University of Bologna | Faculty Engineering and Architecture | Cybersecurity Module

Report for the Industrial Simulation Environment Project

Charlotte Glutting, M.Sc. Information Systems Management, 1900124183
Damian Marvin Atlaß, M.Sc. Digital Media and Technology, 1900125838

Bologna, December 13, 2024
Supervised by Silvio Rosso

Abstract

This project explores the integration and security evaluation of an industrial automation system, utilizing OpenPLC, Factory I/O, ScadaBR, and Docker containers. The goal is to simulate and control an industrial assembly station. In addition, the system undergoes a series of cyberattack simulations to evaluate the stability of the entire setup. This project demonstrates the potential of open-source tools in industrial automation and cybersecurity research and offers a portable and secure framework for testing and development.

Contents

1 Introduction

The increasing integration of industrial control systems with digital networks has significantly raised the importance of securing these infrastructures[1]. These systems require secure and flexible environments throughout development, testing, and deployment to ensure their reliability and functionality[2]. As cyber threats continue to evolve, protecting industrial environments against potential vulnerabilities is crucial[3].

This project focuses on the creation of a mini-industrial infrastructure designed for security testing. The central component is the OpenPLC Project, an open-source tool for simulating industrial components. OpenPLC enables the creation of an abstract yet realistic environment and simulate the behavior of real-world industrial systems. By providing a controlled platform for testing and evaluation, this project highlights the importance of simulation in addressing the challenges modern threats cause.

2 Project's Organisation and Goals

This section outlines the organizational structure and objectives of the project.

2.1 Organisation

All team members pushed their developed code to a shared GitHub repository. Our workflow involved creating new features on separate feature branches. Once a feature was ready, it was submitted as a pull request, underwent peer review, and was then merged into the main branch. This process ensured high code quality and facilitated collaborative development. Regular meetings were held in order to coordinate, plan future developments and give room for discussions and suggestions.

2.2 Goals

To guide our development efforts and ensure that our system met the defined requirements, we established four goals:

- **Simulation of Industrial Environment and Components:** Design and implement a mini-industrial infrastructure to simulate an assembly station and integrate components like OpenPLC, Factory I/O, and ScadaBR.
- **Containerization with Docker:** Use Docker to containerize the simulation to ensure isolation and portability.
- **Automation with Terraform:** Implement Terraform to automate the provisioning of the infrastructure and enable reproducibility across different deployments.
- **Security Evaluation:** Evaluate the security of the simulated industrial environment by performing selected cyberattacks.

1 W. Alsabbagh, C. Kim, P. Langendoerfer, Investigating the Security of OpenPLC: Vulnerabilities, Attacks, and Mitigation Solutions.

2 W. Alsabbagh, C. Kim, P. Langendoerfer, Investigating the Security of OpenPLC: Vulnerabilities, Attacks, and Mitigation Solutions.

3 W. Alsabbagh, C. Kim, P. Langendoerfer, Investigating the Security of OpenPLC: Vulnerabilities, Attacks, and Mitigation Solutions.

3 Concept and Design

3.1 Assembly Station

The concept of the assembly station is of a high level of abstraction compared to real-life assembly stations. As this project's focus is not the detailed reproduction of actual industrial processes, this simple version is completely sufficiant for our purposes. It consists of a mechanical arm that can lift and move pieces of a product from one conveyor to another. The two adjacent conveyors, one that always transports the top piece of a product and the other carrying always the bottom piece, are each powered by a motor. As soon as a product piece is detected to be beneath the mechanical arm, the conveyor stops and waits for the other. When both pieces are in place, the mechanical arm puts the top piece on the bottom piece, releases and retracts. After that, the cycle begins again.

In the following images, the simulation of our assembly station is shown in ladder logic in the OpenPLC Editor. Later, this program will be uploaded as a structured file to the OpenPLC Runtime. This logic simulates the operation of an industrial system and ensures the precise coordination of each component.

