UNIVERSITÀ DI TRENTO

Department of Information Engineering and Computer Science

Master's Degree in
Computer Science

Final Dissertation

Enhancing industrial devices: design and implementation of a security scanning tool

Supervisor

Prof. Alessandro Marchetto

Student

Federico Cucino

Co-supervisors

Arrigo Zanette

Marcello Bellomi (former)

Academic year 2023/24

Acknowledgements

First, I would like to thank my academic supervisor, professor Alessandro Marchetto, for the guidance and the support.

I extend my gratitude to Corvina & Exor International companies for giving me the opportunity to work on this challenging and actual project, in particular to my co-supervisors Marcello and Arrigo for their mentorship, technical assistance and continous feedback during the internship. The hands-on experience and exposure to real-world industrial challenges have been invaluable for both my personal and professional growth.

As always, thanks to my family for their unconditional support, source of strength and motivation.

Last but not least, I would like to say thank you to my friends and all the individuals who have been part of this journey.

Contents

List of Figures

Abstract

The rapid and widespread adoption of Industrial Internet of Things (IIoT) technologies has revolutionised industries such as manufacturing, energy and transportation by enabling real-time monitoring and automation. However, this increased connectivity has expanded the attack surface for cyberattacks, exposing industrial devices and operation technologies networks to new threats. In addition, such legacy and constrained systems often lack modern security features and, due to their critical nature, they are often difficult to update or replace.

This thesis presents an approach to address these challenges by presenting the design and the implementation of a security scanning tool specifically for industrial devices. The tool aims to identify misconfigurations while providing actionable recommendations to enhance the security posture, ensuring the integrity, confidentiality and availability of industrial systems. The research incorporates a comprehensive scanning methodology aligned with the most recent industry standards and regulations requirements.

The scanning tool evaluates key risk factors, including outdated software, weak credentials and insecure configurations and it outputs a detailed report that guides users in mitigating the identified weaknesses. It has been designed to be modular and extensible and it has been tested on a series of real industrial devices.

1 Introduction

Nowadays, cybersecurity is one of the most trending subjects of discussion around the world [1].

The *Internet* is growing: the users are growing, therefore the exchanged amount of data are growing, and so the malicious actors are growing, trying to mislead an increasing number of people, regardless they are newbies or experienced ones. Every day hundreds of new vulnerabilities [8] are discovered and millions of cyberattacks are performed, with an increase of nearly three times compared to the previous year [11].

In today's rapidly evolving industrial landscape, the integration of connected devices and smart technologies, commonly referred to as the Internet of Things (IoT), has transformed traditional industries such as manufacturing, energy and transportation. These connected devices enable real-time monitoring, predictive maintenance and automation, resulting in enhanced productivity and efficiency.

However, the widespread adoption of the Industrial IoT also introduces significant security challenges: industrial control systems and operational technology (OT) networks, which were historically isolated, are now exposed to cyber threats that were once only confined to the realm of information technology (IT).

The increased interconnectivity has expanded the attack surface, making industrial devices vulnerable to cyberattacks, including ransomware, data breaches and operational disruptions. According to the European Commission, hardware and software products are increasingly subject to successful cyberattacks, leading to an estimated global annual cost of cybercrime of EUR 5.5 trillion by 2021 [5].

The current situation of many industrial environments is inadequate to meet the rising threat level. Many legacy systems and industrial devices lack modern security features and traditional security measures are often difficult to implement due to the unique requirements of industrial environments. To address these challenges, there is a pressing need for security solutions tailored to the specific characteristics of industrial devices and networks.

This thesis presents the design and the implementation of an application that performs security scans on industrial devices. The objective is to develop a robust and efficient tool capable of identifying potential risks due to misconfigurations in industrial devices, ensuring the integrity, confidentiality and availability of the device and its associated data and providing actionable recommendations for improving the security of industrial systems. We suggest the best practices and do not directly modify the devices configurations due to the critical nature of industrial systems, where any disruption could have severe consequences on the physical environment.

The internship project and this thesis are part of the curricular activities of the Master's Degree in Computer Science - Cybersecurity branch - at the University of Trento. The project was carried out in the first semester of 2024 in a company called *Corvina*[1], part of the *Exor International*[2] multinational group, located in the province of Verona, which designs and builds industrial devices and provides solutions to make these legacy devices *smart* by connecting them to the Internet and exchanging data with the Corvina cloud platform, in order to let the customers retrieve and elaborate these data or remote controlling the devices.

In the following chapters, the underlying design principles, implementation strategies and practical applications of the proposed security scan tool will be discussed in detail, providing a comprehensive solution to an increasingly critical problem in industrial cybersecurity.

2 Background

This chapter contains an explanation of many fundamental terms used in the thesis, without which it would not be possible to fully understand all the covered topics.

2.1 Software vs. Security

A software is a set of programs and data which provides functionalities. Security is understanding and identifying software-induced security risks and how to manage them. Software and functionalities come with certain risks and software security is about managing these.

Therefore, software plays a crucial role in providing security, but it is also one of the relevant sources of security issues and problems. Many times the developers have limited training on this concept or the goal is the speed of the development rather than the attention to the hidden details; as result, security is often considered a secondary factor, because it is a complex and expansive task, hard to evaluate when nothing bad happens but usually too late to evaluate when something bad happens [37].

A software system is secure if it satisfies specified security objectives, starting from the CIA triad which stands for:

- **Confidentiality**: unauthorized actors cannot have access or read the information;
- **Integrity**: unauthorized actors cannot change or alter the information;
- **Availability**: authorized actors can always have access to the information;

The CIA triad is a common model that stands as the basis for the development of security systems. Ideally, when all three objectives have been met, the security profile of the organization is stronger and better equipped to handle threat incidents [4].

Secure software is not only composed of the CIA triad; there are also other objectives like [37]

- **Authentication**: who is performing a task;
- **Authorization**: what is that actor allowed to do;
- **Privacy**: controlling personal information from being shared;
- **Anonymity**: remaining unidentified to others;

[1] https://corvina.io

[2] https://www.exorint.com

- **Non-repudiation**: actor cannot deny having taken an action;
- **Reliability**: the extent to which a software yields consistent and expected results;
- **Audit**: having traces of performed actions in separate systems or places;
- **Monitoring**: observing the system for any unusual activity;
- **Intrusion detection**: detecting unauthorized access;
- **Intrusion prevention**: preventing unauthorized access;

Total security is unachievable, but the goal is to minimize the risks and the vulnerabilities. Security is a process, not a product, and it is a continuous process [37].

2.2 Risk vs. vulnerability vs. CVE

We talked about risks in the previous section, but what is a risk? A risk is the potential that a dangerous situation becomes reality. It is the probability of a threat exploiting a vulnerability and the impact of that event. A risk is a combination of a threat, a vulnerability and an impact.

Safety is about protecting from accidental risks, while security is about mitigating the risk of dangers caused by intentional and malicious actors.

A system is secure as its weakest element. A vulnerability is a weakness in a system that can be exploited by a threat. A threat is a potential danger that can exploit a vulnerability. A threat agent is an actor that can exploit a vulnerability. An attack is the exploitation of a vulnerability by a threat agent. An exploit is the code that takes advantage of a vulnerability. A graphical flow of these concepts is shown in Figure 2.1.

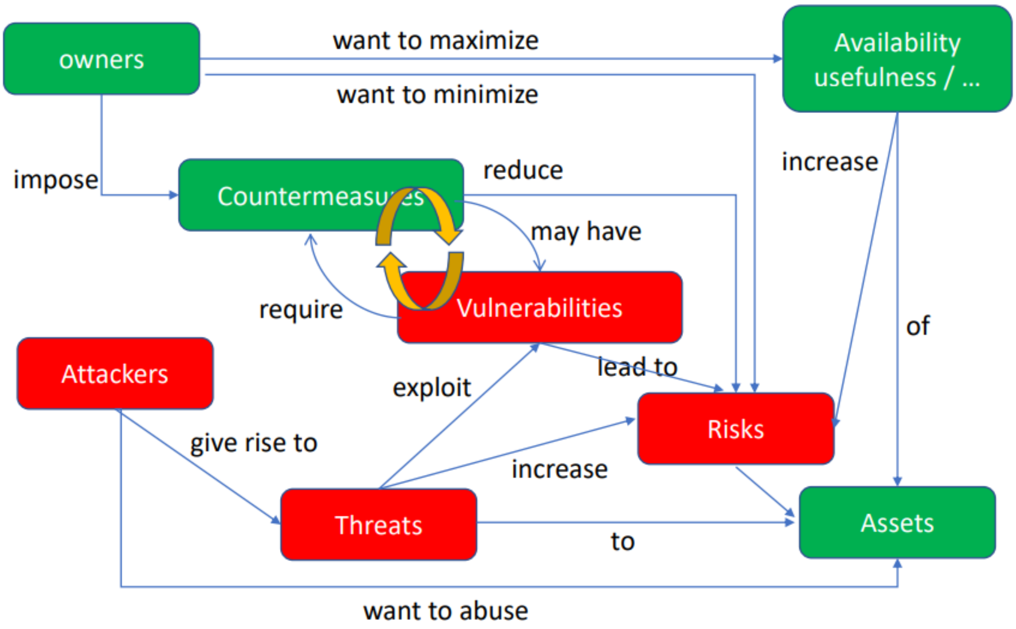


Figure 2.1: Security meta-model definition. Image by prof. A. Marchetto taken from `commoncriteriaportal.org`

CVE, which stands for *Common Vulnerabilities and Exposures*, is a list of publicly known cybersecurity vulnerabilities. The CVE system provides a reference method for publicly known information-security vulnerabilities and exposures. Each CVE entry represents a unique identifier for a specific vulnerability, named *CVE-ID*, making it easier to reference it across different tools.

Each CVE entry is evaluated using the CVSS score, which stands for *Common Vulnerability Scoring System*. This system provides a standardized way to assess the severity of vulnerabilities. It is important to note that CVSS measures severity, not risk. The difference is that severity is the intrinsic

characteristic of a vulnerability while risk is the likelihood of an attacker exploiting the vulnerability and the impact of that event in a specific environment.
CVSS v2.0 and CVSS v3.x include three metric groups: Base, Temporal and Environmental. The Base metrics evaluate the intrinsic characteristics of a vulnerability that are constant over time and across user environments. The Temporal metrics adjust the valutation based on factors like patch availability and exploit code and the Environmental metrics allow organizations to customize the score based on their unique environment. The latest version, CVSS v4.0, introduced in 2023, includes Base, Threat, Environmental and Supplemental metric groups. Compared to the previous version, the Threat metrics evaluate the likelihood of an attacker exploiting the vulnerability, the Environmental metrics evaluate the impact of the vulnerability on the organization's unique environment and the Supplemental ones provide additional information about the vulnerability [9].
The metrics produce a numerical score from 0 to 10, where highest score means highest severity, and the assessment is also represented as a vector string, which is a concise textual representation of the values used to calculate the score [10].

NVD is the *National Vulnerability Database*, a U.S. government database that contains information about known vulnerabilities, including their CVE identifiers and CVSS scores. It is maintained by NIST and serves as a comprehensive resource for organizations to evaluate security risks.

NIST (*National Institute of Standards and Technology*) is a U.S. federal agency that develops cybersecurity standards, guidelines and best practices. Even if it is an American agency, its standards are widely used around the world.

2.2.1 Bug

A bug is a flaw in a system that is not behaving as it is designed to do; a vulnerability is a way of abusing the system, in a security-related way, whether that's due to a design fault or an implementation fault, so a bug.

Many security issues are related to vulnerabilities due to bugs, but not all bugs are vulnerabilities. Exploitation is an activity composed of many steps in which an attacker uses a bug to gain its goal.

2.3 IT and OT Security

IT (*Information Technology*) and OT (*Operational Technology*) focus on protecting different types of systems and they have distinct priorities.

IT is primarily concerned with managing electronic data, supporting business operations and facilitating decision-making through the use of computers and software to securely gather, store, process and share information; basically, it represents the internet access to the cloud we are used to interacting with every day. In contrast, OT focuses on controlling physical processes and equipment in industrial operations such as manufacturing and energy. Unlike IT, OT directly interfaces with industrial machinery and processes, addressing the physical environment and operational requirements [35], as visible in Figure 2.2.

We can devise some comparison points between the two types:

1. **Primary priorities**: IT security prioritizes Confidentiality, Integrity and Availability (CIA), with the main focus set to protect data from breaches and authorized actions, while OT security prioritizes Availability, Integrity and Confidentiality (AIC), ensuring the continuous operation of critical systems with safety and reliability being more important than data confidentiality.

2. **Impact of Security Breaches**: IT breaches typically affect data confidentiality and may result in privacy violations, financial losses or business disruptions. OT breaches can have severe direct consequences, including physical damage and environmental harm.

3. **Security Measures**: IT security can use cybersecurity tools such as firewalls, antivirus, encryption and user authentication. Instead, OT security should require specialized tools like network segmentation, strict access control and real-time monitoring.

4. **Regulatory Requirements**: IT uses regulations like GDPR, the *General Data Protection Regulation* to protect personal data for a European citizen and standards like ISO-27001, while

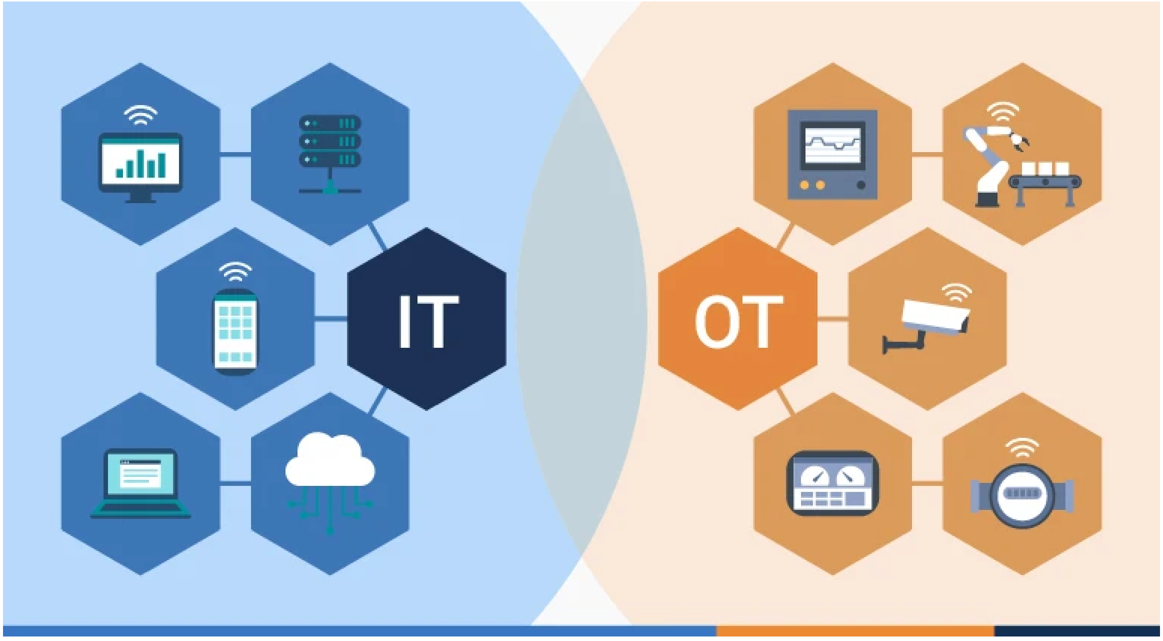


Figure 2.2: IT vs OT: how they differ. Image by `onlogic.com`

OT follows regulations and standards like the IEC 62443, which is a series of standards for industrial automation and control systems management and security. More on these regulations in the next chapters.

5. **Vulnerability management**: IT systems can usually be patched with a software update in a short time without severe impact on operations, while OT systems may require a longer time to be patched, as they are often critical to the operation of industrial processes or incompatible with the running legacy systems.

Despite their distinct differences, IT vs. OT cybersecurity share similarities and are increasingly overlapping and one approach has not to exclude the other.

