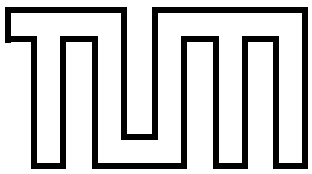


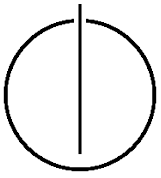
FAKULTÄT FÜR INFORMATIK

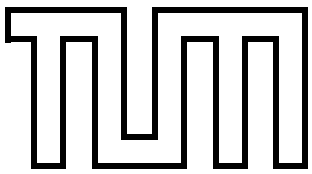
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Development of a Testbed to Demonstrate Attacks on Emulated PLC Networks

Victor Embacher





FAKULTÄT FÜR INFORMATIK

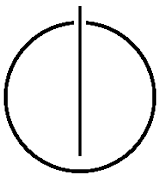
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Development of a Testbed to Demonstrate Attacks on Emulated PLC Networks

Entwicklung einer Testumgebung zur Demonstration von Angriffen auf emulierte SPS Netzwerke

Author:	Victor Embacher
Supervisor:	Prof. Dr. Claudia Eckert
Advisors:	Alexander Giehl, Michael Heinl
Date:	2021-04-15



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Immenstadt i. Allgäu, Victor Embacher

Acknowledgements

I want to thank my advisors Alexander and Michael of the Fraunhofer Institute for Applied and Integrated Security. They were very supportive while I was writing this thesis.
Additionally, I want to thank my friends and family for being there for me.

Abstract

Operational technologies (OT) are industrial information systems that directly interact with the physical world. Previously isolated networks have become more connected to corporate networks and the Internet, highlighting unaddressed problems in regard to information security. Some issues are caused by an established reliance on security-through-obscurity, others are rooted in their initial conception as isolated networks. Consequently, OT networks receive increased attention from both malicious actors and cybersecurity researchers. For research purposes and threat hunting, we require an environment in which we can study apparent information security problems in a safe and realistic manner. Programmable logic controllers (PLC) are important elements of OT environments. However, most pre-existing testbeds do not address PLCs in-depth. The goal of this thesis is to develop a testbed to fulfill the need for a comprehensive testing environment. We analyze requirements posed to such testbeds and then use them to develop a PLC network based on simulation, emulation and virtualization techniques. The realized testbed is used to implement and demonstrate realistic attacks targeting PLC networks. Our testing environment can be used to educate people from OT about risks they face and are potentially unaware of. Additionally, IT personnel can be familiarized with issues unique to cyber-physical systems. Furthermore, we can utilize the testbed to discover previously unknown vulnerabilities.

Zusammenfassung

Operational Technologies (OT) sind industrielle Informationssysteme, die direkt mit der physischen Welt interagieren. Ehemals isolierte Netzwerke werden vermehrt mit den IT-Netzwerken von Unternehmen und dem Internet verbunden. Dies führt dazu, dass bisher unbeachtete Probleme wiederholt in Erscheinung treten. Viele Komplikationen werden durch den Ansatz "Security-through-Obscurity" verursacht, andere basieren darauf, dass diese Netzwerke in Abschottung entstanden sind. Folglich werden OT-Netzwerke gehäuft zum Ziel von Cyberattacken, jedoch stehen sie auch verstärkt im Fokus der Cybersicherheitsforschung. Hierfür benötigen wir eine realistische Umgebung, in der auf sichere Art und Weise verschiedene Aspekte der Informationssicherheit untersucht werden können. Ein wichtiges Element von OT-Umgebungen sind Speicherprogrammierbare Steuerungen (SPS). Viele bestehende Testumgebungen betrachten diese jedoch nur ungenügend. Dafür analysieren wir die Anforderungen, die von einer solchen Umgebung verlangt werden und entwickeln folglich ein Testbed, das auf Simulation, Emulation und Virtualisierung basiert. Die Testumgebung kann dann verwendet werden, um Personal aus dem OT-Bereich auf unbekannte Risiken hinzuweisen. Des Weiteren ist sie dafür geeignet, IT-Sicherheitsexpert*innen mit den speziellen Problemen vertraut zu machen, die in cyber-phyischen Systemen auftreten. Außerdem können wir damit neue, noch unentdeckte Schwachstellen erkennen.

Contents

1 Introduction **1**
- 1.1 Problem . . . 2
- 1.2 Motivation . . . 3
- 1.3 Objectives . . . 4
- 1.4 Outline . . . 4

2 Background **5**
- 2.1 Industrial Control Systems . . . 5
  - 2.1.1 Supervisory Control and Data Acquisition Systems . . . 6
  - 2.1.2 Distributed Control Systems . . . 7
  - 2.1.3 Programmable Logic Controllers . . . 7
  - 2.1.4 Modbus Protocol . . . 8
- 2.2 Attacks on Industrial Control Systems . . . 11
  - 2.2.1 General ICS Attacks . . . 11
  - 2.2.2 Attacks on PLCs . . . 12
- 2.3 Simulation, Emulation and Virtualization . . . 13
  - 2.3.1 Simulation . . . 13
  - 2.3.2 Emulation . . . 14
  - 2.3.3 Virtualization . . . 15
  - 2.3.4 Drawing the Lines . . . 16

3 Related Work **17**

4 Testbed Conceptualization **20**
- 4.1 Required Properties . . . 20
  - 4.1.1 Fidelity . . . 20
  - 4.1.2 Repeatability . . . 22
  - 4.1.3 Measurement Accuracy . . . 22
  - 4.1.4 Safe Execution of Tests . . . 22
  - 4.1.5 Other Desirable Properties . . . 22
  - 4.1.6 Complementary and Conflicting Goals . . . 23

4.2 Required Components 24
4.3 Testbed Components 24
4.4 Choosing between Simulation, Emulation and Virtualization . 26
4.4.1 Advantages and Disadvantages of Different Approaches 26
4.4.2 Choosing the Correct Approach 28
4.5 Simulating an Industrial Process 30
4.5.1 Defining the System 31
4.5.2 Defining the Simulation Model 32
4.5.3 Defining the PLC Output Behavior 33

5 Implementation of the Testbed 34
5.1 Virtualizing the Hosts and Network 34
5.1.1 Choosing an Operating System 34
5.1.2 Choosing a Virtualization Environment 35
5.1.3 Automating the VM Creation Process 36
5.1.4 Virtualized Network 37
5.2 Implementing the PLC 37
5.2.1 OpenPLC 38
5.2.2 Writing the PLC Program 38
5.2.3 OpenPLC configuration 40
5.3 Implementing the Process Simulation 41
5.3.1 The Simulator 41
5.3.2 The Visualizer 42
5.4 Implementing the HMI 42
5.5 Implementing the Data Historian 43

6 Using the Testbed for Attack Demonstration 47
6.1 The Adversary 48
6.2 Manipulating the PLC's View 48
6.2.1 Variant: Redirecting Traffic 49
6.2.2 Variant: Packet Modification 50
6.2.3 Implications 50
6.3 Manipulating the PLC behavior 52
6.3.1 Implications 52
6.4 Other Possible Attacks 53
6.5 Discussion 53
6.6 Limitations 55

7 Conclusion 56
7.1 Status 56
7.1.1 Strengths and Weaknesses 56

7.1.2 Realized Goals 57
7.1.3 Open Goals 58
7.2 Future Work 58

Chapter 1

Introduction

Industrial networks have become more connected to IT networks and the Internet [1]. In order to effectively discuss the problems that industrial networks face we need to define the terms *information technology* (IT), *operational technology* (OT) and *industrial control system* (ICS). Gartner defines IT the following way:

> "*[IT] is the common term for the entire spectrum of technologies for information processing, including software, hardware, communications technologies and related services. In general, IT does not include embedded technologies that do not generate data for enterprise use* [2]."

From this definition we can observe that the primary function of IT relates to information and its processing.

We will also use Gartner's definition of OT, which is:

> "*[OT] is hardware and software that detects or causes a change, through the direct monitoring and/or control of industrial equipment, assets, processes and events* [3]."

From this definition we can draw an important distinction from IT, which is the fact that OT monitors and controls another physical system. A common type of OT system are ICS's, which is a general term for systems used for industrial automation [4]. Depending on interpretation of the term OT is not limited to ICS's, but can also include systems such as planes or trains. We

use both OT and ICS in this introduction, because the described problems apply to many kind of OT systems. Later sections prefer the term ICS, as they are more focused on ICS's rather than OT in general.

This introduction begins with Section 1.1 by explaining the problems faced by OT and ICS's. It is followed by two sections explaining the motivation and objectives of this thesis. The chapter concludes with a section detailing the thesis' outline.

1.1 Problem

IT and OT/ICS's have and are continuing to converge [1]. Connecting IT systems to OT networks affects the security of OT networks [4]. For instance, it makes them more vulnerable to malware that targets operating systems such as Windows. This is worsened by the fact that unsupported OS's are common [4]. Interest in the exploitation of ICS's has increased [5]. However, a number of high level attacks have also heightened the alertness of operators and researchers. An event which this frequently has been attributed to is the Stuxnet attack [1]. The attack was first uncovered in 2010, in the years that followed, we can observe a significant increase in vulnerability disclosure [5]. Despite an improved recognition of the need for better security measures in the world of OT, some still consider its security to be "*still ... in its infancy*"[1].

One common component in ICS's are *programmable logic controllers* (PLC), which can be a desirable target for attackers [6]. The previously mentioned Stuxnet attack is one of the incidences where they were targeted [7]. PLCs are used to control industrial processes, Chapter 2 describes them in more detail and names common attack types.

One key difference of IT and OT is a different valuation of common security goals [4]. For instance, IT usually sets the highest focus on the first two letters of the CIA triad, confidentiality and integrity. The focus is on concealing information from unauthorized parties and ensuring that it is not unintentionally or maliciously modified. On the other hand, OT is primarily focused on the security represented by third letter, availability [4]. An-

Issue	IT	OT/ICS
Real-time requirements	No	Yes
Response time	Should be consistent.	Is time-critical.
Throughput requirements	High.	Medium.
System reboot	Is acceptable.	Needs careful planning and is costly.
Hard and software	Commodity.	Proprietary and specialized.
Communication	Standardized protocols.	Proprietary protocols.
Updates/Upgrades	Can be done timely.	Complicated, legacy OS's frequent.
Life-time	3-5 years	Up to 25 years [8].
Access	Easy.	Hard, remote or isolated locations.

Table 1.1: Differences between IT and OT/ICS, adapted from [4].

other important distinction between IT and OT is the cyber-physical nature of OT. The safety of personnel, the environment and equipment has to be considered. We might need emergency off-switches that could conflict with authentication requirements. Table 1.1 summarizes important differences of IT and OT/ICS. These differences and an overall poor awareness show the need for specialized testing environment for OT/ICS networks.

1.2 Motivation

We observe a need to educate on issues that ICS networks face and feel that this requires a practical display. Awareness can be poor and known solutions to common vulnerabilities are not applied. For instance, HTTP traffic is unencrypted, passwords are sometimes weak, left to their default or even hardcoded [8]. However, there are problems that do not have an easy solution, these need to be highlighted as well. For that, we require an appropriate environment. Additionally, the majority of other ICS testbeds are not focused on field devices such as PLCs [9]. These controllers are one of the key focuses of this document.

1.3 Objectives

The goal of this thesis is to develop a virtual testbed that can be used to demonstrate attacks on PLC networks. We want to have a network centered around a PLC. During the developments we need to address the unique demands posed to an ICS related testbed. We also want to highlight attacks that are specific to ICS's. The purpose of the testbed is to educate on attacks, vulnerabilities and threats. The other major goal is to provide a context in which new vulnerabilities can be discovered in an environment that is both realistic and safe.

1.4 Outline

This thesis contains six more chapters. Chapter 2 is focused on providing important background information that is necessary to understand the context in which ICS's and PLCs are used. Additionally, we also provide information that we will build on in later chapters. It is followed by Chapter 3 which discusses related literature. The testbed is conceptualized in Chapter 4 and subsequently implemented in Chapter 5. Chapter 6 then evaluates the testbed's ability to demonstrate attacks. The thesis finally concludes in Chapter 7 with a summary about the testbed's status and possible future research directions.

Chapter 2

Background

This chapter introduces important concepts required to understand the content of later chapters. It begins by introducing ICS's in 2.1 and its important related topics. The section also contains an introduction to the Modbus protocol, which is used extensively in the testbed and common in industry. Section 2.2 covers general attacks on ICS's and PLCs. The final section defines the terms simulation, emulation and virtualization, which are relevant concepts as the testbed is not a physical one.

2.1 Industrial Control Systems

ICS is a generic term for a wide array of cyber-physical systems. They are used in industry and critical infrastructure to automate and control processes, which can range from electrical grids to chemical plants. They serve as an important backbone for modern society. ICS's can consist of many different pieces of technology and are not a homogenous technology. Typical components include [4, 8]:

- Human machine interfaces (HMI) are used to monitor processes, raise alarms and diagnose failures.

- PLCs read sensor values as input and control actuators as outputs. Refer to section 2.1.3 for more details.

- Remote diagnostic tools
- Maintenance tools
- Historians: Keep a record of the system state.
- Actuators: Interact with physical systems, e.g. a servo.
- Sensors: Measure properties of the physical system, e.g. a thermometer.

ICS's are often hard real time[1] systems, this differs from IT where deadlines are usually not as strict [4]. A typical ICS deadline would be the time it has to react to changing sensor values. Like all OT systems, ICS's have changed drastically in the past decades, they have become more connected to corporate IT networks or the Internet [4]. This leads to security problems, as these systems were not designed with cybersecurity in mind. However, interest in researching ICS related security issues has increased in recent years [1, 5].

Sections 2.1.1 and 2.1.2 introduce systems in which PLCs are used as part of their subsystems. Understanding them is relevant to provide information about the context in which PLCs are typically used.