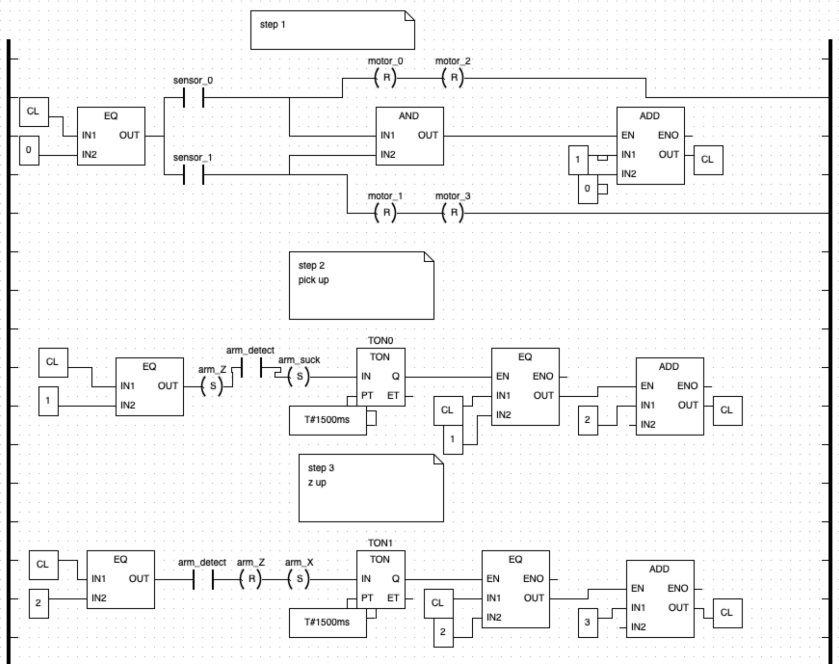

Figure 1

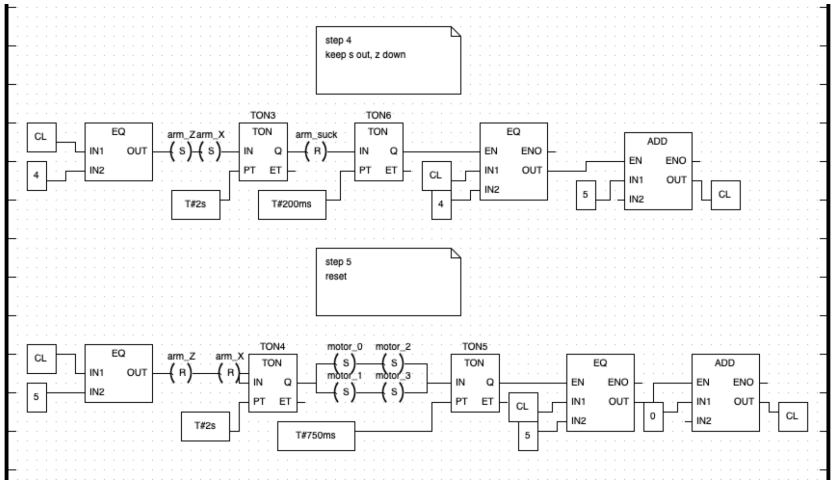

Figure 2

4 Implementation

This section provides an overview of the project implementation process. Details technologies used to bring the project from concept to working solution.

4.1 OpenPLC

OpenPLC is an open-source Programmable Logic Controller. OpenPLC highlights it in terms of compatibility and interoperability across industrial systems. Due to this fact, it mirrors actual industry practices well and has a high value in security testing[4]. OpenPLC has emerged as an affordable and practical alternative to traditional high-cost PLC hardware[5]. It runs on a wide range of platforms, including personal computers and low-cost embedded devices such as Raspberry Pi, Arduino, and other controllers[6]. This flexibility makes OpenPLC an appealing solution for both educational and industrial applications, particularly in scenarios where cost-efficiency and scalability are important considerations[7].

In our project, we utilize two versions of OpenPLC to address different integration needs. The first version runs locally on the host machine, enabling direct integration with Factory I/O, because it requires OpenPLC to be executed locally rather than in a container. This setup ensures communication between OpenPLC and the 3D simulation environment. The second version of OpenPLC runs in a containerized environment within our Docker network. This configuration is used in combination with ScadaBR and ensures efficient communication between OpenPLC and ScadaBR. By running OpenPLC in a container, we benefit from isolation and portability. Another advantage is that our Open PLC project can be easily redeployed with the same configurations in new projects.

4 W. Alsabbagh, C. Kim, P. Langendoerfer, Investigating the Security of OpenPLC: Vulnerabilities, Attacks, and Mitigation Solutions.
5 W. Alsabbagh, C. Kim, P. Langendoerfer, Investigating the Security of OpenPLC: Vulnerabilities, Attacks, and Mitigation Solutions.
6 W. Alsabbagh, C. Kim, P. Langendoerfer, Investigating the Security of OpenPLC: Vulnerabilities, Attacks, and Mitigation Solutions.
7 W. Alsabbagh, C. Kim, P. Langendoerfer, Investigating the Security of OpenPLC: Vulnerabilities, Attacks, and Mitigation Solutions.