According to Figure 2.3 taken by the *2023 State of Operational Technology and Cybersecurity Report* by Fortinet[1], a global leader of cybersecurity solutions and services, nearly one-third of respondents indicated both IT and OT systems were impacted, up from the 21% in 2022.

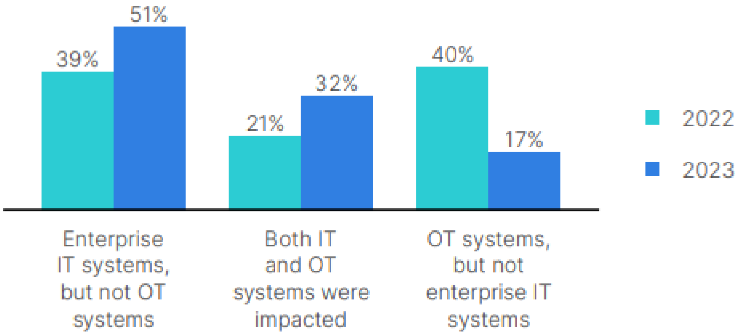


Figure 2.3: Attacks to IT and OT systems: impacted environments

As a general rule, OT devices are traditionally kept separate from the public internet and often internal networks, which means they can only be accessed by authorized employees. However, it is increasingly possible for OT systems to be controlled and monitored by IT systems or remotely via the Internet [34].

[1] `https://www.fortinet.com`

2.4 Industry 4.0

Industry 4.0, synonymous of *smart manufacturing*, is the realization of the digital transformation of the field, delivering real-time decision-making, enhanced productivity, flexibility and agility to revolutionize the way companies manufacture, improve and distribute their products.

This is the Fourth Industrial Revolution, characterized by increasing automation and the employment of smart machines and smart factories. By collecting more data from the factory floor and combining that with other enterprise operational data, a smart factory can achieve information transparency and better decisions.

Some specific technologies are pushing this revolution: [47]

- **Internet of Things (IoT)**: machines on the factory floor are equipped with boards that allow the machines to connect with remote services, making possible for large amounts of valuable data to be collected, analyzed and exchanged.

- **Cloud computing**: the typically large amount of data being stored and analyzed can be processed efficiently and cost-effectively with *the cloud*. Cloud computing can also reduce startup costs for small- and medium-sized manufacturers who can right-size their needs and scale as their business grows.

- **AI and machine learning**: Artificial Intelligence (AI) and machine learning can create insights providing visibility, predictability and automation of operations and business processes. The data collected from these assets can help businesses perform predictive maintenance based on machine learning algorithms, resulting in more uptime and higher efficiency.

- **Edge computing**: to reduce latency and improve security, edge computing can be used to process the needed data closer to the source, rather than sending all the raw statistics to the cloud, wasting bandwidth and time.

- **Cybersecurity**: last but not least, when undergoing a digital transformation to Industry 4.0, it is essential to consider a cybersecurity approach that encompasses IT and OT equipment.

2.5 Industrial devices: PLC and HMI

Industrial devices are used in many different sectors like manufacturing, energy, transportation and so on. We can trace back these devices to a common tablet placed on a wall or on a production line, with the difference that these devices support a wide range of industrial protocols. They are usually not connected to the internet and they take input from a physical human interaction physically on the place.

These types of devices are an example of HMI, which stands for *Human Machine Interface*, and they are defined as a feature or component of a certain entity that enables humans to engage and interact with machines. In the past HMIs started to be command line interfaces handled by a keyboard, then they evolved to graphical user interfaces with the support for the mouse navigation and now they are touchscreens, where the end-user directly touches the screen.
Traditionally, to integrate a manufacturing line with an HMI, the HMI had to be connected to a *Programming Logic Controller* (PLC) to display the data received from the PLC and give the PLC input from users [46].

In terms of the demands of Industry 4.0, industrial HMIs will also see further incorporation of new and emerging technologies that are impacting HMIs as a whole. First of all, there is the need to start integrating the Internet of Things (IoT) into industrial HMIs. This will allow the collection of data from the factory floor and the ability to analyze those data in real-time. This will also allow the ability to control the factory floor from a remote location. Of course, making these devices connected to the internet makes the risks of cyberattacks higher.

2.6 ICS and IACS

ICS and IACS refer respectively to *Industrial Control System* and *Industrial Automation and Control Systems.* In our context, they can be used interchangeably to define the collection of hardware, software and policies that control and manage an industrial process. Basically, it means that anything interacting with the system influencing its safety, security and operations belongs to IACS [27].

We use this terminology in the next chapters, especially related to the security regulations of the systems.

2.7 Standards and Regulations

Until the so-called third industrial age, or Industry 3.0, cybersecurity had minimal impact on manufacturing. Industrial machinery wasn't necessarily connected to the internet or to each other, making external risks unlikely. However, with the dawn of Industry 4.0, smart machinery and smart factories have become vital for the smooth operation of production departments, placing cybersecurity at the forefront of concern.

Cybersecurity involves protecting systems, networks and programs from digital attacks. Given the critical nature of the topic and the attention it demands, companies are embarking on paths to elevate their awareness of cyberattacks, adopting internal policies, or even pursuing specific certifications in cybersecurity. Concurrently, national and international legislators have introduced new legislative measures imposing new obligations on certain entities concerning cybersecurity.

Let's now consider the most recent legislative measures on cybersecurity and the related certifications that manufacturing industries must take into account.
Certifications provide a guarantee of product and company security. The main certifications currently considered include:

- **IEC 62443**: This series of international standards is renowned for enhancing the security of industrial control systems, setting forth fundamental prerequisites to shield industrial systems from cyber threats;
- **ISO/IEC 27001 (2022)**: This certification covers various aspects, including security policy, human resource security, physical and environmental security, communications management and regulatory compliance;

There are then several laws and regulations that address the issue of cybersecurity under various profiles.

Among them it is worth noting: [12]

- **Cyber Resilience Act (CRA)** [2]: While still pending final approval by the European Institutions, the Cyber Resilience Act seeks to enhance the resilience of the European digital market by ensuring that connected devices and digital services are equipped to withstand cyberattacks effectively. It will become effective in 2027;
- **NIS Directive 2** [3]: This directive outlines essential criteria that companies must adhere to in order to maintain a robust level of cybersecurity. These criteria cover the implementation of risk analysis strategies, fortification of information system security and effective incident management protocols. EU Member States must transpose this directive, with enforcement specified in the respective national acts;
- **Italian Cybersecurity Law (June 28th, 2024 No. 90)** [4]: This Italian law pertains to national cybersecurity and applies to both public and private entities whose services are deemed critical. It mandates the implementation of security measures to protect critical digital infrastructures and sensitive information, including specific obligations to notify the Cybersecurity Agency of any cyber incidents;
- **New Machinery Regulation No. 1230/2023** [5]: This regulation will replace the Machinery Directive No. 2006/42/EC, focusing on the overall safety of machinery and semi-machinery and

[2] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:52022PC0454

[3] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32022L2555

[4] https://www.gazzettaufficiale.it/eli/id/2024/07/02/24G00108/sg

[5] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32023R1230

it will become effective in 2027. It emphasizes the essential integration of cybersecurity into the design and manufacturing processes of machinery, recognizing the potential risks that cyber vulnerabilities pose to physical safety;

2.8 Security vulnerabilities scan

A security vulnerabilities scan is a process that looks for vulnerabilities in a system or network. It is a way to identify potential security risks and weaknesses that could be exploited by attackers. The scans can be performed manually or automatically using specialized tools. The goal of this type of scan is to identify vulnerabilities and provide recommendations for improving the security of the system.

There are also different types of security scans, including network scans and penetration tests. Network scans are used to identify devices on a network and detect open ports and services while penetration tests are used to simulate cyberattacks and test the security of a system.

Security scans are an important part of a comprehensive security program. They can help organizations identify and address security vulnerabilities before they are exploited by attackers. By regularly performing security scans, organizations can improve their security attitude and reduce the risk of a security breach.

2.9 JSON and YAML

JSON, acronym of *JavaScript Object Notation*, is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is based on a subset of the JavaScript programming language. JSON is often used to exchange data between a server and a web application. It is a text format that is language-independent and can be used with any programming language. JSON is commonly used for configuration files, API responses and data storage.

YAML, a recursive acronym of *YAML Ain't Markup Language*, is a human-readable data serialization format that is usually used for configuration files. It is a superset of JSON and is designed to be more human-readable and easier to write than JSON. It is commonly used in the cloud computing industry for configuration files and infrastructure as code.

Listing 2.1: JSON example

```json
{
  "name": "John",
  "employed": true,
  "address": {
    "city": "Verona",
    "country": "Italy"
  },
  "hobby": ["reading", "running"]
}
```

Listing 2.2: YAML example

```yaml
name: John
employed: true
address:
  city: Verona
  country: Italy
hobby:
  - reading
  - running
```

The example in Listing 2.1 shows the data represented in JSON, while Listing 2.2 shows the same data as YAML.

2.10 REST API

REST stands for *REpresentational State Transfer*, and it is an architectural style for designing networked applications. It relies on a stateless client-server communication protocol, meaning that each request from a client to a server must contain all the information necessary to understand that request. REST is usually used in web services development as a way to communicate between client and server.

A REST API is an application programming interface that uses HTTP requests to perform operations on a server. The endpoints can be called in a RESTful way, meaning that the user transfers a representation of the state of the resource to the destination endpoint with appropriate HTTP methods to perform standard functions like creating, reading, updating and deleting records (also known as *CRUD*) within a resource [41].

2.11 Microservices deployment

This section explains how software can be deployed in a cloud environment, with a focus on the microservices architecture.

2.11.1 Microservice

Microservice architecture is an architectural style that organizes an application into a set of services that can be deployed independently and are loosely coupled. This means each service is packaged as an executable unit ready for production; in addition, services do not have direct dependencies on each other, allowing for independent development, deployment and scaling on the needs [38].

In doing so, there is no need to scale or deploy the entire application when only a part of it needs to be updated or scaled. This approach allows for faster development and deployment cycles, as well as improved fault tolerance and scalability. The concept is shown in Figure 2.4.

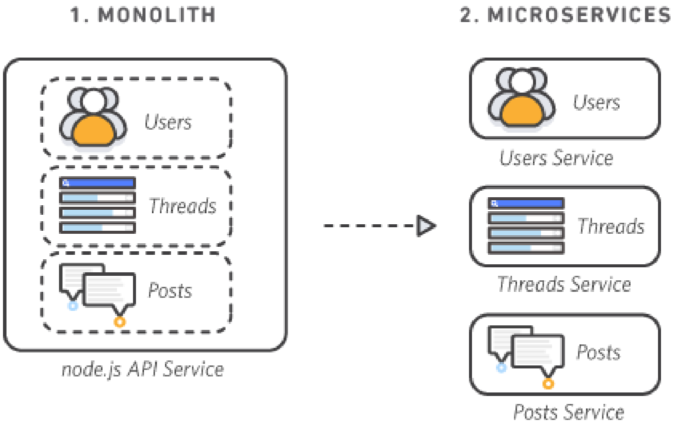


Figure 2.4: Breaking a monolithic application into microservices. Image by `aws.amazon.com`

2.11.2 Virtualization

Virtualization is a technology that enables the creation of virtual versions of physical resources such as processors, storage devices, or network components. By simulating the functions of physical hardware, virtualization allows multiple operating systems to run on a single physical machine. This approach enhances IT agility, flexibility and scalability, while also providing significant cost savings by abstracting the underlying physical hardware [45].

Each virtualized environment runs within its allocated resources. There are two main types of virtualization approaches: container-based virtualization and hypervisor-based virtualization, also known as virtual machines' virtualization.

Container vs. Virtual Machine

A modern and efficient way to execute microservices is to use *containers*.
Containers are lightweight software packages that contain all the dependencies required to execute the contained software application in a virtualized environment. They share the same kernel of the host system, but they are usually isolated from the host and from other containers and they can run different operating systems. Another advantage of containers is that they are immutable, meaning that they could be altered after their execution, but the state is restored at each restart.
Instead, compared to the containers, *virtual machines*'s goal is to virtualize the entire system hardware, therefore they are slower and heavier - resources side and disk usage side - than containers.

Since we are talking of a microservices architecture, there is the need to run multiple containers at the same time, and to manage them, from the creation to the destruction up to the recovery in case of failure or the scaling in case of high load. This is where we need an orchestration tool.

2.11.3 Orchestration

Orchestration is a term referring to the automated configuration, coordination and management of virtualization software. It is used to manage the deployment, scaling and operation of application containers. Orchestration tools are used to automate the deployment and scaling of containers, as well as to manage the networking, the storage configurations of the containers and much more. The most popular orchestration tool is Kubernetes.

2.11.4 CI/CD

A container is a running instance of an image, that is an executable package that includes all the needed software, libraries and dependencies to run an application. Images are mainly built from a Dockerfile, which is a configuration file that contains a series of instructions describing how the image should be built. Tipical steps are the installation of a base image and the runtime software along with the application code, otherwise another alternative could be to directly embed an executable file. Each specific modification or addition, such as installing a new package, represents a new layer in the image. The layering layout allows for their caching and reuse, making the building process faster and more efficient in case of changes.
The building process can be automated by a *Continuous Integration* (CI) tool. Later, the images must be pushed to a registry, that is a repository where the images are stored. The deployment of the images can be automated by a *Continuous Deployment* (CD) tool.

CI and CD are practices that allow the developers to automate the building, testing and deployment of the software. CI is the practice of integrating the code changes of the developers into a shared repository multiple times a day, and CD is the practice of automatically deploying the code changes to the production environment. CI/CD allows the developers to detect and fix the bugs early, to reduce the risk of integration issues and to deliver the software faster and more frequently.

2.12 Agile method

The *Agile methodology* is a project management philosophy that involves breaking the project into phases and emphasizing continuous collaboration and improvement. Teams follow a cycle of planning, executing and evaluating. Teams choose agile so they can respond to changes in the market or feedback from customers quickly without derailing a year's worth of plans. The publication of the Agile Manifesto[6] in 2001 marks the birth of the methodology. Since then, many agile frameworks have emerged such as scrum, kanban or lean. Each embodies the core principles of frequent iteration, continuous learning and high quality in its own way [2].

2.12.1 Scrum

Scrum is an agile project management framework, which is different from Agile, a philosophy.
The definition of scrum is based on empiricism and lean thinking. Empiricism says that knowledge comes from experience and that decisions are made based on what is observed. Lean thinking reduces waste and focuses on essentials. The scrum framework is heuristic; it is based on continuous learning

[6] `https://agilemanifesto.org`

and adjustment to fluctuating factors by acknowledging that the team does not know everything at the start of a project and it will evolve through experience. Scrum is structured to help teams to naturally adapt to changing conditions and user requirements, with re-prioritization built into the process and short release cycles so your team can constantly learn and improve [43].

Scrum artefacts help to define the product, what work has to be done to create it and who has to do it. An *epic* is a large body of work that can be broken down into smaller tasks. Each of these tasks is called *story* and it is a short requirement written from the perspective of an end user. The story is linked to a person in the team that has to carry it out.

The two main artefacts boards are the *product backlog* and the *sprint backlog.*
The former is a list of all the tasks that needs to be done; it is a dynamic list of features, requirements, improvements, fixes and epics, ordered by their priority. Essentially, it is a *"To Do"* list.
The latter is a list of items selected for the current sprint cycle; the sprint cycle is a fixed period of time, usually up to four weeks. Before each sprint, the team selects items from the product backlog to work on. Epics are broken down into stories and stories are moved from the product backlog to the sprint backlog [44].