2.1.1 Supervisory Control and Data Acquisition Systems

Supervisory Control and Data Acquisition (SCADA) systems are hierarchical systems, they are used to control multiple processes that depend on each other and are not limited to a single geographic area [4]. An example for such a system could be a power grid. SCADA systems are focused on central control and data collection [4]. They can have control centers which are not local to the process itself, but rather one that is connected to it over wide area networks like the Internet [4].

[1] *hard real-time:* missing deadlines can cause damage to the system [10]

2.1.2 Distributed Control Systems

An alternative to SCADA systems are *distributed control systems* (DCS). These systems are limited to a single geographic location, for example a power plant. In this scenario the hierarchy is flatter, with fewer levels on top of the localized process controlled by PLCs [4]. These localized processes work together as part of a bigger process, but can be fault tolerant from each other [4].

2.1.3 Programmable Logic Controllers

PLCs[2] are devices used to control processes in industry and critical infrastructure. PLCs are used in DCS, SCADA systems or as independent controllers [4]. This equipment is designed to work in rough environments, resistant to temperature, vibration and other physical disturbances [8]. PLCs essentially run through the following cycle:

1. Read the PLC's inputs.
2. Run the PLC's program.
3. Write to the PLC's outputs.

Inputs can be sensors and outputs can be actuators. The latter having a direct effect on the environment, such as a light or a servo. The programs that control the outputs based on the inputs are standardized in IEC 61131-3 and include five programming languages: Ladder Diagrams (LD), Function Block Diagrams (FDB), Instruction Lists (IL), Structured Text (ST) and Sequential Function Charts (SFC) [11]. Figure 2.1 illustrates a simple LD program. It has two input buttons, if either of them is set to true the light is turned on.

The two biggest PLC vendors are Siemens and Rockwell, who together account for more than half of the market share (31% and 22% respectively) as of 2017, according to Deutsche Bank Markets Research [13].

[2]Frequently referred to as *Speicherprogrammierbare Steuerung* (SPS) in Germany.

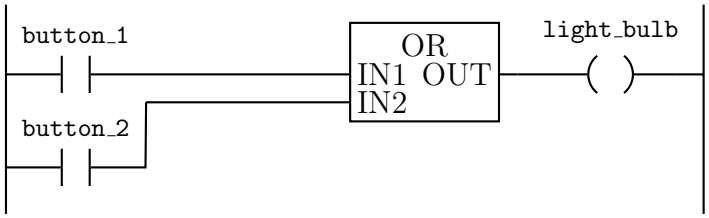


Figure 2.1: Example IEC 61131-3 LD program, adapted from [12].

2.1.4 Modbus Protocol

Modbus is a protocol that is widely used in ICS's and especially with PLCs. It is a client-server protocol, servers are often called slaves and clients are masters. It can be used to communicate between the following endpoints:

- PLC ↔ HMI
- PLC ↔ Historians
- PLC ↔ Sensors
- PLC ↔ Actuators

It is an open protocol standardized by the Modbus Organization and the specification is publicly available free of charge.[3] It is an application layer protocol and can run on top of asynchronous serial connections like RS-485 or TCP/IP [14]. Modbus TCP/IP is covered later in this section, the serial version is not covered, because the testbed currently does not use any serial connections. The general communication of the protocol is shown in Figure 2.2. The client sends a request to the server, which receives it as an indication. The server then responds with a confirmation to the client. Different types of requests are identified by function codes. This section also covers a selection of them.

Modbus TCP/IP

Modbus was originally a serial protocol, but it has become available to be used with TCP as well. The TCP port 502 is reserved for Modbus. Figure 2.3

[3] https://www.modbus.org/specs.php

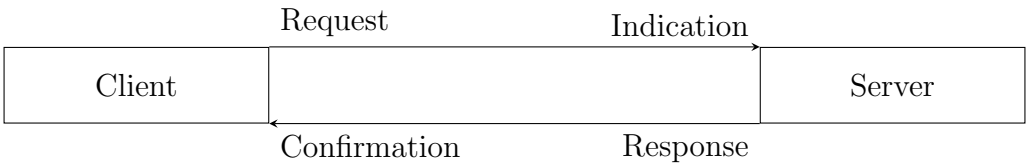


Figure 2.2: Modbus Request and Response, adapted from [15].

shows how it fits into the TCP/IP model as an application layer protocol. The structure of a message slightly differs for serial Modbus connections, but they share an identical Modbus PDU (Protocol Data Unit). It also uses a different method of addressing and includes an error checksum. The checksum in Modbus TCP/IP is already provided by TCP. The PDU and the MBAP together form the application data unit (ADU). Figure 2.4 shows the structure of a Modbus TCP/IP packet [15]. The transaction identifier is used to map responses to requests, it is chosen by the client. The protocol identifier is a constant, it is always filled with zeros. The length field specifies the amount of following bytes, including the unit identifier. The identifier specifies a modbus slave on serial buses, but is also included in the TCP variant for gateway compatibility. The PDU can be longer than the five bytes as shown in Figure 2.4 and as short as a single byte. The maximum length is 253 bytes. The next part covers a few specific Modbus PDUs in more detail.

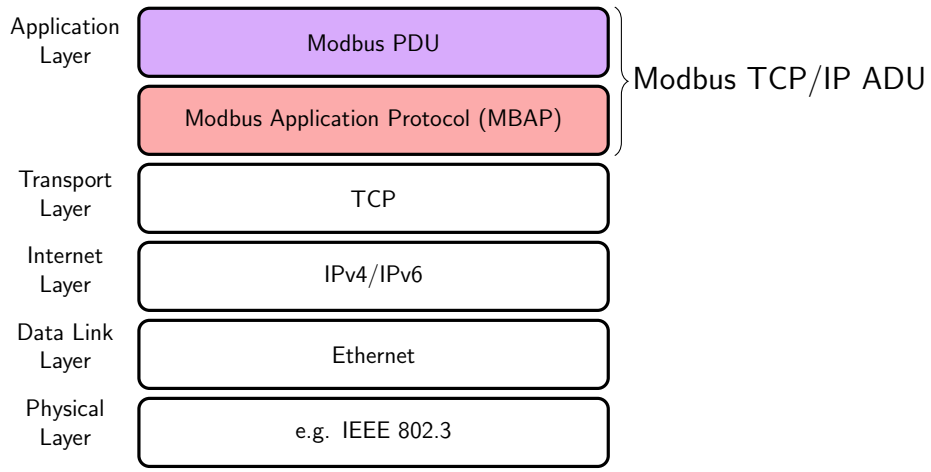


Figure 2.3: Modbus TCP/IP in the five-layer Internet model.

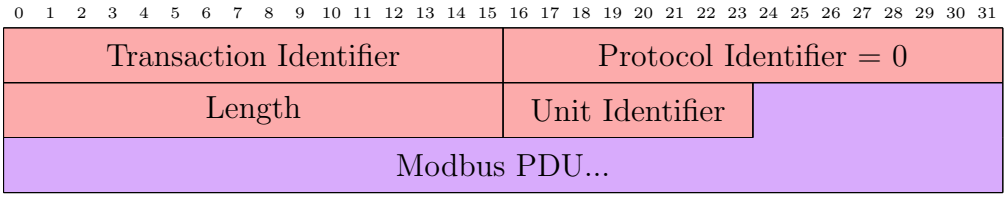


Figure 2.4: Modbus TCP/IP frame showing MBAP and PDU.

Function Codes and Storage Tables

Modbus identifies operations with function codes. The Modbus Organization standardizes some of them, but there exist ranges which can be used for custom functions. To understand the different function codes, one needs to understand how data is stored and how it can be accessed. Figure 2.5 shows the four primary storage tables of Modbus [14]. Coils and input discrete inputs store single-bit values. Holding and input registers store 16-bit values. The entries are accessed via addresses. Only coils and holding registers can be written to by clients, the other two can be mapped to the servers I/O devices. The access rights are reflected in function codes. This thesis covers three

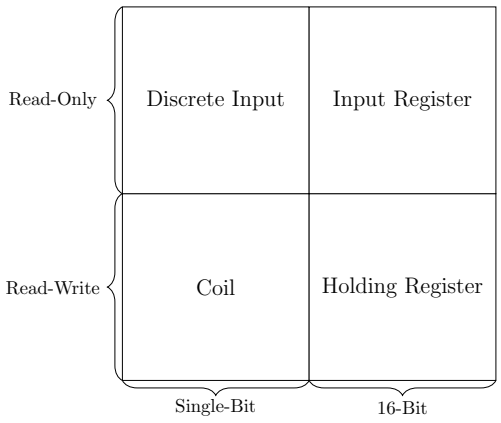


Figure 2.5: The four primary Modbus tables.

groups of them in general. Function codes like exceptions are not covered. The groups are read, write-multiple and write-single functions. Read requests specify a range of continuously numbered addresses and receive a response with all requested values. Read requests can be used for coils (0x01), discrete inputs (0x02), holding registers (0x03) and input registers (0x04). Multiple

coils can be written to with function code 0x0F and multiple holding registers with 0x10. Individual coils and holding registers can be written to with 0x05 and 0x06. Figure 2.6 shows the ADU of a write single register request and response. It writes 0x0100 into the third holding register. Request and response ADU are identical in this Modbus function.

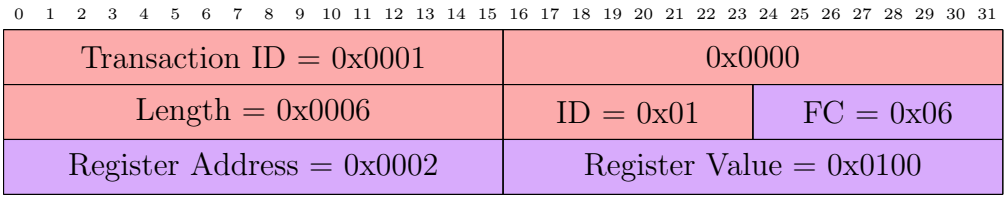


Figure 2.6: Modbus TCP/IP ADU of a write holding register request.

2.2 Attacks on Industrial Control Systems

This section introduces some attacks on ICS's, it cannot cover them in depth and exhaustively. For more information and guidelines refer to NIST 802-82 [4] or the BSI ICS security compendium [8]. The article [5] provides an overview of developments regarding attacks on SCADA systems. For taxonomies related to ICS's refer to: [16] for cyber physical systems in general, [17] for SCADA and [18] for Modbus. For a survey about SCADA incidents with a selection of concrete attacks refer to [19]. Nonetheless, Section 2.2.1 introduces some general attacks on ICS's and Section 2.2.2 covers attacks that are related to PLCs.

2.2.1 General ICS Attacks

ICS's are also vulnerable to many attacks and threats familiar to people with knowledge about IT systems. The overall awareness of threats is lower than in IT [8], but it has improved since the Stuxnet attack [5]. An apparent issue are bad practices such as weak or default passwords or cross-site-scripting (XSS) vulnerabilities, leaving configuration systems ripe for exploitation [8]. Other attacks and vulnerabilities include [8]:

- **Protocols lack encryption, authentication and integrity protections:** Protocols like Modbus do use encryption and message authentication codes.
- **Hard coded passwords:** Insecure default credentials cannot be changed.
- **Inadequate documentation:** Security relevant information such as vulnerabilities are not documented despite being known. Connections between part of the networks are not documented properly to an incomplete view of it.
- **Insecure remote maintenance access points:** Remote access without encryption (e.g. telnet) or management interfaces without proper authentication.

Another issue common to ICS's, which can cause serious security issues is the usage of equipment that has a long lifespan. They can be in service for up to 20-25 years, running on unsupported legacy operating systems. These operating systems might have widely known exploits and are not protected against new attacks [8]. For adversaries it is interesting to begin their attacks targeting corporate networks and from there propagating towards more isolated systems such as SCADA or DCS [4]. As control networks are often not directly connected to the Internet, despite not being air gapped. Initial entry can be gained by phishing or employees accidentally bringing their own infected devices to the office [8].

2.2.2 Attacks on PLCs

PLCs can be desirable targets for exploitation [5]. Stuxnet, one of the most famous ICS attacks, targeted PLCs. It uploaded malicious code to these controllers, which is an important attack type [7]. A consequence of this can be misbehaving actuator output, causing damage to physical systems. Other possible attacks target the communication between PLCs and other components in the network. This stems from the issue that many protocols

do not provide any confidentiality, authentication or integrity protections. It allows Man-in-the-Middle (MitM) attacks which can lead to: a manipulation of the view of a system, the total loss of view or a total loss of control. The protocol described in Section 2.1.4 is vulnerable to this, which becomes apparent in Chapter 6. Other consequences include impersonation and replay attacks. I/O pins of PLCs can also be attacked directly, as shown by [20]. It shows an attack on the integrity and availability of I/O pins used by PLCs. The manipulation of I/O devices can be very hard to spot. PLCs can also be flooded with requests, which might cause delays to the PLC cycle shown in Section 2.1.3 [21]. Another issue common to PLCs is a lack of quality access control, frequently allowing unauthorized access to attackers [6].

In summary, attacks on PLCs usually target either their firmware, their view of the system, their control of it or their access control methodology.

2.3 Simulation, Emulation and Virtualization

We need to define the terms *simulation*, *emulation* and *virtualization*. There are differences between the terms, even though these techniques definitely overlap and have common properties. Common advantages are reproducibility, inferring information about real-world systems in a cost efficient way. Section 4.1 also discusses how these properties relate to requirements when constructing testbeds.

The goal of the following three sections is to define these terms, in order to avoid ambiguities and allow accurate description of components especially in Chapters 4 and 5.

2.3.1 Simulation

In [22] the author J.Banks defines *simulation* the following way:

"*Simulation is the imitation of the operation of a real-world process or system over time. Simulation involves the generation of an artificial history of the system, and the observation of that artificial history to draw*

inferences concerning the operating characteristics of the real system that is represented."

