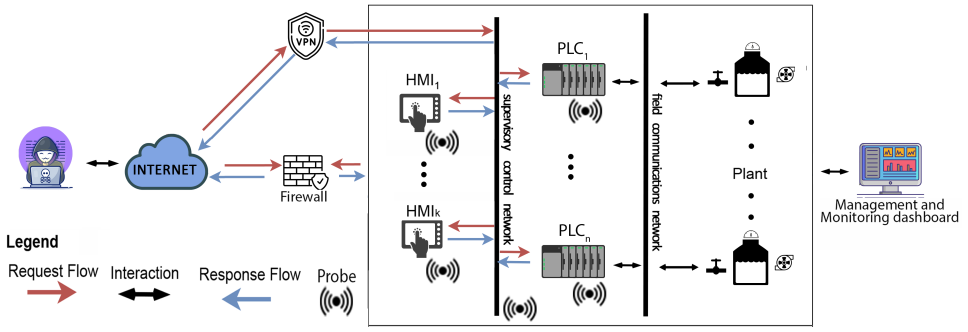

VPN
INTERNET
Firewall
HMI₁
HMIk
PLC₁
PLCn
supervisory control network
field communications network
Plant
Management and Monitoring dashboard
Legend
Request Flow
Interaction
Response Flow
Probe

Figure 4.4: HoneyICS Architecture

Table 4.1: HoneyICS evolution

HoneyICS version	Level of interaction	Configurability	Scalability	Honeypot entry point	Attack monitoring
First 4.1	●	◐	○	○	○
Second 4.2	●	◐	◐	○	○
Third 4.3	●	●	○	○	●
Final 4.4	●	●	●	●	●

- *Direct internet access to specific services*: Attackers can discover our honeypot by searching for internet-facing PLC services on common search engines or platforms like Shodan [49]. These services might include:

  - Modbus/TCP on port 502 (TCP). This communication protocol itself is not inherently secure, and attackers on the same network segment could potentially leverage common Modbus functions to manipulate connected devices.
  - Static HTML information pages of the PLCs on port 80 (TCP).
  - Web interface of the HMI on port 9090 (TCP). We specifically introduced a vulnerability within this HMI web interface, leveraging the CVE-2021-26828 ScadaBR Remote Code Execution flaw. This vulnerability allows attackers to upload and execute malicious code if they can obtain valid user credentials.
- *VPN access*: By gaining access to the VPN the honeypot is connected to, attackers can infiltrate the supervisory control network connecting PLCs and HMIs. This grants them more extensive control compared to internet access, although they still wouldn't have direct access to the field communication networks.

We attempted to introduce a third entry point that bridged the gap between direct internet access and VPN access. This involved incorporating a vulnerability into an internet-facing device, allowing attackers to pivot into the network and achieve a level of control similar to a VPN compromise. This scenario will be explored further in Section 7.1.

In both cases after gaining access, the intruder can use tools like Nmap [46] to fingerprint the targeted PLCs, extracting basic system information, including the PLC model and brand, open and filtered ports, the services operating on those ports, and the supported industrial protocol. Subsequently, the attacker may proceed to read and/or write PLC memory registers and attempt to upload and execute a malicious PLC user program. The effects of program execution on the physical process can be observed by inspecting the values of the PLC registers or through a compromised HMI. Similarly, another possible attack in both entry points is to fingerprint and then exploit weak or default credentials on HMI interfaces through brute-force attempts. This could potentially allow attackers to tamper with the physical state of the system by issuing commands directly through the compromised HMI interfaces.

Furthermore, if the attacker successfully compromises the VPN under which the honeypot operates, they gain the capability to: (i) conduct network traffic sniffing on the supervisory control network, and (ii) establish Man-in-the-Middle (MITM) attacks between two PLCs or between a PLC and its associated HMI. In the latter scenario, the attacker may achieve a dual objective: firstly, manipulating PLC registers (e.g., those storing actuator commands or sensor measurements) to compromise the physical process, and secondly, transmitting false measurements to the corresponding HMI. These manipulated measurements may originate from previous recordings made during an eavesdrop phase on the genuine system. This replay of data allows the attacker to remain stealthy, leveraging previously captured information to maintain a semblance of normalcy in their activities as happened in the notorious Stuxnet attack [89]. Lastly, it is worth noting that by gaining access to the network, the attacker can read the MAC address of the PLC. Therefore, it becomes crucial to align the personality of the PLC with the corresponding MAC address, as each manufacturer possesses its unique identifier.

4.3 Honeypot design novelties

This section discusses the key novelties of our HoneyICS design, focusing on its physics-aware and high-interaction capabilities. These features address critical limitations in existing ICS honeypots and enable a more realistic deception environment for attackers.

Physics-Aware Design:

Traditional honeypots often lack a realistic representation of the underlying physical process that the ICS system controls. This discrepancy can raise suspicion among attackers. HoneyICS incorporates a simulated physical plant, allowing for:

- Feedback loop: The honeypot mimics the behavior of a real system by providing feedback from the simulated plant to the PLCs. This feedback loop influences the PLC's outputs based on the process dynamics, creating a more realistic response to attacker actions.
- Attack impact visualization: By simulating the physical process, HoneyICS allows defenders to observe the potential consequences of an attack on real-world parameters like temperature, pressure, or flow rates. This facilitates a deeper understanding of attacker intent and potential damage.

High-Interaction Design:

Many honeypots limit the level of interaction attackers can have with the system. HoneyICS prioritizes high interaction, enabling attackers to perform actions that closely resemble real-world attacks on ICS environments. This includes:

- Extensive memory access: Attackers can interact with a broad range of PLC memory registers, allowing them to read, write, and modify process control values. This level of access reflects the potential for manipulating critical control parameters within a real ICS system.

- HMI exploitation: The honeypot incorporates a vulnerable HMI interface, mirroring potential weaknesses in real-world HMI systems. Attackers can exploit this vulnerability to gain unauthorized access and potentially manipulate the human operator's view of the process.
- VPN access: HoneyICS offers a VPN entry point, enabling attackers to compromise the network and conduct activities like sniffing traffic or launching Man-in-the-Middle attacks between PLCs and HMIs. This feature provides a more comprehensive attack surface, reflecting the growing use of VPNs in industrial control systems.

Chapter 5

PROTOTYPE IMPLEMENTATION

5.1 Case study: a water treatment system

In this section, we outline the case study used for our project, presenting a model of a water treatment plant structured into three distinct stages [109]. The control and monitoring of this system are overseen by three PLCs. It's essential to note that this model serves as a simplified representation inspired by the Secure Water Treatment system (SWaT) [117].

The Secure Water Treatment (SWaT) system, as originally presented by Mathur et al. [117], was intended to be a sophisticated testbed designed for the study and training in security aspects of industrial control systems, with a specific focus on water treatment processes. It serves as a platform to understand the impact of cyber and physical attacks on water treatment systems, assess the effectiveness of attack detection algorithms, and explore the cascading effects of failures in interconnected ICS. The SWaT system incorporates a six-stage water treatment process, each autonomously controlled by a dedicated Programmable Logic Controller (PLC). In terms of communication, SWaT employs local field-bus communications between sensors, actuators, and PLCs through both wired and wireless channels. The network architecture is based on a conventional Ethernet star topology, integrating all process stages, Human-Machine Interface (HMI), SCADA system, and historian through an industrial switch. Key communication protocols include EtherNet/IP (ENIP) and Common Industrial Protocol (CIP). The control structure of SWaT is characterized by distributed control, where each stage of the process is independently managed by its PLCs, each having a primary and a backup. These PLCs are networked for seamless communication and data sharing. Operators have the ability to manually control all actuators via the HMI and the SCADA system. The integration of sensors and actuators is a critical aspect, with various sensors providing real-time data

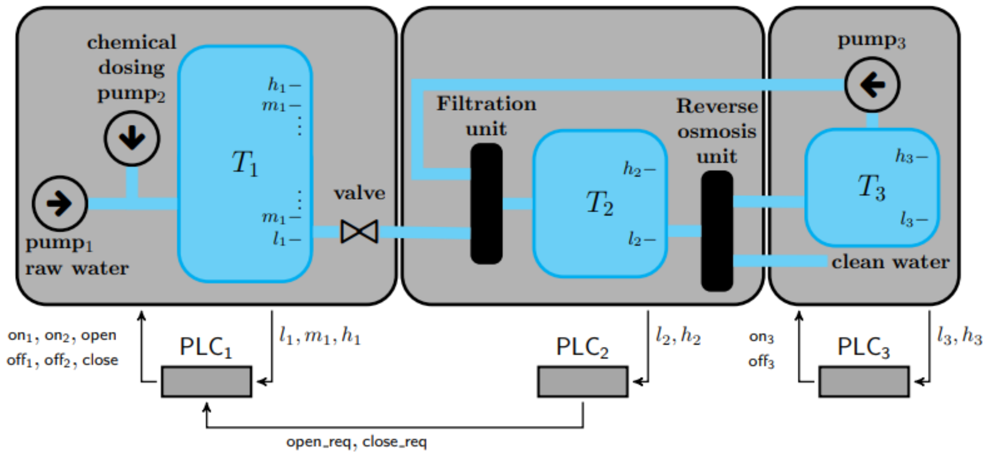


Figure 5.1: Lanotte et al. Simplified SWaT System: Three-Stage Water Treatment Plant with PLC and HMI Control

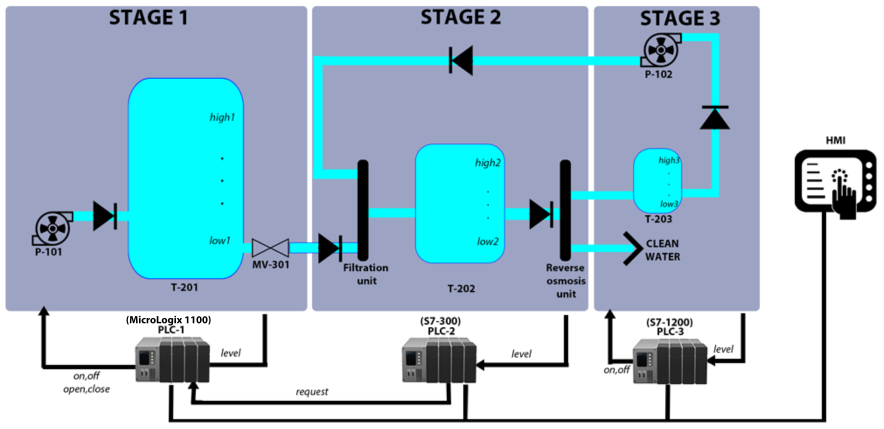


Figure 5.2: Our simplified SWaT System: Three-Stage Water Treatment Plant with PLC and HMI Control

to PLCs for controlling actuators like pumps and valves, using an Ethernet-based ring topology with Device Level Ring (DLR) protocol. The six stages of SWaT involve different processes such as inflow control, chemical dosing, ultrafiltration, reverse osmosis, and membrane cleaning. Each stage is equipped with specific sensors and actuators for monitoring and control purposes.

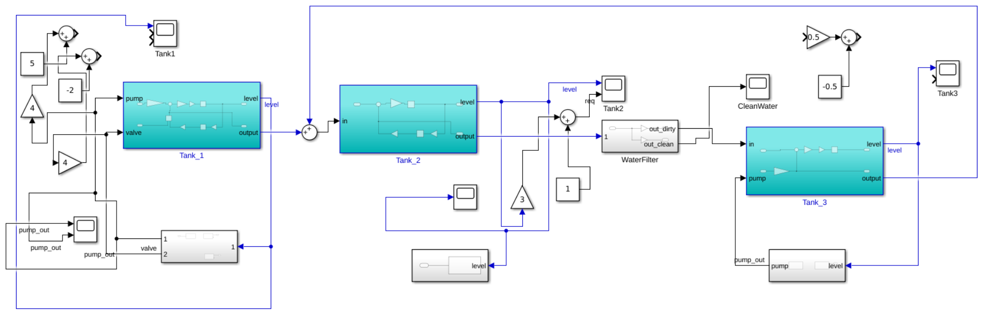


Figure 5.3: Simulink model

Lanotte et al. [109] proposed a scaled-down version of the original SWaT system, shown in Figure 5.1. Unlike the original SWaT's six-stage process, the simplified system narrows its scope to just three stages. These stages primarily involve the chemical dosing pump and raw water pump that let the water flow into a tank *T1*, followed by filtration and reverse osmosis processes. In the simplified system the design assumes specific flow dynamics: the first stage involves chemically dosing and pumping raw water into a tank (T1) using two pumps. The second stage includes a filtration unit that releases treated water into a second tank (T2). The third stage involves a reverse osmosis unit to reduce inorganic impurities. The treated water is either distributed as clean water or stored in a backwash tank (T3) and pumped back to the filtration unit. The authors assume that the flow of incoming water in tank T1 is greater than the outgoing flow through the valve, and tank T2 is large enough to receive the entire content of tank T3 at any moment.

We maintained the core structure of the simplified SWaT system proposed by Lanotte et al., introducing subtle modifications to enhance its functionality. As illustrated in Figure 5.2, we simplified the system by reducing the number of pumps for the incoming water in tank *T-201* to a single pump (*P-101*) for clarity. Additionally, we improved system efficiency by adjusting the *reverse osmosis unit* to raise the cleanliness threshold, resulting in a larger volume of contaminated water to accelerate the filling of *T-203*. In terms of communication, we refined the system to employ an industrial protocol, as elaborated in Section **??**. Alongside these modifications, we introduced an HMI to monitor the evolution of the physical process, through the registers of the PLCs.

5.1.1 Plant

In Figure 5.3, the model used in the Simulink container's is presented. Beginning from the top left, the sub-model *Tank_1* manages the first tank. In Figure 5.4, a detailed view of the sub-model is provided. As shown in Figure 5.5, water

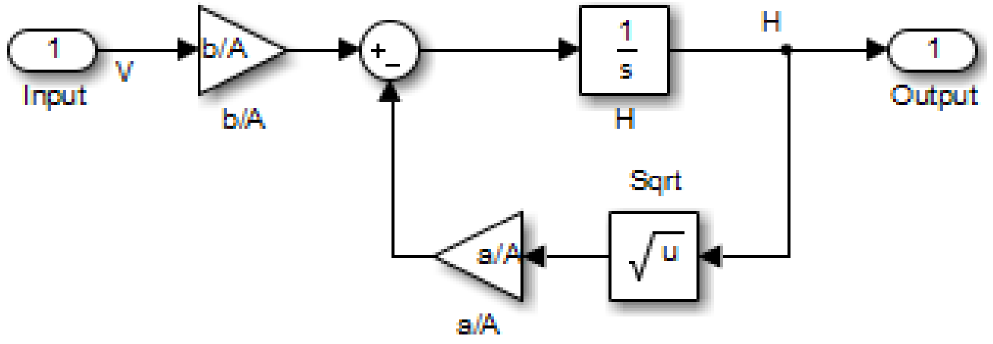


Figure 5.4: Simulink sub-model system: Tank_1

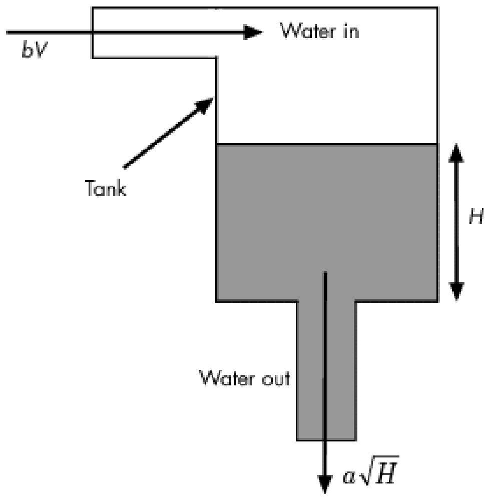
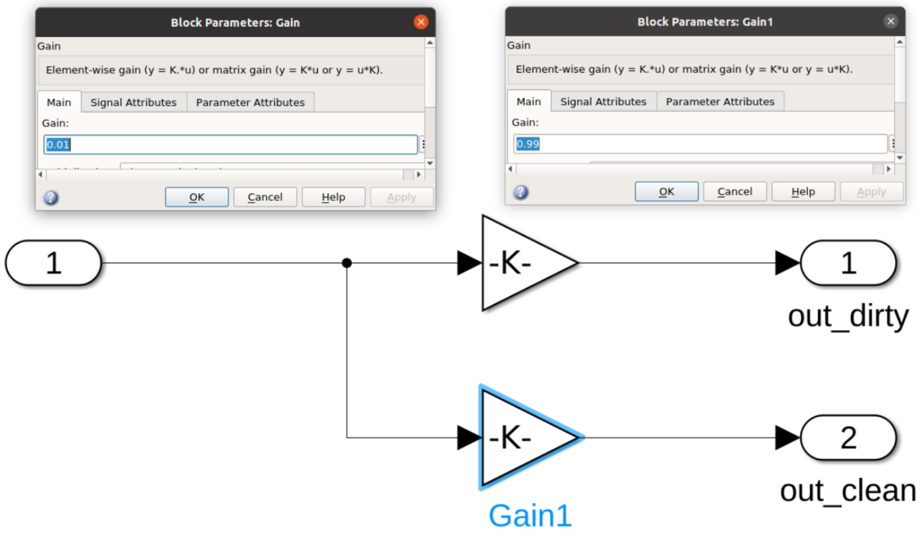


Figure 5.5: Simulink sub-model tank

enters the tank at a rate correlated to the voltage V supplied to the pump. In the PLC logic implementation, we restrict the pump to operate in either an on or off state, therefore the voltage values are 0 or 1. The water exits from the base and flows to the next tank as input, and the outflow rate is associated with the square root of the water height H, this introduces a nonlinear element, making the whole system nonlinear. The other variables shown in the picture include H for water height and Vol for volume, while system parameters are the tank's cross-sectional area A, and constants b and a for inflow and outflow rates. The system dynamics are expressed through the differential equation: $\frac{d}{dt}\text{Vol} = A\frac{dH}{dt} = bV - a\sqrt{H}$, with water height H as both the state and output, and


Block Parameters: Gain

Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Gain: 0.01

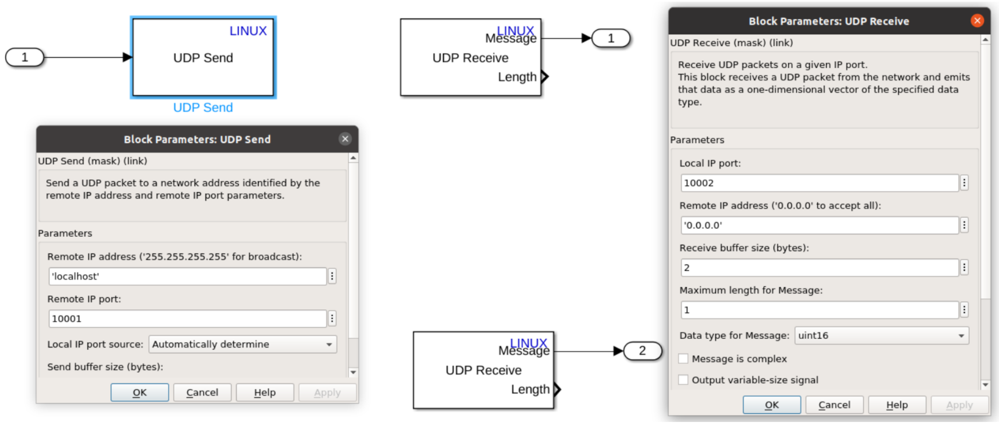
Block Parameters: Gain1

Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Gain: 0.99

Figure 5.6: Reverse osmosis filter


Block Parameters: UDP Send

UDP Send (mask) (link)

Send a UDP packet to a network address identified by the remote IP address and remote IP port parameters.

Parameters

Remote IP address ('255.255.255.255' for broadcast): 'localhost'

Remote IP port: 10001

Local IP port source: Automatically determine

Send buffer size (bytes):

Block Parameters: UDP Receive

UDP Receive (mask) (link)

Receive UDP packets on a given IP port.
This block receives a UDP packet from the network and emits that data as a one-dimensional vector of the specified data type.

Parameters

Local IP port: 10002

Remote IP address ('0.0.0.0' to accept all): '0.0.0.0'

Receive buffer size (bytes): 2

Maximum length for Message: 1

Data type for Message: uint16

Message is complex

Output variable-size signal

Figure 5.7: UDP receiver and sender

voltage V as the input. Tank_2 and Tank_3 are using the same exact equation as Tank_1 and also the same sub-model.

The reverse osmosis filter, positioned at the output of Tank 2, is depicted in Figure 5.6. The water level outputted from Tank 2 is categorized as 99% clean water and 1% dirty water. The 1% dirty water is reintroduced into the loop through Tank 3. When the water level of dirty water in Tank 3 reaches 10, the PLC logic activates the pump linked to the *Tank_3* sub-model, initiating the transfer of water into Tank 2.

The sub-model *Tank_1* takes two input signals through a UDP receiver block: one for the voltage level (on/off) to activate the pump and the other for the valve's voltage value (on/off). It produces an unsigned integer representing the current water level inside the tank. This water level value is sent to the subsequent tank and is also transmitted through a UDP sender block, as illustrated in Figure 5.7.

Tank_2 has a single output representing the water level, as it lacks any linked actuators.

Regarding *Tank_3*, it takes the voltage level (on/off) for pump activation as input and produces the water level as an output.

5.1.2 Controllers

Each PLC is assigned to manage one of the three stages within the simplified water treatment plant. The code dedicated to the PLC overseeing the initial stage, which involves controlling the pump, valve, and level sensor, is provided in Listing 5.1 and the flow chart of the code is illustrated in Figure 5.8. This code, written in Structured Text (ST), is designed to regulate the pump and valve based on the real-time level of the tank.

The variables: *level*, *requestValve*, *pumpStatus*, and *valveStatus* are declared to represent the tank level, requests for valve operations, pump status, and valve status, respectively. Additionally, internal boolean variables such as *isLow* and *isHigh* are introduced to determine if the tank level is below the specified threshold of 40 or above 80. Internal requests for valve operations, denoted by *openRequest* and *closeRequest*, are controlled based on external requests coming from PLC2 and written into the *requestValve* register. Conditions are then evaluated, for instance, the status of the pump is adjusted based on the tank level, ensuring the pump is activated when needed. Similarly, the valve status is updated according to internal requests, external requests, and the tank level. The CONFIGURATION section at the end of the code plays a crucial role in setting up the PLC. It specifies the presence of a resource named Res0 that executes a task named task0 at regular intervals. This task, in turn, ensures the execution of the PLC1 program every 20 milliseconds.

The code running on PLC2 is presented in Listing 5.2 and its logic is illustrated in Figure 5.9. The program controls a pump based on the water level. The level variable represents the current water level. The initially defined variables include the water level (*level*), the request to alter the valve's status to PLC1 (*valveRequest*), the request to open the valve (*valveOpenRequest*), and the request to close the valve (*valveCloseRequest*). The constants *LOW_LEVEL* and *HIGH_LEVEL* denote the thresholds for low and high water levels. The *valveOpenRequest* is activated when the water level is at or below the low level threshold, while the *valveCloseRequest* is triggered when the water level reaches

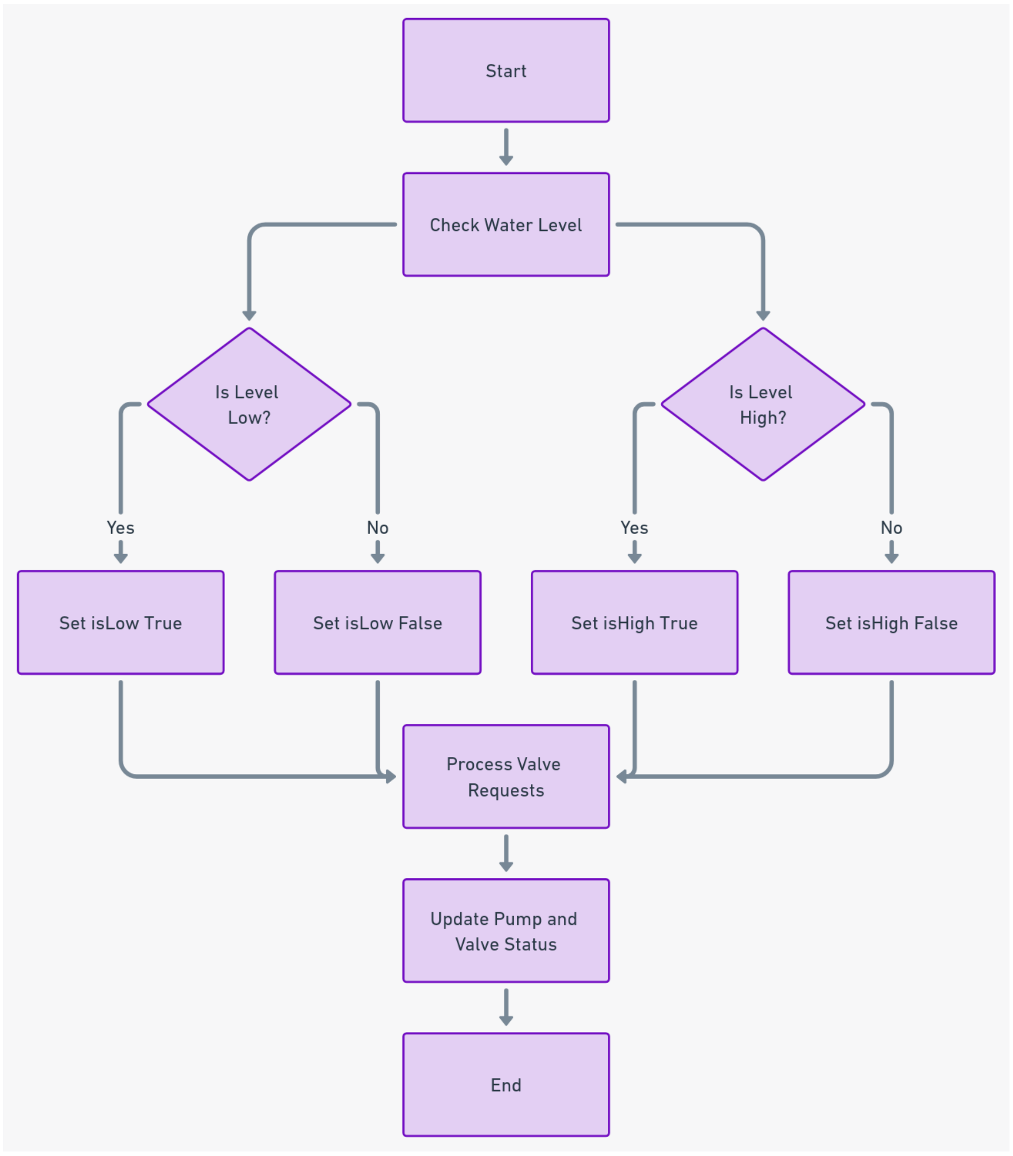


Figure 5.8: PLC1 logic flow chart

or exceeds the high level threshold. The *pumpRequest* is sent to PLC1 when the valve is not in the process of closing, and either the valve is opening or there is a request for valve operation.