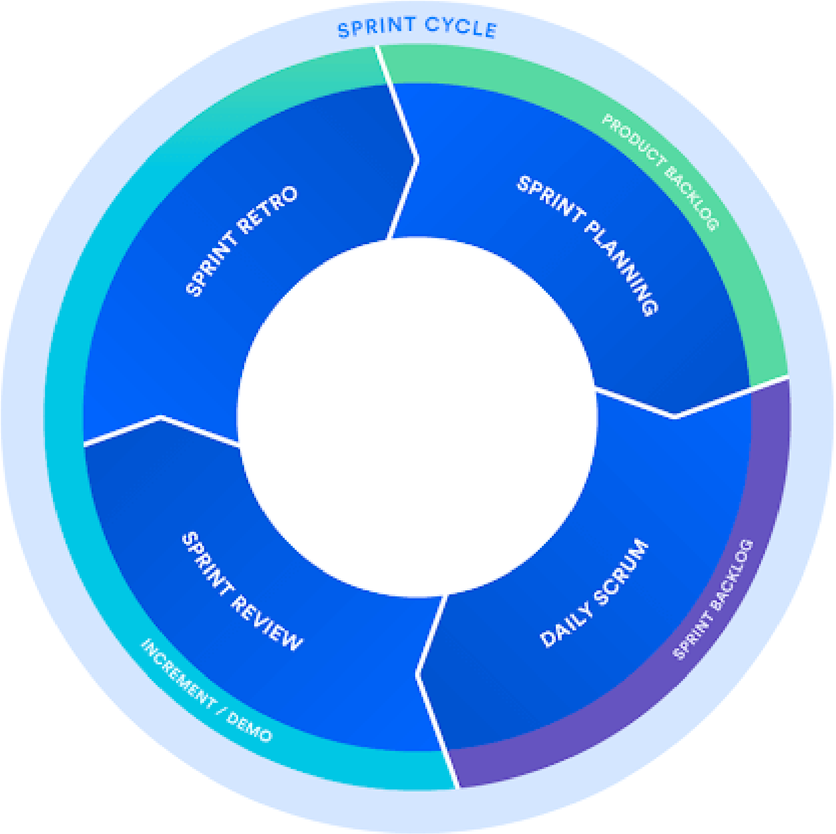


Figure 2.5: Scrum sprint cycle. Taken from `atlassian.com`

As visible in Figure 2.5, Scrum is split in four principal phases:

- The **sprint planning** is the initial phase of a sprint, the actual time period when the scrum team works; the team meets and decides what to do in the sprint by placing the tasks from the product backlog to the sprint backlog;
- The **daily scrum** is a daily meeting where the team members synchronize with each other. It is usually taken stand-up, as a way to not waste time and to keep the meeting short;

- The **sprint review** and **sprint retrospective** are the meetings at the end of the sprint where the team shows what they have done and demonstrates the work to the stakeholders and receive feedback;

3 State of the Art

This chapter describes how the company handles the required processes and takes advantage of the technologies in order to develop software that runs on industrial devices and the cloud platform. It also details which technologies and methodologies are used to develop the software in the company. This chapter represents the context in which the project started.

From now on, the first plural person conjugation is used to refer to the individual experience.

3.1 Company

Corvina is located in the province of Verona in Italy. It is part of the main multinational company called *Exor International*, which is the one that designs and assembles the industrial devices.

Corvina is the business where we are enrolled for the internship; it develops a cloud-based, all-in-one Industrial Automation Platform that provides the technology needed for the upcoming industrial challenges. It is an administration shell for distributed edge systems, integrating data collection, monitoring and control to support the machinery and applications throughout its whole lifecycle providing productivity increase and new business model based on services. It allows the data generated by the Internet of Things (IoT) to be processed and analyzed. It bridges layers between IT and OT architecture, providing effective tools to access all the Industry 4.0 benefits, such as asset performance management, cloud edge computing, predictive maintenance, data modelling and OT remote monitoring. On the devices, a built-in service collects and sends data to the cloud on Corvina servers.

Since the businesses are part of the same holding group, they share administrative resources and also technical resources. From now on, we refer to Exor talking about standards, regulations and devices, and to Corvina talking about the software, the platform and the cloud.

3.1.1 Software company organization

The cloud management platform, developed by Corvina, is composed of five main layers and it is managed by five different teams:

- **Device data**: it should make friendly the adoption of the platform and it handles the cloud lifecycle of a device, by managing the device registration and the handling of the collected data to being able to create flexible dashboards and reports;

- **VPN**: it handles the connection to a device through a VPN and it is in charge of the security of the connection;

- **Apps**: the work of the apps team is to increase the value of the platform, by allowing third parties to integrate their services with the platform and by creating apps that can be installed on the devices. Customers can craft their own apps by using the provided *Software Development Kit* (SDK), that is a set of tools and documentation;

- **Core**: it is the core of the platform and it is responsible for the provided backend REST APIs mainly used for the frontend website. It also handles the management of the user via a custom *IDentity Provider* (IDP);

- **Platform**: last but not least, the platform team orchestrate all the infrastructure needed to run the services and it is in charge of the deployment of the services on the cloud. It is also responsible for the monitoring of the services and for the security of the development lifecycle and deployment process.

The cloud software is developed using the microservices architecture, that is a design pattern that structures an application as a collection of loosely coupled services. Services can be developed, deployed and scaled independently.

We were part of the apps team, and we were in charge of developing a scanning tool that can be used to check the security of the devices powered by the cloud management software and to report the potential issues to the customer.

3.2 Initial design

Under the hood, every device is powered by a custom Linux distribution, provided by *Yocto Project*, an open-source collaboration project that helps developers create custom Linux-based systems regardless of the hardware architecture [48], developed by the internal *Research & Development* (R&D) team.

In order to be able to make the device interact with the industrial machines, Exor provides a proprietary IDE software, needed to scratch projects and deploy them on the device. This software lets the user define the inputs and outputs of the device and the logic that will be executed on the device. For example, the user can define a trigger, an alarm that will be raised when a certain condition is met, a personalized handling of the data received by the many supported protocols and so on.

Furthermore, the same IDE is used to draw the graphical interface that will be displayed on the HMI and to define the behaviour of the interface itself.
The graphical interface is composed of widgets, that are the building blocks of the interface. The user can define the position of the widgets, their size, their colour and their behaviour. The widgets can be buttons, labels, images, graphs or custom-defined ones.

The setup with their IDE software is strictly related to the device interacting with the industrial machines, and it is not the focus of the internship project. The focus is on the device itself, and on the software that runs on it.

Given that, at the beginning stages, our scanning tool could not be installed and released as a firmware update for the devices, because that would require a modification of the R&D team's workflow, we decided to develop the scanning tool as an executable binary that can be run on the device itself. It is a *command-line interface* (CLI) tool able to perform the scan and report an output to the user. Then, the user will be able to fix the issues by itself, by interacting with the device settings.

3.3 Device settings

The device provides a web interface that can be used to view and change the system settings of the device. The user can change the network settings, set the datetime, set the management user password, manage the startup of services like the SSH server and much more. The web interface can be operated directly on the device, interacting with a touchscreen, or remotely by connecting to the local webserver exposed at the device's network address.
Client and server, respectively the graphical interface and the backend, are independent of each other; the web interface is powered by REST APIs.

For the internship project, we will take advantage of the REST APIs to retrieve the status of the device and to potentially change its settings. APIs are documented with the *OpenAPI Specification*, formerly *Swagger Specification*, which is an API description format that depicts the endpoints, the parameters, the responses, the authentication needed to call them and licenses or other information. The Swagger file can be visualized through the Swagger UI, a web interface that renders OpenAPI definitions as interactive documentatione [40].

We can devise two different scenarios: API calls made by a remote host over a network and API calls made by the local host. In the first case, the TLS protocol secures the communication using the public-key authentication first, and then the APIs are protected by basic authentication, that is the client must provide the username of a management account and its related password. In the second case, the webserver listens over a port accessible bound to the same host only, with no further authentication needed. Given that the scanning tool will be run on the device itself, we will take advantage of the second case.
On the security side, we consider the *localhost* perimeter as secure, because the device is a closed system

and the user must have physical access to it.

3.4 *Go* programming language

Go is an open-source programming language supported by Google, designed for building simple, reliable and efficient software. It is statically typed, compiled and syntactically similar to C, with the added benefits of built-in concurrency management, garbage collection, memory safety, a rich standard library and lot of community packages. Go is widely used in cloud computing and web development due to its simplicity, performance and scalability [16] [17].
Go was released in 2007 by Google, and at the time of writing it is at version `1.22`.

Go's standard library provides comprehensive support for networking, encryption and concurrency [21]. The built-in packages for HTTP, TLS and JSON processing simplify the implementation of RESTful APIs, secure communication channels and data serialization/deserialization, respectively. If additional functionalities are required, the language supports third-party packages through the Go module system; in order to download a custom package, the developer simply references it in the code via the link to the repository and the Go compiler will automatically download and install it.

Go also offers a robust testing framework to write unit tests and benchmarks to ensure the reliability and performance of their code.

Another point in favour is the efficiency of the language. Go is compiled into machine code, which makes it faster than interpreted languages like Python. The language's garbage collection mechanism automatically manages memory allocation and deallocation, reducing the risk of memory leaks and improving the overall performance of the application. The language is more than a full order of magnitude faster than Python, with a smaller memory footprint and faster execution times [15].

One of the main reasons for choosing Go for the internship project is its versatility in building executable binaries for multiple platforms, including Windows, macOS and Linux. Go's cross-compilation capabilities allow the developers to build binaries for different operating systems and architectures from a single codebase, simplifying the deployment process and ensuring compatibility across various platforms. Given that industrial devices run on different architectures and operating systems, the ability to build cross-platform binaries is essential. The language natively supports the binaries compilation for over 50 combinations of operating systems and architectures [14]; given that the majority of the Exor devices run either on `Arm 32bit` or `Arm 64bit` or `x86 64bit` architectures, the Go compiler can generate the executables for these architectures with no further configuration needed. The real breakthrough is the native ability to produce static binaries, that is the binary contains all the libraries needed to run the software without any dependency on the target host system.

3.5 Adopted technologies

Referring to Chapter 2, Corvina takes advantage of many technologies to develop the software that runs on industrial devices and on the cloud platform. We are going to describe the most relevant software used in the company.

3.5.1 Security vulnerabilities scan

Corvina uses *Snyk*[1] to scan the software for vulnerabilities and receive alerts for security issues in the code. Snyk is a security company that provides a platform that enables developers to find and fix vulnerabilities in their code and dependencies. Snyk integrates with popular development tools and workflows, making it easy for developers to identify and address security issues early in the development process.

It offers a range of security tools, including vulnerability scanning, dependency monitoring and container security. The platform provides real-time alerts and recommendations for fixing security issues, helping developers to secure their applications and prevent security breaches. It also offers a vulnerability database that contains information about common security vulnerabilities and their impact on software applications.

[1] `https://snyk.io`

3.5.2 Deployment of the microservices

Specific software are used to manage the entire infrastructure of the cloud platform, from the development and testing to the deployment and monitoring of services.

Docker

The company uses *Docker*[2] to create, deploy and run containers. Docker is an application that allows developers to build, package and run applications as containers.

Docker itself enables the user to instantiate a single container at a time from an image, that is a series of instructions defining the environment of the container, starting from the base operating system up to all the steps needed to install the final software.

Docker containers can be deployed on any platform that supports Docker, including Linux, Windows and macOS. The company uses Linux as the operating system for the cloud platform, and the containers are deployed on a Kubernetes cluster.

Kubernetes

The containers can be managed by *Kubernetes*[3], also known as *K8s*, an open-source container orchestration platform that provides a platform for automating the deployment, scaling and operation of application containers across clusters of hosts. It works with a range of container tools, including Docker. Kubernetes was originally developed by Google, and in 2014 it has been open-sourced [36].

Kubernetes is a powerful tool that allows the user to manage the containers in a declarative way, that is the user declares the desired state of the system, and Kubernetes takes care of the rest. The user can define the number of replicas of a container, the resources needed by the container, the network configuration, the storage configuration and much more. Kubernetes will take care of the deployment of the containers, the scaling of the containers, the recovery of the containers in case of failure and the load balancing of the containers.

An instance of Kubernetes is called a *cluster* and it is composed of at least one *master nodes* and one or more *worker nodes*. The master node is in charge of the orchestration of the containers, while the worker nodes are in charge of running the containers. The master node is composed of the API server, the scheduler, the controller manager and *etcd*, that is a distributed key-value store used to store the state of the cluster. The worker nodes are composed of the *kubelet*, which is the agent that runs on each node and is responsible for the communication between the master node and the worker node and the *kube-proxy*, that is a network proxy that runs on each node and maintains network rules.

Every action towards the cluster is performed through the API calls to the API server, that is the entry point for the cluster. The user can interact with the cluster through the *kubectl* command-line tool, that is the Kubernetes command-line tool, or through the many available third-party software that interact with the Kubernetes API.

Helm

Helm[4] is a package manager for Kubernetes that allows the user to define, install and upgrade Kubernetes applications. Helm uses a packaging format called *charts*, that are a collection of files that describe a set of Kubernetes resources. Helm charts can be used to define the resources needed by the application, the dependencies, the configuration and the hooks that should be executed during the installation.

Helm can be used to deploy the microservices on the Kubernetes cluster.

CI/CD pipeline

A series of flows can be used to automate the building, testing and deployment of the software.

[2] `https://www.docker.com`

[3] `https://kubernetes.io`

[4] `https://helm.sh`

Once the code is pushed to the main or to the developer branch of the repository, the CI tool, hosted on the GitHub Actions[5], automatically runs the tests and, if successful, builds the image for `amd64` and `arm64` and it pushes the image to a private container registry, hosted on Google Artifact Registry[6].

3.5.3 Scrum

Corvina takes advantage of the Scrum framework to manage the development of the cloud software, via the *Jira* suite[7], a project management tool developed by Atlassian. The platform is used to manage the product and the sprint backlog, the epics and the stories and to track the progress of the team.

The team uses a board with five columns: *To Do*, *In Progress*, *Standby*, *In review* and *Done.* The stories are moved from one column to another as the work progresses. In addition to the usual columns, as defined by the framework, the third column contains the stories that are blocked by some external factor and the fourth column contains those stories that are in the testing phase. Eventually, the stories are moved to the last column.

We created a new Epic for the scanning tool and we filled it with the stories that we thought were needed to develop the tool. In this specific case, the sprint period was a month because of the internship duration. At the end of the sprint, the product backlog contained some pending stories, because of the limited time and the ongoing updating of the priorities. The daily scrum was achieved by a quotidian quick meeting with the business tutor, where we keep him updated on the progress of the stories. Of course, new stories were added to the backlog and some of them were de-prioritized.

3.6 Platform applications

The cloud management platform has an application store where the developers can upload their web *apps*.

The scaffolding of the web app is provided by a public NodeJS package bundle. It is a drafted working application, so the developer can start to develop the app without the need to set up the project from scratch. Furthermore, it contains the necessary base configuration files to deploy the app to the platform. The app has to be deployed in a dedicated Kubernetes namespace in a cluster, and later it has to be pushed to the app store by using the public platform APIs.

A scaffolding app is composed of two layers: the frontend, which is the user interface, and the backend, which is the logic of the application. The frontend is written in VueJS[8] and the backend is written in NestJS[9], a progressive Node.js framework. The former is a web application that the user can access from the cloud platform and the latter provides REST APIs needed for the frontend to interact with the backend.

4 Methodology and Design Process

This chapter contains the design of the application architecture and scanning methodology. It explains the active studies finalized to the goal of the project.

4.1 Standards and regulations

This section provides an overview of the most relevant standards and regulations for the context of the internship's research, focusing on the industrial sector and the required goals; in particular, we detail more about *IEC 62443*, *ISO 27001*, *NIS 2* and *CRA*.