In summary, we want to replicate a real-world system, in order to approximate the behavior of it over a certain timespan.
A simulation model includes, but is not limited to the following components [22]. The *system* to be simulated, for example a subway system. A *model* that mirrors the system, building on our example, let us say a graph $G = (V, E)$ that represents the train tracks as edges E and stations vertices V. *Events* are points in time that change the system model. There are two kinds of events, internal ones (a train arrives at a station) or external ones (a tree falls on the tracks). *System state variables* describe the system at a moment, may it be the amount of railroad switch failures during rush hour. *Entities* are objects that require further descriptions with *attributes* (e.g. the passenger capacity of a train). They can either be static (railroad switches) or dynamic (trains), the deciding difference is whether they move through the system. *Resources* are required by dynamic entities in order to navigate the system, for instance train tracks. *Activities* are deterministic time periods with a duration that is known in advance. *Delays* are spans of unknown length resulting from the system state. We can use these terms to define and describe simulations. They are used in Section 4.4.2 to describe the model of an industrial process. Overall we can say that simulation is a general term and technique that can be applied to many things.

2.3.2 Emulation

Emulation is a more specific form of simulation [23]. It is a hybrid approach, using simulation for some aspects and real-world technologies for others (usually software). A common usage of emulators is the emulation of processor architectures. For example, the Dolphin Emulator[4] replicates an IBM PowerPC architecture in order to run Nintendo GameCube and Wii games on other architectures such as x86 or ARM. Another use case can be software

[4]Dolphin Emulator FAQ: https://dolphin-emu.org/docs/faq/

that replaces real hardware components. This is what we are going to do with the PLC in Section 5.2.

2.3.3 Virtualization

Virtualization is a method with the goal to decouple the relationship between physical hardware and abstractions of it [24]. Hardware components such as CPUs, memory, storage and network interfaces can be virtualized in order to run multiple operating systems side-by-side [25]. Another use case is virtual memory, which allows computer programs to address more memory than is physically available. Software using the virtualized hardware does not have to coordinate the usage of shared hardware with other guest systems. The virtualized hardware can be used like dedicated hardware and is often indistinguishable from real equipment from an interface perspective.

The two following definitions are taken from the article [25]:

- *Host Machine*: The physical machine that provides the hard- and software necessary for virtualization.
- *Virtual Machine (VM)*: A node that has its own operating system and virtualized hardware. The operating system can differ from the host machine

We also need to define the term *hypervisor*, the article [25] defines it differently to the one we use. It equates hypervisors with type 1 hypervisors and *virtual machine monitors* with type 2 (hosted) hypervisors. We use hypervisor synonymously for both types. Hypervisors manage the operation of the virtual machines and sit between the guest OS and depending on the type either directly on the hardware (bare metal) or on top of the host OS. Bare metal hypervisors offer better real time properties. In summary, virtualization has the primary goal of decoupling hardware from software. It also provides high realism, software components can be real operating systems and technology stacks. That act almost identical to those running on dedicated hardware. Many of those operating systems can run in parallel, using the same physical hardware without knowing about each other.

2.3.4 Drawing the Lines

Each of these techniques offers different levels of control and realism. Control being highest in general simulation and reduced with emulation and virtualization. Virtualization has the highest realism, because real software can be used together with accurate representations of real physical hardware. They all share similar advantage, especially reproducibility, safe test execution and inferring conclusions about real systems [22, 23, 25]. Virtualization provides the highest accuracy, but not everything can be virtualized. It is limited to computational systems and not applicable to physical simulation. We consider the following relationship shown by the euler diagram in Figure 2.7.

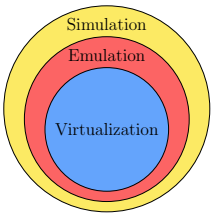


Figure 2.7: Relationship of simulation, emulation and virtualization

Chapter 3

Related Work

There exist many different testbeds that attempt to replicate ICS networks. In our research we found three relevant survey papers that examine these kind of testbeds, one other testbed that applies similar techniques and one paper that proposes a certain approach to SCADA testbeds. The authors Holm et al. investigated 30 ICS testbeds [9]. They analyzed requirements demanded from this kind of testing environment and name a set of requirements and components that need to be addressed during the development of one. They studied whether and how testbeds address these requirements and noted that as of 2015 no testbeds emulated or virtualized field devices. The stated requirements directly affected the conceptualization of our testbed in Chapter 4.

The second survey, authored by Qassim et al. analyzes different realization approaches of individual testbed components and discuss their advantages and disadvantages [26]. The authors analyzed the following approaches: physical replication, simulation, virtual, virtual/physical, and hybrid. They assess how well the approaches address the different kind of requirements one can pose to a testbed. The stated requirements largely overlap with the ones from the first survey paper. The observed strengths and weaknesses were also considered during the conceptualization of our testbed in Chapter 4.

Geng et al. also cover different implementation approaches [27]. For each approach, they give a number of example testbeds and discuss the specific

system. Additionally, they analyze the application scenarios a testbed can attempt to target. They observe a need for virtualized testbeds and ones that employ hybrid approaches.

The testbed named GRFICS[1] by Formby et al. has a similar approach to our testbed [28]. They also use OpenPLC as a PLC, but use AdvancedHMI as a machine interface, we use ScadaBR for this purpose. The testbed also includes visualization, realized with the Unity game. In our testbed we use a simpler approach in a programming language (Python) is commonly known, allowing others to easily extend the testbed. Their testbed topology includes three VMs based on VirtualBox, one for the PLC, one for the simulation and one for the HMI. In our testbed we have two additional nodes, a data historian and an attacker.

Alves et al. propose an approach that can be used as a framework when developing SCADA/ICS testbeds, especially those with multiple PLCs [29]. They do not present a specific testbed, instead they evaluate their modular approach with four case studies. They show that virtualized testbeds are suitable for high fidelity ICS testbeds. The authors also employ OpenPLC and ScadaBR as their PLC and HMI, respectively. In their proposal, they do not include a dedicated data historian, but rather use the HMI for record keeping. We additionally utilize a data historian which provides us a building base for a useful dashboard solution and the ability to extend it in the future. They also simulate the industrial process with software, like most testbeds do [9]. The simulation is based on the proprietary software Simulink, which communicates with virtual sensors and actuators over UDP. Afterwards, they use an additional step to communicate values from and to the PLC over TCP. Our simulation is implemented in a single Python program. One important distinction is that this thesis introduces a specific testbed that can be used, whereas their work proposes a general approach.

The contribution of this thesis is the development of a simple, yet complete ICS/PLC testbed that has a largely automated setup. In our implementation we only use open-source components, making our testbed free from potential licensing costs. We also show that our testbed is suited for the

[1] GRFICS on GitHub: `https://github.com/djformby/GRFICS`

demonstration of attacks in order to raise awareness for industrial security. Furthermore, we can can confirm commonly raised concerns about ICS protocol security evident by our high fidelity implementation of MitM attacks. Another exploit that closely relates to PLCs is covered as well. The attacks we discuss are covered in detail and causes are named. Additionally, we found a potential vulnerability in OpenPLC which substantiates the claim that our testbed is applicable to cybersecurity research.

Chapter 4

Testbed Conceptualization

This chapter constructs the testbed, it begins by analyzing requirements in Sections 4.1 and 4.2. Section 4.4 details the advantages of different realization approaches and chooses a technique for each component we demand in Section 4.3. The final section introduces our simulation model. Figure 4.1 provides a summary of this process.

4.1 Required Properties

In order to construct a useful testbed, we need to address important requirements. The survey paper by Holm et al. names four important properties of ICS testbeds: fidelity, repeatability, measurement accuracy and a safe execution of tests [9]. The following four sections explain each of these requirements. It also introduces further desirable properties in Section 4.1.5. Table 4.1 summarizes the requirements and the goals we have towards each of them.

4.1.1 Fidelity

This property describes the accuracy with which a testbed mirrors a real system we want to study [9]. We want fidelity to be as high as possible, in order to be able to gain meaningful insights. There are different methods

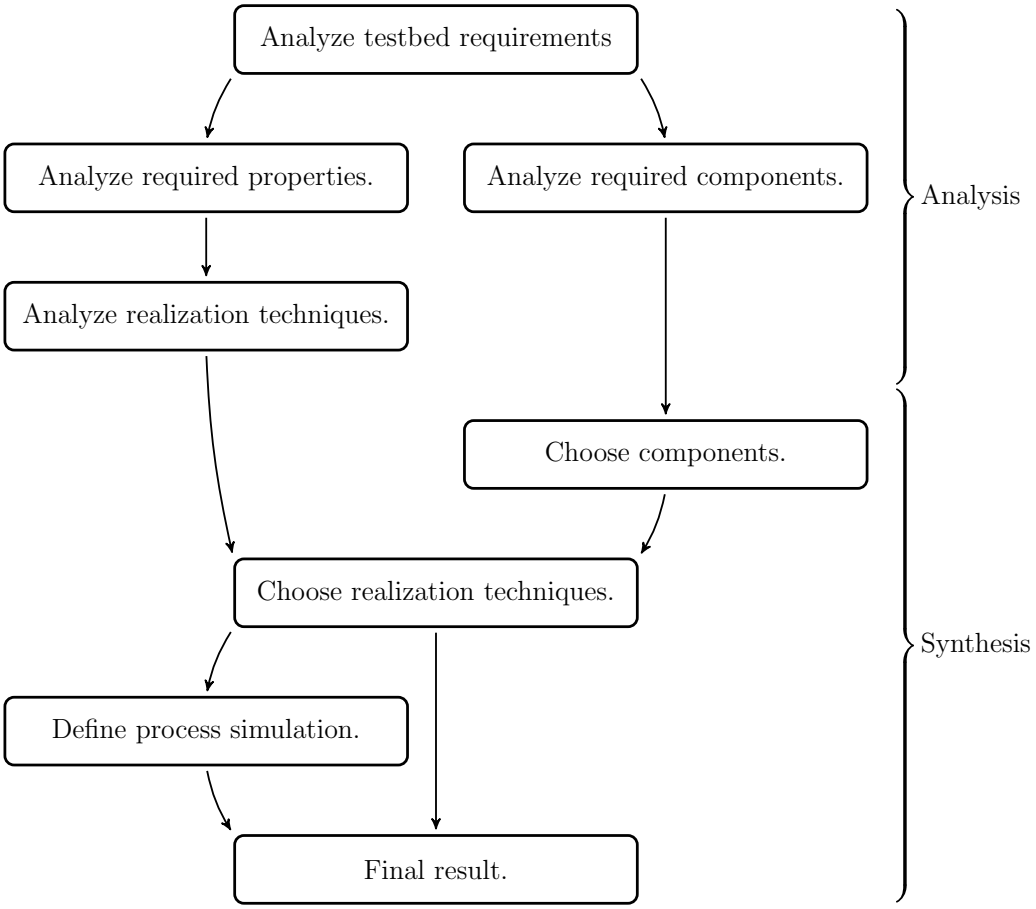


Figure 4.1: Conceptualization steps.

Requirement	Goal
Fidelity	High fidelity, replicate real world systems.
Repeatability	Tests can be repeated, with consistent results.
Measurement accuracy	Results are not affected by measuring.
Safe execution of tests	Testing cause no harm to humans, the environment or the system.
Real-time	Address potential real-time requirements.
Cost	Make the testbed accessible by having a low cost.
Open-source	Utilize open-source components wherever possible.

Table 4.1: Requirements and corresponding goals.

of assuring high fidelity, one is studying real systems, another one is basing your system on standards.

4.1.2 Repeatability

This requirement stems from the fact that results from experiments need to be verified. One way of assuring this is the repetition of tests and making sure that the results stay consistent. There are two ways we want experiments to be repeatable: First they should have consistent results. Secondly the testbed should not enter a state that does not allow repetition. For instance, it could be damaged by the first iteration of tests.

4.1.3 Measurement Accuracy

Measurement accuracy describes the extent to which the outcome of tests is affected by observing them [9]. For example, using network observation and scanning tools can seriously impact the performance and behavior of components [30]. We want to be able to accurately measure the effect of incidents, without the experiment being altered to the point of producing unreliable data. Some minor alterations of the behavior might be acceptable. For instance, we can accept minor latency increases when analyzing vulnerabilities unrelated to real-time constraints.

4.1.4 Safe Execution of Tests

Another important requirement is the safe execution of tests [9]. Experimentation and testing should not endanger any personnel, bystanders, computer or physical systems. Malfunctioning industrial equipment could lead to dangerous effects on the environment. Real malware could infect equipment that is used to conduct tests, but itself is not part of the testbed.

4.1.5 Other Desirable Properties

This section introduces a number of other requirements that are desirable for ICS testbeds.

One important difference between OT/ICS and IT is the extent of real-time requirements. In IT these are usually soft and the occasionally dropped packet does not lead to system failure. This differs for some control systems, where delays can lead to shutdowns or failure. In order to accurately study these systems, we need to account for this requirement. The importance of real time requirements depends on the conducted test. For example, exploiting the weak password of a component.

Another goal is that the testbed should be affordable, in order to reduce the barrier of entry to people who want to use it. For example, a physical testbed can be very expensive [26]. We also want components to be open-source if possible, allowing us to gain a deeper and transparent insight into the root causes of problems. Proprietary soft or hardware might have incomplete or unavailable documentation. Therefore, reverse engineering might be the only possibility to gain more insight, which can be a time consuming process. Using open-source software also reduces the price.

The last goal is the following: the manual setup of a testbed should be minimized. This reduces deviations of the setup caused by different interpretations of instructions or their lack of quality. Automated setup steps should be transparent and comprehensible. This requirement can be considered being part of repeatability. We also want the testbed to be useful for educational purposes, such as demonstrating attacks.

4.1.6 Complementary and Conflicting Goals

Some of the previously named properties can conflict with each other. For instance, wanting the highest possible fidelity at a low cost is impossible. The reason for this is the fact that even single pieces of real-world equipment can be very expensive. They can also be of such a large scale, that its replication adds up to a significant sum. The desire to use open-source software also can conflict with the goal of wanting to have the highest fidelity. Fact of the matter is that many components used in industry are proprietary. Replacing those with open-source elements has the consequence that some kind of attacks are harder to study. This especially includes codebase-specific

attacks such as buffer overflow attacks. Nonetheless, using open-source software greatly reduces the cost of the testbed. A low cost and open-source components will be prioritized, as we consider the benefits of this to be more important than the downsides of having a slightly reduced fidelity and transferability.