The code running on PLC3 is provided in Listing 5.3 and its logic is illustrated in Figure 5.10. The purpose of this code is to regulate the operation of the pump *P-102* based on the water level of tank *T-203* within a specified range. The

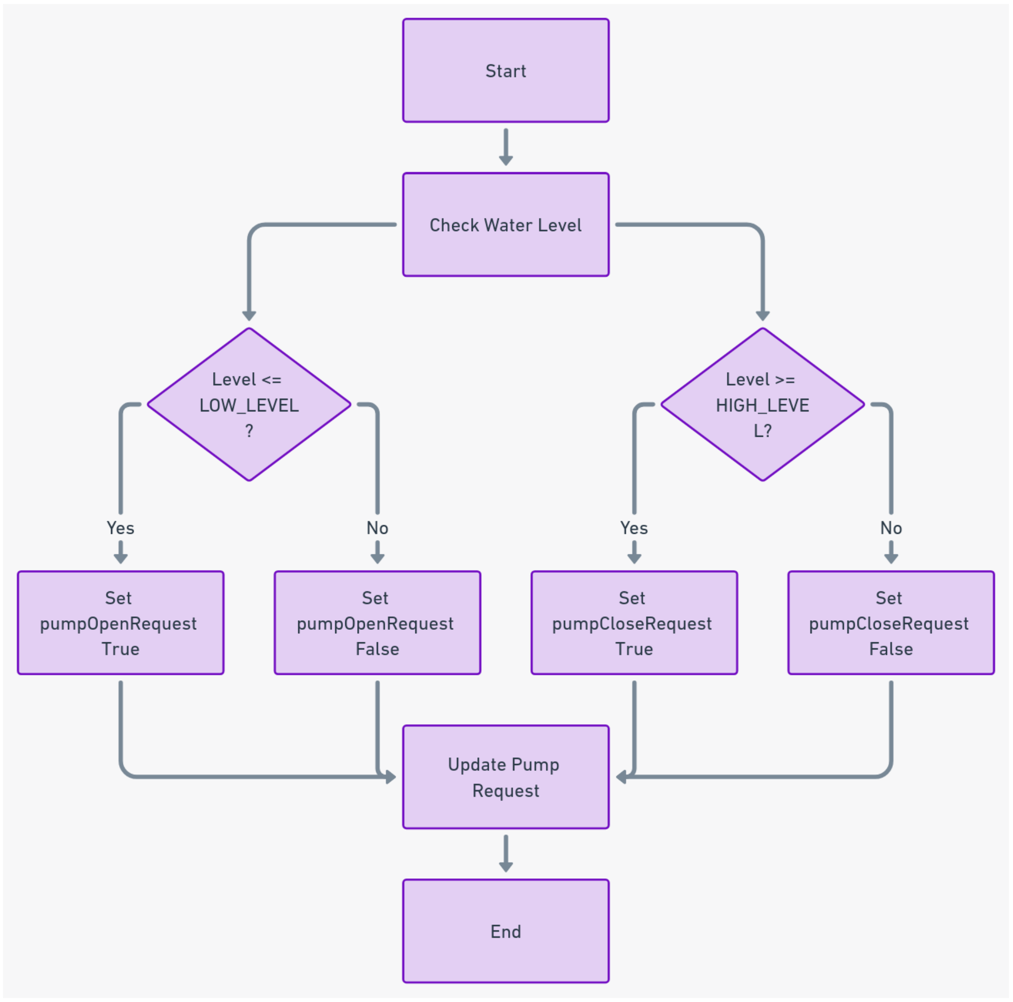


Figure 5.9: PLC2 logic flow chart

variable *level* stores the current water level, while the boolean variables *isLow* and *isHigh* are utilized to determine whether the water level falls below 0 or rises above 10 which are the predefined thresholds. These thresholds are established by the constants *LOW_LEVEL* and *HIGH_LEVEL*. If the water level is less than or equal to LOW_LEVEL, the variable isLow is set to true. Conversely, if the water level is greater than or equal to HIGH_LEVEL, isHigh is set to true.

The status of the pump is determined by the line of code: *pump := NOT(isLow) AND (isHigh OR pump);*. This implies that the pump will be activated if the water level is not deemed low and is considered high, or if the pump is already in an active state. In summary, the pump will be activated when the water

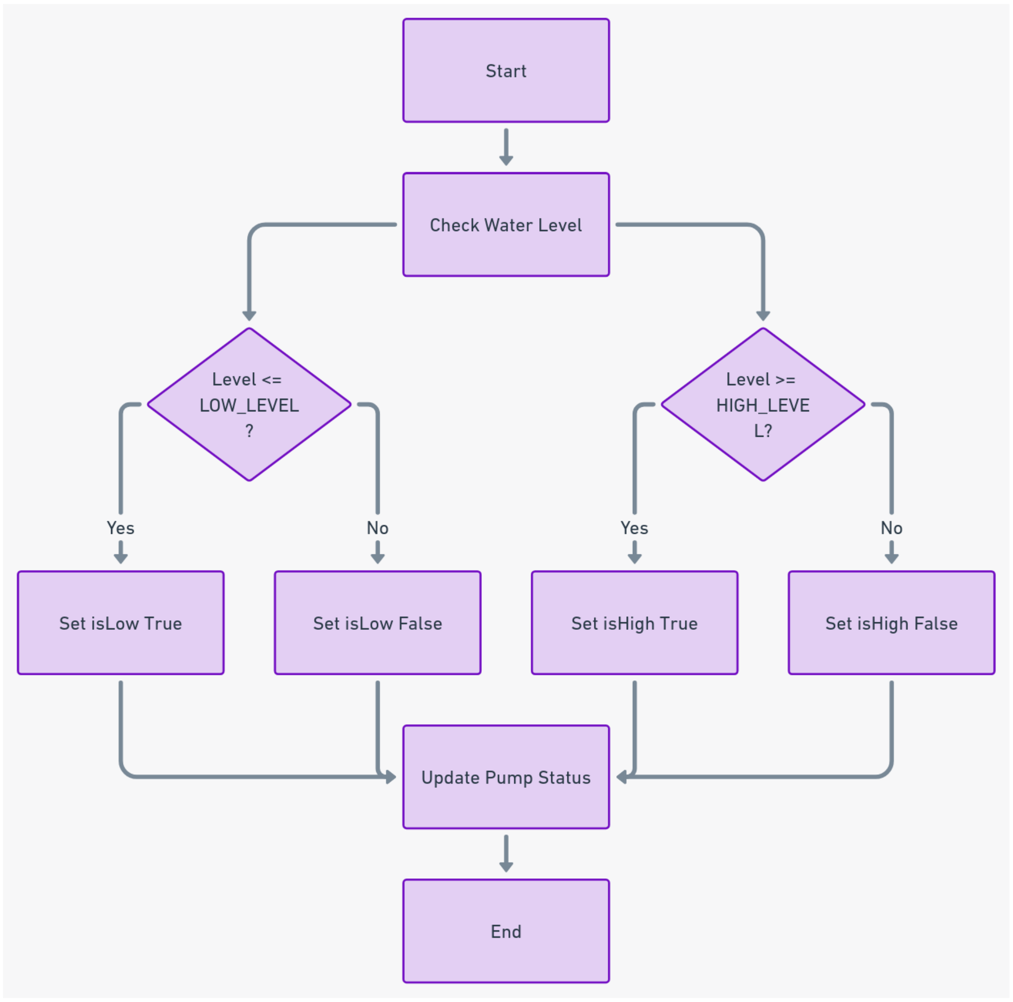


Figure 5.10: PLC3 logic flow chart

level surpasses the high threshold and will remain active until the water level decreases to the low threshold. If the water level resides within the specified range, the pump will maintain its existing state.

Listing 5.1: PLC1 Structured Text code

```
PROGRAM PLC1
  VAR
    level AT %IW0 : INT; // Water level
    requestValve AT %QX0.2 : BOOL; // Request to open/close the valve
    pumpStatus AT %QX0.0 : BOOL; // Current status of the pump
    valveStatus AT %QX0.1 : BOOL; // Current status of the valve
    isLow AT %MX0.0 : BOOL; // Flag to indicate if level is low
    isHigh AT %MX0.1 : BOOL; // Flag to indicate if level is high
    openRequest AT %MX0.3 : BOOL; // Internal request to open the valve
    closeRequest AT %MX0.4 : BOOL; // Internal request to close the valve
    LOW_LEVEL : INT := 40; // Low level threshold
    HIGH_LEVEL : INT := 80; // High level threshold
  END_VAR

  // Check if level is low or high
  isLow := LE(level, LOW_LEVEL);
  isHigh := GE(level, HIGH_LEVEL);

  // Set open and close requests based on valve request
  openRequest := requestValve;
  closeRequest := NOT(requestValve);

  // Update pump status: turn off if level is high, turn on if level is low or
      keep current status
  pumpStatus := NOT(isHigh) AND (isLow OR pumpStatus);

  // Update valve status: close if close request is true, open if open request
      is true and level is not low, or keep current status
  valveStatus := NOT(closeRequest) AND (openRequest AND NOT(isLow) OR
      valveStatus);
END_PROGRAM

CONFIGURATION Config0
  RESOURCE Res0 ON PLC
    TASK task0(INTERVAL := T#20ms,PRIORITY := 0); // Task to run the program
    PROGRAM instance0 WITH task0 : PLC1; // Assign the program to the task
  END_RESOURCE
END_CONFIGURATION
```

Listing 5.2: PLC2 Structured Text code

```
PROGRAM PLC2
  VAR
    level AT %IW0 : INT; // Water level
    pumpRequest AT %QX0.0 : BOOL; // Pump request status
    pumpOpenRequest AT %MX0.0 : BOOL; // Pump open request status
    pumpCloseRequest AT %MX0.1 : BOOL; // Pump close request status
    LOW_LEVEL AT %MW1 : INT := 10; // Low water level threshold
    HIGH_LEVEL AT %MW2 : INT := 20; // High water level threshold
  END_VAR

  // Set pump open request if water level is less than or equal to low level
  pumpOpenRequest := LE(level, LOW_LEVEL);

  // Set pump close request if water level is greater than or equal to high
      level
  pumpCloseRequest := GE(level, HIGH_LEVEL);
  // Set pump request if pump close request is not set and either pump open
      request is set or pump request is already set
  pumpRequest := NOT(pumpCloseRequest) AND (pumpOpenRequest OR pumpRequest);
END_PROGRAM

CONFIGURATION Config0
  RESOURCE Res0 ON PLC
    TASK task0(INTERVAL := T#20ms,PRIORITY := 0); // Task running at 20ms
        interval
    PROGRAM instance0 WITH task0 : PLC2; // Assigning PLC2 program to the task
  END_RESOURCE
END_CONFIGURATION
```

Listing 5.3: PLC3 Structured Text code

```
PROGRAM PLC3
  VAR
    level AT %IW0 : INT; // Water level
    pump AT %QX0.0 : BOOL; // Pump status
    isLow : BOOL; // Is water level low?
    isHigh AT %QX0.1 : BOOL; // Is water level high?
    LOW_LEVEL : INT := 0; // Low water level threshold
    HIGH_LEVEL : INT := 10; // High water level threshold
  END_VAR

  // Check if water level is lower than low level threshold
  isLow := LE(level, LOW_LEVEL);
  // Check if water level is higher than high level threshold
  isHigh := GE(level, HIGH_LEVEL);
  // Pump should be on if water level is not low and (water level is high or
      pump is already on)
  pump := NOT(isLow) AND (isHigh OR pump);
END_PROGRAM

CONFIGURATION Config0
  RESOURCE Res0 ON PLC
    TASK task0(INTERVAL := T#20ms,PRIORITY := 0); // Task running every 20ms
    PROGRAM instance0 WITH task0 : PLC3; // Assign PLC3 program to task0
  END_RESOURCE
END_CONFIGURATION
```

Table 5.1 provides a comprehensive overview of the registers employed in our use case, detailing their specific functions. The table summarizes the mapping between physical devices (including sensors and actuators) and their corresponding PLC registers, along with the range of values for each register.

Table 5.1: Intended use of PLC registers in our use case

PLC	physical device/variable	Range	PLC register
PLC1	level sensor of tank T-201	[40,80]	`InputRegisters_IW0`
	actuator of pump P-101	{0,1}	`Coils_QX00`
	actuator of valve MV-301	{0,1}	`Coils_QX01`
	request from PLC2	{0,1}	`Coils_QX02`
	low setpoint of tank T-201	40	`MemoryRegisters_MW0`
	high setpoint of tank T-201	80	`MemoryRegisters_MW1`
PLC2	level sensor of tank T-202	[10,20]	`InputRegisters_IW0`
	request to PLC1	{0,1}	`Coils_QX00`
	low setpoint of tank T-202	10	`MemoryRegisters_MW0`
	high setpoint of tank T-202	20	`MemoryRegisters_MW1`
PLC3	level sensor of tank T-203	[0,10]	`InputRegisters_IW0`
	actuator of pump P-102	{0,1}	`Coils_QX00`
	low setpoint of tank T-203	0	`MemoryRegisters_MW0`
	high setpoint of tank T-203	10	`MemoryRegisters_MW1`

5.2 Honeypot Technology Stack

This section provides a summary of the core technologies that underpin our HoneyICS design. These components were carefully chosen to create a robust, scalable, and realistic deception environment for attackers.

- *Docker and Containerization*: Docker serves as the foundation for our containerized deployment approach. Each element of the honeynet leverages a dedicated Docker container, offering modularity, scalability, and efficient resource utilization. This allows for easy deployment across various environments and simplifies maintenance.
- *Nginx and Iptables*: Nginx functions as a web server, reverse proxy server, and load balancer within the honeynet. It efficiently delivers static content and manages incoming client requests, enhancing overall responsiveness. Iptables, a powerful firewall tool, provides granular control over network traffic flow within the honeynet. We leverage Iptables to establish rules that govern incoming and outgoing traffic based on predefined criteria, ensuring the security and integrity of the honeypot environment.
- *Matlab and Simulink*: They play a vital role in simulating the physical processes controlled by the honeypot PLCs. Matlab offers a versatile platform for numerical computation, facilitating the creation and analysis of mathematical models. Simulink, a graphical programming environment built on top of Matlab, enables the design and simulation of these models. By incorporating a physics-aware simulation layer, HoneyICS provides attackers with

a more realistic representation of the underlying industrial control system, enhancing the deception capabilities of the honeypot.

- *OpenPLC*: This open-source software PLC serves as the core element of our honeypot PLCs. OpenPLC offers a versatile platform for executing PLC programs, supporting various platforms and industry-standard programming languages defined in the IEC 61131-3 standard. By leveraging OpenPLC, we can create realistic honeypot PLCs that mimic the behavior of real-world industrial controllers.
- *ScadaBR*: As an open-source SCADA system, ScadaBR is used to create the HMI for our honeypot PLCs. ScadaBR allows for the visualization of variables, alarms, and other critical data points, mimicking the functionality of real-world HMI systems. This inclusion provides attackers with a familiar interface and increases the believability of the honeypot environment.
- *Elasticsearch, Logstash, and Kibana (ELK Stack)*: This forms the backbone of our log analysis and visualization system. Elasticsearch, a distributed search and analytics engine, stores and retrieves log data generated by the honeypot components. Logstash acts as a pipeline, collecting, parsing, and transforming log data before feeding it into Elasticsearch. Kibana, an open-source data visualization platform, empowers defenders with interactive dashboards to explore and analyze the collected log data. This comprehensive logging and analysis suite provides valuable insights into attacker behavior and facilitates post-incident analysis.
- *Zeek and eBPF*: Our approach to log collection and processing leverages a dual-technology strategy. Zeek, a passive network traffic analyzer, captures and analyzes network traffic flowing through the honeypot. Meanwhile, eBPF (extended Berkeley Packet Filter) offers a powerful mechanism for in-kernel data capture, providing deep visibility into network activity within the honeypot environment. The combined capabilities of Zeek and eBPF ensure comprehensive network traffic analysis and facilitate the detection of malicious activities.

By combining these technologies, we create a robust and feature-rich Honey-ICS that effectively deceives attackers, gathers valuable threat intelligence, and strengthens the overall security posture of industrial control systems.

5.2.1 Docker and containerization

Docker is an open-source project [12], that completely revolutionised the landscape of software development, offering an efficient and portable approach to containerization. Born out of the necessity to address challenges associated with deploying applications across diverse environments, Docker has reshaped how developers build, ship, and run software. Docker was originally named dotCloud [15], founded by Solomon Hykes in 2010. The initial focus was on creating

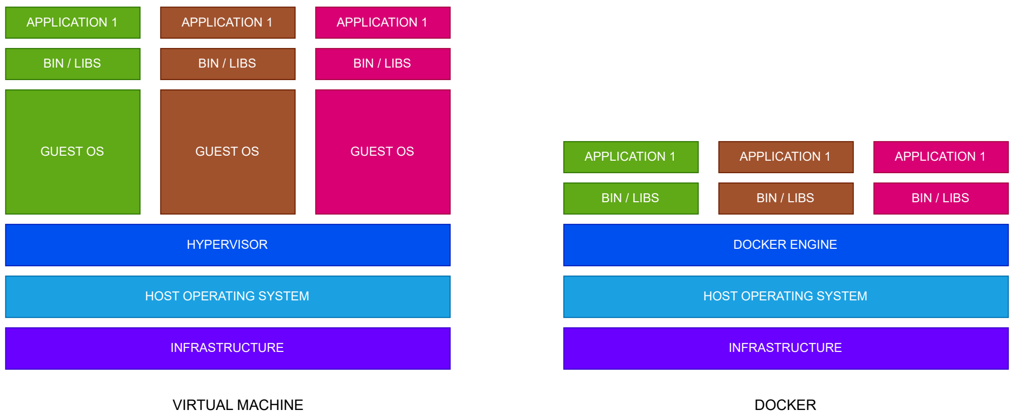


Figure 5.11: Virtual Machine vs. Docker

an internal platform-as-a-service (PaaS). However, recognizing the broader implications of their work, the team open-sourced the project in 2013, officially naming it Docker. At its core, Docker utilizes containerization, a lightweight, stand-alone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools. Unlike Virtual Machines (VMs), which emulate entire operating systems and are resource intensive, Docker containers share the host OS kernel as shown in Figure 5.11, resulting in a more efficient and resource-optimized solution. VMs require a hypervisor, hosted or bare metal, in order to manage the resources and emulate the entire operating system, on the other hand Docker containers leverage the host OS kernel directly. This results in faster startup times, reduced resource overhead, and enhanced scalability. To create a Docker container, developers use Docker images. These images are built using Dockerfiles, which contain a series of instructions for the Docker engine to follow. Dockerfiles automate the process, making image creation reproducible and efficient. Once an image is built, it serves as a template for launching containers. One of Docker's main advantages lies in its ability to encapsulate an application and its dependencies, eliminating environmental inconsistencies as it creates a new clean environment every time. It helps accelerate development cycles and optimizes resource utilization.

The anatomy and functionality of Dockerfile

The cornerstone of a Docker container is the Dockerfile [13], which is composed of a series of instructions along with their corresponding arguments. Every Dockerfile starts with the *FROM* instruction, followed by the name of a base image upon which all subsequent instructions will be executed. Any valid image can be used as a base image and can be easily retrieved from public repositories [14].

After each instruction is executed, a temporary image is generated, which is then passed to the next instruction in the Dockerfile, progressively constructing the final image step by step. The *RUN* instruction executes one or more commands on the current image and verifies the outcome. The commands provided as arguments to RUN are specific to the base image's operating system and are executed as if they would if they were manually inputted into the shell. For example, if a container requires the installation of specific programs like Python, using the command:

```
RUN apt install python
```

the built image would have Python installed, unless errors occur during the execution of the install command.

The *EXPOSE* instruction indicates on which ports the container will listen and the respective protocol, by default TCP. However, the *EXPOSE* command doesn't actually expose the ports; it configures them internally to enable communication between the host and the container. To concretely publish a container's port, it must be specified when starting the container (*docker run*) using the ”*-p*” flag.

The *USER* instruction sets the user performing the *RUN* operations, allowing Docker to retrieve their rights. This enables operations to be executed either as an administrator, using the root user, or to configure the container differently if there's a need to separate functionalities for different users.

The *WORKDIR* instruction allows navigation within folders and selecting the active folder for executing operations.

The *COPY* instruction is used to copy a file from the host machine to the active directory of the container, previously set with *WORKDIR*. This command can also be used repeatedly based on needs. The format is *COPY <source-folder> <destination-folder>*. The *ADD* instruction has the same functionality as *COPY*, except it allows using an URL as ¡source-folder¿ parameter, downloading its content.

The *CMD* and *ENTRYPOINT* instructions are used to set a command that will be executed when the container starts. The main difference is that the command passed as an argument to *CMD*, unlike *ENTRYPOINT*, can be overridden if command-line parameters are specified during the container startup.

Docker networking

A crucial aspect for container orchestration is networking, to allow communication between containers and external systems. Docker supports various network drivers, each serving different purposes. This includes:

- *Bridge Network*: The default network driver, creating an internal network for communication between containers on the same host.

- *Host Network*: Directly uses the host network stack, offering maximum performance but limiting isolation.
- *Overlay Network*: Enables communication between containers across different Docker hosts.
- *Macvlan* and *IPvLAN*: Provide options for connecting containers directly to physical networks.

Docker leverages on technologies like iptables, namespaces, and virtual interfaces to implement network isolation and routing for containers. Docker provides the following set of commands to manage networks:

- *docker network create my-network*: Creates a new Docker network called "my-network".
- *docker network ls*: Lists all available Docker networks on the host. The network called "my-network" we previously created will appear here.
- *docker network inspect my-network*: Offers detailed information about the Docker network called "my-network".
- *docker network connect my-network my-container* and *docker network disconnect my-network my-container*: Connects and disconnects the container called "my-container" from Docker network "my-network".

5.2.2 Nginx and Iptables

Nginx [2] is an open-source web server, reverse proxy server, and load balancer. Originally created by Igor Sysoev, it has gained popularity for its efficiency, performance, and versatility.

Nginx can be used as a standalone web server, since it excels in delivering static content directly to clients, showcasing its efficiency in handling static files. Additionally, Nginx can function as a reverse proxy, managing and distributing incoming client requests to backend servers. This reverse proxy capability is useful for load balancing, optimizing resource utilization, and enhancing the overall responsiveness of web applications. As a load balancer, Nginx distributes incoming network traffic across multiple backend servers, preventing any single server from becoming a bottleneck. Nginx's versatility extends to SSL/TLS termination, where it handles encryption and decryption, serving as a termination point for secure connections. This enhances security but also offloads the resource-intensive task of encryption from backend servers. Furthermore, Nginx supports WebSocket as a reverse proxy, making it suitable for applications requiring real-time communication. Its caching mechanisms add another layer of efficiency by storing and serving frequently requested content directly, thereby reducing the load on backend servers and improving response times. Lastly, Nginx prioritizes security with various features, including access controls, rate limiting, and the capability to mitigate common web vulnerabilities.

Iptables [27] is a versatile and powerful tool for managing packet filtering rules in a Linux-based firewall. It provides a set of rules to govern the flow of incoming and outgoing network traffic based on predefined criteria such as the protocol used, source and destination ports, and the desired action to be performed once a match is found. Listing 5.4 provides a set of Iptables rules to exemplify its usage.

Listing 5.4: Iptables usage

```
 # Allow incoming traffic on port 80 (HTTP)
iptables -A INPUT -p tcp --dport 80 -j ACCEPT

# Drop all incoming traffic on port 22 (SSH)
iptables -A INPUT -p tcp --dport 22 -j DROP

# Allow outgoing DNS requests
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
```

Using the *-A INPUT* and *-A OUTPUT* flags the tool is appending the newly created rules to the INPUT and OUTPUT chains, respectively. The former is responsible for dealing with incoming packets destined for the system, while the latter manages outgoing packets from the system. The *-p tcp* and *-p udp* options specify the transport protocol as TCP or UDP. The *–dport* flag designates the destination port for the rule. Lastly, the *-j ACCEPT* and *-j DROP* flags indicate whether to accept or discard packets matching the specified criteria.

5.2.3 Matlab and Simulink

Matlab is a programming platform developed by MathWorks Inc., made for engineers and scientists, that provides an environment for the analysis and design of systems. Specialized in numerical computation, it facilitates matrix manipulation, function, and data visualization, along with seamless interaction with other programs. Its utility extends to creating models and employing mathematical equations for simulations, enabling a profound understanding of complex physical phenomena and predicting system behavior.

The Matlab environment has four primary windows:

- Command Window: Allows the input of commands and real-time result visualization.
- Workspace: A memory space enabling variable allocation and displaying current allocations.
- Current Directory: Facilitates folder exploration in memory, supporting various operations such as opening and consulting other MATLAB files.
- Command History: Maintains a list of recently used commands for re-execution or reviewing executed instructions.

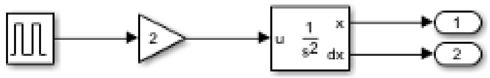


Figure 5.12: Example of a Simulink model

Simulink is a graphical programming environment also developed by MathWorks that allows engineers and scientists to model, simulate, and analyze multidomain dynamical systems. It is a block diagram environment that enables the design of systems with multidomain models, simulation before moving to hardware, and deployment without writing code. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. It can be used for a range of applications, including wireless communications, electrification and control systems. An example of a Simulink model is illustrated in Figure 5.12. It replicates the dynamic interaction between a car's motion and the operation of a proximity sensor. Beginning with the generation of an input signal representing accelerator pedal pressure, the model distinguishes between a value of 1 when the pedal is pressed and 0 when released. This signal is then multiplied by a constant factor to determine its impact on the car's acceleration. The model uses successive integrations to compute the car's position based on the calculated acceleration. One of the model's outputs provides the resulting position of the car. The model then calculates the actual distance between the car and an obstacle situated 10 meters away by subtracting the car's position from the obstacle's position. To emulate real sensor measurement errors, random noise is introduced to the actual distance. The model further emulates a digital sensor operating at discrete time intervals by sampling the noisy distance. The measured distance is then fed to the other output of the model.