[5] `https://github.com/features/actions`

[6] `https://cloud.google.com/artifact-registry/docs`

[7] `https://www.atlassian.com/software/jira`

[8] `https://vuejs.org`

[9] `https://nestjs.com`

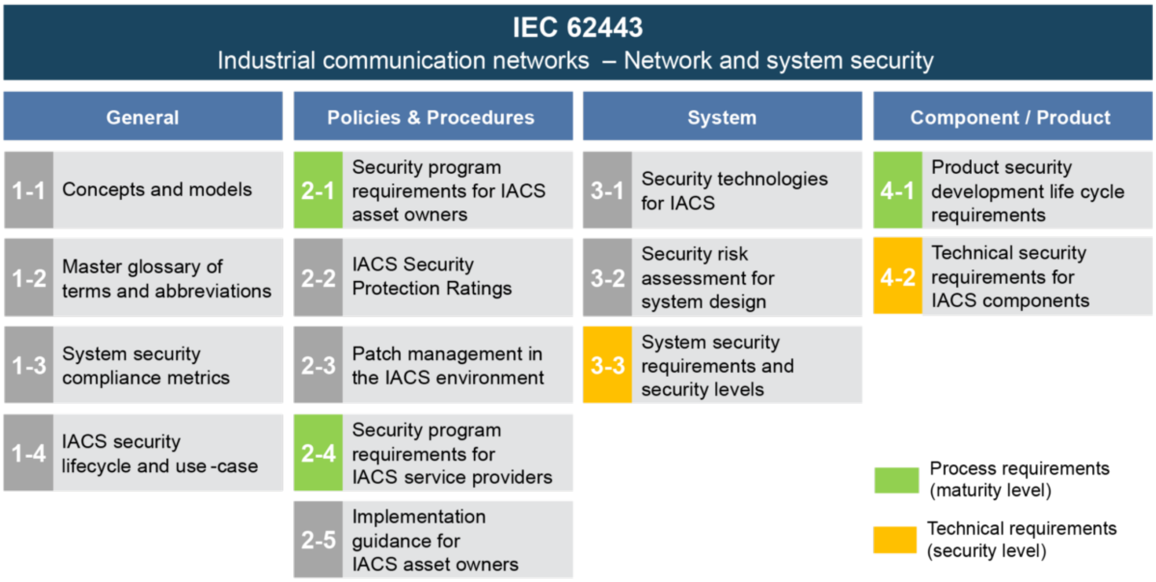
4.1.1 IEC 62443

The IEC 62443 provides guidelines, rules and definitions specifically crafted for Industrial Automation and Control Systems (IACS). It is more focused on the specific sector while ISO 27001 is more universal and open for interpretation depending on the company it applies to.

The market requires the IEC 62443 certification for the companies that are involved in the production of industrial devices, as it is a guarantee of the security of the product and the company itself.

The IEC 62443 is a set of standards drafted by the *Internation Electrotechnical Commission* (IEC) and it is divided into four categories: [30]

1. **General**: it covers topics that are common to the entire series;
2. **Policies and procedures**: focuses on methods and processes associated with IACS security;
3. **System**: it covers the requirements for the secure development and integration of systems;
4. **Component**: it covers the requirements for the secure development and integration of components, defined as final products.



IEC 62443
Industrial communication networks – Network and system security

General	Policies & Procedures	System	Component / Product
1-1 Concepts and models	🟩 2-1 Security program requirements for IACS asset owners	3-1 Security technologies for IACS	🟩 4-1 Product security development life cycle requirements
1-2 Master glossary of terms and abbreviations	2-2 IACS Security Protection Ratings	3-2 Security risk assessment for system design	🟨 4-2 Technical security requirements for IACS components
1-3 System security compliance metrics	2-3 Patch management in the IACS environment	🟨 3-3 System security requirements and security levels	
1-4 IACS security lifecycle and use-case	🟩 2-4 Security program requirements for IACS service providers		
	2-5 Implementation guidance for IACS asset owners		

🟩 Process requirements (maturity level)

🟨 Technical requirements (security level)

Figure 4.1: IEC 62443 categories. Image by Mohamed Wassef O. on `https://www.linkedin.com/pulse/power-iec-62443-safeguarding-industrial-automation-othmani-gsrde`

Each category is divided into several subcategories, corresponding to different subjects. IEC 62443 is a comprehensive standard that covers all the specific aspects starting from the product design up to the development.

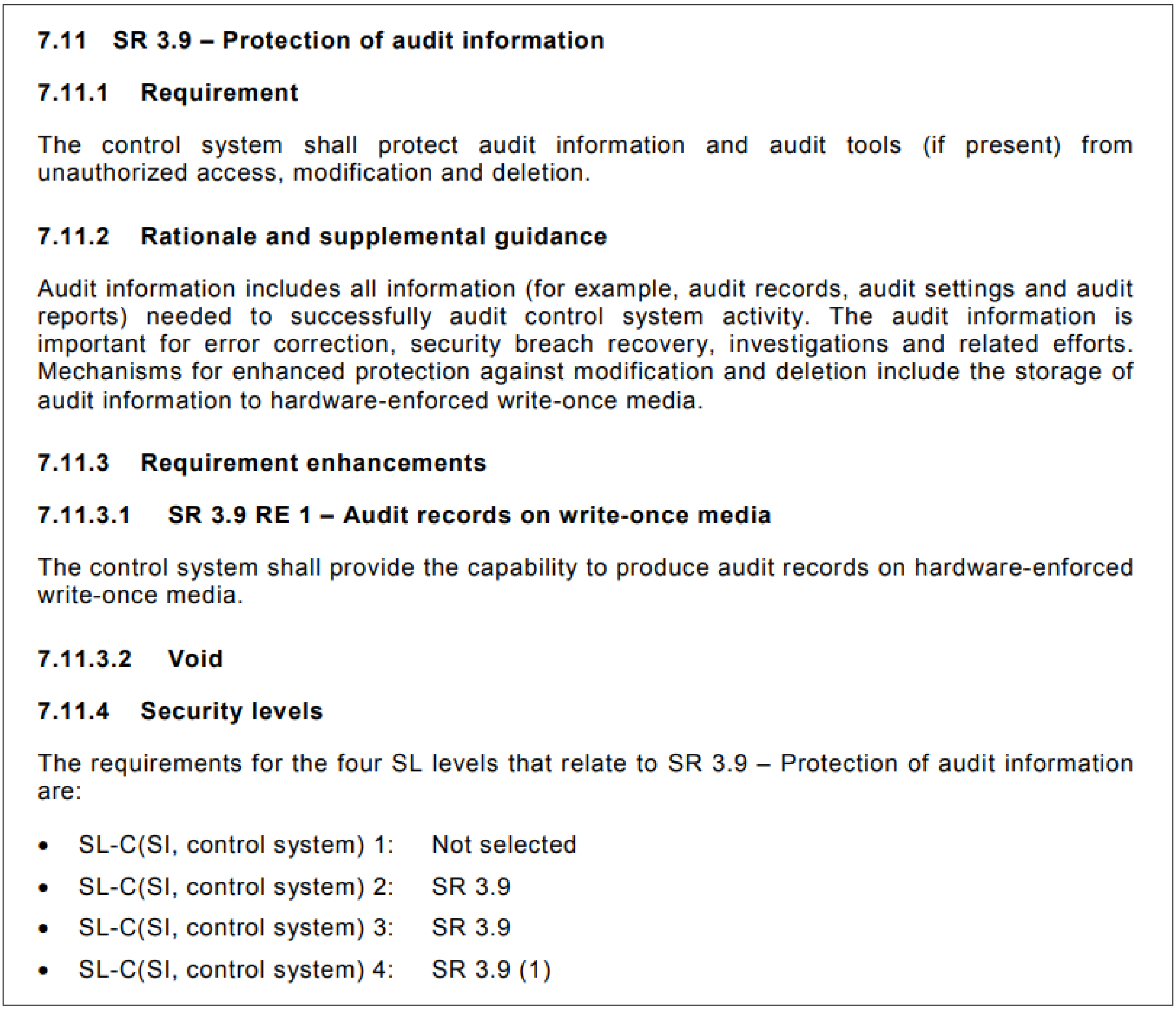
In the context of the internship project we are involved in, since we are directly at the level of product design and component development and deployment, the relevant subcategories we are focusing on are `3-3` and `4-2`, respectively named *System security requirements and security levels* and *Technical security requirements for IACS components*.

`IEC 62443 3-3` and `4-2` categorize seven *Foundational Requirements* (FR) each, expanded respectively into a series of *System Requirements* (SR) and *Component Requirements* (CR) that the company must meet to obtain the certification.

For example, a system requirement taken from the subcategory *System security requirements and security levels* (`3-3`) is the one shown in Figure 4.2. We are going to explain it in detail.

The foundamental requirement is *FR 3 - System integrity* and the the system requirement states the following:

> SR 3.9 - Protection of audit information: The control system shall protect audit information and audit tools (if present) from unauthorized access, modification and deletion.



7.11 SR 3.9 – Protection of audit information

7.11.1 Requirement

The control system shall protect audit information and audit tools (if present) from unauthorized access, modification and deletion.

7.11.2 Rationale and supplemental guidance

Audit information includes all information (for example, audit records, audit settings and audit reports) needed to successfully audit control system activity. The audit information is important for error correction, security breach recovery, investigations and related efforts. Mechanisms for enhanced protection against modification and deletion include the storage of audit information to hardware-enforced write-once media.

7.11.3 Requirement enhancements

7.11.3.1 SR 3.9 RE 1 – Audit records on write-once media

The control system shall provide the capability to produce audit records on hardware-enforced write-once media.

7.11.3.2 Void

7.11.4 Security levels

The requirements for the four SL levels that relate to SR 3.9 – Protection of audit information are:

SL-C(SI, control system) 1:	Not selected
SL-C(SI, control system) 2:	SR 3.9
SL-C(SI, control system) 3:	SR 3.9
SL-C(SI, control system) 4:	SR 3.9 (1)

Figure 4.2: Snapshot of a IEC 62443 3-3 System Requirement

Log audits are essential for the prevention and detection of cyberattacks. When something happens, log audit can help to understand what happened, when and where it happened and who was involved. This is a fundamental requirement for the IEC 62443 certification.

For each requirement listed in the documentation, the standard also describes some potential rationale and supplemental guidance to help the company understand the requirement and how to implement it.

Each system or component requirement is associated with at most four *Requirement Enhancements* (RE) that can be implemented to increase the *Security Level* (SL) of the system or component.

A security level is a measure of the security of a system or component, which is eventually transmitted to the final client: it is a score indicating how much sophisticated the solution is to follow the requirement. The minimum applicable security level is 1, while the maximum is 4 [29].
It is possible to increase the score of the security level by applying more non-trivial solutions, that is a requirement enhancement. An enhancement to increase the security level from 3 to 4 is stated in `SR 3.9 RE 1`:

> SR 3.9 RE 1 - Audit records on write-once media: The control system shall provide the capability to produce audit records on hardware-enforced write-once media.

We would like to note that not every SR has exactly four REs; instead, it could need fewer solutions in order to increase its score. For instance, this is the case for SR 3.9 which has only one RE and the minimum applicable SL is level 2, as shown in Figure 4.2.

Moreover, sometimes the highest security level is not always the best solution or the most suitable for the company. Therefore, it is essential to evaluate the cost and the benefits of the solution before applying it.

At the system level, `IEC 62443 3-3` introduces security levels and corresponding requirements, offering a scalable framework to protect industrial systems. This part enables organizations to customize their security measures based on specific risk profiles and operational demands.

At the component level, `IEC 62443 4-1` lays the foundation for secure product development, emphasizing the critical importance of incorporating security measures right from the design phase. It encourages manufacturers to adopt a security-centric approach to product development, ensuring that components are fortified against cyber threats from the very beginning.
Moving from development to deployment, `IEC 62443 4-2` specifies the technical requirements for IACS components. It defines essential security capabilities such as robust authentication, encryption and intrusion detection, ensuring that each component reinforces the overall security of the system [31].

4.1.2 ISO/IEC 27001

The ISO/IEC 27001 is a globally recognized standard addressed to companies of any size and from all sectors of activity with guidance for establishing, implementing, maintaining and continually improving an information security management system, which is a set of policies and procedures defining and managing controls that an organization needs to implement to ensure that it is sensibly protecting the confidentiality, availability and integrity of assets from threats and vulnerabilities. An information security management system that meets the requirements of ISO/IEC 27001 preserves the CIA triad and gives confidence to interested parties that risks are adequately managed.

Conformity with this standard means that an organization or business has put in place a system to manage risks related to the security of data owned or handled by the company and that this system respects all the best practices and principles enshrined in this International Standard.

It has been jointly published by the *International Organization for Standardization* (ISO) [32] and the *International Electrotechnical Commission* (IEC) [28]. The number indicates that it was published under the responsibility of Subcommittee 27 (on Information Security, Cybersecurity and Privacy Protection) of ISO's and IEC's Joint Technical Committee on Information Technology (ISO/IEC JTC 1).

It is widely used around the world; as per the ISO Survey 2022, over 70000 certificates were reported in 150 countries and from all economic sectors, ranging from agriculture through manufacturing to social services [33].

4.1.3 NIS Directive 2

The Network and Information Security (NIS) Directive is the first piece of EU-wide legislation on cybersecurity, drawn in 2016, and its specific aim was to achieve a high common level of cybersecurity across the Member States. To respond to the growing threats posed by digitalisation and the surge in cyber-attacks, the Commission has submitted a proposal to replace the NIS Directive and thereby strengthen the security requirements, address the security of supply chains, streamline reporting obligations and introduce more stringent supervisory measures and stricter enforcement requirements, including harmonised sanctions across the EU. NIS 2 entered into force on January 2023, and the Member States have until October 2024 to transpose its measures into national law.

The major key features of the updated directive are: [39]

- **Expanded scope**: NIS2 extends NIS scope to cover more sectors and industries. It applies to *essential* and *important* entities: NIS include energy (electricity, district heating and cooling, oil and gas), transport (air, rail, water and road), banking, health, pharmaceutical, water, digital infrastructure (internet exchange points, DNS providers, TLD name registries, cloud computing service providers, data centre service providers, content delivery networks), public administration and space. NIS2 include postal and courier services, waste management, chemicals, food, manufacturing of medical devices, computers and electronics, machinery equipment, motor vehicles and digital providers (online marketplaces, online search engines and social networking service platforms);

- **Incident Reporting**: Companies must report significant cybersecurity incidents within 24 hours of detection, improving coordination between national cybersecurity authorities and helping mitigate the impact of attacks;

- **Fines and Penalties**: NIS2 introduces higher penalties for non-compliance, similar to the GDPR, with fines that could reach up to 2% of an organization's total global turnover.

NIS 2 aims to enhance the resilience of the European digital market, ensuring the continuity of essential services even in the face of sophisticated cyberattacks.

4.1.4 Cyber Resilience Act

While the cybersecurity of providers of digital services is regulated at the EU level by the NIS Directive 2, as analyzed in Section 4.1.3, the security of products with digital elements and software products is so far not subject to any comprehensive piece of EU regulation [5].

Therefore, the European Commission proposed the Cyber Resilience Act in 2022; it was approved in early 2024 [7] and manufacturers have to place compliant products on the Union market by 2027, aiming to enhance the security of ICT products, services and processes.

It applies to products connected directly or indirectly to another device or network except for specific exclusions such as open-source software or services that are already covered by existing rules, which is the case for medical devices, aviation and cars. A non-exhaustive list of products that are in the scope of the CRA includes: [6]

- **end devices**: laptops, smartphones, cameras, smart equipment, network equipment, Industrial Automation Control Systems (IACS);

- **software**: firmware, operating systems, applications;

- **components**: components like CPUs, video cards and libraries;

4.2 How IEC 62443 has driven the design

Recalling Section 4.1.1, the IEC 62443 standard provides a comprehensive framework for securing industrial control systems and operational technology networks. In order to perform a step towards the certification of the devices with the standard, the scanning tool must cover at least the aspects of the standard.

First of all, our project is based on Component Requirement 6.2 as the foundation:

> CR 6.2 - Continuous monitoring: The control system shall provide the capability to continuously monitor all security mechanism performance using commonly accepted security industry practices and recommendations to detect, characterize and report security breaches in a timely manner.