4.2 Required Components

The survey paper by Holm et al. [9] uses NIST 800-82 [4] to derive a number of component types that need to be addressed during the development of an ICS testbed. The components that are demanded from an ICS related testbed are consistent across literature [9, 26, 27]. The first required part are *control centers*. HMIs and data historians are considered to be part of this set of components. Even though not all use-cases of PLCs include an HMI or historian, including these components increases the amount of covered setups. The second important issue that needs to be addressed is the *communication architecture*. We need to discuss this because the type of connection can have huge implications on attacks. For example, denial-of-service attacks are easier with wireless networks. Adversaries can jam them without gaining entry to the network and only need access to the transmission medium.

The third type of component are *field devices*, which include PLCs. The fourth is the *physical processes* that are controlled by the control system itself. This component is also referred to as *industrial process* in this document. The authors of [9] also state that network protocols should not be omitted when discussing an ICS testbed.

4.3 Testbed Components

This section derives most of the required components from the information provided in Section 4.2.

The first group named in the previous section were control centers, for this we include an HMI and a data historian. The HMI allows operators to

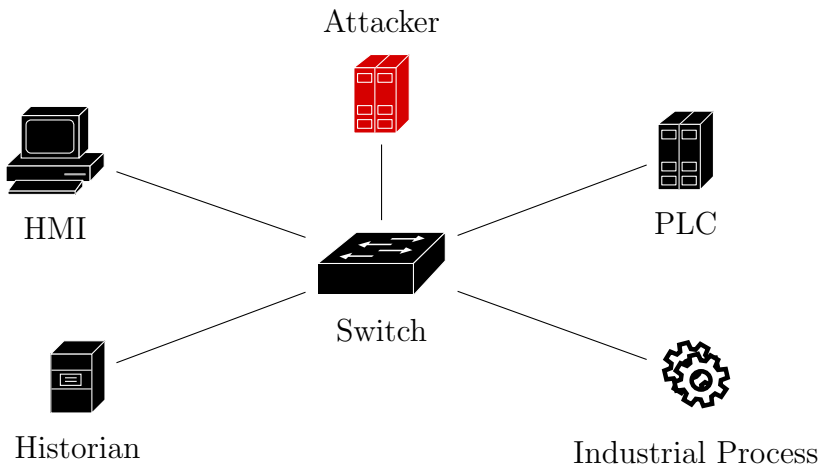


Figure 4.2: Network Topology of the testbed.

control the process, for example, by turning it on or off. Some HMIs provide a feature for data collection and record keeping, but a fully fledged historian offers more features and less coupling. The second component we need to include is a field device, we choose a PLC, as those are a primary focus of the document. Furthermore, PLCs are one of the most important types of field devices. We include an industrial process that is controlled by the PLC and observed by the HMI and historian. The last entity we include in the network is an attacker, who has gained access to the network. The last point from Section 4.2 we need to address is the communication architecture. Most communication between components such as PLCs, HMIs and data historians happens over switched networks [4]. Consequently, we choose a switched network.

Figure 4.2 visualizes the final testbed topology. It includes an attacker, an HMI, a historian, a PLC and a process. All being represented by individual hosts and connected through a switched network.

4.4 Choosing between Simulation, Emulation and Virtualization

In this section we discuss the advantages and disadvantages of different ways to realize the testbed's components. Based on this we choose a technique for each element. It first begins with discussing the general advantages and disadvantages of each approach, as relevant to the ICS testbed requirements from Section 4.4.1. The following section then chooses the methods for each component based on the previous section.

4.4.1 Advantages and Disadvantages of Different Approaches

The paper [26] covers the advantages and disadvantages of different testbed approaches. They examined five different categories of testbeds: physical replication, simulated, virtual, virtual-physical and hybrid ones. Each method was evaluated on how well they address the following requirements: fidelity, repeatability, accuracy, safety, cost-effectiveness, reliability of test results and scalability. The following parts discuss these approaches. In order to stay consistent with our previous definitions, we also discuss it in relation to emulation. Our definition of this term overlaps with the authors' definition of virtual-physical replication.

Approach	Fidelity	Repeatability	Accuracy	Safety	Cost	Result reliability	Scalability
Physical replication	very high	low	high	low	high	very high	low
Simulation	low	high	low	very high	low	Low	very high
Emulation	high	high	medium/high	very high	medium	medium	medium/high
Virtualization	high	high	high	very high	low/medium	high	medium/high
Hybrid	very high	high	very high	high	medium	high	medium

Table 4.2: Advantages and Disadvantages

Physical Replication

The goal of this thesis is to develop a purely virtual testbed, without any physical components. Nonetheless, we need to discuss this approach. It

provides the highest possible fidelity and reliability of test results [26]. This comes at the cost of the lowest repeatability, due to potential equipment damage, and the worst cost efficiency. It also scales poorly, because one might need large physical locations to host the testbed, which also causes a low cost-efficiency. Additionally, tests leading to system failure also might endanger personnel, equipment and the environment.

Simulation

Simulated testbeds can use software to replicate all components such as PLCs, the physical process and communication. This approach can cover almost every component, with the limitation that the fidelity is quite low [26]. No or very few real components are used. Consequently, the approach is not suited to research cyber-attacks. These factors also lead to test results having a lower reliability and accuracy. The benefits of it are: high safety (no harm to physical entities or software components), high cost efficiency, good scaling and a generally good flexibility.

Emulation

This technique can be used for some components, for example, as a hardware-in-the-loop emulation approach [26]. Equipment might be too expensive to include or could be damaged. Sometimes it can be emulated well without sacrificing fidelity. Another possibility is emulating the parts of the physical process that are prone to permanent deterioration, while keeping other segments physical. This technique provides good fidelity, as we can partially use real-world equipment or replace hardware with an accurate software analogue. Other benefits can include high repeatability, accurate and reliable experimentation and great safety. The extent of these benefits depends on the quality of emulated components. A drawback can be high cost if real-world hardware is used.

Virtualization

Virtualization can provide high fidelity for important parts of ICS testbeds, such as SCADA components and networking. Properties where virtualization excels at are repeatability and safety [26]. If testing leads to damaged virtual machines (e.g. corrupted files), they can simply be rolled back to an earlier snapshot. This allows us to repeat tests, even if they cause serious, but non persistent damage to the VMs and other virtualized components. The scaling of virtualization depends on the computational requirements demanded from the host system. For instance, running many VMs might require a lot of CPU cores, memory and storage. This also affects the cost effectiveness. Nonetheless, virtualization allows us to avoid buying specialty hardware. Overall, virtualization is very suited for testbed purposes [9, 26, 27, 29].

Hybrid Approaches

Choosing different techniques for different parts of the testbed can provide the best properties across the board [26]. It is important to pick the most suitable one for the current task or component. The biggest drawback can be cost efficiency, but this chiefly applies to testbeds including real hardware or expensive proprietary software. Combining techniques can still improve the testbed while keeping cost low and still achieving other goals.

4.4.2 Choosing the Correct Approach

HMI and Data Historian

These two components can be covered in the same section because they are both part of the control center and use software that can be run on commodity soft and hardware, such as Windows or Linux running on x86 architectures [9]. This makes the choice relatively easy, we want these components to be fully virtualized and use real-world software. This leads to a high fidelity and accurate representations of real-world systems.

PLC

Choosing the correct technique for the PLC is a lot harder. These components are often specialized hardware, due to tougher environmental demands. Their software is proprietary and information about the CPU architecture or other hardware aspects is not readily available or entirely inaccessible. Another issue is the fact that PLCs are often used for very long time periods and might run on systems that cannot be virtualized at all [9]. We are unable to use real-world hardware for this purpose, as we are developing a virtual testbed. Therefore, we need to find another approach. A possibility is to use software based PLCs that can run on commodity hardware and operating systems. This type of PLC can be seen as an *emulated* PLC. This choice has the consequence that we can only accurately portray software based attacks and hardware exploits are not covered at all. Nonetheless, this approach provides good fidelity, as we use real PLC software. Furthermore, it reduces the cost, considering we do not need to buy specialty hardware. Additionally, we cannot damage the emulated hardware, providing safety in this aspect. We can realize the previous mentioned commodity hardware and operating systems with virtualization. This addresses requirements such as repeatability, accuracy, reliability and additionally increases safety. Experiments targeting the PLC are repeatable because we rollback the VM's state in case of system corruption. Accuracy and reliability are improved by using virtualized hardware and operating systems that can be targeted by common attacks that are not codebase-specific. Safety is increased because we can isolate the system inside virtualized operating systems.

Industrial Process

Real-world industrial processes are inherently physical systems. This leaves us with fewer possibilities to realize them in a testbed. There are three ways to achieve this: physical simulation, software simulation or combining them by simulating the aspects of it that are especially prone to damage. As previously stated, we do not want to use physical equipment. This limits us to using software simulation, which is a good and popular option [9]. It is

the safest and most affordable option. The simulated process is introduced in Section 4.5.

Communication Architecture

The last choice we need to make is on how the hosts communicate. Due to the constraints set by the network topology set by Figure 4.2 and the immediately preceding sections, we do not have much room for decisions regarding the network connections. As we run most components on separate VM hosts, we can simply use fully virtualized ethernet-based networking. This provides good fidelity, as we use the real-world communication software stacks of the VMs and virtualized network interfaces, which appear to the VMs as real hardware interfaces. It allows us to examine network-based attacks between all host connections. Virtualized networking also increases the safety by allowing us to isolate the networking if needed. Additionally, it reduces cost by eliminating hardware elements such as network interface controllers, routers or switches.
Furthermore, we set ourself the constraint to use real-world ICS protocols wherever possible, in order to increase fidelity, accuracy and reliability of test results.

Summarizing the Choices

This section and Table 4.3 summarize the choices we made for each component. We separate the system into the following hosts: attacker, HMI, historian, PLC and industrial process. They run on virtualized operating systems and interface with each other over a virtualized ethernet network. The PLC is emulated as a soft PLC and the process is realized with software simulation. The HMI and historian use real-world software.

4.5 Simulating an Industrial Process

Based on the information and definitions provided by Section 2.3.1, we define the following process and behavior. The goal of the simulation is not to

Component	Type
Operating System	Virtualization
Networking	Virtualization
PLC	Emulation (soft PLC)
HMI	Real-world software
Historian	Real-world software
Process	Software simulation

Table 4.3: Method of realizing important components.

provide a physically accurate model. It primarily attempts to have clear states that represent a misbehaving system. This decision was influenced by the circumstance that no real-world system was available to study and replicate. Furthermore, the author does not have the necessary background to define a physically accurate simulation.

4.5.1 Defining the System

First we need to define the system we want to simulate. It is a boiler or heater filled with water, controlled by a PLC. It can be filled with water and drained of it. For this process, it needs two sensors providing information about the water level. With the two sensors we can differentiate between the tank being empty, partially filled and full. Furthermore, we need a thermometer to measure the temperature and we want a pressure gauge. This results in the following four sensors: thermometer, pressure gauge, lower and upper water level sensors. In order to control it, we need some actuators. One to turn the heater on and off. In order to control the tank's water level, we need another two actuators. One to fill and one to empty it. Additionally we want an on/off-button to control to the process' current operating goal. The process has the following behavior:

- When the heater is turned to *on* the temperature rises. If it is *off*, it sinks over time.
- Pressure is affected by the temperature.

Name	Type	Affects/Affected by	Data Type
`thermometer`	sensor	temperature	integer
`pressure_gauge`	sensor	temperature	integer
`level_sensor_low`	sensor	water level	boolean
`level_sensor_high`	sensor	water level	boolean
`heater_on`	actuator	temperature	boolean
`drain_on`	actuator	water level	boolean
`water_on`	actuator	water level	boolean

Table 4.4: The sensors and actuators of our simulation.

- When the temperature and pressure are above a certain threshold, the process is damaged.
- Damage is accumulated and permanent.
- When the water supply is turned *on*, the water tank fills up.
- When the drain is turned *on*, the tank is emptied.

Table 4.4 summarizes the relationship between the sensors, actuators and the system. It also introduces the names used in later sections.

4.5.2 Defining the Simulation Model

This section defines a simulation model based on the system we described in Section 4.5.1. We describe it with the terms defined in Section 2.3.1. The simulation's I/O model is directly mapped to the sensors.

- **Events:** In our model, we have both internal and external events. The external events are changes to the actuators. There are two kinds of internal events. The first are changes in system state variables leading to different sensor values, such as the water level reaching a certain threshold. Additionally, temperature and pressure moving past a certain boundary lead to system damage.

- **System state variables:** The system state is represented by three variables: temperature, water level and system damage. System damage does not affect any output variables (sensors).
- **Entities:** The drain, water supply and heater can be considered entities.
- **Resources:** The tank can only be in one of the following states: heating up, filling up, emptying or switched off.
- **Attributes:** Our entities do not require any attributes.
- **Activities:** This includes filling up the tank and draining it. Heating up and cooling down.
- **Delays:** The only delays in our model are irregularities caused by the network and computational delays, for example, caused by CPU limitations.

4.5.3 Defining the PLC Output Behavior

The PLC technically is not part of our simulation model, but it interfaces with its I/O. We define the following expected output behavior from the PLC:

- $\texttt{water\_on} := \neg\texttt{level\_sensor\_high} \wedge \texttt{on\_button}$
- $\texttt{drain\_on} := \texttt{level\_sensor\_low} \wedge \neg\texttt{on\_button}$
- $\texttt{heater\_on} := \texttt{on\_button} \wedge \texttt{level\_sensor\_high}$
  $\wedge\ \texttt{thermometer} < \texttt{max\_temp}$
  $\wedge\ \texttt{pressure\_gauge} < \texttt{max\_pressure}$

It is translated into a ladder diagram in Section 5.2.