5.2.4 OpenPLC: an open-source software PLC

Software PLCs, like OpenPLC, offer an alternative to traditional hardware-based PLCs. OpenPLC is an open-source PLC suite designed for industrial automation and research. It supports various platforms, including Arduino, Raspberry Pi, Windows, and Linux, making it versatile for diverse applications. The suite comprises of two main components: (i) The *runtime*: which is the core part of OpenPLC, it is installed on a device that can be either physical or virtual, to execute PLC programs and (ii) the *editor*: which is used for creating PLC programs. The editor in OpenPLC is based on the Beremiz project [6] and supports all five programming languages defined in the IEC 61131-3 standard, which include Ladder Logic (LD), Function Block Diagram (FBD), Instruction List (IL), and Structured Text (ST). Internally, like any PLC, OpenPLC manages data through input, system, and output registers, which can vary in size depending on the contained data. To access a register, its address is required. In OpenPLC, an address consists of:

- the % character;
- letter from the set I, Q, M, specifying whether the register is designated for input, output, or system;
- a letter from the set X, B, W, D, L, signifying the data size (Bit, Byte, Word, Double Word, Long Word);
- a hierarchical address that differentiates addresses that would otherwise be equivalent. Bit addresses follow a hierarchical structure consisting of two parts separated by a dot: the left part signifies the register number ranging from 0 to 1023, while the right part indicates the specific bit within that register. Since only one bit is required, a register can accommodate up to 8 binary values. Conversely, for addresses of different types, only one number is used, denoting the left part.

OpenPLC features a user-friendly web interface, where through a login page, the user logs into OpenPLC and can navigate through various sections: the *Dashboard* provides an overview of the running program and a log table; *Programs* allows the insertion and compilation of .st files for execution by the PLC; *Slave Device* enables the addition of additional devices; *Monitoring* displays variable values, inputs, and outputs; *Hardware* configures the hardware on which OpenPLC is installed; *Users* defines users with respective privileges; *Settings* manages various configurations, such as communication protocols, their ports, and timeout periods; and finally, *Logout* disconnects the user from the interface.

5.2.5 ScadaBR: an open-source software HMI

ScadaBR is an open-source, Java-based, cross-platform application designed to serve as supervision and control system software. Offering a comprehensive set of features, it provides visualization capabilities for variables, graphics, statistics, protocol configurations, alarms, and facilitates the creation of HMI-like screens, along with an array of configuration options. Upon configuring communication protocols with the equipment and defining inputs and outputs for an automated application, ScadaBR enables the construction of web-based interfaces directly through the browser. Furthermore, ScadaBR offers the flexibility to develop custom applications in any modern programming language. Users can leverage the provided source code or utilize the "web-services" API, enabling the creation of tailored solutions that align with specific requirements. ScadaBR relies on data sources, these are its atomic entities that serve as the origin points for incoming data (inputs and outputs). The received data values are then stored in data points. ScadaBR's Graphical Representation feature can be used to craft visualizations of this data. The creation process involves selecting the Graphical Representation option, naming the representation, choosing an image, and adding components such as Simple Data Point, Text, Image, and Shape. Configuration options for these components, including font size, color, and position, allow users to tailor the visual representation. Furthermore, ScadaBR extends its functionality with statistical analysis, configurable alarms and a versatile web services API supporting custom applications.

Selenium

A significant hurdle in deploying our honeypot system lies in ScadaBR's lack of an Application Programming Interface (API) for automated configuration. This absence necessitates manual configuration by users for each honeypot instance, a tedious and error-prone process. To overcome this limitation, we leverage Selenium [20], a powerful web automation framework. By scripting interactions with ScadaBR's user interface through Selenium, we can automate the configuration process.

At the heart of Selenium lies WebDriver, a potent API and protocol. It establishes a language-agnostic interface, enabling communication between Selenium and a multitude of web browsers. Each browser leverages a dedicated WebDriver implementation, known as a driver, to facilitate this communication. The driver acts as a bridge, translating Selenium commands into actions within the specific browser environment.

5.2.6 Elasticsearch Logstash and Kibana stack

ELK is an acronym for three very well known projects: Elasticsearch, Logstash and Kibana. It is used for log data analysis, document search, security information and event management, and observability.

Elasticsearch stands out as a distributed, RESTful search, and analytics engine with the capability to store and retrieve data in real time. Logstash acts as an event and log management tool, enabling the collection, parsing, and storage of logs for subsequent analysis. On the visualization front, Kibana, an open-source platform from Elastic, provides an intuitive and interactive web interface for data exploration. Together, these components form a cohesive framework, empowering users with robust tools for comprehensive data analysis and visualization tasks.

Kibana is based on a versatile search language known as the query domain-specific language (DSL). This language uses a JSON interface to allow users to articulate intricate search queries efficiently. Every single table in the dashboard is built using DSL queries. An elementary search in OpenSearch is shown in Listing 5.5 utilizes the *match_all query*, which, as the name implies, matches all documents within a specified index, somewhat like the *SELECT * FROM table* query in SQL.

Listing 5.5: Basic Match-All Query

```
GET testindex/_search
{
  "query": {
     "match_all": {
     }
  }
}
```

A query can be composed of multiple query clauses, categorised into leaf queries and compound queries. Leaf queries, are atomic and can function autonomously, they are designed to search for specific values in designated fields. They encompass full-text queries for text document searches, term-level queries for exact term searches, geographic and xy queries for spatial data searches and specialized queries encompassing various types. On the other hand, compound queries act as wrappers for multiple leaf clauses, combining results or modifying behavior.

5.2.7 Zeek and eBPF

For log collection and processing, we employ a dual-technology approach, utilizing eBPF for collection and Zeek for data processing and loading into Elasticsearch.

eBPF, or extended Berkeley Packet Filter, represents a groundbreaking technology within the Linux kernel. This project allows the execution of sandboxed programs in privileged contexts, such as the kernel itself. It is useful for extending kernel capabilities without the need for modifying source code or loading new kernel modules and serves a range of purposes, including high-performance networking, security observability, and application tracing. One of its key advantages lies in its ability to enhance kernel functionalities dynamically.

Zeek is a passive, open-source network traffic analyzer which is used as a Network Security Monitor (NSM) for investigating potentially malicious and/or suspicious activities. Zeek also supports various traffic analysis tasks such as performance measurement and troubleshooting. Zeek collects all the information captured into JSON log files, facilitating post-processing with external software such as uploading to a third party software like Elasticsearch. It also offers the option to integrate external Security Information and Event Management (SIEM) products to store and process the data. Zeek is also capable, among the other functionalities, of extracting files from HTTP sessions, interfacing with external registries for malware detection, reporting vulnerable software versions observed on the network, detecting SSH brute-forcing, validating SSL certificate chains, and more.

5.3 Docker implementation

In this section, we describe the details of the implementation, unraveling the key components and methodologies that constitute the foundation of our honeynet architecture. In the implementation of our use case, we decided not to utilize any physical hardware or devices. Instead, we leveraged on well-established simulation frameworks such as HoneyPLC [112], OpenPLC [1], ScadaBR [45], and Simulink [50]. As for industrial network protocols we emulate *Modbus* [102]. To ensure scalability and reconfigurability, each element of our honeynet is deployed within dedicated Docker containers [118], running on either Ubuntu 18.04 LTS or NGINX base images. This containerized approach provides a modular and adaptable deployment environment, aligning seamlessly with the dynamic needs of our honeynet architecture.

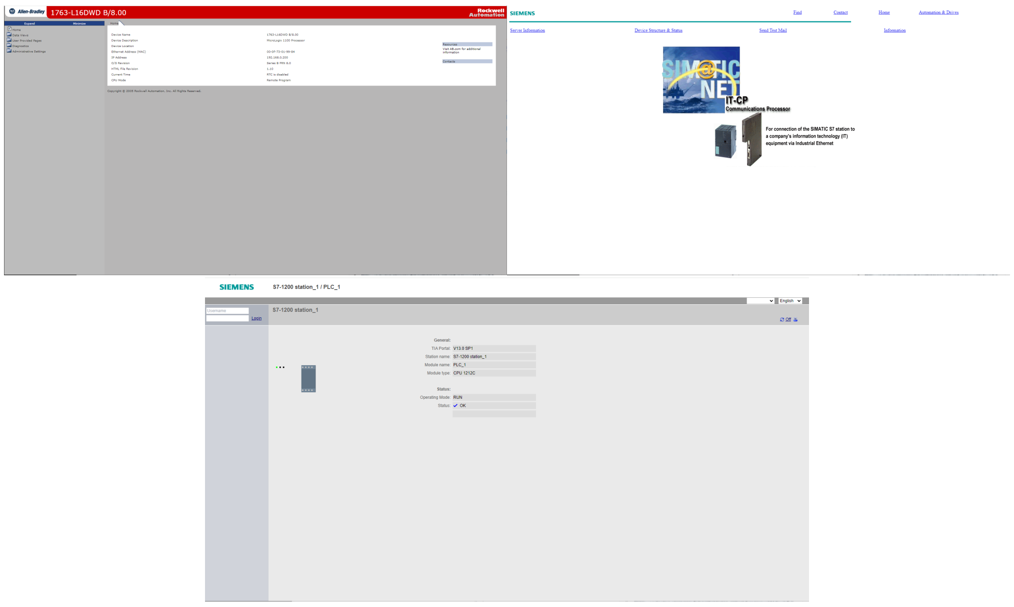
5.3.1 PLC container

Our honeynet incorporates three distinct PLCs, all implemented as software PLCs. Each PLC is implemented using a dedicated container, utilizing the Ubuntu 18.04 LTS as base image, and houses both the *low-interaction* and *physics-aware high-interaction* part of the honeypots.

Initially, we employed Honeyd [127] for our low-interaction honeypot. Honeyd allows the creation of a ”personality engine” that emulates the TCP/IP stack of target devices, such as PLCs. When a scan request is sent from a fingerprinting tool like Nmap, Honeyd selects the relevant PLC fingerprint from an extensive database and responds to the scanning tool. To accommodate a diverse range of PLC models, we seamlessly integrated PLC profiles from HoneyPLC [112] into Honeyd. Currently, HoneyPLC can respond to fingerprint requests for notable PLC models, including ABB PM554-TP-ETH, Allen-Bradley MicroLogix 1100, Siemens S7-300, S7-1200, and S7-1500. For the execution of PLC functionalities, such as code execution, we utilized Honeyd’s subsystem virtualization (detailed in Section 3.2.1) to run OpenPLC [74] and route the packets to the runtime.

However, in light of the challenges detailed in Section 3.3.6, where Honeyd struggled to generate desired fingerprints, we decided to discontinue its use. Consequently, we shifted our focus to developing alternative features to enhance the precision of fingerprints.

To this end, we developed web interfaces for the Allen-Bradley MicroLogix 1100, Siemens S7-300, and S7-1200, starting from real PLCs, as illustrated in



Device Name	1763-L16DWD B/8.00
Device Description	MicroLogix 1100 Processor
Device Location	
Ethernet Address (MAC)	00-0F-73-01-99-84
IP Address	192.168.0.200
O/S Revision	Series B FRN 8.0
HTML File Revision	1.10
Current Time	RTC is disabled
CPU Mode	Remote Program

Visit AB.com for additional information

Copyright © 2005 Rockwell Automation, Inc. All Rights Reserved.

For connection of the SIMATIC S7 station to a company's information technology (IT) equipment via Industrial Ethernet

General:
TIA Portal: V13.0 SP1
Station name: S7-1200 station_1
Module name: PLC_1
Module type: CPU 1212C

Status:
Operating Mode: RUN
Status: ✓ OK

Figure 5.13: Web interfaces for: Allen-Bradley MicroLogix 1100, Siemens S7-300, S7-1200

Figures 5.13. We also added the modified version of Snap7 [51] provided by HoneyPLC to add S7comm capabilities to our honeynet. Snap7 is an open-source communication library for Siemens PLCs. It provides a set of functions for communicating with Siemens S7 PLCs, including reading and writing data, and handling alarms and events. The S7 protocol is used to communicate with Siemens PLCs and is mainly used to connect the PLCs to the PC stations. The communication follows the traditional client-server model, where the PC (client) sends S7 requests to the field device (server). These requests are used to query from or send data to the device or issue certain commands. The library is useful for making realistic honeypots of Siemens PLCs. The authors of Honey-PLC extended the Snap7 Server module to support program block upload and download, added a specific banner for the emulated PLC model, and included additional features. Every Siemens PLC container has its own libsnap7 profile, which displays the banner once a nmap scan is performed using the s7-info.nse script to collect device information. The banner produced is shown in Listing 5.6, as demonstrated in Listing 5.7 on line:

```
RUN cp /home/honeyplc/snap7/build/bin/x86_64-linux/libsnap7-
    .so-300 /usr/lib/libsnap7.so
```

the appropriate profile is built into each container.

Listing 5.6: Honeypot response to a "nmap –script s7-info.nse" scan

```
Details
   System
      PRODUCTION S7-300
   Module
      CPU 315-2 PN/DP
   Copyright
      Original Siemens Equipment
   Serial Number
      S C-U2A203512011
   Module Type
      CPU 315-2 PN/DP
   Reserved For OS
      MMC 276FA33B
   Module ID
      6ES7 315-2AG10-0AB0
```

Listing 5.7: Dockerfile for the PLC container

```
# Base image
FROM ubuntu:18.04

# Install necessary tools and libraries
RUN apt-get update && \
   apt-get -y install lighttpd git python-pip autoconf bison build-essential
       pkg-config bison flex autoconf automake libtool make git python2.7
       python-pip sqlite3 cmake sudo \
   && apt-get clean

# Install additional dependencies
RUN apt-get update \
  && apt-get install -y wget gcc make openssl libffi-dev libgdbm-dev
      libsqlite3-dev libssl-dev zlib1g-dev \
  && apt-get clean

# Clone and setup HoneyPLC
WORKDIR /home/
RUN  git clone https://github.com/sefcom/honeyplc.git

# Clone and install OpenPLC
WORKDIR /home/
RUN git clone https://github.com/thiagoralves/OpenPLC_v3.git
WORKDIR /home/OpenPLC_v3/
RUN sudo chmod +x install.sh \
  && sudo ./install.sh linux \
  && mkdir /home/OpenPLC_v3/scripts

# Copy OpenPLC scripts and configurations
COPY run.sh PLC2.st launch.sh /home/OpenPLC_v3/
RUN sudo chmod +x run.sh launch.sh
```

```
# Setup Lighttpd
COPY ./profile /var/www/html
COPY lighthttpd.conf /etc/lighttpd/lighttpd.conf

# Setup Snap7
RUN cp /home/honeyplc/snap7/build/bin/x86_64-linux/libsnap7.so-300
    /usr/lib/libsnap7.so

# Expose necessary ports
EXPOSE 502 6668/udp 8080 8081 47 67 68 80 102

# Finalize setup
COPY init.sh /home/init.sh
RUN sudo chmod +x /home/init.sh
CMD /home/init.sh
```

By leveraging SNMPsim library [18] and incorporating the MIB files from HoneyPLC (with a minor fix), we successfully configured the SNMP server to respond as if it were a Siemens S7-300. Listing 5.8 presents the Dockerfile used to establish a honeypot that mimics the response of a Siemens S7-300 PLC to an snmpwalk request. Additionally, Listing 5.9 includes the provided MIB file.

Listing 5.8: SNMP Dockerfile

```
# Use Ubuntu 20.04 as base image
FROM ubuntu:20.04

# Update the package lists for upgrades and new package installations
RUN apt update

# Install vim and software-properties-common for managing software repositories
RUN apt install -y vim software-properties-common

# Add deadsnakes PPA to get different Python versions
RUN add-apt-repository ppa:deadsnakes/ppa

# Install Python 3.7
RUN apt install python3.7 -y

# Make Python 3.7 the default Python version
RUN echo "alias python=python3.7" >> ~/.bashrc && \
   export PATH=${PATH}:/usr/bin/python3.7 && \
   /bin/bash -c "source ~/.bashrc"

# Install pip for Python 3
RUN apt install python3-pip -y && \
   python3 -m pip install --upgrade pip

# Install specific versions of snmpsim and pyasn1
```

```
RUN pip install snmpsim==0.4.7 pyasn1==0.4.8

# Copy data from local 'data' directory to '/data' in the container
COPY ./data /data

# Create a new user and switch to it
RUN useradd -ms /bin/bash newuser
USER newuser

# Set the entrypoint for the container
ENTRYPOINT ["snmpsimd.py", "--data-dir=/data",
    "--agent-udpv4-endpoint=0.0.0.0:1024", "--v2c-arc"]
```

Listing 5.9: MIB file

```
.1.3.6.1.2.1.1.2.0 = OID: .1.3.6.1.4.1.9.1.516
.1.3.6.1.2.1.1.1.0 = STRING: "Siemens, SIMATIC, S7-300"
.1.3.6.1.2.1.1.2.0 = OID: .0.0
.1.3.6.1.2.1.1.3.0 = 194500870
.1.3.6.1.2.1.1.4.0 = ""
.1.3.6.1.2.1.1.5.0 = ""
.1.3.6.1.2.1.1.6.0 = ""
.1.3.6.1.2.1.1.7.0 = INTEGER: 72
.1.3.6.1.2.1.2.1.0 = INTEGER: 1
.1.3.6.1.2.1.2.2.1.1.1 = INTEGER: 1
.1.3.6.1.2.1.2.2.1.2.1 = STRING: "Siemens, SIMATIC NET, CP343-1 PN, 6GK7
    343-1EX21-0XE0, HW: Version 2, FW: Version V1.1.13, Fast Ethernet Port 1,
    Rack 0, Slot 4, 100 Mbit, full duplex, autonegotiation"
.1.3.6.1.2.1.2.2.1.3.1 = INTEGER: 6
.1.3.6.1.2.1.2.2.1.4.1 = INTEGER: 1500
.1.3.6.1.2.1.2.2.1.5.1 = Gauge32: 100000000
.1.3.6.1.2.1.2.2.1.6.1 = Hex-STRING: 00 0E 8C 82 38 A9
```

Although working perfectly, we choose not to add a Simple Network Management Protocol (SNMP) functionality to our honeypots. This decision stems from the acknowledgment of a vulnerability (*CVE-2018-18065*) affecting all Siemens S7 devices, specifically a potential Denial of Service (DOS) attack through SNMP [126]. Siemens recommends disabling SNMP as a mitigation strategy for this vulnerability, fully addressing the risks. Considering that the activation of a SNMP server on our PLC could potentially reveal the nature of our honeypot, given that a genuine device with SNMP active would be vulnerable to the identified risk, we chose to prioritize the realism of our honeynet by refraining from implementing support for SNMP. This strategic decision aligns with Siemens' suggested precautions, as it is common to find SNMP disabled on real devices as well.

Communication module

In order to enable communication between PLC2 and PLC1 and allow PLC2 to send requests for opening the valve to PLC1, we developed a dedicated communication module. This module serves as an intermediary, allowing the two PLCs to exchange Modbus commands seamlessly. In the evaluation of Modbus libraries for our specific use case, we conducted a thorough analysis of three prominent options:

- *MinimalModbus*, *pymodbus*, and *modbus-tk*. Each library has its own set of advantages and drawbacks. MinimalModbus is a lightweight module suitable for applications that require reading around 10 registers. However, its performance becomes unacceptable when dealing with approximately 64 registers, exhibiting relatively high CPU load.
- *pymodbus*, on the other hand, distinguishes itself by relying on a serial stream. It offers low CPU load and acceptable performance when the serial timeout is dynamically set. However, there are challenges, such as performance being two times lower compared to modbus-tk even with dynamic timeout adjustment. Additionally, it may experience issues with responses when different reads or reads/writes are performed frequently.
- *modbus-tk* stands out for its quick response assembly, resulting in the best overall performance. While its CPU load is higher compared to pymodbus, improvements have been made to address this through modifications in the library. The code in modbus-tk may not be as elegant as pymodbus, but it's optimized performance makes it a favorable choice.

After extensive testing, we found out that Pymodbus offers certain advantages that align with our specific requirements. We were aiming at lower CPU load and acceptable performance, since we had to keep the system scalable, the resource usage was our main priority.

5.3.2 HMI container

We containerized a ScadaBR instance, incorporating a Python script utilizing the Selenium WebDriver library [47] to load the data source and graphical interface onto ScadaBR. Selenium, a web framework, facilitates web page navigation. The Dockerfile shown in Listing 5.10, begins by installing the necessary dependencies using the Ubuntu 18.04 image, followed by the cloning and installation of ScadaBR. Finally, the startup script is copied with execution rights. After changing the directory, the driver, the Python script, the HMI configuration file (exported from ScadaBR), and the package to install Google Chrome (used by Selenium as a web browser) are copied into the container. Finally, the automated ScadaBR configuration script is run, and ports 502 for Modbus, 8080, and 9090 for web interfaces are exposed.

Listing 5.10: HMI Dockerfile part 1

```
# Use Ubuntu 18.04 as base image
FROM ubuntu:18.04

# Set root as user
USER root

# Update and install necessary packages
RUN apt-get update && apt-get -y install sudo git python-pip autoconf bison
    build-essential pkg-config bison flex autoconf unzip automake libtool make
    git python2.7 python-pip sqlite3 cmake python3 python3-pip wget \
   adwaita-icon-theme at-spi2-core dconf-gsettings-backend dconf-service
       fontconfig fontconfig-config fonts-liberation glib-networking
       glib-networking-common glib-networking-services

# Install Python packages
RUN pip install flask flask-login pyserial pymodbus

# Clone ScadaBR_Installer repository
WORKDIR /home/
RUN git clone https://github.com/thiagoralves/ScadaBR_Installer.git

# Install ScadaBR
WORKDIR /home/ScadaBR_Installer/
RUN sudo ./install_scadabr.sh linux

# Copy and set permissions for run.sh
COPY run.sh /home/ScadaBR_Installer/
RUN sudo chmod +x run.sh

# Create selenium directory
RUN mkdir selenium

# Copy nginx configuration
COPY default /etc/nginx/sites-available/default

# Set working directory
WORKDIR /home/selenium

# Expose ports for Modbus, ScadaBR and some other service
EXPOSE 502 8080 9090

# Run init.sh script and start nginx in foreground when a container is started
CMD /bin/bash -c "/home/selenium/init.sh && nginx -g 'daemon off;'"
```

To make the process of deployment as smooth as possible, we went ahead and created a script to preload the environment on the HMI. Leveraging Selenium, we successfully automated the entire process, from the creation of the graphical interface for end-users, to the linking of datapoints to the corresponding registers on the PLCs.

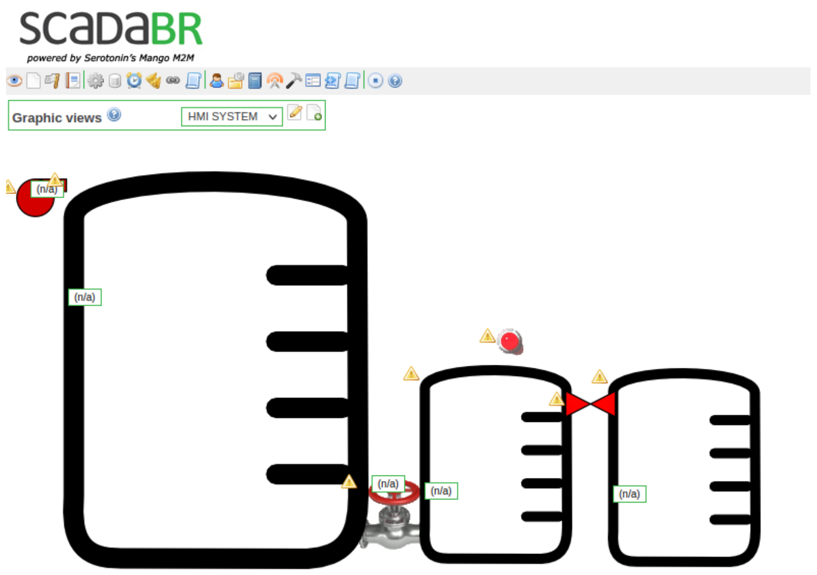


Figure 5.14: ScadaBR HMI graphical view

The web interface displayed on port 9090 TCP is shown in Figure 5.14 for reference.

5.3.3 Networking container

We implemented a proxying technique with dual objectives. First, to act as a shield for each device/component in the honeynet, preventing attackers from infiltrating other containers through alternative ports or services, as the proxy actively rejects such unauthorized attempts.

In more detail, for each PLC container, we deploy a dockerized NGINX proxy serving a dual role: acting as a reverse proxy for external and HMI-originating connections and as a gateway for connections originating from the associated PLC container.

This solution addresses two critical challenges. Firstly, by utilizing the proxy as a gateway, we transparently redirect the traffic generated by attackers, simultaneously deceiving them by presenting a facade that mimics a genuine system. Secondly, it ensures the isolation of honeynet components, mitigating the risk of lateral movements within the supervisory control network in the event of an attacker compromising a device (PLC or HMI).

To fulfill both objectives, each PLC component is logically connected to the supervisory control network through a dedicated *proxy* (refer to Fig. 4.4).

Precisely, every PLC is associated with a private `macvlan` docker network [32], accompanied by a corresponding proxy component situated in front of the PLC. This proxy serves a dual purpose: (i) as a *reverse proxy* for connections from the Internet and the supervisory control network; (ii) as a *gateway* for connections originating from the PLCs. Importantly, proxies can communicate with each other via a shared secondary docker macvlan network, facilitating controlled connections between two potentially compromised PLCs.

We implemented the reverse proxy functionality of the proxy component using NGINX, coupled with iptables post-routing commands for the gateway aspect. This enables us to alter the source address of connections from the associated PLC, ensuring proper routing.

This proxying technique creates the illusion of a flat supervisory control network to the attacker, facilitating potential *MITM attacks*. The proxies appear transparent to the attacker, resembling real PLCs exposing their interfaces on the control networks.

Furthermore, the use of proxies enables the concealment of details about the true nature (virtual/physical) of the PLC from entities connecting to the supervisory control network. It also provides the flexibility to enforce security or filtering policies on the flowing traffic if necessary.

In order to achieve the implementation shown in Figure 5.15 we had to configure Nginx as follows. We set the user that Nginx runs as *'user nginx;'* and the number of worker processes *'worker_processes auto;'*. By utilizing the *auto* value for *worker_processes*, we allow Nginx to dynamically determine an optimal value based on the available CPU cores. However, to enhance security and ensure availability, we also define the maximum number of simultaneous connections that a worker process can open. We then defined the *'stream'* block which is used for configuring TCP and UDP proxying. Inside this block, there are multiple *'server'* blocks, each representing a different server configuration.

For example, consider this *'server'* block:

Listing 5.11: Example of a server block in the nginx.conf file

```
properties
server {
listen 192.168.10.111:502;
proxy_pass 172.21.0.2:502;
}
```

The block in Listing 5.11 is for PLC 1 and tells Nginx to listen on IP address 192.168.10.111 and port 502. Any incoming connections on this address and port will be forwarded to 172.21.0.2:502, which is the address and port of the upstream server.