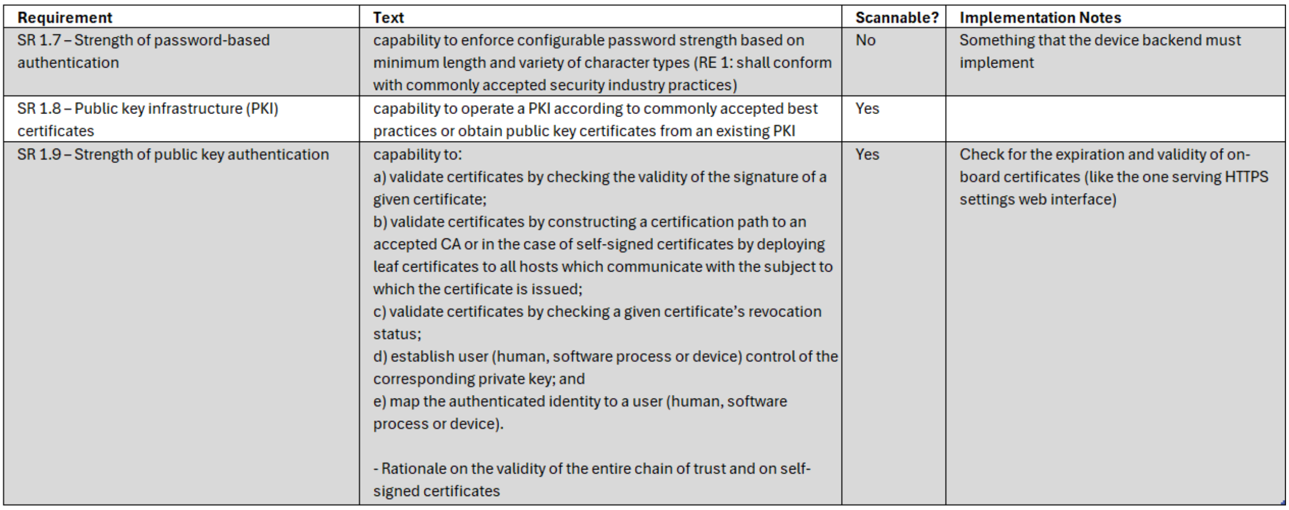
With that said, our scanning tool will become the way to actively monitor the security of the devices and to report the results to the user for the company devices.

We did a case study on the standard documentation: for each of the requirements listed in the `3-3` and `4-2` documents we said:

> *«Can we implement a check for this requirement in such a way to make the user able to fix the potential issue by himself?»*

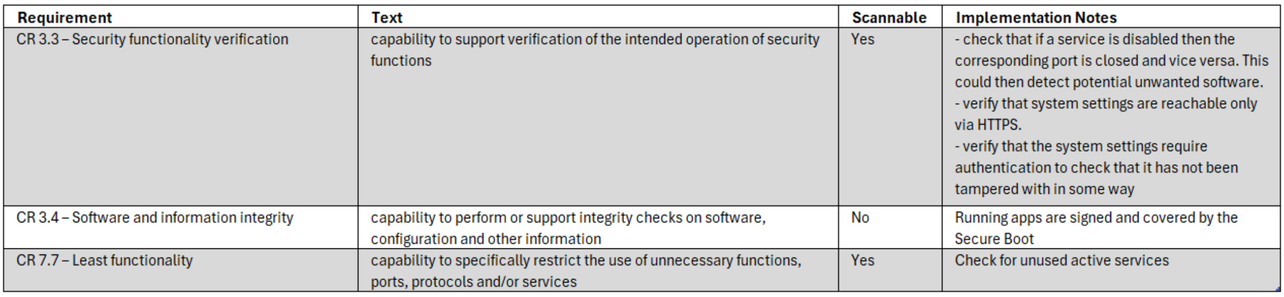
We identified in the user that has a management account on the device, whom can interact on the system settings. The goal was to make the final customer able to fix the reported weaknesses in complete autonomy, by tweaking the settings and configuration of the device or eventually by updating the operative system.

Figure 4.3 and Figure 4.4 show the findings of the case-study. The table is divided into four columns: the first one is the requirement title, the second one is the description, the third states whether we believe that it satisfies with our statement and the fourth one contains some notes about the possible implementation.



Requirement	Text	Scannable?	Implementation Notes
SR 1.7 – Strength of password-based authentication	capability to enforce configurable password strength based on minimum length and variety of character types (RE 1: shall conform with commonly accepted security industry practices)	No	Something that the device backend must implement
SR 1.8 – Public key infrastructure (PKI) certificates	capability to operate a PKI according to commonly accepted best practices or obtain public key certificates from an existing PKI	Yes	
SR 1.9 – Strength of public key authentication	capability to: a) validate certificates by checking the validity of the signature of a given certificate; b) validate certificates by constructing a certification path to an accepted CA or in the case of self-signed certificates by deploying leaf certificates to all hosts which communicate with the subject to which the certificate is issued; c) validate certificates by checking a given certificate's revocation status; d) establish user (human, software process or device) control of the corresponding private key; and e) map the authenticated identity to a user (human, software process or device). - Rationale on the validity of the entire chain of trust and on self-signed certificates	Yes	Check for the expiration and validity of on-board certificates (like the one serving HTTPS settings web interface)

Figure 4.3: IEC 62443 3-3 chosen requirements



Requirement	Text	Scannable	Implementation Notes
CR 3.3 – Security functionality verification	capability to support verification of the intended operation of security functions	Yes	- check that if a service is disabled then the corresponding port is closed and vice versa. This could then detect potential unwanted software. - verify that system settings are reachable only via HTTPS. - verify that the system settings require authentication to check that it has not been tampered with in some way
CR 3.4 – Software and information integrity	capability to perform or support integrity checks on software, configuration and other information	No	Running apps are signed and covered by the Secure Boot
CR 7.7 – Least functionality	capability to specifically restrict the use of unnecessary functions, ports, protocols and/or services	Yes	Check for unused active services

Figure 4.4: IEC 62443 4-2 chosen requirements

We already filtered the requirements that we believe relevant for the scanning tool, and we are going to evaluate them depending on the previous statement.

To better explain our choices and motivations, we now take into account *SR 1.9 - Strength of public key authentication* requirement. The requirement states that there must be the capability to enforce the validity of the used certificates by checking the signature, the expiration date and the revocation status. We believe that this is a check that we should implement in the scanning tool because the user can fix the issue by itself by renewing the certificate, or by changing the certificate authority if needed. The implementation could be done by retrieving the certificate from the device and then checking the signature, the expiration date and the revocation status. If the certificate is self-signed, the tool could suggest the user to replace it with another certificate signed by a trusted certificate authority.

On the other hand, let's consider *SR 1.7 - Strength of password-based authentication*. The requirement states that the password must have a minimum length and a variety of different character types, but if the system could let the user set an invalid password, the tool cannot enforce that requirement because it could only suggest to change the password accordingly, without the possibility to enforce it. Therefore, the issue should be reported to the development team, which should implement a password policy, release a software update and the user should be then forced to change the password at the next login. In this specific case, we believe that the burden of checking and integrating this functionality, required by the standard, should be carried out separately from the execution of the tool by the end customer.

With this approach, we have identified the checks that the scanning tool should cover and we have included them in the backlog of the project.

4.3 Backlog stories

We have a backlog of stories for the scanning tool. Therefore, we drafted a new epic on the board and we filled it with the stories that we thought were needed to develop the tool.

We considered each story as an independent task that could be developed and tested in a short time, usually in a week. So, we split the epic into small stories, each one with a title, a description and a priority.

We started from the basis of a software, so the first stories were about the setup of the project and the thinking about the working architecture, the required parameters and the output formatting of the CLI. We want to keep the functioning of the tool as simple as possible and as much universal and reusable as possible. As a reference, we took inspiration by the help pages of some common software like `npm`, the NodeJS package manager, or `curl`, the tool to perform network requests directly on the terminal. Therefore, we sketched out the manual help page to have an overview of the commands and the options that the user could use.

At first glance, we decided to implement the following flow: the user runs the tool with the `scan` command, then the tool retrieves the device settings and the network status, then it performs the checks and finally it outputs the results to the console. The user can otherwise choose to save the output as an external file. The flag `--verbose` can be used to print more information about the checks and the results. The flag `--help` can be used to print the manual page or to print the documentation for a specific command.

The scan with no further options should perform all the checks that the tool can do as a default option, while the user can choose to perform only a subset of the checks by opting in using the `--list` flag. The list of the available checks is available by issuing the `list` command.

Listing 4.1: Man page

```
$ ./scantool
Corvina Industrial Security Scanning

Usage:
  scantool [command]

Available Commands:
  completion Generate the autocompletion script for the specified shell
  help       Help about any command
  list       Show all the available scans
  scan       Start the device scan (options available typing ‘scan --help‘)
  version    Print the version number

Flags:
  -h, --help    help for scantool
  --verbose     Log developer messages on stdout
  -v, --version version for scantool
```

Moreover, we thought that the tool should be able to perform a scan in every network condition, to not exclude a priori a customer with an offline configuration or with a network that does not allow outgoing connections. Therefore, we decided to implement two different modes: the online mode where the tool can reach the cloud to determine the latest available version of the software and the offline mode where the tool cannot reach the cloud and has to rely on the local timestamp supplied by some of the software and libraries. The choice of the mode is automatic.

From the board side, for each of the checks that the tool should perform, we put at least two different stories, respectively one for the online mode and one for the offline mode.

Then, we sketched out the stories about the checks to perform, starting from those extracted from the IEC 62443 standard, as explained in Section 4.2, but also adding some other checks that we were asked to implement. We also added some stories about the cross-compilation of the tool, the logging library to use and the CLI library to use.

At this point, we have a rich backlog of stories that we can work on and that we can prioritize according to the needs of the company.

The following list shows the stories that we have drafted for the scanning tool at the beginning of the project:

- Choose a CLI library

- Choose a logging library
- Parse input parameters and output
- Split flow into offline and online usage
- Check correct time
- JSON output
- Cross build script
- Scan OS version - date based
- Scan OS version - version based
- Default BSP user credentials
- Default BSP admin credentials
- Scan SSH port
- Check VNC credentials
- Check certificate expiration
- Verbose flag
- Add YAML report formatter

The list is not exhaustive and it is subject to changes during the development of the tool. It is ranked by a first look at the priority of the stories, but the priority can change during the advancement of the project.

4.4 Choosing the libraries

Starting from scratch, we have to take care as first point of some utility libraries that we can use to draft the basis to develop the tool: we have to choose a CLI library to interact with and parse the command-line arguments, and a logging library to show the debug messages on the console or in another location.

4.4.1 CLI library

For the CLI parameters parsing we compared a library called *urfave/cli/v2*[1] (`v2.27.1`) and another one called *spf13/cobra*[2] (`v1.8.0`). Data and evaluation refer to March 2024.

Both libraries have been recommended by the supervisor, and they are both widely used in the Go community. The evaluation between the two libraries is based on the features that they offer, the ease of use and the community support. In particular, we are interested in the ability to define flags and arguments, the ability to define commands and subcommands, the complexity of having a help page or the version of the tool.
Flags and arguments are the parameters that the user can pass to the tool to modify its behaviour, while commands are the different actions that the tool can perform. For example, the `scan` command is a command, while `--verbose` is a flag. Flag could be global, meaning that can be issued with every other command, or local, meaning that it has to be recognized only after a specified command or argument. A pattern to follow is `APPNAME COMMAND ARG --FLAG`. Finally, it's not obvious that a library supports the *context* parameter, that is a way to pass data between the functions across the application.

urfave/cli/v2 is a package for building command line apps in Go. The package provides a way to define flags and arguments, and it also provides a way to define commands. The package has friendly

[1] `https://github.com/urfave/cli`
[2] `https://github.com/spf13/cobra`

documentation and it is actively maintained by the contributors. It is quick to define the actions for each flag or option. It supports the *context* parameter. There is an upcoming major version `v3` in development, but it is not yet documented at all. The package has 21k stars on GitHub.

spf13/cobra is a similar package for building command line apps in Go. The package provides a way to define flags and arguments, and it also provides a way to define commands. The documentation points directly to the source code, which is less friendly than the other package but anyway it is very rich and always updated. It supports the ability to deprecate a flag or a command with a custom message. It supports the *context* parameter and an intelligent system of suggestions on typos, for example, if the user types a command that does not exist, the package suggests the most similar command. The package has 35k stars on GitHub and it is actively maintained.

We initialized an empty project and we tried to implement the same simple CLI with both libraries, one at time. The build size on the development laptop architecture with the first library was 4.8MB; with the second package was 5.1MB. The same working draft required 96 lines of code for the former and 110 lines of code for the latter.

In the end, we decided to use the *spf13/cobra* because of the upcoming major version of *cli/v2* which we do not know whether it will contain breaking changes or not, because the nice feature to deprecate an option, which could be useful given the youth of the project and possible future changes to the command-line interface and because of the capability to suggest the most similar command in case of a typo. The build size and the lines of code to get the same result are negligible for the use case of the tool.

4.4.2 Log library

The comparison for the logging library is between the standard library *log*[3] and another one called *rs/zerolog*[4] (`v1.32.0`). Data and evaluation refer to April 2024.

The standard library `log` is the default logging package in Go. It provides a simple way to log messages with methods for formatting output. The package is easy to use and it is well documented. It supports the ability to log messages with different levels of severity, like `Info`, `Warning` and `Error`. The package is actively maintained by the contributors.

The *rs/zerolog* lib, standing for *Zero Allocation JSON Logger*, is another package for logging messages in Go. The package provides a way to log messages to the console or to a file. It also supports the different severity levels. The authors claim that the package is faster than other libraries because it does not allocate memory for the log messages [26]. Furthermore, it implements pretty logging, a way to format the output in a human-readable way, the hooks to attach custom functions on an event, the *context* integration and much more. It is actively maintained and it has 10k stars on GitHub.

With all being said, we decided to use the *rs/zerolog* library because of the performance and the features that it offers. Also, the package was already used in Corvina, so it was a good choice to keep the same library for the scanning tool.

4.5 Scanning tool web app

The scanning tool is a command-line interface (CLI) application written in Go. The tool is designed to be run directly on the device, so it can be used by the user to check the configuration of the device and possibly receive suggestions on how to improve them. As an initial idea, the tool should be able to perform a scan of the device and output the results to the console. Thinking about the future, the scanning tool should be able to be run periodically and report the results to a central server, so the user can have a dashboard with the status of devices fleet.

The idea to consider is to make available the scanning tool through the applications store, so the user can enable the wider automation directly from the platform; the customer can create a campaign to verify the configuration of devices fleet from the same place and receive aggregated reports, with the

[3] `https://pkg.go.dev/log`

[4] `https://github.com/rs/zerolog`

goal to have a complete dashboard for the security status of the devices.

5 System Implementation

This chapter describes how the system is actually being implemented.

5.1 Go language interfaces

The Go programming language provides a powerful feature called interfaces. An interface is a collection of method signatures that a type, like a struct, can implement. Interfaces allow to define a set of methods that a type must implement to be considered an instance of that interface. This allows to write code that is more flexible, especially for testing purposes.

We setup each check as a separate interface, so that we can easily test them in isolation and also to make easier to manage the codebase and add new checks in the future, like if they are modules that can be added or removed from the system.

The modularity of a project is a key feature that allows one to easily extend the codebase and add new features. Without modularity, the codebase would be a monolithic block of code that is difficult to handle with the time and the natural growth of a codebase.

Therefore, we created different packages for the different categories of modules, like *check* and *version* plus some utility packages like *parsing*, *output* and *mapping*.

The scopes for the different packages are the following:

- *check*: it aggregates the different controls on a property of the device, like for instance the *Check correct time* or the *Check VNC credentials*;
- *version*: contains the modules to determine whether the software running on the device is up-to-date or not;
- *parsing*: it handles the parsing of the input parameters, for example the verbose flag or the list of checks to run;
- *output*: it manages the output of the checks, like the output format and the output file;
- *mapping*: contains the effective available modules list for the tool, and bind them to the underlying implementation.
- *variab* and *utils*: are utility packages that contain some common structures and functions that are used across the codebase.

The structure of a module is composed of the actual interface, named as *<module>Interface*, where *<module>* is the name of the module, a struct that implements the interface, named as *<module>Struct* and a function that returns an instance of the struct, named as *new<Module>*. All the fields of the struct are private, and the public method is the one that returns the method of the interface.