Chapter 5

Implementation of the Testbed

This chapter covers the implementation of the testbed's individual components. At first, it covers the applied virtualization techniques in Section 5.1 Afterwards, Section 5.2 describes our PLC setup. Section 5.3 details the simulation of the industrial process. Sections 5.4 and 5.5, respectively, describe how the HMI and the data historian are realized.

5.1 Virtualizing the Hosts and Network

In Chapter 4, we decide to virtualize the operating systems and the network. This section describes how these components are realized. The following goals are relevant to their implementation:

- Use open-source components,
- address potential real time requirements,
- automate the setup as much as possible and minimized it for testbed users.

5.1.1 Choosing an Operating System

We choose Ubuntu 18.04 LTS as the operating system as it is widely supported and has a mature ecosystem. We also use the cloud variant of it which

reduces the OS size and gives us many automation options. The cloud image runs *cloud-init*[1] during startup. This allows us to define aspects of the VMs in configuration files. For instance, we can:

- install dependencies,
- run a setup script,
- change the network configuration,
- define users,
- add authorized SSH keys,
- Skip a lengthy operating system installation; this allows us to quickly produce the VMs.

We use an infrastructure-as-code approach. The infrastructure is managed with version control similarly to code [31]. In our case, the infrastructure compromises VM and network configurations. It helps us to reduce deviations between different setups of the testbed and therefore improving repeatability.

5.1.2 Choosing a Virtualization Environment

We decide to use *libvirt*[2] to manage the VMs. The reason is the following: We can define the VMs and the network with configuration files, written in XML.[3] The tool can be used with different hypervisors such as KVM, Xen and VirtualBox.[4] Libvirt provides a python API that enables us to automate the creation of VMs. This builds on our previous effort to automate the VM creation as much as possible, in order to increase usability and repeatability. Furthermore, testbed users can generate the VMs from scratch without having to go through an OS installation process.

[1] cloud-init documentation: https://cloudinit.readthedocs.io/en/latest/
[2] libvirt documentation: `https://libvirt.org/docs.html`
[3] libvirt XML format: `https://libvirt.org/format.html`
[4] List of all supported hypervisors: `https://libvirt.org/drivers.html`

5.1.3 Automating the VM Creation Process

It would be possible to maintain VMs by hand and then distribute VM images. However, a key issue of this approach is an unclear view of the configuration and changes to it. This approach would require careful documentation by hand, which is labor intensive and prone to errors. It would also directly conflict with our goal of having a transparent and comprehensible setup. Automation can provide a more transparent and streamlined setup. The automatic VM creation happens the following way.

A python application creates our VMs. It dynamically produces:

- a cloud-init configuration,
- one libvirt domain XML description per VM,
- and two disk images for each VM; The first disk image contains the OS and the second one is used to mount the cloud-init configuration,
- additionally, it also produces a libvirt description of the testbed's network.

After this process has finished, the VMs are defined and can be booted. During startup, the VMs download and install all dependencies. Dependencies that cannot be installed from official repositories are installed by executing a locally hosted script.

The software also has a few functionalities that were implemented, but are not supported or part of the final result. An exploration into different territories is the reason for this code. It could be interesting for people wanting to extend the testbed or for future work. The functionality includes the generation of a more flexible routed network instead of a switched one. It was developed to allow MitM attacks, in case those are not possible otherwise in our virtualized switched network which turned out not to be the case as demonstrated by Chapter 6. This functionality also had a number of technical problems, that do not exist in the final and simpler solution. The downsides include: DNS issues, higher resource requirements and an overall more complex configuration.

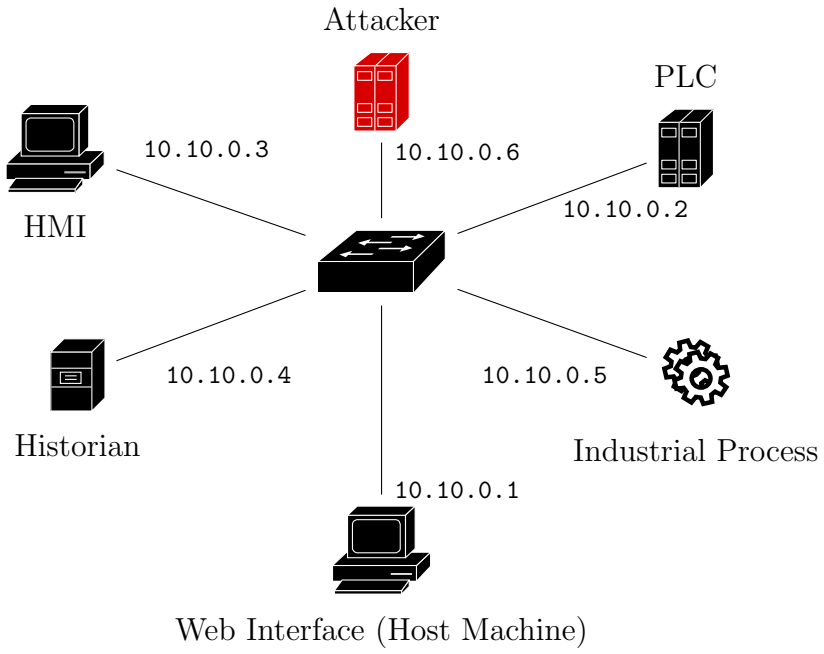


Figure 5.1: Network topology of our testbed, including IPv4 addresses.

5.1.4 Virtualized Network

Figure 5.1 illustrates the final topology, it includes all IP addresses. It additionally includes a host, serving as an access to web interfaces. For practical reasons, we use the host machine. This host is used to access the web interfaces of the PLC, HMI and data historian. Additionally, this network interface is used to communicate between the process simulator and the visualizer. Figure 5.2 shows the latency between the data historian and other hosts, measured with Telegraf's ping plugin and a sample size of 100. It shows the latencies in our virtualized network being very low. Assuring that real time requirements are not violated by our virtual networking solution.

5.2 Implementing the PLC

In Chapter 4 we set the goal to use an open-source software PLC. To the author's knowledge *OpenPLC*[5] currently is the only open-source PLC. This

[5]Project Website: `https://www.openplcproject.com/`

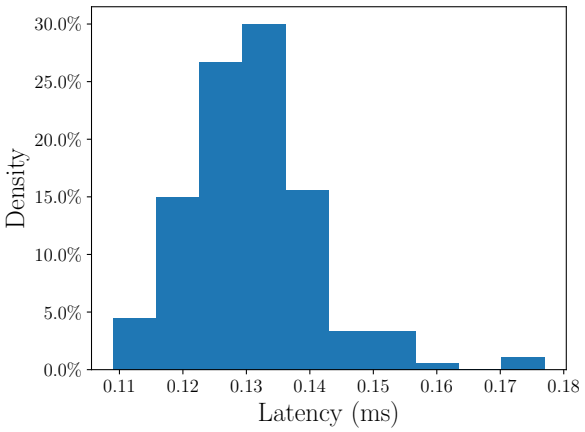


Figure 5.2: Distribution of latencies between testbed hosts.

makes it the only PLC that fits our previously defined criteria.

5.2.1 OpenPLC

OpenPLC supports all five IEC 61131-3 programming languages that were introduced in Section 2.1.3 [32]. It supports both Windows and Linux systems. OpenPLC can be used as a soft and hardware PLC [33]. It can be used on platforms with general purpose I/O pins, for instance a Raspberry Pi. Additionally, it can use Modbus slaves for further physical I/O.

5.2.2 Writing the PLC Program

OpenPLC provides an editor that can be used to write PLC programs.[6] We implement our PLC program as an LD program, which is an easy to understand visual programming language. Figure 5.3 illustrates the program that implements the behavior specified in Section 4.5.3. At first, we ignore PLC addressing and focus on the program's logic. The actuator `water_on` is affected by the upper water level sensor `level_sensor_high` and by the `on_button`. If the process is turned on and the upper limit has not been reached, it is set to true. The second actuator `drain_on` is determined very

[6]Editor: `https://www.openplcproject.com/plcopen-editor/`

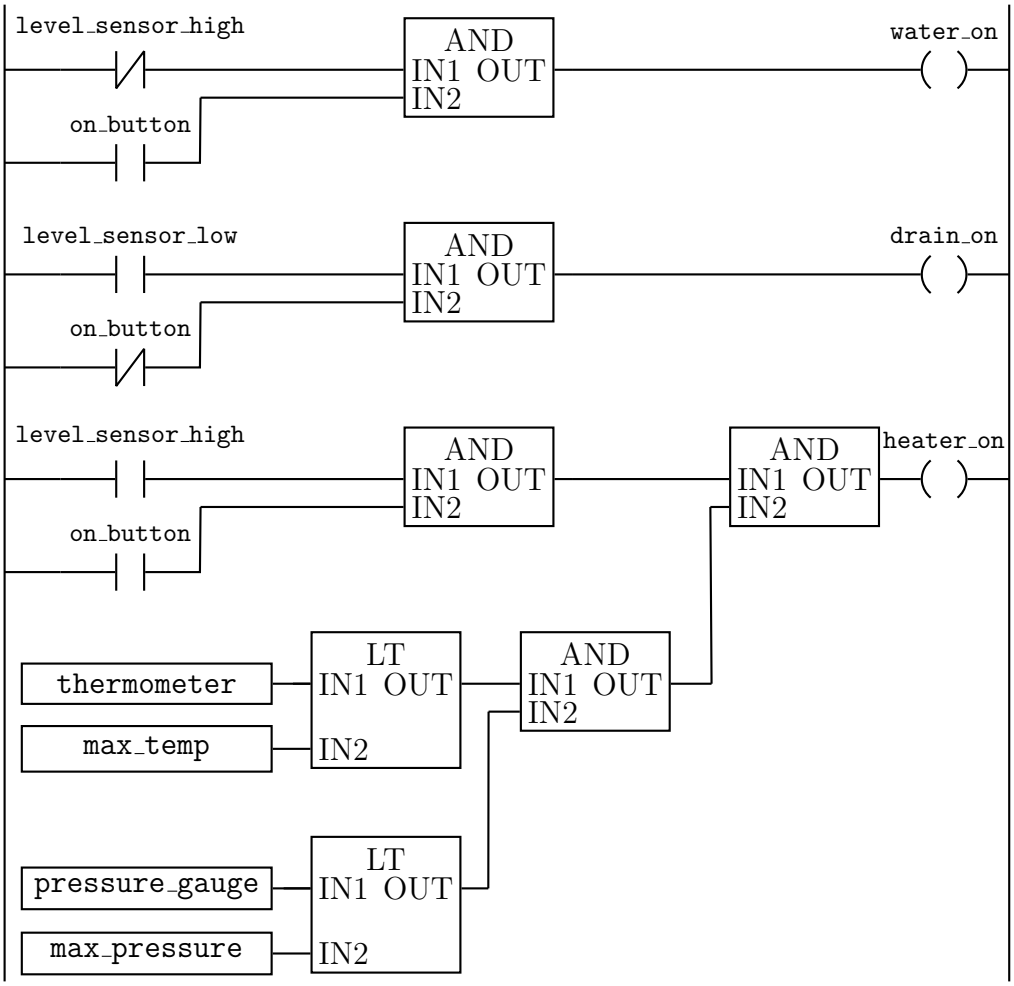


Figure 5.3: LD program controlling the I/O behavior.

similarly. It is set to true when the tank is not empty (`level_sensor_low` is true) and the process is turned off. The last actuator is calculated the following, more complicated way: The upper water limit is reached and process's heater is turned on. Additionally, both the `thermometer` and the `pressure_gauge` have to be below the limits `max_temp` and `max_pressure`.

In the next step we map the PLC program to Modbus addresses. PLC programs use different addresses internally and do not map them to physical Modbus addresses. The PLC runtime handles the translation. The scheme `%QW<Address>` is used for holding registers, `<address>` is an integer limited by the amount of available registers. `%QX<Byte>.<Bit>` is used for coils. The

addressing is influenced by the fact that coils are always sent as byte(s), rounded up to eight coils. For example, if we map the first coil of a server to the address `%QX0.0`, the ninth coil would be addressed with `%QX1.0`. In OpenPLC, addresses of the PLC's Modbus server begin with `0`, all external addresses conventionally start with `100`. In our case, the external addresses are located on the process's modbus server. With this knowledge we now can choose locations for all of the following data points: actuators, sensors and helper variables. We begin with our three actuators: `heater_on`, `water_on` and `drain_on`, all are stored externally and are bit sized. Therefore, we store them on coils beginning from the address `%QX100.0`. Furthermore, we need a place to store the sensor values. Additionally, we have to be able to externally communicate them from the process. As a result of this requirement, we are limited to holding registers or coils, both being located on the PLC. The process has two kind of sensors, boolean and integer ones. The boolean sensors `level_sensor_low` and `level_sensor_high` will be stored in the coils, as they are bit-sized. The two integer sensors `thermometer` and `pressure_gauge` are mapped to the holding registers. The last relevant PLC addresses are `max_temp` and `max_pressure`. They can be mapped to both input registers or holding registers, we choose the latter.

Table 5.1 contains a summary of the relationship of variables, PLC and Modbus addresses. It shows the three actuators having coil addresses starting from `%QX100.0`. Unlike the other addresses, they are located on industrial process's Modbus server, rather than on the PLC. Therefore, the first actuator coil is located at `%QX100.0`, which is where external coils conventionally start in OpenPLC.

5.2.3 OpenPLC configuration

The PLC is configured to have a single Modbus slave which represents the server of the process simulation. It is polled every 100 ms. The PLC itself also runs a Modbus server which is accessible to other Modbus clients. The runtime runs the program shown in Figure 5.3.

Name	Data Type	PLC Address	Modbus Address
max_temp	UINT	%QW0	PLC holding register 1
max_pressure	UINT	%QW1	PLC holding register 2
thermometer	UINT	%QW2	PLC holding register 3
pressure_gauge	UINT	%QW3	PLC holding register 4
on_button	BOOL	%QX0.0	PLC coil 1
level_sensor_low	BOOL	%QX0.1	PLC coil 2
level_sensor_high	BOOL	%QX0.2	PLC coil 3
heater_on	BOOL	%QX100.0	process coil 1
water_on	BOOL	%QX100.1	process coil 2
drain_on	BOOL	%QX100.2	process coil 3

Table 5.1: Mapping of actuators, sensors and helper variables.