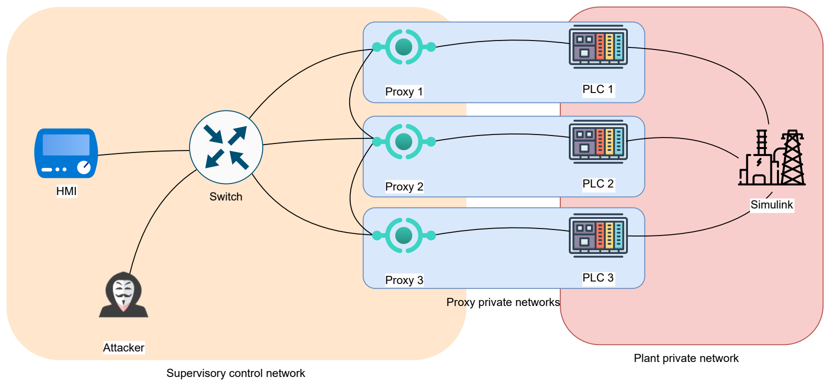


Figure 5.15: Implementation of the supervisory control network

5.3.4 Plant container

In our implementation, we used a container with Simulink to emulate the physical plant. . However, it is also feasible to integrate a real plant into our architecture. This container is built upon the MathWorks Matlab r2021a base image, where some components like *Simulink*, *Signal_Processing_Toolbox*, and *DSP_System_Toolbox* were installed. These tools are indispensable for executing our simulink model accurately. The Dockerfile is very simple, it copies three local files (*'plc.slx'*, *'init.m'*, *'reset.sh'*) to the *'/usr/local/src/'* directory. These files contain respectively the Simulink model, the script for initializing environment variables, and the script for periodically resetting the simulation. Notably, *reset.sh* is executed once every six hours to free up RAM and ensure availability.

Following this, the script updates the package lists for potential upgrades and new arrivals in the repositories. It then installs *'wget'*, a tool for non-interactive file downloads from the web.

After that, it acquires the MATLAB Package Manager (mpm) from the MATLAB website, granting it executable status. This tool is used to install additional MATLAB toolboxes.

Next, it installs through mpm the components mentioned earlier: Simulink, Signal Processing Toolbox, and DSP System Toolbox.

The Dockerfile concludes by opening UDP ports 10001 to 10006 within the Docker container. These ports are going to be used by the model to send and receive data about the sensors and actuators.

Field communication network

To establish the connection between the plant and the controllers (PLCs), we adapted a script named *Simlink* written by the OpenPLC developer [53]. We modified the C++ script in order to suit the requirements of a Docker environment.

The code uses threads and sockets to send and receive data to and from each station (also known as PLC) and the simulink simulation over UDP. The data received from either the simulation or the PLC is organized and stored in a designated structure named *plcData*. Prior to transmission, this data is duplicated into a local buffer and subsequently dispatched to either the simulation or the PLC. The code also includes error checking to ensure data is sent and received correctly. After successfully receiving data, it copies the data back to the main data structure.

We are going to explore the key functions in Simlink, emphasizing their roles in establishing and managing communication between network stations, and Simulink, and ensuring accurate transmission through error-checking mechanisms.

The function, *parseConfigFile*, processes the configuration file by reading it line by line. For each line, the function verifies that it is neither a comment (lines beginning with # are treated as comments) nor empty. The function then identifies the Simulink IP address, the number of stations, the ip address of the station and UDP port for each station and finally the communication delay. The structures plcData and stationInfo are used to store details about each station. Specifically, *plcData* captures analog and digital input/output data, while *stationInfo* stores IP addresses, ports, and analog/digital input/output configurations for each station.

The *connectToPLCStations* function creates a new thread for each station, managing the exchange of data. The *main* function initiates the data exchange process and periodically outputs the data from each station. This data encompasses variables such as pressure, tank levels, pump status, and valve positions. The code implements mutex locks to guarantee data consistency when accessing shared data.

The function named *receiveSimulinkData* is crafted to receive data from the Simulink model via a UDP connection. This function expects a pointer to an integer array as argument, with three information: the station number, variable type, and variable index. Upon receiving the argument, the function casts it to an integer pointer, assigning values to variables such as *stationNumber*, *varType*, and *varIndex*. Subsequently, the function determines the type of data to receive (analog or digital) based on the *varType* variable. Depending on the type, it sets the port variable to the appropriate port number, and the *analogPointer* or *digitalPointer* points to the relevant location in the array. The function proceeds to establish a UDP server on the specified port using the *createUDPServer* function. This auxiliary function initiates a socket, initializes a *sockaddr_in* structure with server details, binds the socket to the specified port, and then returns the socket file descriptor. Entering an infinite loop, the function awaits data reception on the socket. Upon receiving data, it checks for errors, and if none are found, it converts the received data from the buffer to a double using the *convertBufferToDouble* function. To maintain data integrity, the function locks a mutex, updates the value pointed to by analogPointer or digitalPointer with the received value, and subsequently unlocks the mutex.

The *sendSimulinkData* function is a threaded function responsible for transmitting data to Simulink using the UDP protocol. Receives a void pointer as an argument, it casts it to an integer pointer to extract crucial information like the station number, variable type, and variable index. The function then establishes a UDP socket, determining the port and data pointer based on the variable type. The function enters an infinite loop, employing mutexes to secure data integrity. It retrieves the data value, unlocks the mutex, and sends the data to the server. In case of transmission errors, the function issues an error message. Following each iteration, the function sleeps for a specified delay before resuming the loop.

The *exchangeDataWithSimulink* function manages the exchange of data with the Simulink model and multiple stations. For each station, it dynamically creates threads for both sending and receiving operations. The *sendSimulinkData* function manages the data transmission, while *receiveSimulinkData* handles data reception.

The *exchangeDataWithPLC* function manages the data interchange with the PLC (station). It initiates a UDP socket connection with the PLC and enters an infinite loop for bidirectional data transfer. The exchange involves *plcData* structures, with mutex locking implemented for thread-safe access to the shared data buffer. A controlled data exchange rate is maintained through a delay (specified in the configuration file) between each send-receive cycle.

5.3.5 Probes

In order to collect network logs and the system call used on the device, we used a data collection component developed using Aquasecurity Tracee [67], an advanced runtime security forensics tool designed for Linux-based systems. This tool is based upon the extended Berkeley Packet Filter (eBPF) framework [68]. In our specific implementation, we inject the probes into each container to monitor various aspects, including network traffic, system call executions, kernel functions, and alterations to the filesystem. This approach ensures thorough surveillance of the runtime environment, enabling detailed insights into the system's behavior and security posture.

5.3.6 Management and monitoring dashboard

The management and monitoring dashboard offers a GUI Interface functioning as a Command & Control (C&C) Server. Tailored for user interaction, this interface provides a graphical platform for seamless management and control of the honeynet. Serving as an intermediary, it enables the communication between the honeypots and the operator through the Docker API. This streamlines the oversight and manipulation of various components within the honeynet. Its importance is underscored by the fact that, without this component, the operator would have been compelled to engage directly through the command-line interface, adding complexity to the management process.

The dashboard allows the user to manage the resources within the internal network, controlling the Docker engine and overseeing various components, including OpenPLC and ScadaBR. To monitor the activity, the main page of the dashboard as shown in Figure 5.16, displays both the real-time Simulink simulation and the HMI evolution, ensuring that the physical evolution of the system aligns with its simulation in Simulink. The real-time graphics are achieved through an API interface that fetches data from the Simulink model and the ScadaBR web interface, seamlessly embedding them onto the webpage using an iframe.

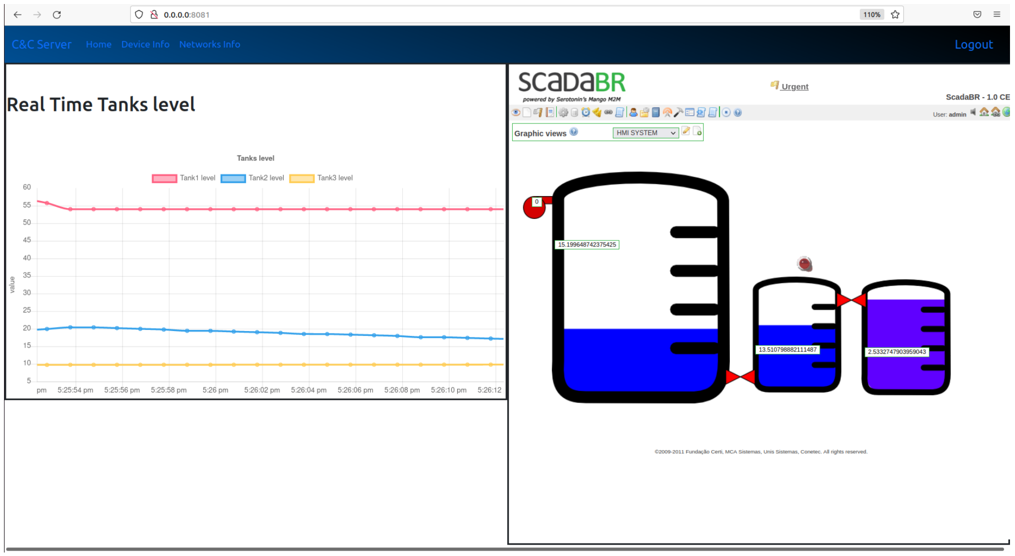
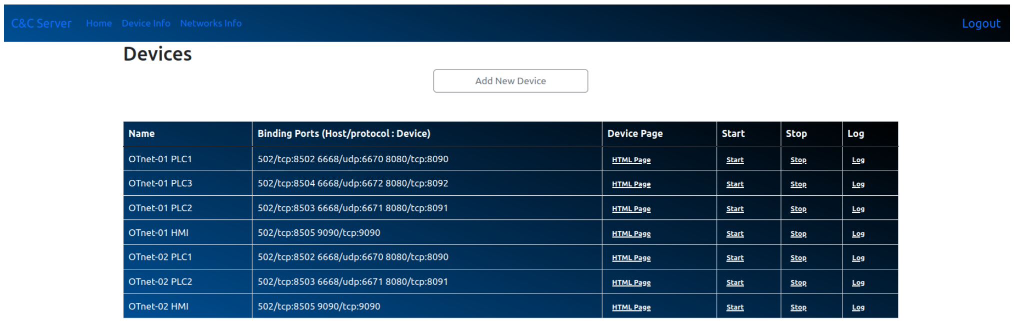


Figure 5.16: Monitoring of the simulink simulation and of the HMI evolution on the dashboard

To showcase the devices and internal networks, the server leverages Docker's API. The Python Docker library is employed to instantiate the Docker client, enabling seamless interaction with the Docker daemon. Figure 5.17 illustrates the devices page, presenting a comprehensive list of active and/or inactive honeypots, including their names, connected ports and links for accessing HTML pages. Users can initiate and halt honeypots, as well as access raw Docker logs. The interface enables the operator to deploy new honeypots and seamlessly integrate them into an existing network.

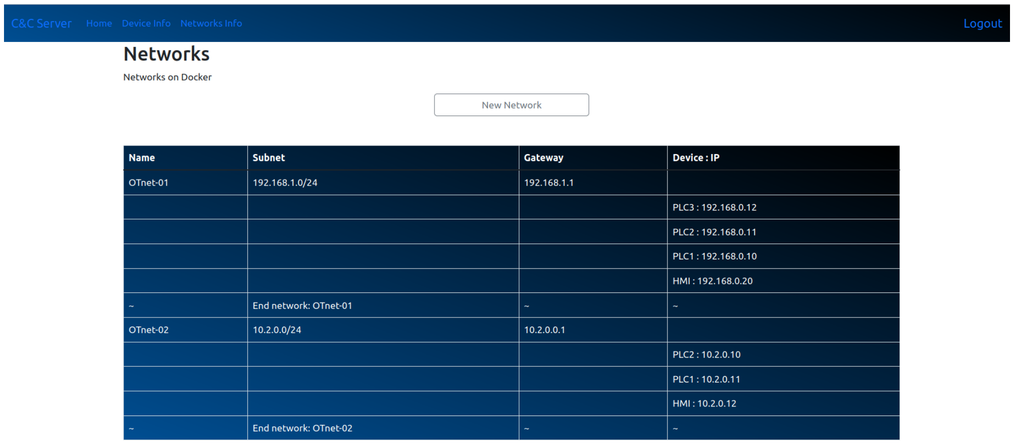


Devices

Name	Binding Ports (Host/protocol : Device)	Device Page	Start	Stop	Log
OTnet-01 PLC1	502/tcp:8502 6668/udp:6670 8080/tcp:8090	HTML Page	Start	Stop	Log
OTnet-01 PLC3	502/tcp:8504 6668/udp:6672 8080/tcp:8092	HTML Page	Start	Stop	Log
OTnet-01 PLC2	502/tcp:8503 6668/udp:6671 8080/tcp:8091	HTML Page	Start	Stop	Log
OTnet-01 HMI	502/tcp:8505 9090/tcp:9090	HTML Page	Start	Stop	Log
OTnet-02 PLC1	502/tcp:8502 6668/udp:6670 8080/tcp:8090	HTML Page	Start	Stop	Log
OTnet-02 PLC2	502/tcp:8503 6668/udp:6671 8080/tcp:8091	HTML Page	Start	Stop	Log
OTnet-02 HMI	502/tcp:8505 9090/tcp:9090	HTML Page	Start	Stop	Log

Figure 5.17: Device management interface: active and inactive honeypots with start and stop controls

The Networks Information page, depicted in Figure 5.18, is accessible from the server's main menu, offering a comprehensive view of the networks configured within the Docker environment. Users can gain insights into existing networks and also can add new networks, customising IP classes based on the planned deployment of devices on each network.



Networks

Networks on Docker

Name	Subnet	Gateway	Device : IP
OTnet-01	192.168.1.0/24	192.168.1.1	
			PLC3 : 192.168.0.12
			PLC2 : 192.168.0.11
			PLC1 : 192.168.0.10
			HMI : 192.168.0.20
~	End network: OTnet-01	~	~
OTnet-02	10.2.0.0/24	10.2.0.0.1	
			PLC2 : 10.2.0.10
			PLC1 : 10.2.0.11
			HMI : 10.2.0.12
~	End network: OTnet-02	~	~

Figure 5.18: Network management interface: list of all the networks

Log monitoring dashboard

In order to analyze the logs collected by the probes, we ran the pipeline displayed in Figure 5.19. Once the probes collect data, it undergoes processing through Zeek. Zeek utilises protocol analyzers, similar to Wireshark's dissectors, to generate JSON files containing organized logs. Subsequently, the script uploads these JSON files to Elasticsearch through an API call.

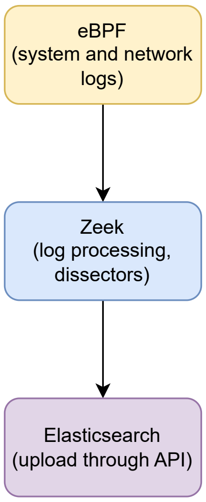
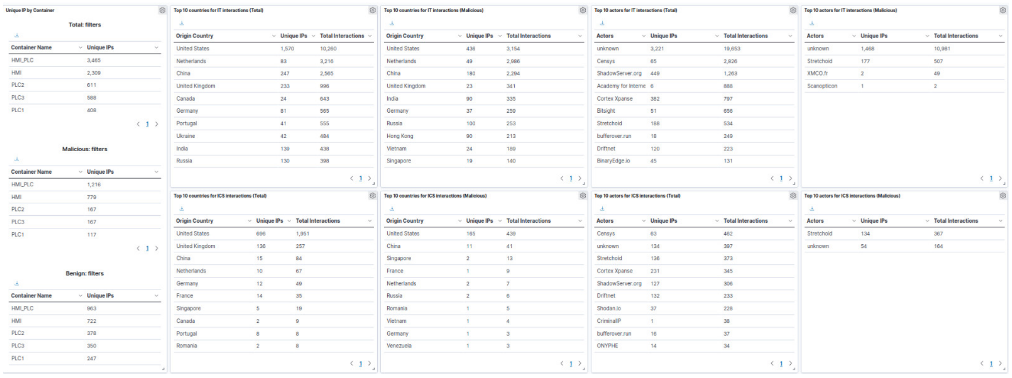


Figure 5.19: System and network logs flow

As shown in Figure 5.20, the Kibana dashboard is meant to display and analyze the logs in real-time. Each table shown in the dashboard was crafted to



Unique IP by Container

Total: filters

Container Name	Unique IPs
HMI_PLC	3,465
HMI	2,309
PLC2	611
PLC3	588
PLC1	408

Malicious: filters

Container Name	Unique IPs
HMI_PLC	1,216
HMI	779
PLC2	167
PLC3	167
PLC1	117

Benign: filters

Container Name	Unique IPs
HMI_PLC	963
HMI	722
PLC2	378
PLC3	350
PLC1	247

Top 10 countries for IT interactions (Total)

Origin Country	Unique IPs	Total Interactions
United States	1,570	10,260
Netherlands	83	3,216
China	247	2,565
United Kingdom	233	996
Canada	24	643
Germany	81	565
Portugal	41	555
Ukraine	42	484
India	139	438
Russia	130	398

Top 10 countries for IT interactions (Malicious)

Origin Country	Unique IPs	Total Interactions
United States	436	3,154
Netherlands	49	2,986
China	180	2,294
United Kingdom	23	341
India	90	335
Germany	37	259
Russia	100	253
Hong Kong	90	213
Vietnam	24	189
Singapore	19	140

Top 10 actors for IT interactions (Total)

Actors	Unique IPs	Total Interactions
unknown	3,221	19,653
Censys	65	2,826
ShadowServer.org	449	1,263
Academy for Interne	6	888
Cortex Xpanse	382	797
Bitsight	51	656
Stretchoid	188	534
bufferover.run	18	249
Driftnet	120	223
BinaryEdge.io	45	131

Top 10 actors for IT interactions (Malicious)

Actors	Unique IPs	Total Interactions
unknown	1,468	10,981
Stretchoid	177	507
XMCO.fr	2	49
Scanopticon	1	2

Top 10 countries for ICS interactions (Total)

Origin Country	Unique IPs	Total Interactions
United States	696	1,951
United Kingdom	136	257
China	15	84
Netherlands	10	67
Germany	12	49
France	14	35
Singapore	5	19
Canada	2	9
Portugal	8	8
Romania	2	8

Top 10 countries for ICS interactions (Malicious)

Origin Country	Unique IPs	Total Interactions
United States	165	439
China	11	41
Singapore	2	13
France	1	9
Netherlands	2	7
Russia	2	6
Romania	1	5
Vietnam	1	4
Germany	1	3
Venezuela	1	3

Top 10 actors for ICS interactions (Total)

Actors	Unique IPs	Total Interactions
Censys	63	462
unknown	134	397
Stretchoid	136	373
Cortex Xpanse	231	345
ShadowServer.org	127	306
Driftnet	132	233
Shodan.io	37	228
CriminalIP	1	38
bufferover.run	16	37
ONYPHE	14	34

Top 10 actors for ICS interactions (Malicious)

Actors	Unique IPs	Total Interactions
Stretchoid	134	367
unknown	54	164

Figure 5.20: Kibana dashboard

address the research questions that will be later discussed in Section 6.3.

Given that our primary focus wasn't to maintain real-time fidelity at 100%, we had the flexibility to incorporate additional analyses. Through a pre-processing script (middle block in Figure 5.19), we enhanced the original captured data with insights from tools like GreyNoise and VirusTotal. This enhancement enriched the information presented in the dashboard tables, allowing us to include details about the interaction's country of origin, involved actors, the malignancy or benign nature of these actors, Autonomous System information, and other relevant data sourced from public datasets. This integration was made possible by associating the IP addresses with the ones contained in the datasets.

Tuning Elasticsearch for precision and real-time performance

Elasticsearch is primarily designed for maintaining real-time performance, which can sometimes lead to imprecise calculations. To address this, we manually crafted SDL queries and adjusted the precision threshold parameter to 100, ensuring a balance between real-time responsiveness and data accuracy.

5.4 Innovative contributions and added value

This section explores the unique contributions of our honeypot implementation and its advancements compared to existing solutions in the field of ICS security and honeypot technology.

5.4.1 Communication protocols

Our system utilizes an industrial protocol to create noise on the network (elaborated on in Chapter 4). This integration creates a more realistic simulation environment, reflecting actual industrial communication standards and enhancing the fidelity of the honeypot for attacker behavior analysis.

5.4.2 Use of docker for containerization

While Docker is widely used for software deployment, our approach leverages it for several novel purposes:

- Isolated ICS Components: Each component of the honeypot, including PLCs, HMI, networking elements, and the physical plant simulation, is containerized. This isolation not only enhances security by preventing lateral movement within the honeypot but also allows for the easy replication and scaling of the testbed environment.

- Simulating Realistic Attack Scenarios: By leveraging Docker, we create a flat supervisory control network that appears transparent to attackers. This setup facilitates realistic man-in-the-middle (MITM) attacks, providing valuable insights into attacker behavior and potential defense mechanisms.

5.4.3 Integration of advanced monitoring tools

Our implementation integrates cutting-edge monitoring tools to provide comprehensive visibility into honeypot activity:

- Aquasecurity Tracee: Utilizing eBPF technology, Tracee collects detailed network logs and system calls in real-time. This deep visibility enables precise detection and analysis of attacks within the honeypot.
- ELK Stack for Real-Time Data Analysis: The Elasticsearch, Logstash, and Kibana stack is fine-tuned for precision and real-time performance. By adjusting the SDL queries and precision threshold, we ensure accurate and timely insights into honeypot activity, allowing for faster response to potential security incidents.

5.4.4 User-friendly management and monitoring interface

Our Command & Control (C&C) Server offers a user-friendly graphical user interface (GUI) that significantly enhances the usability of the honeypot:

- Network Configuration and Management: The GUI allows users to easily configure and manage networks within the Docker environment, customizing IP classes and adding new networks as needed. This streamlines the deployment and management process.
- Real-Time Monitoring and Control: Operators can interact with the honeypots through the Docker API, gaining real-time insights and control over the system. This facilitates a proactive approach to security by enabling operators to respond to suspicious activity promptly.

5.5 Deployment

The deployment of the honeynet is a crucial step, because if done wrong it could ruin the attractiveness of the honeypot and therefore make a well designed honeypot useless. As an example, a very important aspect to take into account is the public IP address the honeypot will use, as it serves as an identifier to extract valuable information. Utilizing tools like WHOIS, geolocation searches, and network location queries, attackers can glean insights into the origin and potential affiliations of the IP address. It is mandatory for a honeypot environment to mirror a realistic network scenario. For instance, in the deployment of an Industrial Control System (ICS) device honeypot such as a PLC, it is imperative to avoid associating it with an IP address belonging to a cloud provider like Amazon AWS for example. By carefully considering the information retrieved from an IP address, honeypot configurations can be tailored to emulate specific network conditions in order to enhance the authenticity of the simulated environment.

HoneyICS was deployed on a Dell PowerEdge T620 server, with an Intel Xeon E5-2640 v2 processor, featuring 16 CPUs, 128 GB of RAM. On the server was installed the VMware ESXi virtualization platform.

Three distinct virtual machines were configured. While Docker can run natively on the server, we opted for a virtualized approach for the following reasons: (i) *Isolation and Security*: Virtual machines provide an additional layer of isolation between the honeypot environment and the underlying server. This isolation enhances security by preventing potential compromises within the honeypot from affecting the host system. (ii) *Resource Management*: Utilizing separate VMs allows for efficient resource allocation. We can assign specific CPU, RAM, and storage resources to each VM based on its specific requirements.

The first virtual machine runs Docker, hosting the PLC, HMI, and proxy containers. This VM was equipped with 4 vCPUs, 8 GB of RAM, a disk of 200 GB, and 5-network adapter. The second virtual machine is dedicated to running Docker with the Matlab Simulink container, 4 vCPUs, 8 GB of RAM, a 200 GB disk, and a single-network adapter. Finally, the third virtual machine hosts Elasticsearch and Kibana. This system features 4 vCPUs, 12 GB of RAM, 500 GB disk, and a singular network adapter.

More precisely, ten docker containers were deployed over the first two VMs:

- a PLC container with an Allen-Bradley MicroLogix 1100 profile, with a web interface on port 8080 TCP (shown in Figure 5.13) and a Modbus server on port 502 TCP
- a PLC container with a Siemens S7-300 profile, with a web interface on port 8080 TCP (shown in Figure 5.13), a Modbus server on port 502 TCP and a S7comm server (using SNAP7 [51]) on port 102 TCP
- a PLC container with a Siemens S7-1200 profile, with a web interface on port 8080 TCP (shown in Figure 5.13), a Modbus server on port 502 TCP and a S7comm server (using SNAP7 [51]) on port 102 TCP
- three NGINX proxy containers, one for each PLC, these dedicated proxies offer several advantages such as security enhancements, NGINX proxies can be configured with additional security features to add an extra layer of protection for the honeypot and flexibility, independent proxies allow for finer-grained configuration and potential future modifications tailored to specific PLC profiles

- a container running an instance of ScadaBR (shown in Figure 5.16)
- a container running the simulated plant in Simulink (shown in Figure 5.3)
- a container running Kibana (shown in Figure 5.20)
- a container running Elasticsearch.

The three PLC containers were connected to our management dashboard for management purposes while the logs were uploaded to Elasticsearch and displayed on the Kibana dashboard using the pipeline described in Figure 5.19.

For each emulated ICS device (PLC or HMI) there is a specific image that is deployed in a dedicated Docker container for easy scalability and reconfigurability (several instances of the same image could be easily deployed in different containers). Images of PLCs share the same base structure and are differentiated by loading templates and configurations from mounted volumes. On the other hand, there is a unique device image for HMIs. In particular, the deployed honeynet instance comprises the following templates:

- a PLC Allen-Bradley MicroLogix 1100;
- a PLC Siemens S7-300;
- a PLC Siemens S7-1200;
- a ScadaBR HMI.

While a real industrial plant might primarily use PLCs from a single brand, our decision to include three different PLC profiles (Allen-Bradley, Siemens S7-300, Siemens S7-1200) was made to possibly expand the potential attacker pool. This broader range should cater to attackers targeting diverse ICS environments, potentially attracting a wider range of adversaries for a more comprehensive analysis of attacker behavior.

Table 5.2 summarizes all the configurations described before for the device instances within our deployment.

A diagram illustrating the deployed honeynet is presented in Figure 5.21, while Figure 5.22 details the individual elements used.