A practical example is the *CheckTime* module, which checks if the time of the device is correct. The interface is defined as follows:

```go
type checkTimeInterface interface {
    checkTime(context.Context) variab.Result
    getNtpHmiConfig(context.Context) (*hmiDateTimeDto, error)
    getRealTimeWrapper() (*time.Time, error)
}
```

The structure is defined as follows:

```
type checkTimeStruct struct {
    delta           time.Duration
    getCliTime      func() (*string, error)
    getRealTime     func(string, string, string, time.Duration) (*time.Time, error)
    [omitted]
    timeout         time.Duration
}
```

There is an instance of the struct that is returned by the function *newCheckTime*:

```
func newCheckTime() checkTimeInterface {
    return &checkTimeStruct{
        delta:          DELTA_DURATION,
        getCliTime:     getCliTime,
        getRealTime:    getRealTime,
        [omitted]
        timeout:        utils.TIMEOUT,
    }
}
```

and the public method is the one that returns the method of the interface:

```
var checkTime checkTimeInterface = newCheckTime()

func CheckTime(ctx context.Context) variab.Result {
    return checkTime.checkTime(ctx)
}
```

The interface has a method that checks if the time of the device is correct, and it always returns a *variab.Result* struct that contains the result of the check.

The *Result* structure of the package *variab* is a common structure that is used to return the result of the checks. It contains a concise message for the output, an optional recommendation to solve the issue, a severity level ranging from *none* to *high* and an operation status, where *OK* means that the module completed the task without noticing any vulnerability, *ISSUE* means that the module contains a potential weakness, *EXECUTION_ERROR* in the case of an unhandled exception, *UNKNOWN* otherwise. This structure is used by all the modules to return the result of a module. Any exception is handled in order to guarantee the robustness of the tool in case of any failure, allowing the user to always obtain a valid output.

Listing 5.1: Result struct

```
{
  "message": string,
  "recommendation": string,
  "severity": none|low|moderate|high,
  "status": OK|ISSUE|EXECUTION_ERROR|UNKNOWN
}
```

This composition is common to all the modules, and it allows us to easily test the modules in isolation, by mocking the dependencies of the struct. This is done by creating a new struct that implements the same interface, but with mocked methods, and then injecting this struct in the test. This way, we can test the module without having to actually run the code that is being mocked. This is a powerful feature of the Go language that allows to write more robust and maintainable code. More to follow about the testing phase in the next chapter.

5.2 First sprint

The first sprint has the goal of implementing the first set of modules, as expressed in the sprint planning. We are going to detail the implementation of every story from the sprint backlog.

5.2.1 CheckTime module

The *CheckTime* module checks if the datetime of the device is correct. This is a fundamental thing to verify because the issues with the devices not connecting to the internet and to the cloud are most of the time due to the incorrect datetime, leading to an error with the SSL certificate. In fact, if the date is not correct, the SSL certificate could result as expired or not yet valid, warning the user that the connection is not secure.

It does so by getting the time from the device and the real-time from a remote server, and then comparing the two times. If the difference between the two times is greater than a certain threshold, expressed in a range of few minutes, then the module returns an *ISSUE* status, otherwise it returns an *OK* status. Of course, this module works with an active internet connection only; if the device is not connected to the internet, then the module returns a documented execution error.

To perform the execution, it runs the following steps:

- first, it gets the UNIX timestamp, an integer number representing the elapsed seconds since 1st January 1970, by calling the native `time.Now().Unix()` language function. This is the datetime of the device;
- then it calls a remote endpoint that returns the real datetime;
- it parses the two datetimes;
- finally, it compares their difference and returns the result.

If any of the steps fails, the module returns the execution error status.

5.2.2 JSON output

Every module returns a structure representing the result of the execution, a brief message and a recommendation to solve the potential issue. The modules runner collects all the results from the modules in a *Report* type variable, that is defined as a map of maps, in order to have the direct mapping of the result of the module under the category and the name of the module.

The resulting type is therefore given to the *output* package, which is responsible for the desired destination format. The default formatter is JSON, but it can be easily extended to support other formats like YAML by importing or implementing the corresponding marshaler, that is a function that transforms the encoding of an *any* value to the desired format.

By default, the output is printed to the standard output of the terminal, but the destination can be redirected to a file by using the built-in method with the `-d` flag.

The output can be printed in a pretty format by using the `--pretty` flag, useful for human readability. Anyway, the predefined formatting is minified. The formatter can be chosen by using the `-o` flag, followed by the desired format. For example, to output the result in YAML format in the terminal the command is `./scantool scan -o yaml`, or `./scantool scan -o yaml -d output.yaml` to redirect the output to a file.

The list of the active modules is available with the command `./scantool list`, printed in the standard output of the terminal, one for line.

For example, given the execution of the datetime module only, issued with the command `./scantool scan --list "check.time" --pretty` with no further parameters, the output structure would be as the following:

Listing 5.2: Output of the datetime module

```
{
"check": {
  "time": {
    "message": "The current datetime is synced",
    "recommendation": "No action required",
    "severity": "none",
    "status": "OK"
  }
}
```

5.2.3 Cross build script

The Go programming language provides a powerful feature called cross-compilation. This allows to build the binary for a different architecture than the one of the machine that is building it. This is useful to build the binary for a different architecture, like ARM, that is the architecture of some of the devices that we are going to use for debugging purposes.

Go provides the built-in command `go build <file>` that builds the binary for the current architecture, but it also provides the `GOOS` and `GOARCH` environment variables that can be set to the desired values to build the binary for a different architecture.

The former is the operating system, and the latter is the architecture. Possible values useful for the project are `linux` for the operating system and *arm*, *arm64* and *amd64* for the architecture. All the available values can be found in the official documentation of the Go programming language [25].

By default, the binary includes debug symbols, that are useful for debugging purposes, but they are not necessary for the production environment. They increase the size of the binary to several megabytes, and they are not needed by the end user. In particular, we add two flags to the build command: [18, 19]

- `-s` turns off the generation of the Go symbol table;
- `-w` turns off debugging information, not being able to trace the resulting binary with the `gdb` debugger.

In the context of the automatic pipeline that builds the binary for the different architectures, we also inject the version number of the binary in the build command, by using the `-X importpath.name=value` flag. This flag allows to set the value of the string variable in *importpath* named *name* to *value* [18].

Furthermore, we also explicitly set the environment variable `CGO_ENABLED=0` to disable the use of the machine C compiler. Anyway, that is disabled by default when cross-compiling and the go builder chooses the appropriate compiler for the target architecture [13].

Finally, we set the `-trimpath` flag to remove the absolute path of the source files from the binary, in order to make the binary reproducible. This is a good practice to follow, as it does not leak any information about the source code and the environment where the binary was built [24].

The resulting binary for each architecture is placed in a tree of directories, where the root directory is the version of the binary, and the subdirectories are the architectures. The filename of the binary is the same as the name of the project, that is `scantool`.

The final build command is the following:

Listing 5.3: Go tool cross-build command

```
env GOOS=$GOOS GOARCH=$GOARCH CGO_ENABLED=0 go build -trimpath -ldflags="-s -w -X ’main.VERSION=
    ↪ $version’" -o $output_base_dir/$version/$arch/$filename main.go
```

The difference between the size of an unoptimized build and an optimized one is quite relevant. Taking as example the architecture *arm64* on the *Linux* operating system, the former takes 12MB and the latter takes 7.7MB. On industrial devices, such size difference matters a lot in terms of storage and bandwidth to download it, plus the amount of memory that it takes when running. We always have to consider that the devices have limited resources, and we have to optimize the software as much as possible.

5.2.4 Scan OS version

The custom board of the industrial devices runs a custom Linux distribution, that is based on the Yocto Project. The updates are managed by Exor internal department, with an internal version manager. The updates are not automatic, and they have to be requested by the customer.

This module monitors whether the operating system is at its latest version or not. Given that the estimated work to implement it as a single block is not trivial, we can split it into two different specializations: the first one is the one that checks the version of the operating system by retrieving the build date, and the second one is the one that checks the version of the operating system by reading the online repository.

The Device Settings API, a set of endpoints used to retrieve the information about the status of the device and to potentially change them, provides among other things the version number and the build date of the operating system. There is the `/api/v1/management/mainos` endpoint that returns a JSON object structured as follows:

```
{
  "version": "4.2.323",
  "date": "2024-04-17T06:00:00.000Z"
}
```

The module is implemented by the *versionOsInterface* and *versionOsStruct* structure. The interface is defined as:

```
type versionOsInterface interface {
  getOsInfo(context.Context) (*versionOsApiDto, error)
  checkOsVersion(context.Context) variab.Result
}
```

and the structure is formed of the following fields:

```
type versionOsStruct struct {
  xMonthAgo     int
  [omitted]
}
```

The *xMonthAgo* field is the threshold expressed in a range of few months. The *getOsInfo* method retrieves the information about the operating system from the Device Settings API, and the *checkOsVersion* method actually performs the comparison.

Offline date based

The first specialization of the module checks the version of the operating system by reading the build date of the system. This is done by retrieving the build date of the system from the Device Settings API, that is the *date* field of the API response. The date is then compared with the current date, and if the difference is greater than a certain threshold then the module returns an issue, otherwise it returns an OK status through the *Result* structure.

Online version based

This specialization should compare the local version of the operating system with the latest version available somewhere online. Since Exor does not provide a repository for the updates, this module cannot be implemented yet. The idea is to have a list of the latest versions for the different device models and to compare the local version with the latest one. If the local version is not updated, then the module returns an issue, otherwise it returns an OK status. This is a future work that can be implemented when Exor will start to provide an accessible repository for the updates. As a matter of fact, this story has been suspended in the context of our project.

5.2.5 Default BSP credentials

Setting common passwords is a bad practice, as they let an attacker to potentially gain unauthorized access to the management of the device.

The devices ship with some default credentials for the two users that are available on the system, named *admin* and *user*. The credentials are required to interact with the Device Settings.

The module checks whether the default credentials are still in use by trying to login with a pre-defined set of passwords for the two users. The module performs $2 * |harcodedPasswords|$ HEAD requests to an arbitrary endpoint of the Device Settings API and it checks the status code of the response. If one of the responses' status codes is *200 Success*, meaning that the authorization has been granted and therefore username and password match, then the module returns an issue with a high severity, otherwise it returns an OK status.

This security control is possible without stumbling into the device's built-in authentication rate-limiting because the total number of requests is below the threshold of the requests per minute that the device accepts before temporarily blocking the access. This built-in mechanism is derived from the `IEC 62443 3-3` standard, precisely from the `SR 1.11`.

SR 1.11 - Unsuccessful login attempts: The control system shall provide the capability to enforce a limit of a configurable number of consecutive invalid access attempts by any user (human, software process or device) during a configurable time period.

The module is implemented by the *checkCredentialsSystemSettingsInterface* and *checkCredentialsSystemSettingsStruct* structure. The interface is defined as:

```
type checkCredentialsSystemSettingsInterface interface {
    checkCredentialsSystemSettings(context.Context) variab.Result
}
```

and the structure is made of the following fields:

```
type checkCredentialsSystemSettingsStruct struct {
  [omitted]
  hmiHeadRequest func(string, time.Duration, *string, *string) (*http.Response,error)
  users          []string
  passwords      []string
}
```

The *hmiHeadRequest* function is a utility function that performs a HEAD request to the given URL with a certain user and password. The *users* and *passwords* fields are the list of users and passwords to try to login with.

5.2.6 Scan SSH port

The SSH protocol, which stands for *Secure Shell*, is a cryptographic network protocol that allows to securely connect to a remote device. It is widely used in the industry to connect to the devices for the purposes of debugging, maintenance and monitoring. The SSH protocol is based on the client-server model, where the client connects to the server and authenticates itself by providing a username and a password or a public key. The server then verifies the credentials and grants access to the client if they are correct.

The SSH protocol is based on the TCP protocol, and it uses port 22 by default. The server, that is the industrial device, listens on port 22 for incoming connections. Because of the widespread adoption of the SSH protocol, that port is a common target for attackers who try to gain unauthorized access to the devices.

This story is a specialization of a wider set of stories that check the status of the different services running on the device. The idea is to check whether the status of the service reported by the Device Settings API matches with the actual status of the service, and if enabled to warn that a port is exposed and therefore to be informed about the potential risks.

The module is implemented by the *checkServicesInterface* and *checkServicesStruct* structure. The interface is defined as:

```
type checkServicesInterface interface {
  checkServices(context.Context, string) variab.Result
  hmiConfig() (*hmiServicesDto, error)
}
```

and the structure is composed of the following fields:

```
type checkServicesStruct struct {
  isPortOpen    func(uint, time.Duration) (bool, error)
  [omitted]
}
```

The *hmiConfig* function retrieves the information about services from the Device Settings API, and the *checkServices* function actually performs the check. The API endpoint is located at `/api/v1/services` and it returns a JSON object structured as a list of objects for each service, where each object contains its id, the name and the status.

The *isPortOpen* function is a utility function that checks if the port is open on the device. It works by trying to open a socket on the port defined by a predefined map that contains a list of default ports for each of the services. Depending on the protocol, the function tries to connect to each of the ports and returns true if the connection is successful, otherwise it returns false.

By combining the two functions, the module checks if the SSH port status reported by the Device Settings API matches the actual status of the port.

5.2.7 Check VNC credentials

As part of the services that are running on the device, one of them is the VNC service. VNC, which stands for *Virtual Network Computing*, is a cross-platform graphical desktop sharing system to remotely control another computer. It transmits the screen of the remote device to the client, and it allows to interact with it by sending the mouse and keyboard events. It uses the RFB - *Remote Framebuffer* - protocol that governs the format of the data that passes between the client and server within the VNC system. The protocol itself does not require a mandatory authentication method, but it supports many of them, like a password or a certificate.

If the service is not properly configured and does not require a password to connect, then an attacker could potentially gain unauthorized direct access to the graphical interface and perform any operation that a legitimate user could do.

The module checks whether the VNC service is configured with a password or not. It does so by retrieving the configuration, in particular the port on which the service is running, from the Device Settings API, and then implementing the protocol itself to get the required authentication method. If the authentication method is not set, then the module returns an issue with a high severity, otherwise it returns an OK status.

RFB protocol

The RFB protocol is defined in the RFC 6143 [42]. It works using the TCP protocol for the transport layer. The server, that is the industrial device, listens on the default port 5900 or on a custom one defined by the customer in the Device Settings.

The protocol expects the following steps:

1. the client connect to the server and the server sends the protocol version; the *ProtocolVersion* message consists of 12 bytes interpreted as a string of ASCII characters in the format `RFB xxx.yyy\n` where `xxx` and `yyy` are the major and minor version numbers, left-padded with zeros: the only published protocol versions at this time are 3.3, 3.7, and 3.8. Other version numbers are reported by some servers and clients but should be interpreted as 3.3 since they do not implement the different handshake in 3.7 or 3.8.

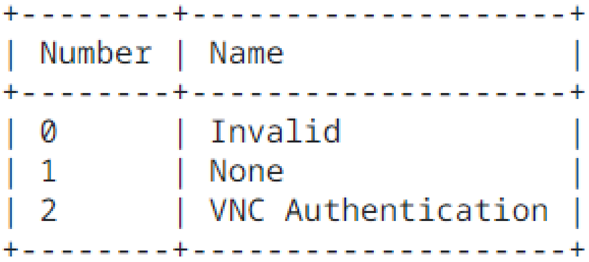
2. the client replies to the server agreeing with its supported version, which must be less or equal to the one of the server;

   - if the version is 3.3, then the server directly replies with a single integer number representing the security type;
   - otherwise, the server replies with a list of numbers representing the security types that the server supports;

3. at this point, instead of proceeding with the protocol, we close the connection and we return the security type.

If the security type is equal to the number 1, the server does not require any authentication. If the number is equal to 0, this is not expected and probably something went wrong in the implementation of the protocol. Both cases return as result a high severity issue.
If the security type is greater or equal to number 2, then the server correctly requires some access control. Note that we said greater than two and not strictly equal to 2 because other security types exist but are not publicly documented.