5.3 Implementing the Process Simulation

The implementation of the simulator and its corresponding visualizer are described in this section. The visualizer is a component that can optionally be run, in order to get a live view of the true system state. Its view is unobstructed by any attack that currently might be happening.

5.3.1 The Simulator

This section describes the implementation of the simulator and its corresponding visualizer. For this, we use the *PyModbus*[7] library. The python simulation reads the actuator values that were written to the Modbus server by the PLC. Afterwards, it changes its internal state based on the input. As defined in Section 4.5.2, the internal state consists of three variables: the water level, the temperature and the system damage. The next step, is to derive the sensor values from the system state variables. For example, the level sensors are set to true if the water level has reached a certain minimum or maximum, respectively. The application uses the Modbus client to write

[7]PyModbus documentation:
https://pymodbus.readthedocs.io/en/latest/readme.html

the sensor values directly to the PLC's Modbus server. The changed sensor values then affect the actuator output, as calculated by the PLC program. Figure 5.4 displays how the systems interact and state is changed.

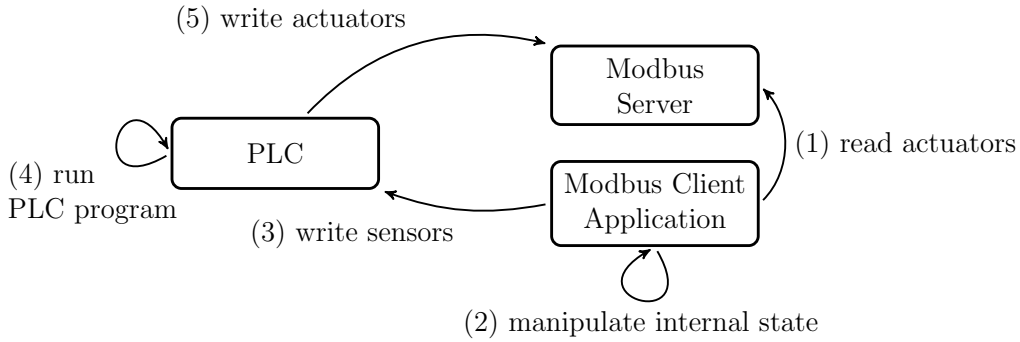


Figure 5.4: Interaction model of our simulator.

5.3.2 The Visualizer

The visualizer gives us a dynamic view of the simulation. It is accessed through the browser. JavaScript is used to fetch data from a backend server and to dynamically modify the visualizations. The backend is written in python using the FastAPI library[8] and a PyModbus client. The simulator additionally stores the sensor values and the system damage variable in its own Modbus server inaccessible to the PLC, HMI and data historian. With the visualizer we can access them and always see the true system state, regardless of any manipulation that is happening at the moment. Figure 5.5 displays a screenshot of the visualizer during regular operation.

5.4 Implementing the HMI

We considered two different open-source HMI systems: ScadaBR[9] and AdvancedHMI[10]. ScadaBR was chosen because AdvancedHMI does not provide

[8]FastAPI: `https://fastapi.tiangolo.com/`
[9]ScadaBR: `https://sourceforge.net/projects/scadabr/`
[10]AdvancedHMI: `https://sourceforge.net/projects/advancedhmi/`

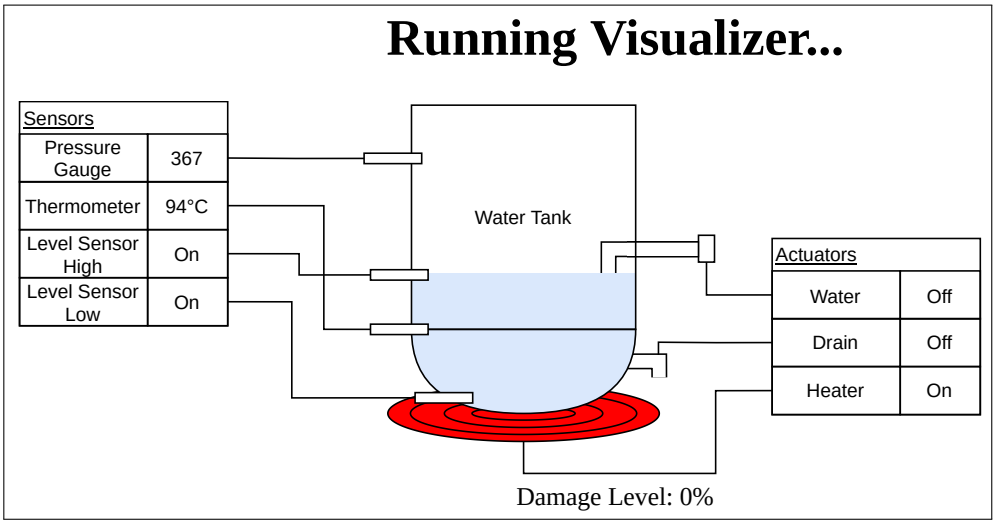


Figure 5.5: Screenshot of the visualizer during operation.

official Linux support. During a later stage of development, we became aware of another promising HMI software that could have been explored, FUXA[11]. This happened too late for it to be properly considered, but it should be examined in future work as it appears to have more active development than the other two candidates. Overall, we observed a lack of polished and commonly used open-source HMI frameworks.

The ScadaBR HMI is configured the following way: We add both Modbus servers to it as data sources and use them to build a view of the process. It displays sensor and actuator values. Additionally, it gives us the ability to turn the process on and off. Furthermore, an alarm is raised when the temperature passes a certain threshold. Figure 5.6 shows a screenshot of the HMI.

5.5 Implementing the Data Historian

The data historian collects values from both Modbus servers. It keeps a record of the history of all the variables shown in Table 5.1. At first, a self written data historian was implemented, using MySQL and a very simple

[11]FUXA: `https://github.com/frangoteam/FUXA`

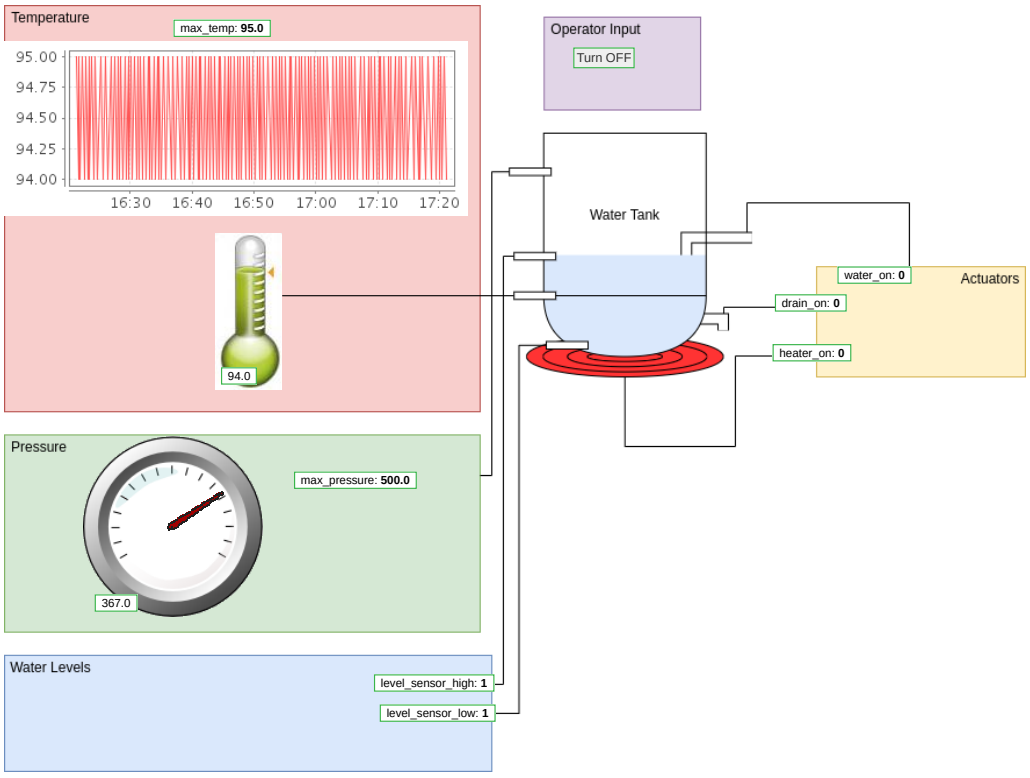


Figure 5.6: Screenshot of the HMI view during operation.

web interface. It was then decided, that it was insufficient and a proper time series database should be employed. This kind of database is developed to store the state of metrics over time [34]. The function of a data historian aligns with this. To realize the historian, we chose the TICK stack. We also investigated other systems such as TimescaleDB (time series database), Grafana (dashboard) and openHistorian. The latter being dismissed because of its lack of official Linux support. The TICK stack is chosen because it provides an all-in-one solution that has portable configuration files. Grafana or TimescaleDB would have required a more complicated setup without any additional benefits, therefore, we dismissed these options.

The TICK stack consist of four components. The first is *Telegraf*[12], it collects data from inputs (e.g. a Modbus server) and writes it to outputs (e.g. a database). The second component is the times series database, *InfluxDB*[13]. As of March 2021, InfluxDB is the most popular time series database according to DB-Engines [35]. *Chronograf*[14] is a dashboard solution. The last element is *Kapacitor*[15], a data processing framework. We are only using Kapacitor as a dependency of Chronograf at this moment.

Telegraf has many plugins that are distributed with it by default.[16] This includes a Modbus plugin, giving us first-class support for our use-case.[17] Telegraf is configured to collect all Modbus values from both the PLC and the process every ten seconds and write them to the database. This way, we can keep track of actuator and sensor values over time.

Chronograf allows us to create a dashboard and visualize the data the we collect. We can easily add graphs and other visualizations. Figure 5.7 displays a screenshot of our dashboard.

[12]Telegraf: `https://docs.influxdata.com/telegraf/v1.18/`
[13]InfluxDB: `https://docs.influxdata.com/influxdb/v1.8/`
[14]Chronograph: `https://docs.influxdata.com/chronograf/v1.8/`
[15]Kapacitor: `https://docs.influxdata.com/kapacitor/v1.5/`
[16]Telegraf plug-ins: `https://docs.influxdata.com/telegraf/v1.18/plugins/`
[17]Modbus plugin:
`https://github.com/influxdata/telegraf/tree/master/plugins/inputs/modbus`

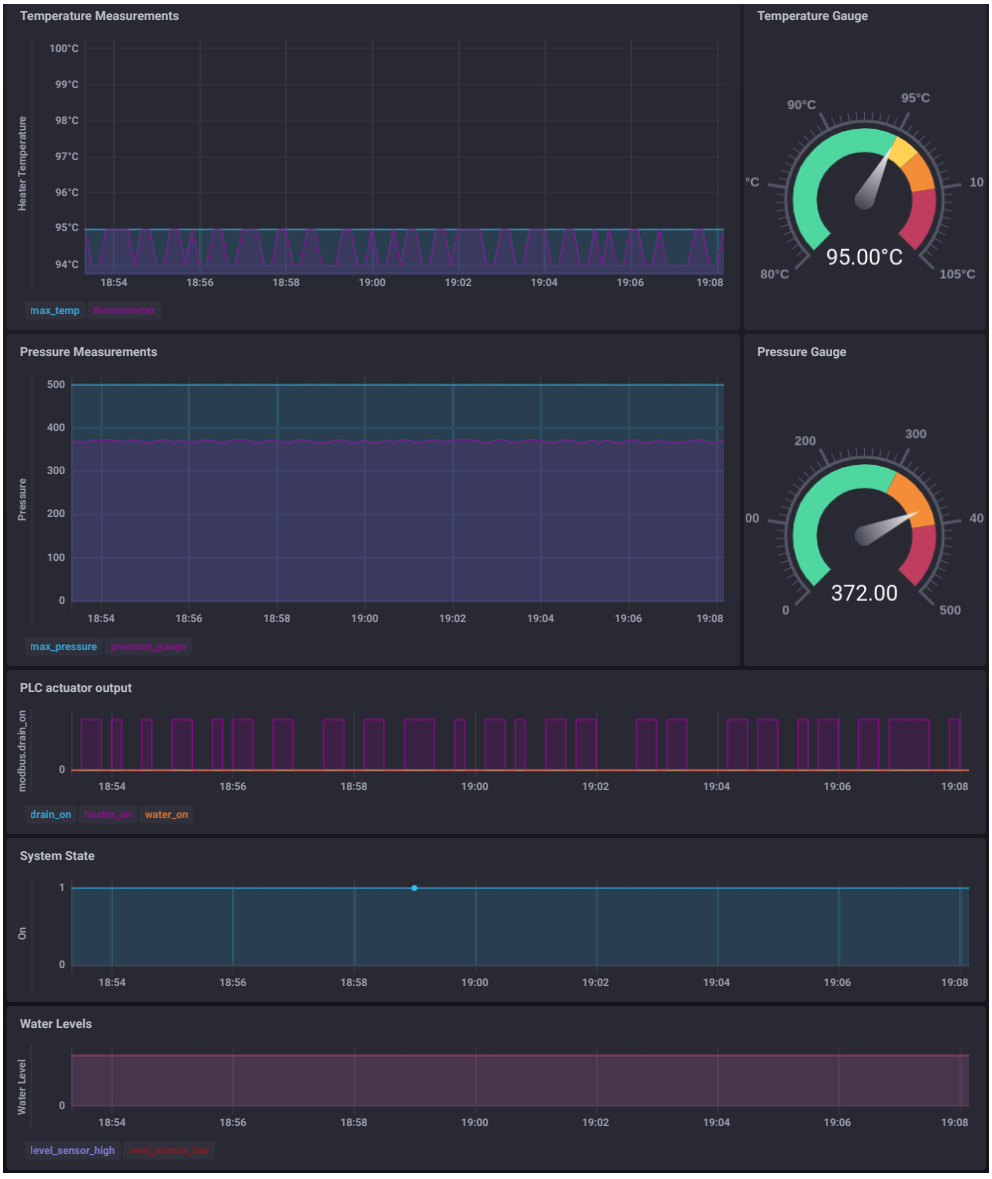


Figure 5.7: Screenshot of the Chronograf dashboard during operation.