Device images run on a centralized host (*main host*) and are exposed through four *remote hosts*, each accessible via a different public IP address. It is worth noting that we exposed two different instances of the same HMI image on two different IP addresses, a dedicated IP address called IP_4 and an IP address in which a PLC is also exposed called IP_2 (see Table 5.2). This allows us to evaluate whether different deployment strategies have distinct outcomes in terms of attractiveness. The remote hosts are connected to the main host via a VPN; all traffic received on the public IP addresses is redirected via IPtables to the VPN interface, and directed to the IP addresses assigned to the Docker containers of the devices exposed on that IP. As an example, consider the scenario where *PLC1* is exposed on IP_1. When an attacker sends a packet to IP_1, the packet first enters the remote host interface named *vmbr0*. Subsequently, IPtables rules

Table 5.2: Device instances in the deployment

Instance	Model	Module type	Service	Port	IP
PLC1	AB M.logix 1100	1763-L16DWD	Modbus/TCP Webserver	502 8080	IP_1
PLC2	Siemens S7-300	CPU315-2PN/DP	Modbus/TCP Webserver Snap7	502 8080 102	IP_2
PLC3	Siemens S7-1200	CPU 1212C	Modbus/TCP Webserver Snap7	502 8080 102	IP_3
HMI1	ScadaBR	–	Webserver	8080	IP_4
HMI2	ScadaBR	–	Webserver	9090	IP_2

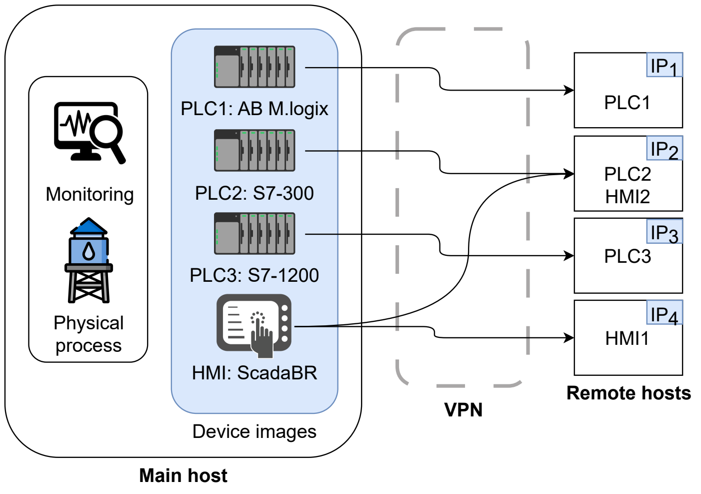


Figure 5.21: Deployment of the honeynet

guide the packet through the VPN virtual network interface, called *utun1*. The packet is then delivered to the internal IP address of *PLC1*. The response from *PLC1* follows the same path but in reverse. The main host also comprises the monitoring system of the honeynet and runs a simulation of the physical process.

The honeynet instance has been deployed on four contiguous IP addresses across the Italian Academic Consortium GARR [66]. GARR was chosen over more widely recognized cloud service providers, such as Amazon AWS, for two key reasons. First, universities and research institutions often legitimately uti-

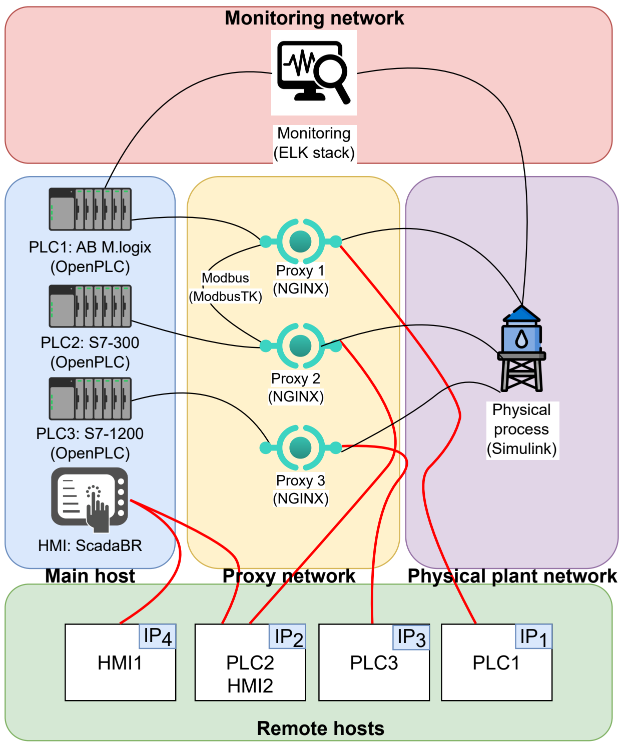
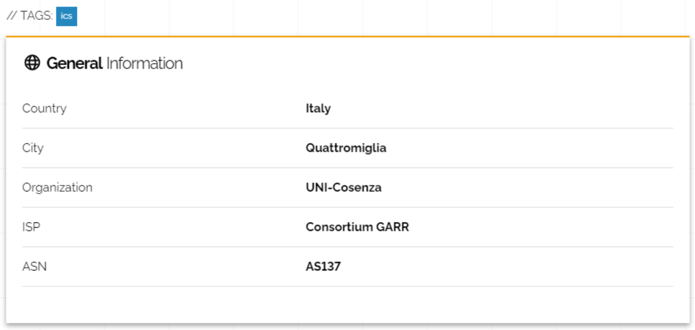


Figure 5.22: Block diagram of the honeynet's elements


// TAGS: ics

General Information

Country	Italy
City	Quattromiglia
Organization	UNI-Cosenza
ISP	Consortium GARR
ASN	AS137

Figure 5.23: Shodan results for one of the PLCs

lize PLCs within their networks. Second, GARR's lesser-known status compared to AWS makes it appear more like a residential or industrial network to an untrained attacker. After a few days since the deployment, the first three IP addresses have been tagged as *ICS* by Shodan 5.23 and *Scada* by Censys. Both Shodan and Censys assigned a generic tag to the fourth IP address. This categorization is due to the fact that the IP address exposes only a standard port (9090 TCP) running an HTTP service. This service, based on the port alone, could be associated with any web server and may not necessarily indicate the presence of a Scada HMI.

Chapter 6

DATA ANALYSIS

In this chapter, our analysis of the data collected by HoneyICS over three months of continuous Internet exposure is primarily drawn by the paper titled "ICS Honeypot Interactions: A Latitudinal Study" [114]. This work conducts a latitudinal study on a dataset encompassing both IT and OT interactions obtained from an ICS honeynet emulating OT devices exposed on the Internet.

Our exploration begins with an in-depth review of existing literature, seeking insights into previous instances of similar research, including the methodologies employed and the results obtained. It's worth noting that, we distinguish between IT interactions, involving protocols like HTTP, HTTPS, FTP, SSH, and others, and OT interactions, which revolve around protocols such as Modbus, S7Comm, Ethernet/IP, and the like. Our exploration of data analysis is guided by a set of research questions. The results section presents the findings derived from our data analysis. We report the patterns, anomalies, and trends identified within the attacks our honeynet received providing a comprehensive overview of the implications for security and potential areas of concern.

6.1 Related work

In the related works presented in Section 3.2, our focus was primarily directed towards the details of honeypot architecture and implementation, exploring the methodologies employed in their development. However, the scope of our exploration has now shifted to a distinct facet of our research: the analysis of the data these honeypots have collected. This discussion will revolve around a thorough examination and interpretation of the voluminous data collected over several days of continuous exposure to Internet traffic by various works. Additionally,

Table 6.1: Qualitative analysis of works affording data analysis on OT data. Legend: ●= Present, ◐= Partially present, ○= Not present

ICS honeypot	**Type**	**Interaction**	**Origin**	**Pattern**	**Duration**	**Dataset size**
Dodson et al. [87]	OT	◐	○	○	13 months	~200K
Ferretti et al. [91]	OT/IT	◐	●	○	4 months	4986
Cabana et al. [79]	OT	◐	◐	◐	1 year	1G
Mirian et al. [120]	OT	○	◐	○	10 weeks	5252
Serbanescu et al. [131]	OT/IT	◐	●	○	4 weeks	-
Vasilomanolakis et al. [136]	OT/IT	○	○	◐	12 weeks	-
Hilt et al. [98]	OT/IT	●	◐	◐	7 months	-
Navarro et al. [122]	OT/IT	◐	○	◐	2 years	-

we will analyze the methodologies used by these works to conduct the analysis, exploring the diverse approaches utilized in studying the data gathered.

While a significant number of ICS honeypot designs have been proposed in recent years (e.g., [71,92]), few studies delve deeply into analyzing the data these honeypots capture. This gap exists despite the potential for rich insights from network traffic monitors, such as honeypots, or network telescopes [121] deployed in unused portions of the internet address space. This limited coverage restricts our understanding of the impact that design and deployment decisions have on collected data and the dynamics of attackers' interactions with honeypots. Such analyses often rely on the specific properties of the deployed honeypot. It's important to emphasize that in this work, our central focus revolves around examining network communication protocols associated with the IT world and those associated with the OT world within the context of industrial control systems.

Specifically, our examination encompasses three orthogonal aspects of these interactions:

- *Level of Interaction:* We try to address how design choices, such as the supported protocols, the types of devices supported (PLCs and/or HMIs), the brands of devices utilized in the honeypot, and other deployment strategies, influence the honeypot's attractiveness in terms of the number of interactions received.
- *Origin of Interactions:* We scrutinize the geographic regions from which interactions originate, the actors involved, and explore possible associations between geographic regions and the nature of interactions.
- *Interaction and Attack Patterns:* Our focus extends to the identification of interaction and attack patterns within the honeypot data.

The detailed comparison and comprehensive overview of related works, considering these aspects, along with specific details about the datasets under consideration, are presented in Table 6.1.

6.1.1 Interaction analysis

While some studies [79, 87, 120] focus solely on examining OT interactions, others [91, 98, 122, 131, 136] conduct a comprehensive analysis analysing both OT and IT interactions, enabling a unified quantitative comparison between these two interaction types. For example, Serbanescu et al. [131] explored the attractiveness of various protocol combinations, including both industrially specific and those from the conventional IT realm. These investigations consistently reveal a substantial prevalence of IT interactions over OT interactions, typically around 10 times more. However, certain design choices remain underexplored. For instance, only a limited number of works [79,87,91,98] have tried to compare the attractiveness of different brands and/or models of controllers. Additionally, only works such as [98, 122] have considered HMIs among the exposed ICS devices.

6.1.2 Origin of interactions

The origin of interactions is explored thoroughly in [91,131]. In [91], IP addresses are categorized into distinct actors using DNS PTR records and Autonomous Systems. DNS PTR records, or Pointer records, are a type of Domain Name System (DNS) resource record that associates an IP address with a domain or hostname. These records are used in reverse DNS lookups, providing a mapping from an IP address to a corresponding domain or hostname. While an Autonomous System (AS) is a collection of IP networks and routers under the control of a single organization or entity that presents a common routing policy to the Internet.

For unknown actors, aggregation by country provides valuable insights. Similarly, Serbanescu et al. [131] analyze the IP addresses, associated GeoIP source country information (when available), and corresponding DNS PTR records (if any) for each peer interacting with their honeynet. In contrast, Miriam et al. [120] focus their analysis on identifying scanners within their honeypots. Hilt et al. [98] go a step further by deriving the reverse DNS of interacting IPs and conducting GeoIP lookups. Navarro et al. [122] contribute a unique perspective by providing a heat map of interactions and investigating IPs that potentially belong to the same actor.

6.1.3 Interaction and Attack patterns

Vasilomanolakis et al. [136] conducted an analysis of multi-stage attacks characterized by two or three requests within the same interaction, all originating from the same actor. In [98], the targeted honeypot faced complex IT attacks, encompassing malware designed for cryptocurrency mining and various forms of ransomware. Although the (physical) PLCs received valid command requests related to CPU functions from scanning tools, no discernible ICS-specific attack pattern was identified. Cabana et al. [79] employed machine learning, to classify the sources behind collected interactions and to identify threat actors such as botnets or malicious attackers. Meanwhile, Navarro et al. [122] focused on identifying coordinated attacks from different IPs associated with the same attacker. On a different note, a few works [87,91,122] solely tried to investigate actor behavior, including requests made in interactions and campaigns over time, without specifically exploring patterns associated with the identified attacks.

6.2 Thread intelligence services

From the table 6.2, we can observe that `GreyNoise` provides the most complete and diverse set of information about IP addresses such as on whether an IP address is malicious or not (*classification*), whether it is a registered Tor Exit Node (*TOR*), or whether it is part of a VPN provider service (*VPN*). In addition, `GreyNoise` provides information about the *actor* associated with an IP address, which can be a company, a search engine, or an individual and its *geographic location*. Information about the actor associated with an IP address is also provided by `VirusTotal` but this information is limited to the organization. `VirusTotal` provides other detailed information about the IP address, such as Whois metadata, which includes the Autonomous System Number and country, thus offering an accurate representation of the geographical location.

Although all services provide a *classification* of IP addresses, this information is provided in different forms. `GreyNoise` classifies IP addresses as *benign*, *malicious*, or *unknown*. In contrast, the other services provide numeric measurements or the amount of evidence collected by the service, which is difficult to interpret. For instance `IBM X-Force Exchange` provides a risk score from 0 to 10 based on the amount of spam coming from the IP address and other potentially malicious activities performed from the IP address, `AlienVault` provides the number of suspicious observations (called "pulses") in combination with information on the presence of the IP on whitelists, and `VirusTotal` provides the number of services that observed suspicious, malicious or harmless activities originating from the IP address.

Table 6.2: Data provided by top threat intelligence services

	Classification/Score	*Actor*	*Whois (ASN)*	*History*	*General location*	*Detailed location*	*Reverse DNS*	*VPN*	*TOR*
`GreyNoise` [96]	●	●	○	○	●	●	●	●	●
`AlienVault` [72]	●	○	○	●	●	○	○	○	○
`IBM X-Force Exchange` [72]	●	○	○	○	●	●	●	○	○
`VirusTotal` [138]	●	●	●	●	●	●	○	○	○

6.3 Research questions

Several honeypots have been proposed and deployed to investigate attack patterns within ICSs [92]. However, the focus of existing studies is mainly to demonstrate the credibility and attractiveness of proposed honeypots, while understanding and characterization of the observed interactions with the honeypot and attack patterns is still unclear. Moreover, the attack surface exposed by several honeypots is rather limited, narrowing the scope of the interactions that can be studied. To address this gap, we analyze the network traffic gathered using HoneyICS. This analysis is guided by the following research questions (summarized in Table 6.3), which delve into three key areas: the level of interaction with the honeypot, the origin of these interactions, and the interaction and attack patterns observed.

6.3.1 Level of interaction

Although an analysis of the requests captured by the honeypot can provide useful information about the entities interacting with the honeypot, several requests might come from the same entity as part of the same interaction with the honeypot. Capturing these interactions as a whole can give additional information about the goals of the entity interacting with the honeypot. For instance, the size and patterns of a single interaction can tell us whether it is a single request to check service availability, a script executing some specific routine, a human user manually trying various actions, etc. Our first question aims to provide a first indication of the level of engagement with the honeynet, distinguishing between OT interactions, via the industrial protocols Modbus/TCP and S7comm, and IT interactions, via the protocol HTTP.

RQ1: *To what extent are ICS honeypots involved in OT interactions compared to IT interactions?*

Our second research question aims to understand whether OT interactions carried via Modbus/TCP or S7Comm have a different level of complexity (measured in terms of the number of requests forming the interaction) compared to IT interactions via HTTP. For instance, cyber-physical attacks aiming at altering the runtime evolution of the physical process may require extensive interactions with PLCs to manipulate actuator commands and monitor the process by accessing sensor measurements. We therefore ask:

RQ2: *To what extent do ICS honeypots attract complex OT interactions compared to IT interactions?*

The deployment strategy of a honeynet (e.g., IP address configuration, network topology) might affect the quantity and nature of the interactions it attracts. A critical choice for the believability of a honeynet concerns the deployment of the HMI. Indeed, the HMI is typically accessed via HTTP, which does not necessarily characterize the corresponding IP as part of an ICS. To evaluate such a choice, in our deployment, we exposed the HMI both on an IP address in which a PLC was also exposed and on a dedicated IP address without any ICS-specific tag from Shodan or Censys (cf. Section 5.5). The following question aims to understand whether an HMI exposed on an "ICS" or "Scada" IP address is more or less attractive when compared to an HMI exposed on a generic IP address.

RQ3: *To what extent does an HMI exposed on the same IP address of a PLC attract IT interactions compared to an HMI exposed on a dedicated IP address?*

Another relevant choice regards the PLCs to be employed in a honeypot as different brands/models of PLCs might have different attractiveness. For instance, certain models of specific brands could have known unpatched vulnerabilities (e.g., Siemens S7-300 involved in the Stuxnet attack [89]) and, thus, be more attractive for an attacker.

RQ4: *To what extent does a specific brand of PLC attract interactions compared to other brands?*

6.3.2 Origin of interactions

Characterizing the origin of interactions can shed light on the entities targeting ICSs and, consequently, on the potential threat landscape [92]. We aim to understand whether there is an interest in both IT and OT systems or whether these two types of technology are targeted by different actors. The following two questions aim to characterize the origin of OT and IT interactions in terms of geographic location and actors initiating those interactions.

RQ5: *From which geographic regions do the observed OT interactions originate compared to IT interactions?*

RQ6: *From which actors do the observed OT interactions originate compared to IT interactions?*

By leveraging the increased connectivity between IT and OT systems, a cyber-attack on one system can potentially impact the other [92]. For example, a cyber-attack on an IT system could cause disruptions in the OT network controlling the physical process. The next question aims to explore the relationship between OT and IT interactions, categorizing these interactions by the actors. This allows us to detect whether attackers pursue various paths to infiltrate an OT system or potential movement from one system to the other [71].

RQ7: *To what extent actors originating from the same IP address perform both OT and IT interactions?*

6.3.3 Interaction and attack patterns

The growing convergence of OT and IT systems emphasizes the need to understand the patterns and characteristics of their interactions. Gaining such insights can shed light on potential vulnerabilities, threats, and the behavior of cyber actors when targeting these systems. The following research questions delve into the specific patterns emerging from the observed OT and IT interactions, pointing out possible exploits in the OT setting.

RQ8: *Which pattern or characteristic do the observed IT interactions exhibit?*
RQ9: *Which pattern or characteristic do the observed OT interactions exhibit?*
RQ10: *To what extent the observed OT and IT interactions exploit known vulnerabilities?*

6.4 Methodologies

This section presents the methodology employed in addressing the research questions presented in Section 6.3. A robust and well-structured methodology is crucial in ensuring the reliability and validity of our findings. We outline the steps taken to gather, analyze, and interpret the data, emphasizing the rationale behind each choice.

6.4.1 Dataset

The dataset was built using all the requests directed at the deployed instance of HoneyICS, as detailed in Section 5.5, spanning a three-month duration from June 4 to September 4, 2023. We collected a dataset encompassing a total of

6324 OT requests. Notably, within this dataset, we observed 4216 requests categorized as S7comm interactions, 2108 requests as Modbus/TCP interactions and a substantial count of 36486 requests on HTTP.

Table 6.5 provides a detailed breakdown of the number of requests received for each PLC and HMI in our honeynet. The table is organized into two main categories: "well-formed" and "bad" requests, further subcategorized by the type of interaction, including HTTP, Modbus/TCP, and S7.

We consider *well-formed* requests those that align with the protocol assigned to the destination port and *bad* requests, which deviate from the expected protocol for the designated port. For instance, a *well-formed* request for Modbus/TCP adheres to the appropriate protocol for the assigned port, whereas a *bad* request in this context might involve an HTTP request directed to the Modbus/TCP port 502.

Our analysis focused exclusively on *well-formed* requests, as *bad* requests, by definition, do not constitute valid interactions with the honeynet. By excluding *bad* requests, which deviate from the expected communication protocols assigned to the destination ports, we ensure that our evaluation centers on genuine and protocol-compliant interactions.

Each request in our dataset is characterized by a set of common features, including source IP, destination IP, timestamp, source port, and destination port. Additionally, to provide a detailed understanding of the interactions, we considered specific features tailored to the respective protocols:

- For S7comm interactions:
  - Function codes
  - Data addresses
- For Modbus/TCP interactions:
  - Unit identifiers
  - Function codes
- For HTTP interactions:
  - Request methods
  - Body content
  - URL paths

6.4.2 Threat Intelligence Gathering

To enhance our dataset, we integrated threat intelligence data from two widely used services: `GreyNoise` [96] and `VirusTotal` [138], as described in Section 6.2. Specifically, `GreyNoise` was employed to gather information about the requests and the IP addresses' origins, including details about the *actor*, *organization*, *country*, and `Virustotal` for *whois* metadata. This data allows for the generation of a detailed profile of threats. The threat intelligence data were retrieved using the public APIs provided by GreyNoise, along with the VirusTotal academic API.

From GreyNoise, we obtained information about the *actor* associated with an IP address, which could be a company, search engine, or individual, along with its *geographic location*. While VirusTotal provided information about the organization associated with an IP address, it also offered additional details through its Whois metadata, including the Autonomous System Number and country, providing an accurate representation of the geographical location.

Classification

To determine whether an IP address belongs to a *malicious* actor, we relied on the *classification* provided by GreyNoise since it is based on concrete evidence of malicious and benign activities. Moreover, this service periodically audits and reevaluates its classifications to ensure it matches the actual behavior observed on the Internet [96]. Thus, GreyNoise provides a solid foundation for classifying the IP addresses in our dataset.

The requests in our datasets originated from 4874 unique IP addresses. Table 6.4 reports the classification of these IP addresses based on the information retrieved from GreyNoise.

It is worth noting that GreyNoise did not provide a clear-cut classification for all IP addresses, where 779 were classified *unknown*, indicating that no concrete evidence of either malicious or benign behavior was observed for those IP addresses. In addition, we did not obtain any information for other 330 IP addresses (*blank*), indicating that they have not been captured by the sensors used by GreyNoise to monitor the Internet. We did not use the other services to fill GreyNoise's lack of information as they provide numeric measurements or the amount of evidence collected by the service, which is difficult to interpret. Thus, in our analysis, we answer the research questions presented in Section 6.3 with respect to the overall IP addresses.

Actor

We retrieved the information about the actor associated with an IP address from GreyNoise. According to GreyNoise, an actor can be a company, a search engine, or an individual. In case this information was not available for an IP address, we extracted the organization associated with that IP address from VirusTotal.

6.5 Results

Our analysis yields key insights, summarized as follows:

- Dominance of IT Interactions: IT interactions significantly outnumber OT interactions, emphasizing their prevalence. Despite this, OT interactions exhibit greater complexity.
- Specialized Actors in OT interactions: Actors specifically engaging with OT devices using industrial protocols demonstrate a higher level of knowledge and sophistication. These individuals leverage tools like Nmap to gather valuable information, suggesting the potential for targeted attacks.
- Protocol-Dependent Patterns: The patterns of OT interactions and attacks are strongly influenced by the characteristics of the targeted industrial network protocol. These patterns notably differ from those commonly observed in the IT-C realm.
- Prompt Exploitation of ICS Vulnerabilities: Skilled actors swiftly exploit vulnerabilities specific to OT devices, such as HMI, aiming to take control of the targeted OT devices.

Therefore, based on these discerned IT/OT patterns, we assert the imperative need for a multi-faceted defense approach to enhance OT security, as further discussed in Section 8.

6.5.1 Level of Interaction

RQ1: To what extent are ICS honeypots involved in OT interactions compared to IT interactions?

Table 6.6 (last column) reports the number of IT and OT interactions observed in the data collection period, where IT interactions include the HTTP requests targeting the HMIs and the PLCs' web interfaces and OT interactions include the Modbus and S7comm requests received by the PLCs. We can observe that IT interactions are roughly ten times more than OT interactions (299 IT interactions per day vs. 30 OT interactions per day on average). Among these interactions, 51% of the observed IT interactions are malicious (154 interactions per day), whereas only 27% of the OT interactions are malicious (8 interactions per day). This suggests that IT interactions are still predominant compared to OT ones even in an industrial control system setting.

In the context of Table 6.6, we define a malicious interaction as one classified as malicious by GreyNoise based on its associated tags. These tags capture behaviors GreyNoise has directly observed an IP address engage in. Some of these tags are classified as "malicious" for demonstrably harmful behaviors.

GreyNoise [96] classifies IPs into two categories:

- Malicious Classification: An IP address is classified as malicious if it has at least one malicious tag associated with it and is not classified as benign.

- Benign Classification: GreyNoise assigns a benign classification to IPs associated with legitimate entities such as search engine crawlers, universities, or security research organizations. A benign classification overrides all associated malicious tags.

GreyNoise's classification system offers valuable insights into the potential maliciousness of interacting IPs. However, it's crucial to acknowledge inherent limitations. GreyNoise might not have observed all malicious IPs or the full spectrum of their activities. Additionally, the classification can generate false positives, where seemingly suspicious activity from a specific IP could be benign, such as a legitimate security researcher performing a scan. Therefore, while GreyNoise classification informs our analysis, it's essential to recognize these limitations.

RQ2: To what extent do ICS honeypots attract complex OT interactions compared to IT interactions?

From Table 6.6, we can observe that IT interactions typically comprise a single request, although there are a few IT interactions comprising up to 100 requests. On the other hand, OT interactions are, in general, more complex, with 25% comprising at least 6 requests. Malicious interactions follow a similar trend, although they appear to be less complex.

RQ3: To what extent does an HMI exposed on the same IP address of a PLC attract IT interactions compared to an HMI exposed on a dedicated IP address?

Table 6.7 shows the number of interactions with the PLCs and HMIs. We observed that the HMI exposed on a dedicated IP (HMI1) attracted more interactions than the HMI exposed together with a PLC (HMI2). In the first case, HMI1, we observed 9623 interactions, whereas for HMI2, we observed 2710 interactions. It is worth noting that the number of malicious interactions is significantly higher (ten times more) in the first setting, i.e., when the HMI is exposed on an IP address with no "ICS" or "Scada" tag. This suggests that malicious IT actors show less interest in IPs with ICS/Scada tags.

RQ4: To what extent does a specific brand of PLC attract interactions compared to other brands?

From Table 6.7, we observe that the PLCs in the honeynet attracted a comparable number of IT interactions, regardless of the brand. As regards OT interactions, Siemens PLCs (PLC2 and PLC3), which support both Modbus/TCP and S7comm protocols, generally received more interactions than the Allen-Bradley MicroLogix 1100, which only supports Modbus/TCP. Notably, when focusing on Modbus/TCP, the Micrologix 1100 received approximately 20% more interactions when compared to the two Siemens PLCs.

6.5.2 Origin of Interactions

RQ5: From which geographic regions do the observed OT interactions originate compared to IT interactions?