The module is implemented by the *checkVncCredentialsInterface* and *checkVncCredentialsStruct* structure. The interface is defined as:



Number	Name
0	Invalid
1	None
2	VNC Authentication

Figure 5.1: RFC 6143: security types

```go
type checkCredentialsVNCInterface interface {
    checkCredentialsVNC(context.Context) variab.Result
}
```

and the structure contains the following fields:

```go
type checkCredentialsVNCStruct struct {
  [omitted]
  vncConn            func(string, time.Duration) (bool, net.Conn)
  vncProtocolVersion func(net.Conn) ([]byte, error)
  vncAuthType3       func(net.Conn) (uint32, error)
  vncAuthType7_8     func(net.Conn) (uint32, error)
}
```

The *vncConn* function creates the connection to the VNC server, the *vncProtocolVersion* function retrieves the protocol version, the *vncAuthType3* function retrieves the authentication method for version 3 of the protocol, and the *vncAuthType7_8* function retrieves the authentication method for the versions 7 and 8 of the protocol.

The *checkCredentialsVNC* function performs the check by combining the functions above.

5.2.8 Check certificate expiration

The device uses the HTTPS protocol to expose the Device Settings UI and API and the optional web page of the running project. The HTTPS protocol is a secure version of the HTTP protocol that uses the SSL/TLS protocol to encrypt the data that passes between the client and the server. The SSL/TLS protocol uses a certificate to establish the identity of the server and to encrypt the data that passes between the client and the server. This behaviour is needed to prevent an attacker from intercepting the data that passes between the client and the server.

The certificate is issued by the Exor authority and it is valid for a certain period of time, usually years. A customer can upload a custom certificate to the device. The certificate is stored in the device.

The module checks whether the certificate is expired or not. Go provides built-in packages called `encoding/pem` and `crypto/x509` that allow to parse the certificate and to extract the expiration date. The former implements the PEM data encoding, which originated in *Privacy Enhanced Mail*. The most common use of PEM encoding today is in TLS keys and certificates [20]. The latter allows parsing and generating certificates, certificate signing requests, certificate revocation lists, and encoded public and private keys. It also provides a certificate verifier with a chain builder [23].

In particular, the certificate is read as bytes from the file system, then it is parsed by the `pem.Decode` function that returns a `pem.Block` structure. The `Block` structure contains the type of the block and the bytes of the block. The bytes are then passed to the `x509.ParseCertificate` function that returns a `x509.Certificate` structure. The `Certificate` structure also contains the expiration date of the certificate.

The expiration date is then compared with the current date: if the expiration date happens in less than 30 days, the module returns an issue. If the valid days left are less than 30, the severity is low; if they are less than 15, the severity is moderate; if they are less than 1, the severity is high. - The implementation of the module is defined by the *checkCertificatesInterface* and *checkCertificatesStruct* structure. The interface looks like:

```go
type checkCerticatesInterface interface {
    checkCertificates(context.Context) variab.Result
}
```

and the structure is made up of the following fields:

```go
type checkCertificatesStruct struct {
    hmiCertPath string
    loadCertFile func(string) ([]byte, error)
}
```

The *hmiCertPath* field is the path to the certificate file, and the *loadCertFile* function reads the certificate file from the file system. The *checkCertificates* function performs the check by combining the functions above.

5.2.9 Verbose flag

This flag is a common one that is used in the command line interface of the tool to increase the verbosity of the output. By default, the output is minimal and it contains only the result of the checks, but by using the `--verbose` flag the output is increased to include more information about the execution of the modules.

The flag is implemented by taking advantage of the hooks provided by *spf13/cobra*, that is the package that we use to implement the command line interface. The `cobra.Command` structure provides a `PersistentPreRun` callback that is called before the execution. We use this callback to set the verbosity level of the output for the *zerolog* package with the `zerolog.SetGlobalLevel` function.

This flag is defined as persistent, meaning that it is available for every command and subcommand and it is only defined once.

5.2.10 Add YAML report formatter

In addition to the JSON output, we also add the YAML output format. The YAML format is a human-readable data serialization standard that can be used in conjunction with all programming languages and is often used to write configuration files. The YAML format is more human-readable than the JSON format. It is not native in the Go programming language, but there is an official porting available at the `gopkg.in/yaml.v3` package.

As described in Section 5.2.2, the output package is responsible for the output of the results in the desired format. The default format is JSON, but it can be easily extended to support other formats like YAML by importing the corresponding marshaler.

In this case, we add the `yaml.Marshal` function to the output package the *Report* structure. If the user specifies the `-o yaml` flag, then the output is formatted in YAML to the desired destination.

The high modularity of the Go language and of the project allows to easily extend the output formats by simply adding few lines of code. This is a powerful feature of the Go language that allows to write more maintainable code.

5.3 First sprint retrospective

The first sprint has been completed successfully. All the expected modules but one have been implemented. For each of the tests, we have written comprehensive unit tests that run on demand. The implementation of the modules has been straightforward. The use of the interfaces has allowed to easily test the modules in isolation, by mocking the dependencies of the struct.

The only module that has not been implemented is the one that checks the version of the operating system by reading the online repository. This is because there is not a repository for the updates, and therefore we cannot implement the module yet. This has been marked as future work.

Given that the first sprint has been completed successfully, we can move on to the second sprint. We did a meeting to show the results of the sprint to the office and to discuss the next steps. The feedback has been positive, and the team is happy with the results. The next sprint focuses on the implementation of more server side utilities and some more modules.

5.4 Second sprint

The second sprint has the goal of implementing the following set of modules:

- Create cloud web app
- Implement proxy network call
- Automatic build and deployment
- Enable to download latest version of the tool
- Support internationalization

These are new features useful for the future adoption of the tool.

The first two are server side utilities that are needed to prepare the infrastructure for future development of the tool in terms of strictly network policies and user billing. In fact, Corvina is interested in the possibility to offer the tool as a service to the customers, and therefore it is necessary to have a server side application in order to handle the functioning. Furthermore, as previously said, industrial network policies are very strict and usually they do not allow to connect directly to the internet, and therefore a proxy is needed to perform the network calls.
For example, given the case of a version check, if the remote location on which the version is stored happens to be on a third-party endpoint, like for an external dependency, each network administrator for each customer should manually allow the connection to that location, and beforehand the company should provide the list of the locations to allow and keep it updated. In order to avoid this, Corvina provides a dedicated proxy server that is the only one that connects to the internet. The tool can then connect to the proxy server. This way, network administrators only have to allow the connection to the proxy server, that is already expected for the standard operation of the device.

The second and third points are about the automatic build and deployment of the tool. The company is interested in the possibility to automatically build the tool for the different architectures and to deploy it on the devices. This is also useful for testing purposes, as the build is preceded by the execution of the tests and also for the continuous integration and continuous deployment of the tool.

The latest point aims to provide internationalization, that is the support for different languages. The company is interested in the possibility to offer the tool in different languages, in order to make it more accessible to customers.

5.4.1 Create cloud web app

Recalling Section 3.6, the cloud app is an entire runtime made by a frontend and a backend that is visible through the platform website. The frontend is a VueJS application and the backend uses NestJS as the framework for handling the controllers, the models and the logic. Both the frontend and the backend are written in TypeScript and both implement the authentication with the platform, as it is really integrated with the platform itself and cannot be used in a standalone mode.

The skeleton of the application is generated by a public utility developed by Corvina, which is available on the public *npm* registry. This utility generates the boilerplate code for the frontend and the backend, with many features already implemented, like the authentication with the platform, basic routing, controllers and models needed to complete the first installation on the platform.

The app also provides a running configuration of *PostgreSQL* database that is used to store its data. The database is also deployed on the Kubernetes cluster.

The application has to be deployed on a Kubernetes cluster. The code also contains the required chart files, customized for the application, that are used by the Helm package manager to deploy the application on the cluster, and the Dockerfile that is used to build the Docker image of the application.

Once the deployment is live on the cluster, the application can be linked to the platform store by using the public REST API of the platform. Doing so, the application is visible on the platform website and it can be accessed by the customers.

5.4.2 Implement proxy network call

As said, usually the industrial networks attached to the devices are very strict and do not allow to communicate with every host on the internet. This is a security measure that is taken to reduce the risks of exposing relatively weak devices. In fact, according to internal statistics, unfortunately the devices are usually not updated frequently by the customers and they are not patched for the latest vulnerabilities. The goal of the tool described in this thesis is also to make aware the customers of a potential vulnerability that could be exploited by an attacker and new available updates. In order to do so, the tool has to communicate with the internet to check the latest version of the software. How to do that given all the constraints?

The solution we thought is to provide a proxy server that is the only one that connects to the final endpoints. The tool can then connect to the proxy server. This way, network administrators only have to allow the connection to the proxy server, that is already expected for the standard operation of the device.

The proxy server is managed by a controller on the cloud app backend. The controller is responsible for handling the incoming requests from the tool and forwarding them to the respective methods. The controller is also responsible for handling the authentication with the platform. It is written in TypeScript and it uses the NestJS framework.

The proxy controller is a class that is decorated with the `@Controller` decorator, that is provided by the NestJS framework, with the parameter that is the base path of the controller. All the requests starting with that path are handled by the controller. A decorator is a special declaration that can be attached to a class, method or property, and it is used to modify their behaviour. The class is also decorated with `@ApiTags`, needed to generate the OpenAPI documentation, and by a `@UseInterceptors` directive with the *LoggingInterceptor* custom one that logs the incoming requests.
The class contains the declaration of the services that are used by the controller to perform the actual operations. Services are injected into the constructor of the class, aka they are already instantiated when the class is defined. The class also contains the methods that are the actual endpoints of the controller. The methods are decorated with the `@Get`, `@Post`, `@Put` or `@Delete` decorators, depending on the HTTP method that they handle. The methods also contain the logic to handle the incoming requests and to return the responses. The responses are returned in the form of specific *DTO*s, which stands for *data transfer object* and it is an object that carries data between processes, automatically serialized by the NestJS framework to the JSON format.

The decorators starting with `@Api` are used to generate the OpenAPI documentation. The `@ApiTags` decorator is used to specify the category of the endpoint and the `@ApiHeader` decorator is used to specify the header parameters that are required. The `@ApiBearerAuth` decorator is used to specify that the endpoint requires a bearer token to be accessed. The `@UseGuards` decorator is used to specify the guards that are used to protect the endpoint from unauthorized access. In our case, we use the two custom guards `AuthGuardFromHeaders` and `InstallationRoleGuard` to respectively obtain the bearer token and the expected parameters from the headers and to confirm that the request comes from a valid app installation.

An example of the proxy controller class is shown in Listing 5.4.

Listing 5.4: Proxy controller class

```
[... imports ...]

@Controller('v1/proxy')
@ApiTags('Proxy')
@ApiHeader({ name: 'x-instance-id', required: true })
@ApiHeader({ name: 'x-organization-id', required: true })
@ApiHeader({ name: 'x-device-id', required: true })
@ApiBearerAuth()
@UseInterceptors(LoggingInterceptor)
@UseGuards(AuthGuardFromHeaders, InstallationRoleGuard)
export class ProxyController {
  private readonly _logger: Logger;

  private readonly _proxyTimeService: IProxyTimeService;

  constructor(
    @Inject('IProxyTimeService') proxyTimeService: IProxyTimeService,
    logger: Logger
  ) {
    this._proxyTimeService = proxyTimeService;
    this._logger = logger;
  }

  @Get('time')
  @UseGuards(RateLimiter({ max: 1 }))
  async time() {
    this._logger.info('time');

    const filteredApi = await this._proxyTimeService.ntp();

    return filteredApi;
  }
}
```

The *proxyTimeService* service is the one that actually retrieves the real-time from a public NTP server. NTP stands for *Network Time Protocol* and it is a protocol used to synchronize the clocks of computer systems over a network. The service is injected in the constructor of the controller, which means that there is no need to explicitly instantiate it. The service is defined by an interface that contains the methods that are needed to perform the operations. The service is then implemented by a class that implements the interface and that contains the actual logic to perform the operations. The service is responsible for handling the business logic of the application, and it is used by the controller to perform the actual operations.

An example of the proxy service class is shown in Listing 5.5.

Listing 5.5: Proxy service class

```
[... imports ...]

export class NtpTimeDto {
  time: Date;

  constructor(timestamp: Date) {
    this.time = timestamp;
  }
}

export interface IProxyTimeService {
  ntp(): Promise<NtpTimeDto>;
}

@Injectable()
export class ProxyTimeService implements IProxyTimeService {
  private readonly _logger: Logger;

  private readonly _ntpServerUrl = ’time.cloudflare.com’;
  private readonly _ntpServerTimeout = 5000;

  constructor(logger: Logger) {
    this._logger = logger;
  }

  private async getNtpTime(url: string, timeout: number): Promise<NTPPacket> {
    this._logger.info(‘NTP client call to url ’${url}’‘);
    const ntpClient: NTP = new NTP(url, 123, { timeout });
    return ntpClient.syncTime();
  }

  async ntp(): Promise<NtpTimeDto> {
    this._logger.info(’ntp’);

    const ntpTime = await this.getNtpTime(this._ntpServerUrl, this._ntpServerTimeout);

    return new NtpTimeDto(ntpTime.time);
  }
}
```

The interface contains the public definition for the `ntp` method, then implemented by the `ProxyTimeService` class. The *NtpTimeDto* class is a data transfer object that is used to carry the data between the service and the controller.

5.4.3 Automatic build and deployment

The continuous integration and continuous deployment of the tool is a crucial part of the development process: it allows to automatically build the tool for the different architectures and deploy them to different environments. It is also useful for testing purposes, as the build is preceded by the execution of the tests.

The environments need the following set of images:

- the tool written in Go language is built for three different architectures: *amd64*, *arm* and *arm64*;
- the frontend and the backend are built into two different sets of images, each exported for both *amd64* and *arm64*;

About the tool, different images are needed for the devices running the *arm** architectures. Instead, the frontend and the backend images are built for the cluster environment, that is the *amd64* architecture on the Google Cloud Platform Kubernetes cluster, and the development runtime where the laptops are based both on *amd64* and *arm64* architecture. This way, each developer can run its own instance of the cluster on its laptop and directly test the application as it would be deployed on the real cluster.

The automatic build and deployment is managed by the Google Cloud Build service. It is directly integrated with the project's repositories and it is triggered by the push of the code on a specific tag or on the master branch. The service is programmed by a configuration file, called *Dockerfile*, which contains all the steps that are needed to build the images. The images are then pushed to a remote registry and then they are automatically deployed to the cluster. Instead, the built tool binaries are stored in a public storage bucket. The bucket always contains the reference to the latest master version and all the previous tagged versions. We decided to store the binaries in a public storage bucket because of the simplicity of the service for our internship project.

Therefore, the storage looks like a tree of directories, where the root directory is the version of the binary, and the subdirectories are the architectures. The filename of the final binary is the name of the tool. The tree is structured as follows:

Listing 5.6: Storage bucket tree of directories

```
.
|-- 0.1.0
|   |-- linux-amd64
|   |   |-- scantool
|   |-- linux-arm
|   |   |-- scantool
|   |-- linux-arm64
|       |-- scantool
|-- 0.1.1
|   |-- linux-amd64
|   |   |-- scantool
|   |-- linux-arm
|   |   |-- scantool
|   |-- linux-arm64
|       |-- scantool
|-- latest
    |-- linux-amd64
    |   |-- scantool
    |-- linux-arm
    |   |-- scantool
    |-- linux-arm64
        |-- scantool
```

Doing so, the developers can focus on the development of the tool and they do not have to worry about the building and deploying process.

5.4.4 Enable to download latest version of the tool

Recalling the tool, it is a command line software that is used to scan the device for potential misconfigurations on its settings. The tool is run on the terminal of the device, and therefore it is required to obtain the executable file. To enable the customers to download the latest version of the tool, we provide a public endpoint that returns the latest version for the different architectures.

The cloud app frontend is powered by these endpoints to assist the testers and the customers in downloading the latest version of the tool. The frontend interface is a VueJS application, part of the cloud app. For the internship purposes, we shipped a simple frontend composed of a dropdown that lists the available architectures, a *download* button that gets binary and a *copy command* to copy the *curl*[1] command, to simplify the download process by not involving the copy between the local host and the remote device. This way, the customer can easily copy and paste the command and execute it on the terminal of the device. Figure 5.2 shows the frontend.