OpenPLC provides an cost-efficient solution for controlling and simulating industrial processes in our project, with both local and containerized deployments. The OpenPLC system consists of two primary components: the OpenPLC Editor and the OpenPLC Runtime.

OpenPLC Editor The Editor is used to design control logic for industrial processes using standard programming languages like Ladder Logic and Structured Text [8]. In our project, we use the OpenPLC Editor to create the control logic for our assembly station.

OpenPLC Runtime The designed control logic is executed in the OpenPLC Runtime, which acts as the execution environment for the PLC program [9]. The Runtime also provides real-time monitoring and control capabilities. OpenPLC Runtime enables us to observe the behavior of the assembly station and make adjustments as needed. Furthermore, OpenPLC Runtime communicates seamlessly with external tools like Factory I/O and ScadaBR using the Modbus protocol. This integration allows for both visualization and control, with Factory I/O providing a 3D simulation of the assembly station and ScadaBR serving as a comprehensive monitoring interface.

4.2 Factory I/O

Factory I/O is a 3D simulation software that is widely used for industrial automation training and process visualization [10]. The software allows users to create realistic and interactive factory simulations with various industrial components and it is a valuable tool for testing and designing control systems[11]. Factory I/O is available as a free trial for 30 days, but its use is limited to Windows operating systems [12]. In our project, Factory I/O can only be integrated with OpenPLC when the OpenPLC Runtime is running locally on the host system, rather than within a containerized environment. In our project, we use Factory I/O to visualize the simulation process, providing a dynamic 3D representation of the industrial assembly station. This setup includes elements such as conveyors, sensors, and a pick-and-place unit, all controlled by OpenPLC Runtime. This visualization enhances the understanding of the assembly process and helps in identifying and resolving potential issues in the control logic.

4.3 ScadaBR

ScadaBR is an open-source Supervisory Control and Data Acquisition (SCADA) system that also functions as a Human-Machine Interface (HMI)[13]. It provides a platform for monitoring and controlling industrial processes in real time [14]. In our project, ScadaBR is used as the HMI to monitor the program running on the OpenPLC Runtime. By integrating ScadaBR with OpenPLC, both running within containers on our Docker network, we achieve seamless communication. ScadaBR allows us to monitor the status of the simulated the components of the assembly station.

8 OpenPLC Documentation, https://autonomylogic.com/docs/openplc-overview/, 13.12.24.
9 OpenPLC Documentation, https://autonomylogic.com/docs/openplc-overview/, 13.12.24.
10 Factory I/O Documentation, https://docs.factoryio.com, 13.12.24.
11 Factory I/O Documentation, https://docs.factoryio.com, 13.12.24.
12 Factory I/O Documentation, https://docs.factoryio.com, 13.12.24.
13 ScadaBR Documentation, https://github.com/ScadaBR, 13.12.24.
14 ScadaBR Documentation, https://github.com/ScadaBR, 13.12.24.

4.4 Docker

4.4.1 Container

A Docker container is a lightweight, isolated environment that allows applications and their dependencies to run consistently across different systems [15]. Unlike virtual machines, which visualize hardware, Docker visualizes at the operating system level[16]. Containers share the host operating system's kernel but run as independent processes, which makes them more efficient and faster to start [17]. This isolation ensures that the application inside the container does not interfere with other applications or the host system[18]. In our project, Docker containers are used for three main purposes: Running OpenPLC Runtime, Running ScadaBR, Running Cyberattacks. It also guarantees a well defined, isolated and controlled environment for all of those mentioned processes.

Running OpenPLC Runtime We use a Docker container to run the OpenPLC software, which serves as the central control unit in the automation system. By containerizing OpenPLC, we ensure that the runtime environment is consistent and easily portable. The containerized OpenPLC can be deployed with our configurations. By running the OpenPLC container in the same network as the simulated devices, it is ensured that there are accesible.

Running ScadaBR We use a Docker Container to run the ScadaBr software. The ScadaBR container is configured to operate within the same Docker network as OpenPLC to communicate seamlessly between the two using the Modbus protocol. This setup enables ScadaBR to access OpenPLC's real-time data and provide system status updates. The integrated environment ensures reliable interaction between the HMI and the control system.