Table 6.8 and Table 6.9 present an overview of the countries with the largest number of IT and OT interactions, respectively. We observe that the United States is the predominant source of IT interactions, with 1421 unique IP addresses (including 602 malicious ones) contributing to a total of 8978 interactions. Other significant contributors include the Netherlands, China, Germany and Romania. Similarly, the United States is the primary source of OT interactions, with 512 IP addresses (including 213 malicious ones) generating 1602 interactions. Notably, the total number of IP addresses involved in OT interactions is significantly lower, indicating that IT actors are still predominant. A small number of IP addresses (mostly malicious ones) from Hong Kong and India are responsible for a significant number of interactions. Notice that while the tables focus on the top 10 countries with more interactions, regional data provide extra insight. For instance, in Lithuania 5 IP addresses (3 of which malicious) were responsible for 443 IT interactions (260 from malicious IPs); in California 11 IP addresses were responsible for 417 interactions (both OT and IT ones); a single IP in South Korea was responsible for 58 IT interactions.

RQ6: From which actors do the observed OT interactions originate compared to IT interactions?

Table 6.10 and Table 6.11 present an overview of the actor that had the largest number of IT and OT interactions, respectively. Among those actors, Aggros Operations Ltd. stands out as the most active actor involved in IT interactions, contributing to around 30% of the interactions in the top 10 IT actors. It is worth noting that the IPs associated with Aggros Operations Ltd., China Mobile Comm. Group, DigitalOcean LLC, and Stretchoid are predominantly malicious. On the other hand, Stretchoid emerges as the most active actor involved in OT interactions, contributing to about 38% of the interactions of the top 10 OT actors; this actor is also responsible for a significant number of IT interactions from malicious IPs. Censys is the second most common actor for both IT and OT interactions, accounting respectively for approximately 15% and 13% of the total interactions.

RQ7: To what extent actors originating from the same IP address perform both OT and IT interactions?

Table 6.12 provides an overview of actors involved in both IT and OT interactions, highlighting malicious interactions. While Table 6.6 indicates that in general IT interactions are ten times more than OT interactions, Table 6.12 shows that this ratio is significantly lower when focusing on IP addresses involved in both kinds of interactions. The analysis reveals a balanced engagement by some actors originating from the same IP address, with approximately 50% of their interactions categorized as both OT and IT. This observation is especially true for those actors involved in malicious interactions only, such as Stretchoid, CT-HangZhou-IDC, and Hurrican Elec.

6.5.3 Interaction and Attack Patterns

RQ8: Which pattern or characteristic do the observed IT interactions exhibit?

Table 6.13 presents a summary of the observed attack patterns in the recorded IT interactions, categorized by vulnerability class, along with examples of detected payloads pertinent to each class. The majority of detected patterns are HTTP Connect Proxy attacks. More precisely, we observed 6642 interactions directed towards both the web interface of PLCs and the HMI's ScadaBR software, aiming to exploit an HTTP Connect Proxy attack for unauthorized tunneling and denial of service. Notably, these interactions did not originate from a single malicious actor but were initiated by a diverse group of actors, each attempting to establish connections to various external domains such as myipb1a.mrrage.xyz:80, www.shadowserver.org:443 and ip138.com. The other recorded interactions encompass a wide range of attack types, including command injections, file inclusions, SQL injections (SQLi), and Server-side template injections (SSTI). Interestingly, these attack patterns were rarely specifically tailored for the webservers, indicating that adversaries often engage in blind attacks without checking what web application is installed. Nonetheless, we did identify a small number of attempts to exploit a very well-known vulnerability of the ScadaBR software and other known vulnerabilities, which are discussed later on.

RQ9: Which pattern or characteristic do the observed OT interactions exhibit?

An analysis of *Modbus/TCP* interactions shows that only 32 out of 1927 the observed interactions (cf. Table 6.7) were composed of requests with legal function codes as also observed in other works [91, 120]. These 32 interactions include exactly one request; an overview of the identified patterns is reported in Table 6.14. Even though the set of interactions under consideration is rather small (32 interactions of length 1), we observed some interesting cases. For instance, we noticed an interaction originating from Constantine Cybersecurity Ltd. (AS211298) comprising a *Write Single Register* request on PLC3 executed immediately after an IT interaction with the associated PLC web interface. Such interaction is relevant because, within this context, the *Write Single Register* is used as a fuzzing technique. The actor submits a purposely malformed input (e.g., an illegal data address packet) to the system aiming to induce a crash.

Moreover, we observed 18 interactions using the *Read Holding Registers* and *Read Input Registers* functions, which allow an actor to access the internal registers of PLCs, thereby providing insights into the current state of the system. We also noticed three interactions comprising a single *Read Coils* request; these interactions originated from a malicious IP address linked to Alibaba.com Singapore E-Commerce Private Ltd. (AS45102) and targeted all PLC devices of our honeynet. Finally, we observed an interaction using the *Read Coils* function from a malicious IP linked to "Linode.com" (AS63949) addressed to PLC1. Interactions that exploit *Read Coils* are interesting since they indicate that these actors engage in reconnaissance and scanning activities. After a successful attack, the actor might use the acquired knowledge to manipulate coil statuses leading to a denial of service. For completeness, note that we identified 1895 Modbus/TCP interactions with illegal function codes. Most of them (95%) were composed of a single illegal request with function code 43 and subcode 14 (Read Device Identification). This specific code denotes an Encapsulated Interface Transport Modbus function with MEI type equal to Read Device Identification and could be the result of a Nmap scan using the modbus-discover.nse script. The remaining interactions are more complex (up to four requests) and include requests with function code 17 (*Report Slave ID*) as well as function code 90 (*Unity*), which is proprietary to Schneider Electric devices. These kinds of interactions had targeted all exposed PLCs, indiscriminately.

The analysis of the *S7comm* protocol (reported in Table 6.15) shows two consistent patterns for the two PLCs of our honeynet supporting the protocol. We received 42 interactions with one S7 Communication Setup request (used to establish an S7 connection) followed by three consecutive Read SZL requests (used to collect information about the device) and 419 interactions exhibiting one Communication Setup request and two consecutive Read SZL requests. The analysis of the pattern shows that the actors systematically attempted to retrieve the status of the Siemens PLCs and to collect information about the firmware version, module type, hardware specifications, and system name. All interactions targeting our PLCs that did not originate from a known scanner service were performed using the script s7-info.nse, a Nmap script used to enumerate Siemens S7 PLC devices.

RQ10: To what extent the observed OT and IT interactions exploit known vulnerabilities?

We observed attempts of vulnerability exploitation only related to IT interactions. An analysis of the IT interactions targeting the HMI1's web interface revealed 19 interactions that exploited the ScadaBR CVE-2021-26828. This vulnerability allows remote authenticated users to upload and execute arbitrary JSP files via view_edit.shtm, leading to a remote code execution. Among these interactions, 12 were traced back to an IP address affiliated with "Mullvad.net" (AS6233), 5 to "PenTeleData House Account" (AS3737), and the remaining two to "Datacamp Limited" (AS212238). Interestingly, both the first and third actors used a VPN connection. Next, another interesting common characteristic of these two actors was their attempt to connect to a specific IP address associated with "ALEXHOST SRL" (AS200019) to download malware. In the context of the exploitation process, it is crucial to note that the ScadaBR CVE-2021-26828 vulnerability requires successful authentication. Consistent with this prerequisite, we registered seven login attempts from the three actors in question, who exploited a weak credentials vulnerability to get the JSESSIONID token necessary for executing the RCE. In another case, we observed four different interactions from "Neterra Ltd." (AS44477) that targeted our exposed devices on HTTP. Specifically, the attack pattern was a blind attempt to exploit a very old php-CGI injection known vulnerability against PHP web servers and associated with CVE-2012-2336. Additionally, we observed a pattern occurring in 74 interactions targeting all devices through HTTP. These interactions attempted to upload files such as "Mozi.m" and "Mozi.a", which are associated with the well-known Mozi malware family and CVE-2018-10561, CVE-2018-10562 targeting GPON Routers. Similarly, we discovered eight interactions originating from "Interserver, Inc" (AS19318) and exploiting a recent remote command execution vulnerability, specifically CVE-2023-1389, which targets TP-Link routers.

6.5.4 Contributions to ICS Honeypot Data Analysis

Building upon the foundation established by prior research (as outlined in Section 6.1), this work makes several key contributions that advance the state-of-the-art in ICS honeypot data analysis.

We analyze the complexity of ICS interactions (RQ2), revealing that they often involve multiple requests, which provide valuable insights into potential attacker strategies.

Additionally, we investigate the influence of HMI placement on IT interactions (RQ3). Our findings demonstrate that isolating HMIs on dedicated IP addresses reduces interactions compared to co-locating them with PLCs, offering a crucial step towards hardening industrial control system security.

Finally, we move beyond the focus on interaction volume seen in prior research by examining brand-specific interaction patterns for PLCs (RQ4). This analysis highlights that Siemens PLCs supporting both Modbus/TCP and S7comm protocols attract more interactions, and the brand itself can influence the type of interactions observed. These findings offer valuable insights that go beyond simply confirming the dominance of IT interactions. They shed light on the evolving tactics of attackers targeting industrial control systems, emphasizing the need for a multi-faceted defense strategy that considers interaction complexity, HMI placement, and brand-specific vulnerabilities.

Table 6.3: Research Questions

Category	**Research Question**	**Description**
Level of Interaction	RQ1: To what extent are ICS honeypots involved in OT interactions compared to IT interactions? RQ2: To what extent do ICS honeypots attract complex OT interactions compared to IT interactions?	Quantifies OT vs. IT interactions for initial engagement level assessment. Investigates complexity (number of requests) in OT vs. IT interactions.
HMI Deployment	RQ3: To what extent does an HMI exposed on the same IP address of a PLC attract IT interactions compared to an HMI exposed on a dedicated IP address?	Evaluates the impact of HMI deployment strategy on IT interaction attraction.
PLC Brand	RQ4: To what extent does a specific brand of PLC attract interactions compared to other brands?	Analyzes brand-specific attractiveness for potential vulnerabilities.
Origin of Interactions (Geography)	RQ5: From which geographic regions do the observed OT interactions originate compared to IT interactions?	Characterizes geographic distribution of actors targeting OT vs. IT systems.
Origin of Interactions (Actors)	RQ6: From which actors do the observed OT interactions originate compared to IT interactions?	Identifies potential differences in actors targeting OT vs. IT systems.
Relationship between OT and IT Interactions	RQ7: To what extent do actors originating from the same IP address perform both OT and IT interactions?	Examines potential coordinated attacks involving both OT and IT interactions.
Interaction and Attack Patterns (IT)	RQ8: Which pattern or characteristic do the observed IT interactions exhibit?	Analyzes specific patterns and characteristics of observed IT interactions.
Interaction and Attack Patterns (OT)	RQ9: Which pattern or characteristic do the observed OT interactions exhibit?	Analyzes specific patterns and characteristics of observed OT interactions.
Exploit Detection	RQ10: To what extent do the observed OT and IT interactions exploit known vulnerabilities?	Identifies potential exploit utilization within the captured interactions.

Table 6.4: `GreyNoise` classification of the observed IP addresses

Classification	**#IPs**	**#Requests**
benign	1433	21916
malicious	2332	14629
unknown	779	8360
blank	330	17874
Total	4874	62779

Table 6.5: Requests received per device

Devices	**well-formed**			**bad**		
	HTTP	**Modbus/TCP**	**S7**	**HTTP**	**Modbus/TCP**	**S7**
PLC 1	5408	688	–	0	25	–
PLC 2	5198	691	1951	0	24	136
PLC 3	6640	660	2005	0	20	124
HMI 1	11389	–	–	0	–	–
HMI 2	7851	–	–	0	–	–
Total	36486	2039	3956	0	69	260

Table 6.6: IT and OT interactions and their complexity

Interactions	**Min**	**Q1**	**Median**	**Q3**	**Max**	**Count**
IT (total)	1	1	1	1	100	27525
IT (malicious)	1	1	1	1	20	14194
OT (total)	1	1	1	6	10	2561
OT (malicious)	1	1	1	2	10	696

Table 6.7: Number of interactions on the exposed devices

Devices	**Total**			**Malicious**		
	HTTP	**Modbus/TCP**	**S7**	**HTTP**	**Modbus/TCP**	**S7**
PLC1	5187	721	–	2884	189	–
PLC2	5004	602	333	2714	202	84
PLC3	5001	604	344	2660	174	92
HMI1	9623	–	–	5380	–	–
HMI2	2710	–	–	556	–	–
Total	27525	1927	677	8814	565	176

Table 6.8: Top 10 countries for IT interactions

Country	**Total**		**Malicious**	
	#IP	**#Interactions**	**#IP**	**#Interactions**
United States (US)	1421	8978	602	3356
Netherlands (NL)	201	4550	113	3436
China (CN)	333	2753	257	2569
Germany (DE)	116	1201	56	375
Romania (RO)	17	934	11	687
United Kingdom (GB)	285	903	34	207
Italy (IT)	77	822	21	71
Hong Kong (HK)	155	642	139	316
India (IN)	253	573	131	343
Ukraine (UA)	59	530	45	93

Table 6.9: Top 10 countries for OT interactions

Country	**Total**		**Malicious**	
	#IP	**#Interactions**	**#IP**	**#Interactions**
United States (US)	512	1602	213	564
United Kingdom (GB)	138	263	–	–
Germany (DE)	29	143	2	5
Netherlands (NL)	22	105	6	26
Taiwan (TW)	63	91	–	–
Brazil (BR)	62	88	–	–
Belgium (BE)	55	87	–	–
China (CN)	17	86	15	59
France (FR)	14	35	1	9
Singapore (SG)	6	20	2	13

Table 6.10: Top 10 actors/org involved in IT interactions

Actor/org	**Total**		**Malicious**	
	#IP	**#Inter.**	**#IP**	**#Inter.**
Aggros Operations Ltd.	51	3007	47	2996
Censys	65	2826	-	-
China Mobile Comm. Group	73	1903	72	1902
FranTech Solutions	38	1285	16	1152
ShadowServer.org	449	1263	-	-
DigitalOcean LLC	268	1087	170	783
Unmanaged LTD	12	899	7	656
Academy for Internet Research	6	888	-	-
Cortex Xpanse	382	797	-	-
Stretchoid	279	782	277	777

Table 6.11: Top 10 actors/org involved in OT interactions

Actor/org	**Total**		**Malicious**	
	#IP	**#Inter.**	**#IP**	**#Inter.**
Stretchoid	185	502	180	488
Censys	63	462	-	-
Cortex Xpanse	231	345	-	-
ShadowServer.org	127	306	-	-
Driftnet	132	233	-	-
Shodan.io	35	196	-	-
Hurricane Electric LLC	18	40	18	40
CT-HangZhou-IDC	2	38	2	38
CriminalIP	1	38	-	-
bufferover.run	16	37	-	-

Table 6.12: Top 10 actor/org involved in both OT and IT interactions

Actor/org	**#IP**	**OT interactions**		**IT interactions**	
		Total	**Malicious**	**Total**	**Malicious**
Censys	58	449	-	2776	-
Cortex Xpanse	183	278	-	428	-
ShadowServer.org	105	265	-	392	-
Driftnet	62	98	-	123	-
Shodan.io	13	67	-	58	-
Stretchoid	19	50	50	62	62
CT-HangZhou-IDC	2	38	38	143	143
bufferover.run	16	37	-	222	-
Acad. Int. Res.	4	32	-	667	-
Hurricane Elec.	13	32	32	30	30

Table 6.13: A summary of observed IT patterns by vulnerability class

Vulnerability class	**#Interactions**	**Payload example**
HTTP Connect Proxy	6642	CONNECT myipb1a.mrrage.xyz:80
Command injection	139	/cgi-bin/luci/...&country=$(cd /tmp;rm zizuo.sh...
File inclusion	60	/cgi-bin/../../../../etc/passwd
SSTI	2	id=%{{{11}}*{{11}}}
SQLi	1	'+union+select+(select+concat(0x223e3...

Table 6.14: Observed Modbus/TCP interaction patterns

Legal function codes	**#Interactions**	
	Total	**Malicious**
Read Holding Registers	14	5
Read Discrete Inputs	9	0
Read Input Registers	4	0
Read Coils	4	4
Write Single Register	1	0

Table 6.15: Observed S7 interaction patterns

S7comm Function Pattern	**#Interactions**	
	Total	**Malicious**
Comm. Setup + 3 Consecutive Req. Read SZL	42	5
Comm. Setup + 2 Consecutive Req. Read SZL	419	145

Chapter 7

SUPPORTED ATTACKS

In this chapter, we explore a series of proof-of-concept cyber-physical attacks designed to exploit vulnerabilities in our honeynet, specifically targeting the PLCs and HMIs. Each attack scenario is crafted to showcase potential risks and vulnerabilities within the deployed system. Each scenario provides a detailed narrative of the attack methodology and its potential impact on the system.

Additionally, we introduce an automated tool developed to execute the outlined attacks. This tool utilizes specific technologies for crafting Modbus and ARP packets, enabling sophisticated attacks on the ICS. We describe the functionalities of the tool, such as the tool's ability to modify inputs, outputs, and other register data within the PLC, as well as its integration with interception tools for Modbus request frames.

7.1 Proof of concept attacks

In our evaluation of HoneyICS against the attacker model outlined in Section 4.2, we conducted a set of simulations replicating three distinct attacks, drawing inspiration from the annual Critical Infrastructure Security Showdowns (CISS) on the SWaT system. Our use case implementation relies on the Modbus protocol as the underlying ICS communication protocol. Modbus maps temporary memory within PLC programs to discrete output coils, discrete input contacts, analog input registers, and analog output holding registers. Our methodology was based around the execution of three attacks, summarized in Table 7.1, each characterized by specific parameters, including the attack vector, the intended goals of the attacker, specific targets, and the level of observability or stealthiness when scrutinizing SCADA Human Machine Interface (HMI) interfaces. To assess the impact and efficacy of these attacks, we used our management

Table 7.1: Summary of Attacks

Attack Type	**Target**	**Objective**
DoS on Pump P-101	Pump P-101 (governed by PLC-1)	Achieve tank overflow of T-201
MITM Attack with HMI Compromise	HMI (ScadaBR)	Drag the system into a deadlock
Stealthy DoS on Pump P-102	Pump P-102 (governed by PLC-3)	Achieve pump failure

Table 7.2: Summary of Attack 1

Attack Type	Denial of Service (DoS)
Target	Pump P-101 (governed by PLC-1)
Objective	Achieve Tank Overflow of T-201
Exploited	Modbus TCP
Function Codes	- Read Analog Input Register (0x04) - Write Single Discrete Output Coil (0x05)
Steps	1. Continuously transmit Modbus packets to read tank T-101 level. 2. When level reaches $high_1$, transmit Modbus packet to manipulate pump P-101. 3. Achieve tank overflow independently of valve MV-301 state.

and monitoring dashboard. This tool helped us to visualise simultaneously the HMI interface and the real-time dynamics of the simulated system under attack. Additionally, we collected and analyze the logs from the probes within the honeynet, to provide insights into the sequential progression of malicious activities. This evaluation framework sets the stage for the subsequent detailed exploration of the three supported attacks, shedding light on HoneyICS's robustness and detection capabilities.

7.1.1 Tank overflow via DoS attack on a pump

In this attack scenario, summarised in Table 7.2, we make the assumption that both PLCs and HMIs are openly accessible on the Internet, thereby providing potential entry points for malicious actors. The primary objective of this attack is to execute a Denial-of-Service (DoS), once a certain condition is met, on an actuator, specifically aimed at pump P-101, which is under the control of PLC-1, ultimately leading to the tank T-201 to overflow.

The attack unfolds as follows: The attacker uses a script that continually transmits Modbus packets to the TCP port 502 of the PLC. The initial phase involves sending Modbus packets with the function code ”read analog input register” (0x04). These packets target the PLC register associated with monitoring the level of tank T-201. The attacker monitors this value until the tank’s level is equal to the predefined high setpoint ($high_1$), set at 80 gallons.

Upon reaching the specified threshold, the attacker seamlessly transitions to the next phase, initiating the transmission of a different Modbus packet. This packet is crafted with the function code ”write single discrete output coil” (0x05). The purpose of this packet is to manipulate the PLC register linked to the operation of pump P-101. The attacker forces the pump to remain in an active state.

The end result of this attack is the achievement of tank overflow in T-201. This occurs due to the continuous operation of pump P-101, driven by the attacker’s manipulation of the PLC register. Importantly, the attacker can achieve this outcome independently of the valve MV-301’s state, showcasing the potency of the attack in circumventing traditional control mechanisms. This scenario underscores the critical importance of securing and monitoring industrial control systems to thwart such malicious exploits.

7.1.2 System deadlock based on a shell intrusion

In this attack scenario, summarised in Table 7.3, the adversary targets the HMI, which is exposed on the Internet. The HMI container specifically hosts ScadaBR, version 1.0CE. The attacker leverages the *authenticated arbitrary file upload* vulnerability (CVE-2021-26828 [38]) within ScadaBR to exploit and compromise the HMI, leading to the establishment of a reverse shell on the container.

Once executed the reverse shell, the attacker gains the capability for remote command execution. The attacker identifies the network topology and the devices involved, particularly the target devices like PLC-2 and PLC-1. Using this command channel, the attacker proceeds to upload a statically linked binary. This binary facilitates the following malicious activities:

1. continuously transmitting a Modbus packet with function code *read discrete output coil* (0x01) to read the PLC register associated with valve MV-301
2. sending a second Modbus packet with function code *read analog input register* (0x04) to read the PLC register linked to the level of tank T-101.

ARP is a protocol used to map IP addresses to MAC addresses in a local network. The attacker takes advantage of the fact that ARP does not have built-in security mechanisms, making it susceptible to manipulation. The attacker sends ARP spoofed packets to both PLC-2 and PLC-1, providing false MAC address mappings. PLC-2 is tricked into associating the attacker’s MAC address with

Table 7.3: Summary of Attack 2

Attack Type	Remote compromise and Denial of Service
Target	HMI
Objective	Establish a deadlock state in the physical process
Exploited Vulnerability	Authenticated arbitrary file upload (CVE-2021-26828)
Exploited Protocols	Modbus TCP, ARP Spoofing
Exploited Function Codes	- Read Discrete Output Coil (0x01) - Read Analog Input Register (0x04)
Attack Steps	1. Exploit CVE-2021-26828 to gain control and establish reverse shell. 2. Identify network topology and target PLC-2 and PLC-1. 3. Upload a binary for malicious activities via the reverse shell. 4. Continuously transmit Modbus packets to read PLC register of valve MV-301. 5. Send Modbus packets to read PLC register associated with tank T-101's level. 6. Execute ARP Spoofing to intercept communication between PLC-2 and PLC-1. 7. Drop Modbus packets from PLC-2 to PLC-1 when tank's level nears $high_1$. 8. Induce a deadlock state by preventing the opening of valve MV-301.

the IP address of PLC-1, and vice versa. With the ARP cache poisoned, PLC-2 and PLC-1 unknowingly send their network traffic to the attacker, thinking the attacker's machine is the legitimate destination. When the tank's level nears the predefined threshold of $high_1$ (80 Gal.) and the targeted valve MV-301 remains closed, the attacker intervenes by dropping all Modbus packets transmitted from PLC-2 to PLC-1. This interception disrupts the communication to open valve MV-301. Consequently, the valve remains closed, leading the system into a pro-

Table 7.4: Summary of Attack 3

Attack Type	Stealthy Denial of Service
Target	Pump P-102 (governed by PLC-3)
Objective	Sabotage Pump P-102 without immediate detection
Exploited	Modbus TCP
Function Codes	- Read Analog Input Register (0x04) - Write Single Discrete Output Coil (0x05)
Steps	1. Continuously transmit Modbus packets to read tank T-203 level. 2. When level reaches *low*$_3$, transmit Modbus packet to manipulate pump P-102. 3. Set up MITM attack on HMI interface to convey false pump state. 4. Deceptively communicate to HMI that the pump is off while it is still operational.

longed *deadlock state*. In this state, tank T-201 becomes full without overflowing, while tanks T-202 and T-203 both experience underflow conditions.

7.1.3 Stealthy DoS on a pump with deceptive HMI communication

We consider a scenario where the attacker enters the network through a breached VPN, allowing an attacker to conduct a stealthy Denial of Service attack on pump P-102, controlled by PLC-3. This attack is designed to be subtle and aims to sabotage the pump without raising immediate alarms.

The attack unfolds when tank T-203's level falls to the predetermined low setpoint *low*$_3$ (0 Gal.). At this point, the attacker initiates the first phase of the attack by sending Modbus packets with the function code *read analog input register* (0x04) to query the PLC register associated with the water level of tank T-203. Once the water level reaches *low*$_3$, the attacker proceeds to the second phase.

In the second phase, the attacker switches to transmitting a different Modbus packet, this time with the function code *write single discrete output coil* (0x05). This packet manipulates the PLC register linked to pump P-102, forcing the pump to remain active even in the absence of water. The continuous operation without water eventually leads to the pump's malfunction or breakdown.

To maintain stealthiness and disguise the ongoing malicious activity, the attacker orchestrates a Man-in-the-Middle (MITM) attack on the HMI interface. The attacker starts a Modbus server replaying an old state of the PLC to falsely convey to the HMI that pump P-102 is turned off when, in reality, it is still operational. This deceptive communication aims to mislead operators and monitoring systems, creating a false sense of normalcy despite the compromised state of the pump. This attack has been summarised in Table 7.4

7.2 An automated tool to perform cyber physical attacks

This section explores the creation and deployment of an automated attack tool designed to evaluate the effectiveness of our ICS honeypot (described in previous sections). By simulating real-world attacker behavior, this tool enables us to assess the honeypot's ability to:

- Detect and record attack attempts: Can the honeypot identify malicious activity launched by the tool?
- Mimic realistic industrial environments: Does the honeypot provide a believable target for attackers, enticing them to interact and reveal their tactics?
- Collect valuable forensic data: Can the honeypot capture valuable information about attack methods for further analysis?

By incorporating features like Modbus communication and PLC behavior emulation, the attack tool creates a realistic scenario that tests the honeypot's ability to withstand and reveal insights into various attack types. These include Man-in-the-Middle (MITM) attacks, where the honeypot's ability to detect attempts to intercept communication and manipulate data between devices is assessed. The tool can also simulate Denial-of-Service (DoS) attacks, allowing us to evaluate how the honeypot handles efforts to overload specific PLC functions or disrupt communication. Furthermore, the attack tool can mimic state manipulation attempts, enabling us to determine if the honeypot can identify efforts to alter critical process variables stored in PLC registers. The detailed evaluation process using this attack tool is described in a later section (section 7.2.5).