Chapter 6

Using the Testbed for Attack Demonstration

This chapter discusses a selection of attacks that can be demonstrated with the testbed. We evaluate the testbed's ability to demonstrate attacks in order to raise awareness for industrial security.

Section 6.1 provides information about potential adversaries. Section 6.2 covers two very similar attacks. The following section describes an attack type that is very PLC-specific. Section 6.4 gives further direction about other potential attacks. The content of the previous sections are summarized and discussed in Section 6.5. The final section examines potential limitations of our observations and transferability of the results.

Table 6.1 provides information about the testing environment.

Name	Used
Host OS	Manjaro (Kernel Version: 5.4)
CPU	AMD Ryzen 5 3600 @ 3.6 GHz
Memory	32 GB
Hypervisor	KVM
libvirt	v6.5.0
vCPUs	1 per VM (2 for the PLC)

Table 6.1: Environment that was used to conduct tests.

6.1 The Adversary

In our scenario, the attacker has previously gotten access to the network. There are multiple ways they could have obtained this position. One possible initial entry point is a corporate network from which the field network was not properly shielded off with employing firewalls/demilitarized zones (DMZs) [4]. Another way is the exploitation of an insecure remote point, for instance, an unencrypted telnet connection [8]. The necessary capabilities of an attacker to successfully implement the following attacks are limited to basic knowledge about network security concerns, such as Address Resolution Protocol (ARP) cache poisoning. ARP is used to map IP addresses to MAC address, this is necessary to communicate in local networks. An attacker can abuse this protocol to create false entries in the ARP storage tables of other hosts. As a consequence of the attack, the hosts associate IP addresses with the wrong MAC address (e.g. the one of the attacker) and a MitM position can be obtained. Our ARP cache poisoning implementation is based on an example from the book *Black Hat Python* by Seitz [36]. Other required information to successfully launch these exploits is publicly available, such as the Modbus specification [14].

6.2 Manipulating the PLC's View

This section covers two attacks that affect the system in a very similar way. One redirects the Modbus traffic to two servers running on the attacker host. The other attack directly modifies the requests and responses. Both attacks use the same python lambda expression to modify the values, it is equivalent to: `lambda x: 94 if 94 < x < 98 else min(x, 95)`. Sensor values $x \in \{95, 96, 97\}$ are underreported as 94°C, anything above is reported as 95°C. This effectively makes 98°C the new upper boundary which previously had been 95°C. The raised operating temperature causes damage to our simulated system.

6.2.1 Variant: Redirecting Traffic

In this attack, the malicious host is running two Modbus servers and four clients. One server for the PLC's server and the other one for the industrial process. We redirect requests away from the original servers, to the ones of the attacker. The four clients are necessary to connect to each of the running Modbus servers. The clients read the values written to the attacker's servers and are used to either pass them along or alter them. IP addresses are modified, making both the PLC and industrial process oblivious to MitM attack. The attack is possible due to a lack of authentication in the Modbus protocol. We have the following attack vector.

1. Use ARP cache poisoning to establish MitM position between the PLC and industrial process.

2. Disrupt established TCP connections and redirect new ones to the attacker's server.

3. Manipulate the view by underreporting thermometer measurements.

4. Industrial process enters state that leads to system damage.

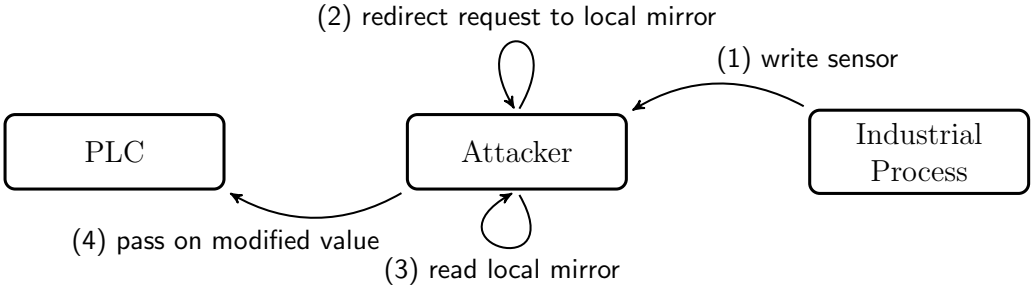


Figure 6.1: Redirection and modification of write requests containing sensor information.

6.2.2 Variant: Packet Modification

This variant directly identifies Modbus packets that need to be modified. It uses *netfilter-queue*[1] and the corresponding python bindings[2] to modify the packets with a python application. The packet modification is done with *ScaPy*[3]. The attack is possible because Modbus TCP/IP has no integrity protections apart from TCP's cyclic redundancy check (CRC). CRC cannot protect against modifications made by malicious actors since it is not a cryptographically secure hash function. Therefore, we do not have proper message authentication. The attack vector has the following steps:

1. Use ARP cache poisoning to establish MitM position between the PLC and industrial process.
2. Look for requests from the process to the PLC with function code 0x06 and address 0x0002.
3. Modify the requested value and forward modified packet.
4. Look for corresponding response to the previous request.
5. Modify the confirmation back to the originally requested value.
6. The industrial process enters a state that leads to system damage.

Figure 6.2 shows what information is sent and what is eventually received.

6.2.3 Implications

Both of these attack variants accomplish the same goal: damaging the physical system by manipulating the PLC's view. The first variant is more intrusive, because it possibly has to disrupt established TCP connections. Additionally, it does not scale well, as we need n Modbus servers and $2n$ clients. The second variant can accomplish the same thing, while being more stealthy and using less resources. If used correctly, neither of the attacks is recognizable by observing the PLC, the HMI or the data historian. Figure 6.4 shows

[1] netfilter-queue : `https://www.netfilter.org/projects/libnetfilter_queue/`
[2] netfilter-queue python: `https://github.com/kti/python-netfilterqueue`
[3] ScaPy: `https://scapy.readthedocs.io/en/latest/`

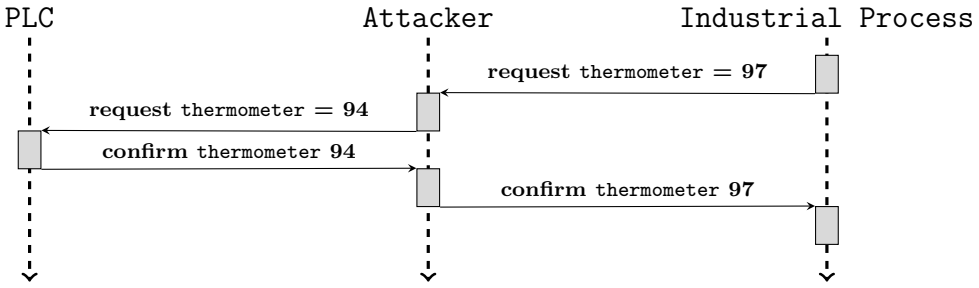


Figure 6.2: Sequence diagram: packet modification attack.

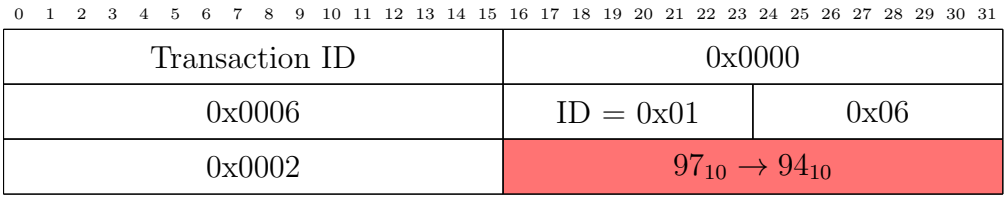


Figure 6.3: Modification of a write-single-register request.

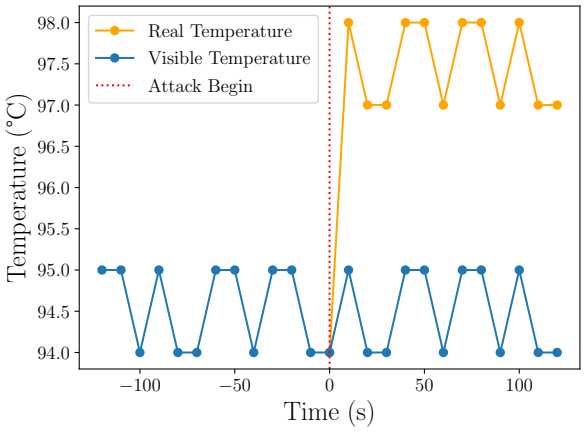


Figure 6.4: Real and visible temperature, applies to both attacks from Section 6.2.

the difference between the actual temperatures and the one that is reported to the PLC, HMI and historian. The data was collected with the second variant, the first variant has an identical effect. This kind of attack can also be applied to all other Modbus connections in the testbed, for instance, between the HMI and PLC.

6.3 Manipulating the PLC behavior

This attack exploits the fact that the PLC's web interface does not use HTTPS to protect traffic. If an attacker obtains a MitM position, they can sniff the password and alter the PLC's configuration. Another way an adversary could gain access to the system is a weak password, that can be brute-forced or is still set to the default. We have the following attack vector:

1. Use ARP cache poisoning to establish MitM position between the PLC and a host used to access its web interface.
2. Wait until an operator logs in and sniff the password.
3. Log into the web interface and upload a malicious PLC program.
4. Industrial process enters state that leads to system damage.

To construct a malicious program, the attacker has either to obtain the original program or to reverse engineer it. Both are possible options. The program could be sniffed the same way because the password and the I/O behavior of the PLC is simple enough to allow a reconstruction of the program. In our case, we edited the original program to have a higher maximum temperature.

6.3.1 Implications

The attack can lead to system damage if it is not detected in time. It is less stealthy than the previously discussed attacks. The true sensor values are available to the PLC, the HMI and the data historian. Altering the actuator behavior without it being reflected in sensor values is hard to do in

our testbed scenario. An example where a similar kind of attack was done in a more subtle way is Stuxnet [7]. Additionally, one can combine the attacks in order to construct an attack vector that is less visible. We could apply the techniques from Section 6.2.2 to alter the values that are reported by the PLC to components such as the HMI. Another option is to directly overwrite the values before the PLC is polled by the historian or the HMI.

6.4 Other Possible Attacks

This chapter only covers a selection of attacks in detail. There are others, for example the most trivial attack would be using a Modbus client to write to any of the two other Modbus servers. This is possible due to a lack of authentication. Another potential attack is flooding the PLC with Modbus requests.

While using the testbed, we also found a code injection vulnerability in the OpenPLC web interface. To the best of the author's knowledge it has either not been previously discovered or at least not publicly disclosed. The vulnerability requires an adversary to have access to account credentials. If they do, they can use a lack of input sanitation for persistent XSS attacks. This vulnerability still needs to be investigated further.

6.5 Discussion

In the previous sections, we showed that there are a number of attacks that can be launched from the adversary's initial position. All of the attacks we covered in detail abuse the lack of secure connection. The final goal of all of them is to damage a physical system. Additionally, they could seriously endanger human lives, the environment or cause substantial monetary damage. These kind of attacks are unique to ICS's or other cyber-physical systems. The first two of our launched attacks are very difficult for operators to detect. No reliable information reaches components they interact with, and they might only become aware when a failsafe is triggered or the system is

critically damaged. Our covered attacks confirm ARP cache poisoning is an important potential entry point for many attacks, especially those abusing a lack of encryption or message authentication.

Furthermore, we notice that the testbed can be utilized to implement high fidelity network-based attacks. The initial assumptions to the adversary's position are limited to them having access to a network. A MitM position is not assumed and first has to be obtained. We can also study types of attacks that are unique to PLCs, such as our third one. Sniffing would not be the only initial access point to such a position, for example dictionary or brute-force attacks against weak passwords are another possibility [8]. Nonetheless, it should be noted that this kind of attack is very codebase-specific and attack vectors of such an exploit can vary widely between implementations. Another type of attack that directly relates to the PLCs is the manipulation of view. Our first two attacks used this to damage the physical system. This kind of attack is less codebase-specific, rather it is highly influenced by the used protocol or the way systems are connected. Altering the sensor values could be very different if we were using a pin-based connection instead. By finding a potential XSS vulnerability in OpenPLC, we also show that the testbed can be leveraged to discover new exploits.

All of the attacks we cover in this chapter should be familiar to a person familiar with basic IT security proficiency. For instance, abusing unprotected HTTP traffic to sniff a password could be considered a solved problem, SSL/TLS have existed and been used with HTTPS for many years. ICS's certainly have their own unique challenges, utilizing cryptography can be hard because legacy hard and software might not support it. This does not apply to HMIs and data historians, which usually run on commodity hardware and operating systems. However, their communication partner might not support cryptography. Nonetheless, this should not be an issue with web traffic. In addition, many ICS protocols have a variant that uses TLS, for instance Modbus TCP/IP [37]. Despite its existence, none of our components except PyModbus support it.

At the same time, not everything is solved by adding encryption. There are attacks that still would be very much viable. Modbus clients are not

authenticated, therefore we can write falsified values to registers without requiring a MitM position. We observe that commonly listed ICS and PLC attacks can be implemented in our testbed setup, for instance, ARP cache poisoning, packet modification and a modification of the PLC program.

6.6 Limitations

In our previous analysis, we were not able to cover every topic that possibly could be addressed. We did not focus on real-time requirements and how they are affected by attacks. Likewise, transferring some of our attacks to other and more commonly used ICS soft and hardware might be limited and was not investigated in depth. ScadaBR and OpenPLC are niche software. However, the first two attacks are not codebase-specific and mirror common communication setups. The third attack is very implementation-specific, but the attack type itself does apply to other PLCs as well.