Running Cyberattacks We use Docker containers to simulate selected cyberattacks, with each type of attack running in its own dedicated container. These containers were placed within the same Docker network as OpenPLC and ScadaBR to allow seamless communication between the attack containers and the target system. This setup enabled us to isolate and test the effects of each cyberattack while maintaining a realistic network environment that closely mirrors industrial system interactions.

4.4.2 Network

Docker networks enable containers to communicate with each other and with the outside world in a controlled and isolated environment[19]. By default, Docker provides several network types, such as the Bridge network[20]. The Bridge Network is the default Docker Network and allows containers to communicate with each other using private IP addresses while being isolated from the host network unless specific ports are exposed[21].

15 Docker Documentation, https://docs.docker.com/get-started/docker-overview/, 13.12.24.
16 Docker Documentation, https://docs.docker.com/get-started/docker-overview/, 13.12.24.
17 Docker Documentation, https://docs.docker.com/get-started/docker-overview/, 13.12.24.
18 Docker Documentation, https://docs.docker.com/get-started/docker-overview/, 13.12.24.
19 Docker Network Documentation, https://docs.docker.com/engine/network/, 13.12.24.
20 Docker Network Documentation, https://docs.docker.com/engine/network/, 13.12.24.
21 Docker Network Documentation, https://docs.docker.com/engine/network/, 13.12.24.

We use a custom Docker bridge network for all the containers in the project. This network is configured with a specific subnet and gateway to ensure that all containers are within the same network and can communicate directly with each other. This setup isolates the simulation environment from the host system and other networks. Through the isolation of the network a secure and reliable communication between containers is ensured. Each container is assigned a static IP address within the network. This approach is essential for industrial automation systems, where devices must have consistent and predictable communication routes to ensure efficient operation.

4.5 Terraform

Terraform is a infrastructure-as-code tool that is used in our project to automate the management and provisioning of docker resources and make the project redeployable. We use Terraform to manage all of our Docker components of our infrastructure. These configurations are stored in version-controlled state files, which provides transparency and allows us to track changes over time. Using Terraform enhances infrastructure management by allowing us to automate the deployment and updates of Docker resources. By automizing the deployment, we reduce manual intervention and the risk of configuration drift. Its ability to deploy resources through code ensures a consistent approach to managing our Docker infrastructure. Another advantage is that our specific Docker infrastructure configurations are redeployable in any other Docker project, which faciliates a new setup of our project.

4.6 Cybersecurity Attacks

We simulate several types of cyberattacks to test the robustness and security of the OpenPLC system running in Docker, which communicates with ScadaBR. The cyberattacks we used include DDoS (Distributed Denial of Service), HTTP Flood, Man-in-the-Middle (MITM), and Modbus Flooding. Each of these attacks targets different aspects of the system, helping us assess its vulnerability to various threats.

DDoS A DDoS attack aims to overwhelm the system by flooding it with excessive traffic from multiple sources [22]. The goal of the attack is to exhaust the resources of the target system, causing it to slow down or crash[23]. In our setup, we simulated a DDoS attack on the OpenPLC system to test its resilience when faced with a high volume of incoming network traffic. The attack attempts to disrupt communication between the Docker containers running OpenPLC and ScadaBR.

HTTP Flood An HTTP Flood is a type of application layer attack that targets the web servers of a system by sending numerous HTTP requests. Our configured HTTP Flood attack specifically targets the OpenPLC system by attempting to prevent user login. In this attack, a large volume of HTTP requests is sent to the login page of the OpenPLC system, mimicking legitimate login attempts. Ultimately the attack is overloading the server and making it impossible for legitimate users to authenticate.

22 DDoS Explanation, https://www.cloudflare.com/de-de/learning/ddos/ddos-attack-tools/how-to-ddos/, 13.12.24.

23 DDoS Explanation, https://www.cloudflare.com/de-de/learning/ddos/ddos-attack-tools/how-to-ddos/, 13.12.24.

MITM A Man-in-the-Middle (MITM) attack occurs when a malicious actor intercepts the communication between two systems,[24] such as between OpenPLC and ScadaBR. The attacker can alter, inject, or steal data being transmitted between the systems without either party realizing it [25]. To assess the security of our OpenPLC setup, we simulated a MITM attack to see if sensitive information could be intercepted or if the integrity of the commands between OpenPLC and ScadaBR could be compromised.

Modbus Flooding Modbus Flooding targets the Modbus protocol, which is commonly used in industrial automation systems to facilitate communication between PLCs, sensors, and other devices[26]. In this attack, the attacker floods the network with a high volume of Modbus requests, overwhelming the target system's ability to process legitimate requests[27]. We simulated a Modbus Flooding attack on the OpenPLC system running within Docker, which communicates with ScadaBR using Modbus.