7.2.1 Related works

To address the context of developing an automated tool for performing cyber-physical attacks on Industrial Control Systems (ICS), it is important to explore related work focusing on tools and methodologies targeting ICS in the literature. Two notable studies offer insights into the landscape of threats and defenses in OT environments, providing a comprehensive background for understanding the dynamics of cyber-physical attacks.

The work titled "Vulnerabilities and Attacks Against Industrial Control Systems and Critical Infrastructures" [116] presents an extensive survey of the most prominent threats against ICS, emphasizing the integration of modern Information Technology (IT) elements into Operational Technology (OT) architectures. This study highlights the ever increasing interest towards OT for various groups of adversaries due to their interconnectedness and inherent complexity. It categorizes threats and vulnerabilities, providing a detailed overview to guide the development of an automated attack tool.

One notable tool that exploits vulnerabilities in Siemens S7 PLCs is PLCInject [73], which leverages a novel approach by injecting a Time-of-Day (TOD) interrupt code to activate malicious operations at a predetermined time, without requiring persistent access to the target system. This method allows external adversaries to patch their malicious codes once they gain access to exposed PLCs, keeping their attack idle inside the infected device, and then activate the attack at a later time without being connected to the target. This approach does not require continuous access to the target system at the attack's zero point, allowing attackers to compromise PLCs when they are offline.

PLCInject is an interesting project that enables the uploading of malicious code to Siemens PLCs, a capability our tool does not possess. Instead, our tool adopts a broader strategy, focusing on attacking the physical process rather than the PLC itself. This means our tool supports a non-proprietary and general industrial protocol, allowing it to interface with a wider range of PLC models and manufacturers. While we do not incorporate code injection, since that would limit our tool's applicability and necessitate dealing with each PLC model individually, our approach offers versatility in addressing security within diverse industrial environments.

7.2.2 Desiderata

Drawing inspiration from [119] where the authors explores the use of SCADA honeypots for detection of malicious tampering within SCADA networks, we were able to derive strategies and desiderata for the tool. We delineated four specific requirements for our tool, ensuring its functionality aligns with a real world attack scenario. These requisites are:

1. *Portability and Seamless Installation*: The tool necessitated effortless distribution and installation, prioritizing accessibility for potential attackers, even in environments without Internet connectivity.
2. *Industrial Protocol Support*: An imperative desiderata was the tool's ability in sending Modbus commands. Furthermore, it was essential for the tool to emulate a PLC convincingly, complete with the ability to replicate collected register values during attacks.

3. *Uninterrupted Operation and Modularity*: Recognizing the dynamic nature of attacks, our tool needed to combine multiple operations. Using this non-blockable nature and modular structure ensure adaptability to diverse attack scenarios, accommodating varied objectives an attacker might pursue.
4. *Suite of attacks*: The tool should be able to execute various attacks, including sending arbitrary commands to the coils/memory registers, initiating DoS on coils/memory registers, and implementing chattering of coils/memory registers (chattering involves rapidly changing the state of a control element). To facilitate Man-in-the-Middle attacks, the tool should also have the capability to broadcast unsolicited ARP replies. This strategic feature enables the interception of communication between devices, a critical component of our tool's functionality.

Portability and Seamless Installation

This tool is based around setuptools [48], a Python package that serves as an enhancement of the distutils (Distribution Utilities) standard for building and distributing Python packages. It was chosen as it allows to easily package Python projects and distribute them via the Python Package Index (PyPI) and also because it aligns with the need for easy distribution and installation, even in environments lacking Internet connectivity.. Setuptools can automatically download and install dependencies specified by a package. This feature simplifies the installation process for end-users, as they don't have to manually install each required package.

Industrial Protocol Support

We integrated pymodbus [59] to address the requirement of Industrial Protocol Support, particularly for sending Modbus commands and emulating a Modbus server. Pymodbus is a comprehensive Modbus protocol implementation in Python, offering client and server functionalities along with asynchronous and synchronous APIs. It also includes simulators for various scenarios. With pymodbus, we were able to create a ModbusTcpClient, connect to a device, and perform actions like writing to a coil or reading from it. Furthermore, pymodbus is instrumental in emulating a PLC. This capability is critical in our attack scenario where the tool must generate a new Modbus server to replay the values of registers that were previously captured.

Uninterrupted Operation and Modularity

Our tool's design focuses on ensuring uninterrupted operation and modularity, which are important features for adapting to the dynamic and unpredictable nature of cyber attacks, particularly in OT. To achieve this, we have employed a multi-threaded architecture, which is instrumental in maintaining continuous operation and enabling the tool to manage multiple tasks simultaneously. The utilization of threads in our tool's architecture allows for parallel processing of different functionalities. It enables the tool to simultaneously execute diverse operations such as monitoring, data manipulation, and communication interception without hindering each other's performance. The threads operate independently but share common resources, ensuring efficient use of system resources. Modularity, another key aspect of our tool, is achieved through a well-designed, component-based architecture. Each module in our tool is designed to perform a specific function and can be independently used. This structure enhances its adaptability.

Suite of Attacks

Thanks to the library pyModbus [59] the tool possesses the capability to send arbitrary commands to coils and memory registers, initiate Denial of Service attacks on coils and memory registers, simulating chattering behavior on coils and memory registers. To facilitate MITM attacks, the tool incorporates the capability to broadcast unsolicited ARP replies. This feature allows the tool to intercept communication between devices within the network, enabling attackers to manipulate and monitor data exchanges. We leveraged arpspoof, a component of the dSniff package [16], by incorporating it into our tool's architecture. To integrate this binary, we implemented a subprocedure specifically designed to invoke arpspoof, ensuring its execution on a separate thread. This strategic design enhances the tool's efficiency and agility, allowing it to perform ARP spoofing attacks in combination with other attacks offered by the tool.

7.2.3 Phases

In the forthcoming sections, we delineate the key phases of our tool, providing insights into the preliminary steps and configurations necessary for its deployment and operation. While the attack tool we developed focuses on established attack techniques within the Modbus protocol, it offers the capability to explore new combinations of these attacks. This allows us to assess the honeypot's ability to handle unforeseen attack scenarios that combine existing methods in novel ways. For instance, the tool could be used to simulate a DoS attack targeting specific PLC functions while simultaneously manipulating critical process variables. However, it's important to acknowledge that the tool itself is not designed to discover entirely new attack vectors beyond the Modbus protocol.

Tool configuration: network reconnaissance

The tool requires proper configuration with specific device details, as demonstrated in Listing 7.1. In this configuration, assigning human-readable device names are encouraged for improved clarity in the tool's interface, replacing raw IP addresses. As exemplified in Listing 7.1, the IP address *172.17.0.5* is represented as "HMI." Additionally, the tool is designed to handle situations where security practices involve altering the default port of the network protocol (for Modbus, typically 502 TCP in our use case). Users should specify a port for each device, with the tool defaulting to the standard port associated with the chosen protocol if none is provided.

Listing 7.1: Configuration file: config.ini

```
[hmi]
ip = 172.17.0.5
port = 502

[plc1]
ip = 172.17.0.2
port = 502
```

In order to obtain this information, the tool user must conduct a comprehensive network scan, identifying devices and the associated protocols. Typically, this process involves determining the network mask and employing tools like Nmap to scan the entire network. Activation of the service discovery flag within Nmap facilitates the identification of the protocols the devices utilize.

```
nmap -sV <target_IP_range>
```

Sniffing of PLCs' registers values

The tool can effectively emulate a Modbus server, a valuable feature for covert attacks aiming to conceal the ongoing attack on a PLC. In such scenarios, the attacker initiates a server that the HMI queries instead of the actual PLC. Meanwhile, the attacker gains control over the authentic PLC, inducing anomalous states, while the HMI displays a seemingly normal situation.

To achieve this, we require a dataset containing the values to be served by the emulated Modbus server. In Ceccato et al. [81] we propose a tool that produces a dataset containing the values associated with the registers of the target PLC within a specified time interval. We utilized the IP Protocol Scan feature provided by the Nmap Python module [43] to identify the target PLC within a range of IP addresses. It is noteworthy that our scanning approach extends beyond the conventional Modbus TCP port 502, acknowledging that in many ICSs, the Modbus protocol may operate on different ports for security through obscurity. Once both the IP address of the PLC and the Modbus port are determined, the tool initiates the capture of register values.

To efficiently handle the reading of register values, we employ the Ray module [60] to parallelize and distribute the task. Our tool systematically reads and records the values of all PLC registers within a specified time frame, with a user-defined time granularity. The data collected during our scans encompass crucial information, as illustrated in Listing 7.2:

Listing 7.2: Example of a Registers capture

```
"127.0.0.1/8502/2022-05-03 12_10_00.591": {
    "DiscreteInputRegisters": {"%IX0.0": "0"},
    "InputRegisters": {"%IW0": "53"},
    "HoldingOutputRegisters": {"%QW0": "0"},
    "MemoryRegisters": {"%MW0": "40", "%MW1": "80"},
    "Coils": {"%QX0.0": "0"}
}
```

The captured details include:

- IP addresses of the scanned PLCs,
- Port used by the Modbus protocol,
- Timestamps of the scan,
- Values saved in each PLC register.

7.2.4 Tool interface

The user interacts with the tool through the interface depicted in Figure 7.1. In this interface, the user can choose the type of attack they intend to launch. Once a specific attack is selected, the tool prompts the user for additional details, as illustrated in Figure 7.2.

7.2.5 Evaluation on a proof of concept attack

Establishing the context for the attack: as outlined in Table 7.3, the goal of attack 2 is to successfully execute a reverse shell, carry out a MITM attack, and induce a Denial of Service by causing the physical process to enter a deadlock state. In Figure 7.3 is displayed the monitoring tab of the three PLCs, which is displaying the state of all the register in a plc at a given point in time.

The targeted victim in this scenario is the operator of the HMI. The tool will be configured to launch an attack on the HMI's ARP table, directing queries to our machine functioning as a Modbus server rather than the actual PLC 1. Concurrently, the tool will intercept and discard the Modbus packets from PLC 1 to PLC 2, which would normally request the opening of the valve facilitating the flow of water from tank 1 to tank 2.

In pursuit of network pivoting [52], we leverage the CVE-2021-26828 vulnerability to establish a shell on the HMI machine. This move enables us to pivot

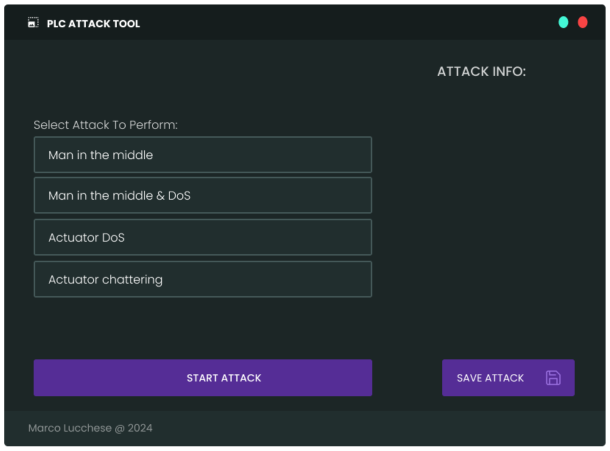


Figure 7.1: Main tool interface

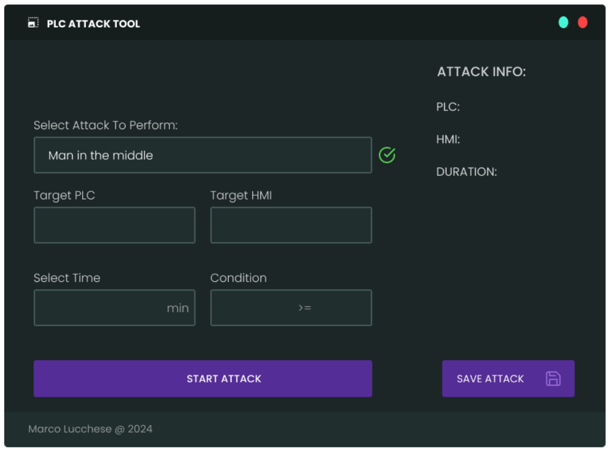
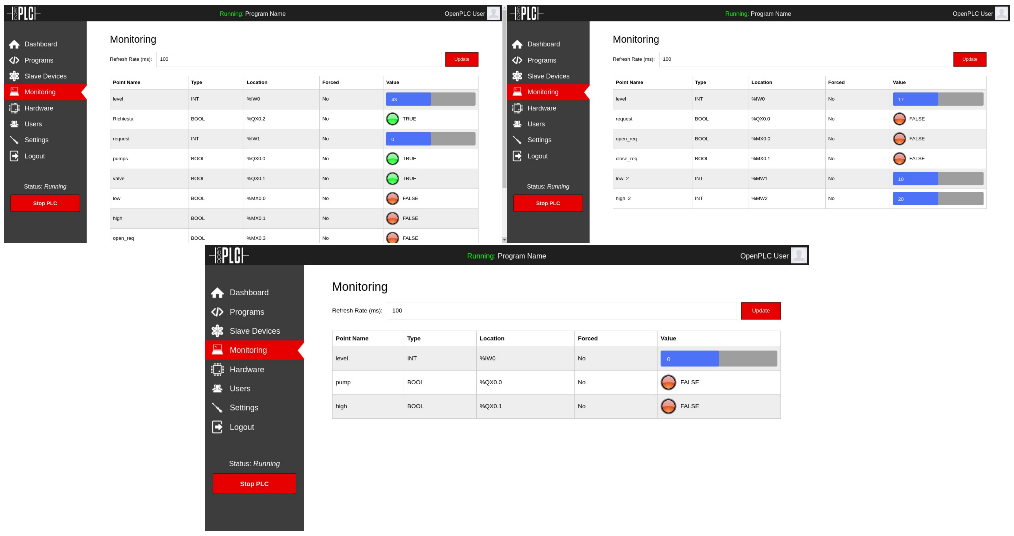


Figure 7.2: Attack configuration

inside the network, granting the capability to execute a man-in-the-middle attack effectively.

To execute network pivoting through Remote Code Execution (RCE), the attacker exploits vulnerabilities within the targeted system to gain unautho-



Monitoring

Refresh Rate (ms): 100

Point Name	Type	Location	Forced	Value
level	INT	%IW0	No	43
Richiesta	BOOL	%QX0.2	No	TRUE
request	INT	%IW1	No	0
pumps	BOOL	%QX0.0	No	TRUE
valve	BOOL	%QX0.1	No	TRUE
low	BOOL	%MX0.0	No	FALSE
high	BOOL	%MX0.1	No	FALSE
open_req	BOOL	%MX0.3	No	FALSE

Monitoring

Refresh Rate (ms): 100

Point Name	Type	Location	Forced	Value
level	INT	%IW0	No	17
request	BOOL	%QX0.0	No	FALSE
open_req	BOOL	%MX0.0	No	FALSE
close_req	BOOL	%MX0.1	No	FALSE
low_2	INT	%MW1	No	10
high_2	INT	%MW2	No	20

Monitoring

Refresh Rate (ms): 100

Point Name	Type	Location	Forced	Value
level	INT	%IW0	No	0
pump	BOOL	%QX0.0	No	FALSE
high	BOOL	%QX0.1	No	FALSE

Figure 7.3: PLC 1, PLC 2 and PLC 3 monitoring tab, showing the current state of the registers

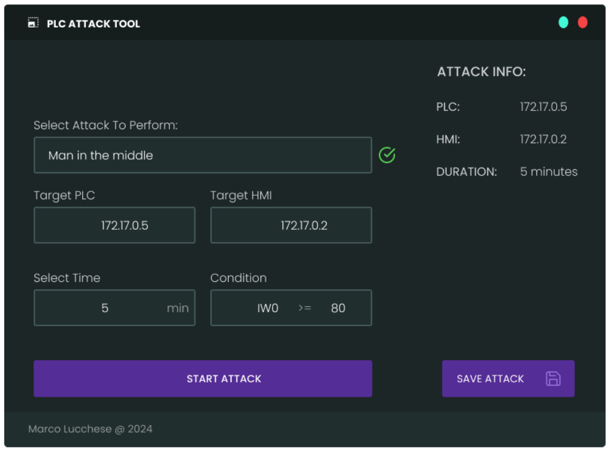
rized access. This method enables the execution of arbitrary code on a remote machine, providing the attacker with a foothold to navigate and manipulate network environments. Once an RCE vulnerability is successfully exploited on the system, it serves as a pivot point for lateral movement within the network, allowing the attacker to compromise additional systems. This technique facilitates the exploration and exploitation of interconnected systems, potentially enabling privilege escalation, sensitive information gathering, and persistent access within the compromised network.

The attacker obtains the information presented in Table 7.5 by executing a Nmap scan.

Name	**IP Address**	**MAC Address**
HMI	172.17.0.2	02:42:ac:11:00:02
PLC 1	172.17.0.5	02:42:ac:11:00:05
PLC 2	172.17.0.4	02:42:ac:11:00:04
PLC 3	172.17.0.6	02:42:ac:11:00:06
Attacker	172.17.0.7	02:42:ac:11:00:07

Table 7.5: Network Devices Information

Leveraging this information, the attacker initiates the process of capturing the values of PLC 1's registers. Once a sufficient number of values are stored, the attacker configures the tool on the designated page. As depicted in Figure 7.4, the attack is configured to target the HMI at IP *172.17.0.2* and PLC 1 at IP *172.17.0.5* for a duration of 5 minutes. The attack starts when the condition IW0 reaches or exceeds 80.



Select Attack To Perform: Man in the middle

Target PLC: 172.17.0.5

Target HMI: 172.17.0.2

Select Time: 5 min

Condition: IW0 >= 80

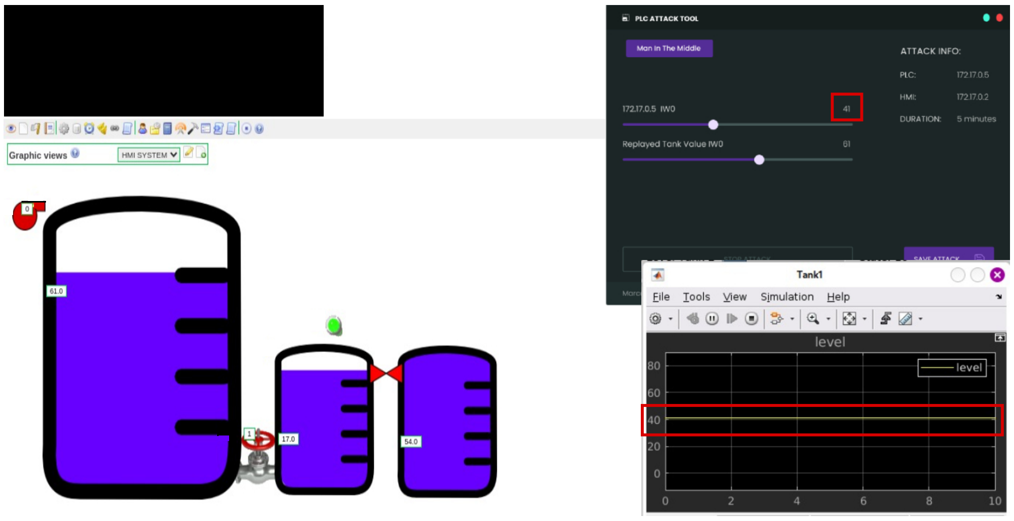
ATTACK INFO:

PLC:	172.17.0.5
HMI:	172.17.0.2
DURATION:	5 minutes

Figure 7.4: Attack tool configuration

Illustrated in Figure 7.5, the genuine physical process is depicted in the bottom right, while the values read from the actual PLC are displayed on the tool at the top right.

In Figure 7.6, what is presented to the HMI operator is depicted. The level 61 is falsified, originating from the previous capture. As illustrated at the bottom right, the authentic level of the tank is 41. Displaying that the attack is successful.


[redacted]

ATTACK INFO:

PLC: 172.17.0.5

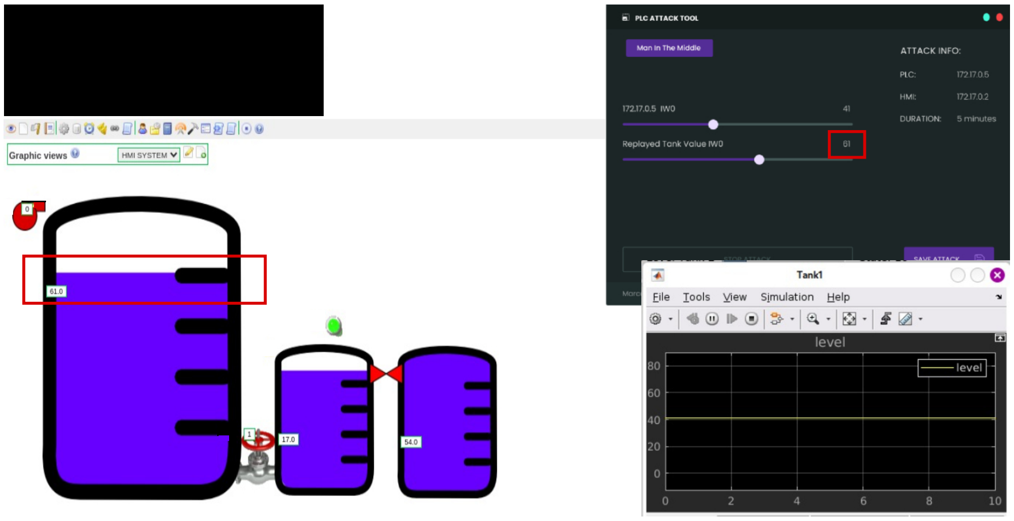
HMI: 172.17.0.2

DURATION: 5 minutes

172.17.0.5 IW0 41

Replayed Tank Value IW0 61

Figure 7.5: System under attack: real process


[redacted]

ATTACK INFO:

PLC: 172.17.0.5

HMI: 172.17.0.2

DURATION: 5 minutes

172.17.0.5 IW0 41

Replayed Tank Value IW0 61

Figure 7.6: System under attack: replaying old register values

Chapter 8

DISCUSSION, CONCLUDING REMARKS AND FUTURE WORK

Our investigation into the deployment and analysis of HoneyICS, a high-interaction, physics-aware honeynet for Industrial Control Systems (ICS), has yielded valuable insights into the effectiveness of honeynet and honeypots configurations, attack patterns, and potential avenues for future research.

8.1 Discussion

8.1.1 Honeynet Deployment

Our analysis of HoneyICS deployment revealed several key lessons for optimizing honeynet effectiveness. Notably, different industrial protocols exhibited varying levels of attractiveness based on factors such as protocol vulnerabilities, industrial domain, and geographic location. To thoroughly explore these dimensions, deploying similar honeynets in diverse geographic regions supporting different industrial protocols becomes essential. Interestingly, exposing an HMI in the same IP alongside a PLC may reduce attractiveness for actors interacting with the honeynet. Most interactions and exploits targeting HMIs were observed when the HMI was the sole exposed device, suggesting that actors in the IT domain may not prioritize targeting OT/Scada IPs.

8.1.2 Attack Patterns

The analysis of IT interactions highlighted a prevalence of automated and indiscriminate attacks, showcasing a 'blind' approach where adversaries exploit well-known vulnerabilities across diverse systems. In contrast, OT interactions displayed a more targeted and sophisticated nature. Actors engaged in OT interactions demonstrated a higher level of knowledge, employing utilities like Nmap for information gathering, indicative of potential development of refined attack patterns. Differentiation between IT and OT interactions emphasizes the need for a multi-faceted defense approach, tailoring responses to the observed level of sophistication.

8.1.3 Ethical Considerations

To address ethical concerns, strict controls have been implemented in HoneyICS, including firewall rules, strong authentication for the management dashboard, and limitations on egress traffic. The honeypot is designed to simulate realistic scenarios without exposing sensitive data or allowing interactions with cross-domain destinations.

8.1.4 Security Measures

While the high-interaction requirement introduces potential security risks, the containerization of HoneyICS components, namespace isolation, and firewall rules mitigate these concerns. Access to the management dashboard is regulated and protected through strong authentication, further securing the honeypot environment. Egress traffic is restricted to prevent unauthorized communication.

8.2 Concluding Remarks

In conclusion, our work provides valuable insights into effective honeynet deployment, attack patterns, and the nuanced differences between IT and OT interactions. The study underscores the importance of tailored defense strategies based on the nature of interactions observed. The deployment of HoneyICS and the analysis of resulting data contribute to the broader understanding of cybersecurity challenges in the realm of Industrial Control Systems.

8.3 Future Work

Moving forward, there are several promising directions for future research that could significantly enhance the capabilities and impact of HoneyICS. The integration of additional threat intelligence data could enhance the identification of attacker tactics and techniques, providing a more comprehensive understanding of the attack process. This could involve developing more sophisticated algorithms to correlate observed behaviors with known threat actors or emerging attack patterns. Further research is needed to explore how attackers traverse between IT and OT networks, identifying combinations of IT and OT interactions. This could involve developing more complex honeypot architectures that simulate both IT and OT environments, allowing for a deeper understanding of lateral movement techniques.

Leveraging lightweight simulation methods for physical processes is crucial for expanding the realism of HoneyICS. This could include developing more sophisticated models that can simulate complex industrial processes in real-time, potentially incorporating machine learning techniques to adapt to different scenarios. Supporting other industrial network protocols, such as OPC-UA and Ethernet/IP, is critical for expanding the capabilities of HoneyICS. This would allow the system to attract a wider range of potential attackers and provide insights into protocol-specific vulnerabilities.

As we consider larger deployments, it's important to investigate potential scalability limitations of HoneyICS. Future work could focus on optimizing the system architecture to handle a significantly larger number of simulated devices and more complex attack scenarios, potentially leveraging cloud technologies or distributed computing approaches. Incorporating more sophisticated attacker behavior models could provide deeper insights into adversary tactics. This might involve developing machine learning algorithms that can predict attacker intentions based on observed behaviors, or creating more nuanced simulations of different attacker profiles.

Exploring advanced data analysis and visualization techniques could help extract more valuable insights from the vast amounts of data collected by HoneyICS. This could include applying big data analytics, developing new metrics for quantifying attack sophistication, or creating more intuitive visualization tools for security analysts. Investigating the potential for applying HoneyICS principles beyond ICS could significantly broaden the impact of this research. Future work could explore how the core concepts of HoneyICS could be adapted to secure other critical infrastructure systems, such as smart grids, transportation systems, or healthcare networks.

Research into seamlessly integrating HoneyICS with existing security information and event management (SIEM) systems and security orchestration, automation, and response (SOAR) platforms could enhance its practical value for organizations. As honeypot technologies become more sophisticated, there's a need for ongoing research into the ethical and legal implications of their deployment, particularly in critical infrastructure contexts.

By pursuing these research directions, we can continue to advance the state-of-the-art in ICS security, develop more robust defense strategies, and ultimately contribute to the protection of critical infrastructure against evolving cyber threats.