Additionally, we might be affected by researcher bias. For instance, the feasibility of reverse engineering the PLC program might be overestimated. This stems from the fact that the program was implemented by the author and familiar to them.

Chapter 7

Conclusion

In this thesis we developed a testbed, which replicates part of an ICS network. At first we introduced import background knowledge, including general ICS terminology, the Modbus protocol, common ICS attacks and established a clear definition for the terms simulation, emulation and virtualization. Afterwards, we specified the requirements we demand from an ICS testbed and decided on what general approach we take to realize included components. Then we implemented a testbed that attempts to address all of our previous demands. This section first reflects on the final status of our testbed in Section 7.1 and then concludes by covering possible future work and research directions.

7.1 Status

In this part we cover the final status of the testbed by evaluation the strengths and weaknesses of the testbed. Furthermore, we reflect on realized goals and address potentially open goals.

7.1.1 Strengths and Weaknesses

In previous sections we analyzed the strengths and weaknesses of our testbed. In this section, we give an comprehensive overview on them. We showed that our testbed is suited towards the demonstration of attacks. Additionally, we

can also directly implement attacks at a high fidelity, and observe vulnerabilities and results that are consistent with literature such as NIST SP 800-82 [4] or the ICS security compendium by the BSI [8]. Our testbed is particularly suited for researching attacks that target communication and directly exploit MitM vulnerabilities such as sniffing or packet modification. We can launch attacks such as ARP cache poisoning, which targets very lower networking layers. This is attributed to the decision to use virtualized networking, which provides the highest fidelity for this use case. Additionally, we also can showcase an attack type that is very specific to PLCs. We demonstrated this kind of attack when we uploaded a malicious PLC program.

However, our design decisions also introduce limitations to our approach. For instance, strictly choosing open-source components reduces the fidelity of our testbed in some regards. Furthermore, we cannot infer too much about codebase-specific attacks. We also cannot examine pin-based attacks, PLCs often use but are not limited to a pin-based I/O. In addition, the testbed is currently limited to Linux, because of libvirt.

7.1.2 Realized Goals

During the development we successfully realized all of our goals. We developed a fully virtual testbed, that includes all components demanded from an ICS related testbed in Chapter 4. These include control centers (HMI and data historian), a field device (PLC), an industrial process, the communication architecture (virtual networking) and the utilized protocols (Modbus). Additionally, we also accomplished to build a testbed that directly addresses desirable properties, these include fidelity, repeatability, safety, measurement accuracy, affordability and others. Furthermore, we also analyzed the suitability of testbed for its intended purpose: attack demonstration. We concluded that we can successfully implement and demonstrate realistic attacks, especially those related to PLC networks.

7.1.3 Open Goals

As stated by the previous section we constructed a useful testbed. However, we did not manage to address all requirements as thoroughly as possible. For instance, we only briefly covered real-time requirements in Chapter 5 and did not investigate the effects of our implemented attacks on the system's real-time behavior. Additionally, the simulation of our industrial process is not physically accurate, which reduces its fidelity.

7.2 Future Work

There are many different research directions future work can head towards. An appropriate course is to address the open goals from the previous section. For instance, one could research the real-time behavior of the system and examine differences when using different realization techniques. It would be interesting to test real-time operating systems combined with bare-metal hypervisors. This could be combined with the inclusion of deadlines in our simulation, which could demand timely responses from the PLC. If one wants to take such an approach it might be advisable to utilize a more deterministic and performant programming language than Python. Furthermore, the simulation model could be improved, to provide a more immersive experience. The author of this thesis has a strictly informatics background. Therefore, this might require expertise from other domains, such as control engineering. Additionally, some of the testbed components offer more functionality that was not utilized to the fullest extent. For instance, the historian is left mostly to the default configurations which is not secure by default. This includes the possibility to employ TLS. Moreover, the testbed could be developed further into one that has distinct roles which can be taken up by users. For example, it could distinguish between an operator and an attacker, both working against each other. This could be very beneficial for training purposes. Kapacitor from the TICK stack can also be used for anomaly detection [38], it might provide an possibility to realize an intrusion detection system for our type of network. Finally, another research direction would be to explore more

lightweight virtualization techniques such as containerization. However, this could potentially decrease fidelity in some regards.

Bibliography

[1] S. Mansfield-Devine, "The state of operational technology security", *Network Security*, vol. 2019, no. 10, pp. 9–13, 2019, ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(19)30121-7.

[2] G. Glossary, *Information Technology (IT)*. [Online]. Available: https://www.gartner.com/en/information-technology/glossary/it-information-technology (visited on Apr. 8, 2021).

[3] G. Glossary, *Operational Technology (OT)*. [Online]. Available: https://www.gartner.com/en/information-technology/glossary/operational-technology-ot (visited on Apr. 8, 2021).

[4] K. A. Stouffer, J. A. Falco, and K. A. Scarfone, *SP 800-82 Rev. 2 - Guide to Industrial Control Systems (ICS) Security*, Gaithersburg, MD, USA: National Institute of Standards & Technology, 2011. DOI: 10.6028/NIST.SP.800-82r2.

[5] S. D. Antón, D. Fraunholz, C. Lipps, F. Pohl, M. Zimmermann, and H. D. Schotten, "Two decades of SCADA exploitation: A brief history", in *2017 IEEE Conference on Application, Information and Network Security (AINS)*, 2017, pp. 98–104. DOI: 10.1109/AINS.2017.8270432.

[6] H. Wardak, S. Zhioua, and A. Almulhem, "PLC access control: a security analysis", in *2016 World Congress on Industrial Control Systems Security (WCICSS)*, 2016, pp. 1–6. DOI: 10.1109/WCICSS.2016.7882935.

[7] T. M. Chen and S. Abu-Nimeh, "Lessons from Stuxnet", *Computer*, vol. 44, no. 4, pp. 91–93, 2011. DOI: 10.1109/MC.2011.115.

[8] Federal Office for Information Security (BSI), *ICS Security Compendium Version 1.23*, 2013. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/ICS/ICS-Security_compendium.html.

[9] H. Holm, M. Karresand, A. Vidström, and E. Westring, "A survey of industrial control system testbeds", in *Nordic Conference on Secure IT Systems*, Springer, 2015, pp. 11–26.

[10] H. Kopetz, *Real-time Systems: Design Principles for Distributed Embedded Applications*. Springer Science & Business Media, 1997, ISBN: 0-306-47055-1.

[11] R. Ramanathan, in *2inproceedings98-603*. DOI: 10.1109/WAC.2014.6936062.

[12] O. Project, *Introduction to Ladder Logic*. [Online]. Available: https://www.openplcproject.com/reference/basics/introduction-to-ladder-logic (visited on Apr. 5, 2021).

[13] G. de-Bray, J. Hoersch, J. Hurn, A. Koski, A. Tout, and J. G. Inch, "Winners and Losers of the Industrial Internet", *Deutsche Bank Markets Research - Global Capital Goods*, 4 April 2017. [Online]. Available: http://www.fullertreacymoney.com/system/data/files/PDFs/2017/April/7th/0900b8c08c914e35%20(1).pdf.

[14] Modbus Organization, *Modbus Application Protocol Specification V1.1b3*, 2012. [Online]. Available: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (visited on Apr. 5, 2021).

[15] Modbus Organization, *Modbus Messaging on TCP/IP Implementaion Guide V1.0b*, 2006. [Online]. Available: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf (visited on Apr. 5, 2021).

[16] M. Yampolskiy, P. Horvath, X. D. Koutsoukos, Y. Xue, and J. Sztipanovits, "Taxonomy for Description of Cross-Domain Attacks on CPS", in *Proceedings of the 2nd ACM International Conference on High Confidence Networked Systems*, ser. HiCoNS '13, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 2013, pp. 135–142, ISBN: 9781450319614. DOI: 10.1145/2461446.2461465.

[17] B. Zhu, A. Joseph, and S. Sastry, "A Taxonomy of Cyber Attacks on SCADA Systems", in *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, 2011, pp. 380–388. DOI: 10.1109/iThings/CPSCom.2011.34.

[18] P. Huitsing, R. Chandia, M. Papa, and S. Shenoi, "Attack taxonomies for the Modbus protocols", *International Journal of Critical Infrastructure Protection*, vol. 1, pp. 37–44, 2008, ISSN: 1874-5482. DOI: 10.1016/j.ijcip.2008.08.003.

[19] B. Miller and D. Rowe, "A Survey SCADA of and Critical Infrastructure Incidents", in *Proceedings of the 1st Annual Conference on Research in Information Technology*, ser. RIIT '12, Calgary, Alberta, Canada: Association for Computing Machinery, 2012, pp. 51–56, ISBN: 9781450316439. DOI: 10.1145/2380790.2380805.

[20] A. Abbasi and M. Hashemi, "Ghost in the PLC Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack", *Black Hat Europe*, vol. 2016, pp. 1–35, 2016.

[21] M. Niedermaier, J.-O. Malchow, F. Fischer, D. Marzin, D. Merli, V. Roth, and A. Von Bodisco, "You snooze, you lose: measuring {PLC} cycle times under attacks", in *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[22] J. Banks, "Introduction to Simulation", in *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*, vol. 1, 2000, 9–16 vol.1. DOI: 10.1109/WSC.2000.899690.

[23] R. Beuran, *Introduction to Network Emulation*. Pan Stanford, 2012, ISBN: 978-981-4364-09-6.

[24] M. Rosenblum and C. Waldspurger, "I/O Virtualization: Decoupling a Logical Device from Its Physical Implementation Offers Many Compelling Advantages.", *Queue*, vol. 9, no. 11, pp. 30–39, Nov. 2011, ISSN: 1542-7730. DOI: 10.1145/2063166.2071256.

[25] S. Campbell and M. Jeronimo, "An Introduction to Virtualization", *Published in "Applied Virtualization", Intel*, pp. 1–15, 2006. [Online]. Available: https://software.intel.com/sites/default/files/m/d/4/1/d/8/An_Introduction_to_Virtualization.pdf (visited on Mar. 15, 2021).

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

[27] Y. Geng, Y. Wang, W. Liu, Q. Wei, K. Liu, and H. Wu, "A Survey of Industrial Control System Testbeds", *IOP Conference Series: Materials Science and Engineering*, vol. 569, p. 042 030, Aug. 2019. DOI: 10.1088/1757-899x/569/4/042030.

[28] D. Formby, M. Rad, and R. Beyah, "Lowering the Barriers to Industrial Control System Security with GRFICS", in *2018 USENIX Workshop on Advances in Security Education (ASE 18)*, Baltimore, MD: USENIX Association, Aug. 2018. [Online]. Available: `https://www.usenix.org/conference/ase18/presentation/formby`.

[29] T. Alves, R. Das, A. Werth, and T. Morris, "Virtualization of SCADA testbeds for cybersecurity research: A modular approach", *Computers & Security*, vol. 77, pp. 531–546, 2018, ISSN: 0167-4048. DOI: `https://doi.org/10.1016/j.cose.2018.05.002`.

[30] S. Lüders, "CERN tests reveal security flaws with industrial network devices", *The Industrial Ethernet Book*, vol. 35, no. CERN-OPEN-2006-074, 12–23. 7 p, Oct. 2006. [Online]. Available: `https://cds.cern.ch/record/1000756`.

[31] S. Guckenheimer, *What is Infrastructure as Code?*, Microsoft, 2017. [Online]. Available: `https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code` (visited on Mar. 26, 2021).

[32] OpenPLC, *What is a PLC?* [Online]. Available: `https://www.openplcproject.com/reference/basics/what-is-a-plc` (visited on Mar. 27, 2021).

[33] OpenPLC, *OpenPLC Runtime*. [Online]. Available: `https://www.openplcproject.com/reference/basics/what-is-a-plc` (visited on Mar. 27, 2021).

[34] InfluxData, *What is a time series database?* [Online]. Available: `https://www.influxdata.com/time-series-database/` (visited on Mar. 28, 2021).

[35] DB-Engines, March 2021. [Online]. Available: `https://db-engines.com/en/ranking/time+series+dbms` (visited on Mar. 28, 2021).

[36] J. Seitz, *Black Hat Python*. No Starch Press, 2014, ISBN: 9781593275907.

[37] M. Organization, *MODBUS/TCP Security*, 2018. [Online]. Available: `https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf` (visited on Apr. 5, 2021).

[38] InfluxData, *Custom anomaly detection using Kapacitor*. [Online]. Available: `https://docs.influxdata.com/kapacitor/v1.5/guides/anomaly_detection/` (visited on Apr. 8, 2021).

List of Figures

2.1 Example IEC 61131-3 LD program, adapted from [12]. 8
2.2 Modbus Request and Response, adapted from [15]. 9
2.3 Modbus TCP/IP in the five-layer Internet model. 9
2.4 Modbus TCP/IP frame showing MBAP and PDU. 10
2.5 The four primary Modbus tables. 10
2.6 Modbus TCP/IP ADU of a write holding register request. . . 11
2.7 Relationship of simulation, emulation and virtualization 16

4.1 Conceptualization steps. 21
4.2 Network Topology of the testbed. 25

5.1 Network topology of our testbed, including IPv4 addresses. . . 37
5.2 Distribution of latencies between testbed hosts. 38
5.3 LD program controlling the I/O behavior. 39
5.4 Interaction model of our simulator. 42
5.5 Screenshot of the visualizer during operation. 43
5.6 Screenshot of the HMI view during operation. 44
5.7 Screenshot of the Chronograf dashboard during operation. . . 46

6.1 Redirection and modification of write requests containing sensor information. 49
6.2 Sequence diagram: packet modification attack. 51
6.3 Modification of a write-single-register request. 51
6.4 Real and visible temperature, applies to both attacks from Section 6.2. 51

List of Tables

1.1 Differences between IT and OT/ICS, adapted from [4]. 3

4.1 Requirements and corresponding goals. 21
4.2 Advantages and Disadvantages 26
4.3 Method of realizing important components. 31
4.4 The sensors and actuators of our simulation. 32

5.1 Mapping of actuators, sensors and helper variables. 41

6.1 Environment that was used to conduct tests. 47