5 Evaluation

5.1 Cybersecurity Attacks

In our evaluation, we focussed on DDOS attacks and HTTP Flooding. Both of these attacks provides valuable insights into a specific type of vulnerability in the OpenPLC and ScadaBR setup, allowing us to evaluate the effectiveness of security measures and identify areas for improvement in the system's defense against cyber threats.

Interestingly, the DDOS attack against the Modbus port of the OpenPLC Runtime did not have any significant effect. We expected to experience a drop in the responsiveness of the ScadaBr webserver, since its measurings depend on OpenPLC providing accurate information on time. But appart from two small spikes in CPU ussage which can be seen in Figure 3, as well as an increase in incomming network traffic. The functionality and communication quality between OpenPLC and ScadaBR was seemingly untouched. This outcome might be due to OpenPLC recognizing the sent packets as somehow invalid and discards the immediately.

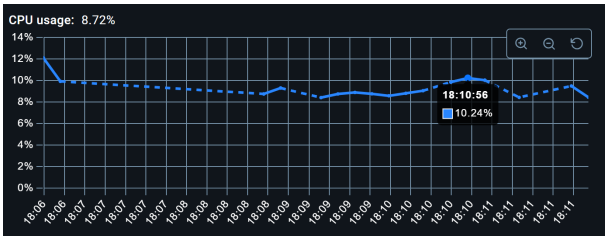

Figure 3

24 MiTM Explanation,https://owasp.org/www-community/attacks/Manipulator-in-the-middle$_attack$, 13.12.24.
25 MiTM Explanation, https://owasp.org/www-community/attacks/Manipulator-in-the-middle$_attack$, 13.12.24.
26 Practical Modbus Flooding Attack and Detection, https://dl.acm.org/doi/pdf/10.5555/2667510.2667517, 13.12.24.
27 Practical Modbus Flooding Attack and Detection, https://dl.acm.org/doi/pdf/10.5555/2667510.2667517, 13.12.24.

Even when exposed to over a dozen processes attempting an HTTP DDoS attack on the OpenPLC web interface, no significant impact on reliability or responsiveness was observed, aside from a spike in network traffic and a slight increase in CPU usage. The same was true for other attacks, such as Modbus Flooding and MITM, where none had any noticeable effect on the simulation environment. Web server mechanisms like rate limiting and limiting the number of connection attempts likely prevented the attack's success. Furthermore, our attacks were configured with basic settings and were relatively simple in nature. Due to their simplicity, they could easily be detected and mitigated by security systems.

5.2 Goals Achievement

In this project, we successfully achieved all of the defined goals that guided our development efforts:

Simulation of Industrial Environment and Components We designed and implemented a mini-industrial infrastructure that simulates an assembly station. Our setup includes the integration of two versions of OpenPLC: one running locally to interface with Factory I/O and the other running in Docker containers, communicating with ScadaBR. This dual setup allowed us to simulate real-world industrial processes while ensuring proper separation of system components.

Containerization with Docker Docker was used to containerize the entire simulation, ensuring isolation and portability across different environments. We also employed Docker's networking features to allow containers to communicate with each other within the Docker network. This setup facilitated interaction between the OpenPLC and ScadaBR and ensures seamless data exchange and coordination.

Automation with Terraform Terraform was implemented to automate the provisioning and configuration of the infrastructure, streamlining the deployment process and ensuring consistent and reproducible deployments across different environments. This automation improved efficiency and made the infrastructure easy to manage and scale.

Security Evaluation We evaluated the security of the simulated industrial environment by performing several cyberattacks, including DDoS, HTTP Flood, MITM, and Modbus Flooding. None of the attacks were successful, likely due to the simplicity of our configured attacks combined with the inherently robust network settings configurations.

6 Conclusion

In this chapter, we summarize the key achievements of the project. This project successfully implemented a mini-industrial simulation environment that replicates an assembly station using OpenPLC, Factory I/O, and ScadaBR. By employing Docker for containerization and Terraform for automation, we created a system that is both portable and reproducible, with seamless communication between components ensured via Docker networks. We tested the system's security by simulating various cyberattacks, including DDoS, HTTP Flood, MITM, and Modbus Flooding. None of these attacks were successful, probably

because of their basic configuration.

In summary, the project achieved its goals by creating a functional and secure industrial simulation environment. The work highlights the potential of open-source tools in industrial automation and security testing.