8.4 Extensions

HoneyICS, initially based on software PLCs, can support real PLCs (hardware) with minimal modifications, expanding the applicability of the framework. Extensions to support additional industrial network protocols, such as DNP3, and the adoption of lightweight simulation methods for physical processes, such as finite state machines or pre-recorded sensor data playback, are planned. Furthermore, integrating the snap7 library for S7comm server implementation and SNMP protocol support are envisioned extensions to enhance HoneyICS capabilities. HoneyICS can further enhance its realism and introduce an additional attack surface by integrating APIs capable of dynamically updating the web page exposed on port 8080 TCP for the PLCs. Currently, the web page is static HTML; however, incorporating an API gateway that interfaces with OpenPLC core functions would not only make the honeypots more believable but also expand the potential attack vectors.

REFERENCES

1. 2.5 modbus addressing – autonomy. https://autonomylogic.com/docs/2-5-modbus-addressing/. (Accessed on 10/09/2023).
2. Advanced load balancer, web server, & reverse proxy - nginx. https://www.nginx.com/. (Accessed on 12/12/2023).
3. Agreement. https://modbus.org/docs/Client-ServerPR-07-2020-final.docx.pdf. (Accessed on 10/09/2023).
4. Attacks on industrial sector hit record in second quarter of 2023 — kaspersky. https://www.kaspersky.com/about/press-releases/2023_attacks-on-industrial-sector-hit-record-in-second-quarter-of-2023. (Accessed on 12/28/2023).
5. Aurora generator test - wikipedia. https://en.wikipedia.org/wiki/Aurora_Generator_Test. (Accessed on 10/10/2023).
6. beremiz.org — beremiz home. https://beremiz.org/. (Accessed on 11/23/2023).
7. Carloslannister/ihs: Industrial hacking simulator. https://github.com/CarlosLannister/IHS. (Accessed on 11/08/2023).
8. Claroty uncovers vulnerabilities in ovarro's tbox rtus — claroty. https://claroty.com/team82/research/vulnerabilities-in-tbox-rtus. (Accessed on 10/11/2023).
9. Crashoverride malware — cisa. https://www.cisa.gov/news-events/alerts/2017/06/12/crashoverride-malware. (Accessed on 10/09/2023).
10. Cyberattack on critical infrastructure: Russia and the ukrainian power grid attacks - the henry m. jackson school of international studies. https://jsis.washington.edu/news/cyberattack-critical-infrastructure-russia-ukrainian-power-grid-attacks/. (Accessed on 10/09/2023).
11. Dangerous stuff: Hackers tried to poison water supply of florida town - the new york times. https://www.nytimes.com/2021/02/08/us/oldsmar-florida-water-supply-hack.html. (Accessed on 10/09/2023).
12. Docker: Accelerated container application development. https://www.docker.com/. (Accessed on 11/28/2023).
13. Docker docs. https://docs.docker.com/. (Accessed on 06/16/2024).
14. Docker hub container image library — app containerization. https://hub.docker.com/. (Accessed on 11/29/2023).
15. dotcloud, inc. is now docker, inc. — docker. https://www.docker.com/press-release/dotcloud-inc-now-docker-inc/. (Accessed on 11/28/2023).
16. dsniff — kali linux tools. https://www.kali.org/tools/dsniff/. (Accessed on 01/24/2024).
17. Ethernet/ip - wikipedia. https://en.wikipedia.org/wiki/EtherNet/IP. (Accessed on 06/04/2024).

18. etingof/snmpsim: Snmp simulator. https://github.com/etingof/snmpsim. (Accessed on 11/22/2023).
19. Extended finite-state machine - wikipedia. https://en.wikipedia.org/wiki/Extended_finite-state_machine. (Accessed on 11/02/2023).
20. Getting started — selenium. https://www.selenium.dev/documentation/webdriver/getting_started/. (Accessed on 06/17/2024).
21. Hack of oldsmar water plant reported two years ago could have been employee error. https://www.fox13news.com/news/hack-of-oldsmar-water-plant-reported-two-years-ago-could-have-been-employee. (Accessed on 10/09/2023).
22. Hackers caused power cut in western ukraine - us - bbc news. https://www.bbc.com/news/technology-35297464. (Accessed on 10/09/2023).
23. Honeyd downloads and releases — honeyd. https://www.honeyd.org/releases/. (Accessed on 10/16/2023).
24. The honeynet project. https://www.honeynet.org/. (Accessed on 11/01/2023).
25. Iec 61131-3:2013 — programmable controllers - part 3: Programming languages. https://webstore.iec.ch/publication/4552#additionalinfo. (Accessed on 10/09/2023).
26. Industrial cybersecurity technology for ics/ot asset visibility — dragos. https://www.dragos.com/. (Accessed on 02/28/2024).
27. iptables - wikipedia. https://en.wikipedia.org/wiki/Iptables. (Accessed on 12/12/2023).
28. Iranian hackers claim cyber attack on new york dam. https://www.nbcnews.com/news/us-news/iranian-hackers-claim-cyber-attack-new-york-dam-n484611. (Accessed on 10/09/2023).
29. Is this project dead? · issue #91 · datasoft/honeyd. https://github.com/DataSoft/Honeyd/issues/91. (Accessed on 10/31/2023).
30. ld.so(8) - linux manual page. https://man7.org/linux/man-pages/man8/ld.so.8.html. (Accessed on 10/30/2023).
31. The llvm compiler infrastructure project. https://llvm.org/. (Accessed on 11/01/2023).
32. Macvlan network driver — docker docs. https://docs.docker.com/network/drivers/macvlan/. (Accessed on 01/09/2024).
33. Meet the latest players in industrial control system cyber intrusions. https://www.asisonline.org/security-management-magazine/latest-news/online-exclusives/2022/Meet-the-Latest-Players-in-Industrial-Control-System-Cyber-Intrusions/. (Accessed on 12/28/2023).
34. Mininet: An instant virtual network on your laptop (or other pc) - mininet. https://mininet.org/. (Accessed on 11/08/2023).
35. Modbus - wikipedia. https://en.wikipedia.org/wiki/Modbus. (Accessed on 10/05/2023).
36. The Modbus organization. https://modbus.org/. (Accessed on 06/04/2024).
37. Nmap: the network mapper - free security scanner. https://nmap.org/. (Accessed on 10/16/2023).
38. Nvd - cve-2021-26828. https://nvd.nist.gov/vuln/detail/CVE-2021-26828. (Accessed on 12/28/2023).
39. OpenSCADA - wiki. http://oscada.org/wiki/About. (Accessed on 10/03/2023).
40. Overview of dnp3 protocol. https://www.dnp.org/About/Overview-of-DNP3-Protocol. (Accessed on 06/04/2024).
41. Plc4x – [untitled]. https://plc4x.apache.org/protocols/s7/s7comm.html. (Accessed on 06/04/2024).
42. The Purdue enterprise reference architecture — industrial cybersecurity. https://subscription.packtpub.com/book/security/9781788395151/5/ch05lvl1sec30/the-purdue-enterprise-reference-architecture. (Accessed on 10/04/2023).

43. python3-nmap · pypi. https://pypi.org/project/python3-nmap/. (Accessed on 01/20/2024).
44. Regulation - 2016/679 - en - gdpr - eur-lex. https://eur-lex.europa.eu/eli/reg/2016/679/oj. (Accessed on 06/04/2024).
45. ScadaBR Wiki. https://sourceforge.net/p/scadabr/wiki/Home/. (Accessed on 10/03/2023).
46. Security collector: Honeypot / honeyd tutorial. https://securitywander.blogspot.com/2012/02/honeypot-honeyd-tutorial.html. (Accessed on 10/31/2023).
47. Selenium. https://www.selenium.dev/. (Accessed on 11/23/2023).
48. setuptools · pypi. https://pypi.org/project/setuptools/. (Accessed on 01/18/2024).
49. Shodan search engine. https://www.shodan.io/. (Accessed on 11/16/2023).
50. Simulink - simulazione e progettazione model-based - matlab. https://it.mathworks.com/products/simulink.html. (Accessed on 11/20/2023).
51. Snap7 homepage. https://snap7.sourceforge.net/. (Accessed on 10/31/2023).
52. Sp 800-53 rev. 5, security and privacy controls for information systems and organizations — csrc. https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final. (Accessed on 01/24/2024).
53. thiagoralves/openplc_simulink-interface: Simulink interface program for openplc. https://github.com/thiagoralves/OpenPLC_Simulink-Interface. (Accessed on 12/04/2023).
54. Threats against industrial control systems on the rise in h2 2020, growing by nearly 8 percentage points in the engineering sector — kaspersky. https://www.kaspersky.com/about/press-releases/2021_threats-against-industrial-control-systems-on-the-rise-in-h2-2020. (Accessed on 12/28/2023).
55. Triton malware — attackers deploy new ics attack framework. https://www.mandiant.com/resources/blog/attackers-deploy-new-ics-attack-framework-triton. (Accessed on 10/10/2023).
56. Twisted. https://twisted.org/. (Accessed on 11/01/2023).
57. Unified architecture - opc foundation. https://opcfoundation.org/about/opc-technologies/opc-ua/. (Accessed on 06/04/2024).
58. Virus alert about blaster worm and its variants - windows server — microsoft learn. https://learn.microsoft.com/en-us/troubleshoot/windows-server/security-and-malware/blaster-worm-virus-alert. (Accessed on 10/10/2023).
59. Welcome to pymodbus's documentation! — pymodbus 3.7.0dev documentation. https://pymodbus.readthedocs.io/en/latest/. (Accessed on 01/18/2024).
60. Welcome to ray! — ray 2.9.0. https://docs.ray.io/en/latest/. (Accessed on 01/20/2024).
61. What is HMI and what is it for? — esa automation. https://www.esa-automation.com/en/what-is-hmi/. (Accessed on 10/03/2023).
62. What is HMI? human machine interface. https://inductiveautomation.com/resources/article/what-is-hmi. (Accessed on 10/03/2023).
63. What is upnp? yes, it's still dangerous in 2023 — upguard. https://www.upguard.com/blog/what-is-upnp. (Accessed on 10/16/2023).
64. Which ports are released for modbus/tcp communication and how many modbus clients - industry support siemens. https://support.industry.siemens.com/cs/document/34010717/. (Accessed on 10/09/2023).
65. Words matter. https://www.acm.org/diversity-inclusion/words-matter. (Accessed on 10/09/2023).
66. GARR cloud, 2022.
67. Aqua Tracee: Runtime eBPF threat detection engine, 2023.
68. Extended Berkeley Packet Filter, 2023.

69. A. Murillo, L. Combita Alfonso, A. Gonzalez, S. Rueda, A. Cardenas, and N. Quijano. A Virtual Environment for Industrial Control Systems: A Nonlinear Use-Case in Attack Detection, Identification, and Response. In *ICSS*, pages 25–32, 2018.
70. S. Abe, Y. Tanaka, Y. Uchida, and S. Horata. Developing deception network system with traceback honeypot in ics network. *JCMSI*, 11:372–379, 2018.
71. M. Alanazi, A. Mahmood, and M. J. M. Chowdhury. SCADA vulnerabilities and attacks: A review of the state-of-the-art and open issues. *Computer Security*, 125:103028, 2023.
72. AlienVault. The world's first truly open threat intelligence community, 2023.
73. Wael Alsabbagh and Peter Langendörfer. Patch now and attack later - exploiting s7 plcs by time-of-day block. In *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, pages 144–151, 2021.
74. T. Alves, M. Buratto, F. Souza, and T. Rodrigues. OpenPLC: An open source alternative to automation. In *IEEE GHTC*, pages 585–589, 2014.
75. Daniele Antonioli, Hamid Reza Ghaeini, Sridhar Adepu, Martin Ochoa, and Nils Ole Tippenhauer. Gamifying ics security training and research: Design, implementation, and results of s3. In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, CPS '17, page 93–102, New York, NY, USA, 2017. Association for Computing Machinery.
76. Daniele Antonioli and Nils Ole Tippenhauer. Minicps: A toolkit for security research on cps networks. In *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy*, CPS-SPC '15, page 91–100, New York, NY, USA, 2015. Association for Computing Machinery.
77. M. Bailey, E. Cooke, F. Jahanian, and D. Watson. The blaster worm: then and now. *IEEE Security Privacy*, 3(4):26–31, 2005.
78. Boldizsár Bencsáth, Gábor Pék, Levente Buttyan, and Mark Felegyhazi. The cousins of Stuxnet: Duqu, Flame, and Gauss. *Future Internet*, 4:971–1003, 12 2012.
79. Olivier Cabana, Amr M. Youssef, Mourad Debbabi, Bernard Lebel, Marthe Kassouf, Ribal Atallah, and Basile L. Agba. Threat Intelligence Generation Using Network Telescope Data for Industrial Control Systems. *IEEE Transactions on Information Forensics and Security*, 16:3355–3370, 2021.
80. Defense Use Case. Analysis of the cyber attack on the ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)*, 388(1-29):3, 2016.
81. Mariano Ceccato, Youssef Driouich, Ruggero Lanotte, Marco Lucchese, and Massimo Merro. Towards reverse engineering of industrial physical processes. In *Computer Security. ESORICS 2022 International Workshops*, pages 273–290, Cham, 2023. Springer International Publishing.
82. Thomas M. Chen and Saeed Abu-Nimeh. Lessons from Stuxnet. *Computer*, 44(4):91–93, 2011.
83. M. Conti, F. Trolese, and F. Turrin. ICSpot: A High-Interaction Honeypot for Industrial Control Systems. In *ISNCC*, pages 1–4. IEEE, 2022.
84. D. Antonioli, A. Agrawal, and N.O. Tippenhauer. Towards High-Interaction Virtual ICS Honeypots-in-a-Box. In *CPS-SPC*, page 13–22. ACM, 2016.
85. D. Pliatsios, P.G. Sarigiannidis, T. Liatifis, K. Rompolos, and I. Siniosoglou. A novel and interactive industrial control system honeypot for critical smart grid infrastructure. In *IEEE CAMAD*, pages 1–6, 2019.
86. D.I. Buza, F. Juhász, G. Miru, M. Félegyházi, and T. Holczer. CryPLH: Protecting Smart Energy Systems from Targeted Attacks with a PLC Honeypot. In *Smart Grid Security*, pages 181–192. Springer, 2014.
87. Michael Dodson, Alastair R. Beresford, and Mikael Vingaard. Using global honeypot networks to detect targeted ICS attacks. In *CyCon)*, volume 1300, pages 275–291. IEEE, 2020.

88. K. Fall. Network emulation in the vint/ns simulator. In *Proceedings IEEE International Symposium on Computers and Communications (Cat. No.PR00250)*, pages 244–250, 1999.
89. Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, symantec corp., security response*, 5(6):29, 2011.
90. P. Ferretti, M. Pogliani, and S. Zanero. Characterizing background noise in ICS traffic through a set of low interaction honeypots. In *CPS-SPC*. ACM, 2019.
91. Pietro Ferretti, Marcello Pogliani, and Stefano Zanero. Characterizing Background Noise in ICS Traffic Through a Set of Low Interaction Honeypots. In *CPS-SPC@CCS*, pages 51—-61. ACM, 2019.
92. Javier Franco, Ahmet Aris, Berk Canberk, and A. Selcuk Uluagac. A survey of honeypots and honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems. *IEEE Commun. Surv. Tutorials*, 23(4):2351–2383, 2021.
93. G. Bernieri, M. Conti, and F. Pascucci. MimePot: a Model-based Honeypot for Industrial Control Networks. In *IEEE SMC*, pages 433–438, 2019.
94. Dieter Gollmann, Pavel Gurikov, Alexander Isakov, Marina Krotofil, Jason Larsen, and Alexander Winnicki. Cyber-physical systems security – experimental analysis of a vinyl acetate monomer plant. 04 2015.
95. Andy Greenberg. The untold story of notpetya, the most devastating cyberattack in history. *Wired, August*, 22, 2018.
96. GreyNoise. Turning Internet noise into intelligence, 2023.
97. Prageeth Gunathilaka, Daisuke Mashima, and Binbin Chen. Softgrid: A software-based smart grid testbed for evaluating substation cybersecurity solutions. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, CPS-SPC '16, page 113–124, New York, NY, USA, 2016. Association for Computing Machinery.
98. S. Hilt, F. Maggi, C. Perine, L. Remorin, M. Rösler, and R. Vosseler. Caught in the act: Running a realistic factory honeypot to capture real threats. *Trend Micro, Shibuya City, Japan, White Paper*, 2020.
99. S. Hilt, F. Maggi, C. Perine, L. Remorin, M. Rösler, and R. Vosseler. Caught in the act: Running a realistic factory honeypot to capture real threats, 2020.
100. J. Cao, W. Li, J. Li, and B. Li. *DiPot: A Distributed Industrial Honeypot System*, pages 300–309. Springer, 2018.
101. S. Jaloudi. Communication protocols of an industrial internet of things environment: A comparative study. *Future Internet*, 11, 03 2019.
102. S. Jaloudi. Communication protocols of an Industrial Internet of Things environment: A comparative study. *Future Internet*, 11:66, 2019.
103. Bojan Jelacic, Daniela Rosic, Imre Lendak, Marina Stanojevic, and Sebastijan Stoja. Stride to a secure smart grid in a hybrid cloud. In *Computer Security: ESORICS 2017 International Workshops, CyberICPS 2017 and SECPRE 2017, Oslo, Norway, September 14-15, 2017, Revised Selected Papers 3*, pages 77–90. Springer, 2018.
104. K. Wilhoit and S. Hilt. The gaspot experiment: Unexamined perils in using gas-tank-monitoring systems. In *Trend Micro*, volume 6, pages 3–13, 2015.
105. Rafiullah Khan, Kieran McLaughlin, David Laverty, and Sakir Sezer. Stride-based threat modeling for cyber-physical systems. In *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pages 1–6, 2017.
106. M. Krotofil, K. Kursawe, and D. Gollmann. Securing industrial control systems. In *Security and Privacy Trends in the Industrial IoT*, pages 3–27. Springer, 2019.
107. S. Kuman, S. Gros, and M. Mikuc. An experiment in using imunes and Conpot to emulate honeypot control networks. In *MIPRO*, pages 1262–1268. IEEE, 2017.
108. L. Rist, J. Vestergaard, D. Haslinger, A. De Pasquale, and J. Smith. Conpot ICS/SCADA honeypot, 2013.

109. Ruggero Lanotte, Massimo Merro, and Andrei Munteanu. Runtime enforcement for control system security. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, pages 246–261, 2020.
110. Robert M Lee, Michael J Assante, and Tim Conway. German steel mill cyber attack. *Industrial Control Systems*, 30(62):1–15, 2014.
111. S. Litchfield, D. Formby, J. Rogers, A. Meliopoulos, and R. Beyah. Rethinking the Honeypot for Cyber-Physical Systems. *IEEE Internet Comput.*, 20:9–17, 2016.
112. Efrén López-Morales, Carlos Rubio-Medrano, Adam Doupé, Yan Shoshitaishvili, Ruoyu Wang, Tiffany Bao, and Gail-Joon Ahn. Honeyplc: A next-generation honeypot for industrial control systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 279–291, New York, NY, USA, 2020. Association for Computing Machinery.
113. Marco Lucchese, Massimo Merro, Federica Paci, and Nicola Zannone. Towards a high-interaction physics-aware honeynet for industrial control systems. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, SAC '23, page 76–79, New York, NY, USA, 2023. Association for Computing Machinery.
114. F. Lupia, M. Lucchese, M. Merro, and N. Zannone. ICS Honeypot Interactions: A Latitudinal Study. In *IEEE International Conference on Big Data*, pages 1–10. IEEE, 2023.
115. Tyson Macaulay and Bryan L. Singer. *Cybersecurity for Industrial Control Systems: SCADA, DCS, PLC, HMI, and SIS*. Auerbach Publications, 2012.
116. Georgios Michail Makrakis, Constantinos Kolias, Georgios Kambourakis, Craig Rieger, and Jacob Benjamin. Industrial and critical infrastructure security: Technical analysis of real-life security incidents. *IEEE Access*, 9:165295–165325, 2021.
117. Aditya P. Mathur and Nils Ole Tippenhauer. Swat: a water treatment testbed for research and training on ics security. In *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, pages 31–36, 2016.
118. Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.
119. Mohamed Mesbah, Mahmoud Said Elsayed, Anca Delia Jurcut, and Marianne Azer. Analysis of ics and scada systems attacks using honeypots. *Future Internet*, 15(7), 2023.
120. Ariana Mirian, Zane Ma, David Adrian, Matthew Tischer, Thasphon Chuenchujit, Timothy M. Yardley, Robin Berthier, Joshua Mason, Zakir Durumeric, J. Alex Halderman, and Michael Bailey. An Internet-wide view of ICS devices. In *PST*, pages 96–103, 2016.
121. David Moore, Colleen Shannon, Geoffrey M Voelker, and Stefan Savage. Network telescopes: Technical report. 2004.
122. O. Navarro, S. Balbastre, and S. Beyer. Gathering Intelligence Through Realistic Industrial Control System Honeypots - A Real-World Industrial Experience Report. In *CRITIS*, volume 11260, pages 143–153. Springer, 2019.
123. Osborn Nyasore, Pavol Zavarsky, Bobby Swar, Raphael Naiyeju, and Shubham Dabra. Deep packet inspection in industrial automation control system to mitigate attacks exploiting modbus/tcp vulnerabilities. pages 241–245, 05 2020.
124. O. Navarro, S.A.J. Balbastre, and S. Beyer. Gathering Intelligence Through Realistic Industrial Control System Honeypots - A real-world industrial experience report. In *CRITIS*, volume 11260, pages 143–153. Springer, 2019.
125. Carlos E Pereira and Peter Neumann. Handbook of automation, 2009.
126. Siemens ProductCERT. Ssa-978220: Denial of service vulnerability over snmp in multiple industrial products. https://cert-portal.siemens.com/productcert/pdf/ssa-978220.pdf. (Accessed on 11/22/2023).
127. Niels Provos. A virtual honeypot framework. pages 1–14, 01 2004.
128. Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: The next computing revolution. In *Design Automation Conference*, pages 731–736, 2010.

129. Ram Sandesh Ramachandruni and Prabaharan Poornachandran. Detecting the network attack vectors on scada systems. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 707–712, 2015.
130. S. Arndt S. Lau, J. Klick and V. Roth. POSTER: Towards Highly Interactive Honeypots for Industrial Control Systems. In *CCS*, page 1823 – 1825. ACM, 2016.
131. Alexandru Vlad Serbanescu, Sebastian Obermeier, and Der-Yeuan Yu. ICS threat analysis using a large-scale honeynet. In *International Symposium for ICS & SCADA Cyber Security Research*, pages 20—-30, 2015.
132. L. Spitzner. *Honeypots: Tracking Hackers.* Addison-Wesley Longman Publishing Co., Inc., USA, 2002.
133. L. Spitzner. The honeynet project: trapping the hackers. *IEEE Security Privacy*, 1(2):15–23, 2003.
134. André Teixeira, Daniel Pérez, Henrik Sandberg, and Karl Henrik Johansson. Attack models and scenarios for networked control systems. In *Proceedings of the 1st International Conference on High Confidence Networked Systems*, HiCoNS '12, page 55–64, New York, NY, USA, 2012. Association for Computing Machinery.
135. V. Pothamsetty and M. Franz. SCADA Honeynet Project: Building honeypots for industrial networks, 2004. (CIAG) Cisco Systems.
136. E. Vasilomanolakis, S. Srinivasa, C. G. Cordero, and M. Mühlhäuser. Multi-stage attack detection and signature generation with ICS honeypots. In *IEEE/IFIPNOMS*, pages 1227–1232, 2016.
137. Emmanouil Vasilomanolakis, Shreyas Srinivasa, Carlos Garcia Cordero, and Max Mühlhäuser. Multi-stage attack detection and signature generation with ics honeypots. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1227–1232, 2016.
138. VirusTotal, 2023.
139. Artemios G. Voyiatzis, Konstantinos Katsigiannis, and Stavros Koubias. A modbus/tcp fuzzer for testing internetworked industrial systems. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–6, 2015.
140. Wikipedia. Programmable logic controller — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Programmable%20logic%20controller&oldid=1160294231`, 2023. [Online; accessed 26-July-2023].
141. Wikipedia. Purdue Enterprise Reference Architecture — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Purdue%20Enterprise%20Reference%20Architecture&oldid=1163152179`, 2023. [Online; accessed 26-July-2023].
142. T.J. Williams. The purdue enterprise reference architecture. *IFAC Proceedings Volumes*, 26(2, Part 4):559–564, 1993. 12th Triennal Wold Congress of the International Federation of Automatic control. Volume 4 Applications II, Sydney, Australia, 18-23 July.
143. M. Winn, M. Rice, S. Dunlap, J. Lopez Jr., and B. E. Mullins. Constructing cost-effective and targetable industrial control system honeypots for production networks. *Int. J. Crit. Infrastructure Prot.*, 10:47–58, 2015.
144. Wenjun Xiong and Robert Lagerström. Threat modeling – a systematic literature review. *Computers Security*, 84:53–69, 2019.
145. Feng Yu, Hong Sun, and Shou Zhang. A new timing method based on PLC scan cycle time and its application in automatic marine power station. *Advanced Materials Research*, 774-776:1489–1492, 09 2013.
146. Chunhui Zhao and Sujuan Qin. A research for high interactive honepot based on industrial service. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 2935–2939, 2017.

LIST OF PUBLICATIONS

The thesis is grounded in the works of "HoneyICS: A High-interaction Physics-aware Honeynet for Industrial Control Systems," presented at the 18th International Conference on Availability in 2023, and "Towards a High-interaction Physics-aware Honeynet for Industrial Control Systems," presented at the 38th ACM/SIGAPP Symposium on Applied Computing in 2023.

- HoneyICS: A High-interaction Physics-aware Honeynet for Industrial Control Systems
  M Lucchese, F Lupia, M Merro, F Paci, N Zannone, A Furfaro
  Proceedings of the 18th International Conference on Availability, 2023
- Towards a High-interaction Physics-aware Honeynet for Industrial Control Systems
  M Lucchese, M Merro, F Paci, N Zannone
  Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, 2023
- Towards Obfuscation of Programmable Logic Controllers
  V Cozza, M Dalla Preda, M Lucchese, M Merro, N Zannone
  Proceedings of the 18th International Conference on Availability, 2023
- ICS Honeypot Interactions: A Latitudinal Study
  F Lupia, M Lucchese, M Merro, N Zannone
  IEEE international conference on big data, 2023
- Towards Reverse Engineering of Industrial Physical Processes
  M Ceccato, Y Driouich, R Lanotte, M Lucchese, M Merro
  European Symposium on Research in Computer Security, 2022