The endpoint is a public REST API that is implemented by a controller on the cloud app backend. The controller is responsible for handling the incoming requests from the frontend and forwarding them to the respective methods. The controller is also responsible for handling the authentication with the

[1] `https://curl.se/`

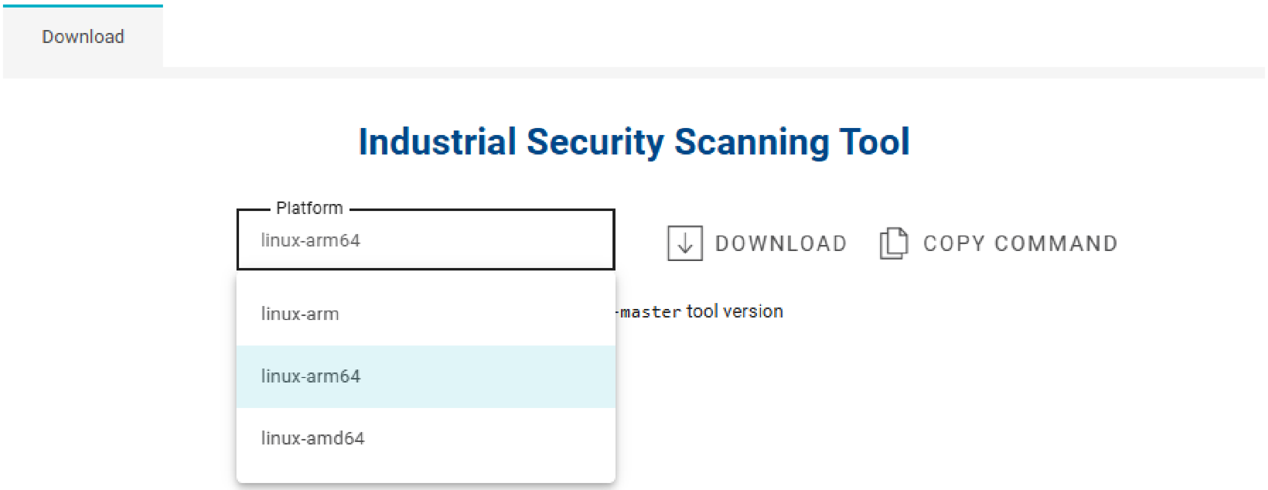


Figure 5.2: Cloud app frontend interface

platform. The controller asks the related service model to actually list the available architectures and returns the final download link.

The endpoint is available at the `/v1/cli-release` base path; in particular, the `/platforms` path returns a dynamic list of the available architectures at that time, and the `/:version/:platform` path returns the actual download link for the specific version and platform. The download link is a direct link to the storage bucket that contains the binaries.

Notice that, although the app is accessible using a private account the download link is public: we do not consider this a security issue because of the initial stage of the project and the limited access to the preview version of the app. In fact, only the company testers can access it at this time. Future work will implement a proper authentication system to access the download link.

5.4.5 Support internationalization

The support for different languages is a fundamental step for a company with a global presence. The company is interested in the possibility to offer the tool in different languages, in order to make it more accessible to customers.

In order to support internationalization, also known as *i18n* because of the 18 letters between the *i* and the *n*, we use the `vue-i18n` package to translate the cloud app. The package is a plugin for VueJS that provides the i18n features to the application and it is available on the public *npm* registry. The package allows to define the translations in a JSON file and to use them in the application by using the `$t` function. The package also provides the ability to switch the language at runtime.

Also the tool is translated into different languages, primarily in English and Italian. The translations are again stored in different YAML files, one for each language, and they are loaded at runtime by the tool. The tool uses the `go-i18n` package. We did some wrapper functions to make the package similar to the way it works in the application by exposing a `t` function used to return the translation of a key. We decided to use a YAML translation file because it is even easier to read and write than a JSON file.

Both the translation methods support the variable replacement, that is the ability to replace a placeholder in the translation with a value. This is useful for example to translate a sentence that contains a variable part, like a number or a name. Also, they support pluralization, that is the ability to translate a sentence in different ways depending on the number of items. A clear example is the translation of the word *item* in English, which is translated in two different ways depending on the number of items: *item* for one item and *items* for zero or more than one item. The pluralization is done by using the `one`, `other` and `few` keys in the translation file. The `one` key is used for the singular form, the `other` key is used for the plural form, and the `few` key is used for the few form. The package automatically selects the correct form based on the number of items passed as parameter.

Usually, the names of these functions are very short to reduce the verbosity in the code, given that they are used really often in the code.

5.5 Second sprint retrospective

With this last task done, the second sprint has been completed successfully. All the expected modules have been implemented. The implementation of the modules has been straightforward. Compared to the first sprint, we adjusted the estimation of the complexity and the time needed to develop the new features and we have been able to complete the sprint in time. This is the end of the internship period. Exor and Corvina are satisfied by our job and they look forward to further development.

6 Testing

This chapter describes the testing of the system, a fundamental part of the development process. It is important to perform testing to ensure that it works as expected. The testing process is divided into two parts: unit testing and system testing. Unit testing is the process of testing individual components of the system while system testing is the process of testing the system as a whole.

6.1 Automated testing

There are several reasons why testing is important. We can group them into four categories:

- First of all, testing helps to prevent regressions. A regression is a bug that is introduced when a feature is added or especially modified in the system. It helps to prevent them by testing the system after each change.
  The unit tests can be defined after the definition of the project, but before writing any code implementation. This way, the developer defines the expected behaviour of the system before writing the code. It is an iterative approach combining unit test creation, programming, and refactoring. This method is called Test-Driven Development (aka TDD), and it is also originated from the Agile manifesto principles [3]. The TDD approach is shown in Figure 6.1;

- Documentation: testing helps to enrich the documentation of the source functions. The test cases are a good way to understand the expected behaviour of the system. They can be used as examples to understand how the system works. Compared to separate documentation, the test cases are obviously always updated with the source code, avoiding possible inconsistencies due to the missing update of the documentation after a refactor;

- The testing process speeds up the development process. It is faster to test the system after each change than to test the system at the end of the development process. It is also easier to find the bugs when they are introduced. The developer can easily identify the cause of the bug and fix it. This way, the developer can focus on the development process rather than the debugging process;

- Knowledge: testing helps to master the programming language you use. It also helps to understand the best practices of the programming language.

6.1.1 Go

Go is a very versatile programming language. In addition to the description available in Section 3.4 about the characteristics of the language, the testing framework is integrated into the language and provides a simple and efficient way to write tests for functions, methods and interfaces. The testing framework also supports the generation of code coverage reports to identify untested code paths and improve the overall test coverage. Furthermore, it supports the use of table-driven tests, which allow the developer to define test cases in a structured format and iterate over them to execute the tests and also it supports fuzzing, a technique used to discover vulnerabilities in software by providing random or invalid inputs to the program [22].

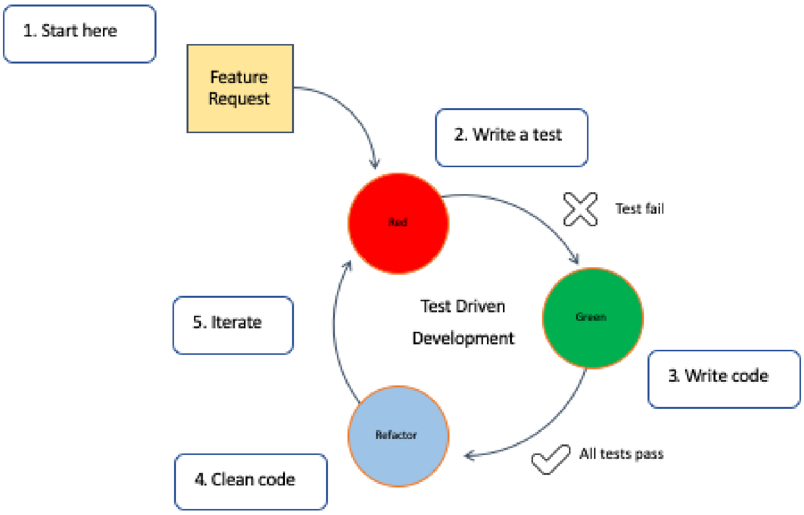


Figure 6.1: Test Driven Development workflow. Image by `developer.ibm.com`

Given that the package is a native one, the standard library is enough. Also, it is not included in the compiled binary, enabling the developer to take advantage of the framework without increasing the size of the final binary, where indeed the testing code is not needed at all.

It supports both *Black Box* and *White Box* testing. Black Box testing is a testing technique that focuses on the functionality of the software, used to validate the software against the requirements. White Box, instead, is a testing technique that focuses on the internal structure of the software, used to validate the software against the implementation.

A test file ends with the suffix `"_test.go"`. The code in the file can be part of the same package of the source file, in order to have visibility on all the private methods and variables, therefore referring to White Box Testing, or in a corresponding package with the *_test* suffix in order to test the real behaviour, known as Black Box Testing. The test file contains the test functions starting with the prefix "Test" followed by the name of the function to be tested.

To be valid, the testing package must be imported by all the test functions; the main exposed types are:

- `func TestMain(m *testing.M)`: sometimes it is needed to do extra setup before starting the testing; this function is the entry point and runs before tests. The function takes a pointer to an object which provides methods to control the flow;
- `func TestXxx(t *testing.T)`: the function takes a pointer to an argument providing methods to report the test results;
- `func TestXxx(f *testing.F)`: it is used to perform a fuzz test. If the test fails, we again can use the `*testing.T` object to report the failure.
- `func BenchmarkXxx(b *testing.B)`: it is used to perform a benchmark test.

The Go test runner scans all the test files in the project, runs them based on the given configuration and reports the results to the standard output. The test runner can be configured to run the tests in parallel, in a specific order, or to run only the tests that match a specific pattern. It can also be configured to generate code coverage reports, which can be used to identify untested code paths and improve the overall test coverage.

The `testing.T` object is used to perform atomic tests, that are the smallest unit of testing. Fuzzing is part of this category and finds out new test cases that might lead to crashes or failures and where the outcomes are not predictable; it is especially useful for strings-based functions and for dealing with user inputs.

Listing 6.1: Go testing framework example file parsing_test.go

```go
package parsing

func TestParseVersionInvalidStringNonNumbers(t *testing.T) {
  _, err := parseVersion("abc")
  if !errors.Is(err, CustomError) {
    t.Errorf("expected CustomError, got %v", err)
  }
}

func TestParseVersionInvalidStringExtraWhitespace(t *testing.T) {
  _, err := parseVersion(" 100")
  if !errors.Is(err, CustomError) {
    t.Errorf("expected CustomError, got %v", err)
  }
}

func TestParseVersionValidStringReturnsCorrectNumber(t *testing.T) {
  result, err := parseVersion("100")
  if err != nil {
    t.Errorf("unexpected error: %v", err)
  }
  expected := 100
  if result != expected {
    t.Errorf("expected %d, got %d", expected, result)
  }
}
```

An alternative to the previous syntax is to use the `t.Run` method and subtests. Doing this way, the syntax would be very similar to how Typescript testing is performed. Anyway, it is not the default convention for the Go coders.

6.1.2 Typescript

Of course, also the cloud app needs to be tested. The chosen testing framework is Jest[1], part of the OpenJS foundation. Indeed, it is quite similar in terms of concept to the Go testing framework, although it is an external dependency.

A notable syntactical difference is placed in how the files are named and how the evaluation system works; in our configuration, the framework looks for files with the extension ending with *.spec.ts*. Then, all we need is the method named *test* which runs a test. By default, we do not need to define test functions with a uniquely identifiable name, but we *describe* the expected behaviour. For example, the following snippet describes the testing for a utility to parse a valid version number:

[1] `https://jestjs.io`

Listing 6.2: Jest testing framework example file parsing.spec.ts

```
[...]
describe('parse version number', () => {
  describe('given an invalid string', () => {
    test('composed of non-numbers throws CustomError', () => {
      expect(() => parseVersion('abc')).toThrow(CustomError);
    });

    test('with extra whitespace throws CustomError', () => {
      expect(() => parseVersion(' 100')).toThrow(CustomError);
    });
  });

  describe('given a valid string', () => {
    it('returns the correct number', () => {
      expect(parseVersion('100')).toBe(100);
    });
  });
});
```

The `expect` function is used to assert the expected behaviour of the system and the `toThrow` function is used to assert that the function throws an error. The `toBe` function is used to assert that the function returns the correct value. The `describe` function is used to group the test cases and the `test` function is used to define the test cases. The `it` function is an alias for the `test` function.

6.1.3 Snyk

We also use Snyk to scan for vulnerabilities in the dependencies of the tool project. Snyk offers a CLI tool embedded in a container image[2]. We run the `golang-1.22` tagged version to get the results. The tool is very useful to find out the vulnerabilities in the dependencies and to fix them. In our case, we found a vulnerability in a package, that we fixed by updating the package to the latest available version.

This software is also specialized in finding vulnerabilities in NodeJS projects; therefore, we use it to scan the cloud web app project, in addition to the *audit* command provided by the NodeJS package manager `npm`. The tagged version we use is `node-22`. Due to the way the dependencies proliferate and the number of discovered vulnerabilities each day, it is important to run the tool frequently to keep the project secure.

An example of commands to issue for scanning dependencies in a Go project is the following:

```
$ export SNYK_TOKEN=token
$ docker run --rm -it --env SNYK_TOKEN -v $(pwd):/app snyk/snyk:golang-1.22
```

The `SNYK_TOKEN` is an environment variable that contains the token to authenticate the user to the Snyk service. The `-v` flag is used to mount the current directory to the `/app` directory in the container. The `--rm` flag is used to remove the container at the end and the `-it` flag is used to run the container in interactive mode to see the output of the command.

6.2 Manual system testing

In addition to the automated testing, the functionalities of the products are tested by a real group of people. In fact, Corvina has a dedicated testing team composed of six people, located in the Indian offices. They take care of the manual testing of the system. To synchronize the work, we do a weekly stand-up meeting focused on the completed tests and on the upcoming ones. The team is divided into different groups: there is the group in charge of testing the infrastructure, the group in charge of testing the frontend, and the group in charge of testing the backend.

Once the implementation of the stories related to a functionality is completed, they are taken in charge by this team which performs several tests, with and without reading the documentation: the

[2] `https://hub.docker.com/r/snyk/snyk`

latter to test the usability of the system as the final customer and the former with the goal to test the expected behaviour in a real environment.

If the manual test does not pass the minimum requirements, the ticket related to that story is reopened and therefore handled by the developer again. After that the fixes are applied, the ticket is assigned back to the testers, and so on. If the testers are satisfied with the implementation, the ticket is closed and the story is considered done, and the feature is ready to be released.

6.3 Considerations

We wrote more than two hundred test cases, including various parameter combinations and edge cases. We used the two syntaxes described in the previous subsections respectively for the two programming languages and we also took advantage of the table-driven tests to test the same function with different parameters.

Despite the automated testing, sometimes the testers still report troubles with test cases not handled by the developer. This is another good way to improve the test coverage.

In our personal projects, we never had a so defined and strict testing methodology. TDD is really such a new concept for us, but it is essential to be efficient in the development process because it helps to focus on the actual requirements of the system. It is also a good way to understand the expected behaviour of the system and to identify the edge cases that need to be handled.

7 Conclusions

The increased connectivity of legacy devices, driven by the integration of the Industrial Internet of Things and digital technologies, has significantly expanded the cyberattack surface in industrial sectors.

This thesis has presented an approach to secure industrial devices by acknowledging the customers of potential risks in the configuration of their devices fleet.

The analysis of the industrial regulations and standards raised the need for a security framework that can be used to assess the security of industrial devices. Therefore, the presented scanning tool is a response to this need.

First, we have identified the main security requirements for such devices, by analyzing the most common regulations and standards. Some common requirements are shared among the regulations and others are specific to a single regulation due to the different sectors that they are focused on. The scanning tool has been designed to cover those requirements for which a solution can be taken by the customer through the configuration of the device itself.

At the end of the internship period, the scanning tool is able to identify the misconfigurations of the industrial devices and to provide the customers with a textual report that can be used to improve their security, by lowering the risks.

Future work should be focused on the improvement of the user experience of the tool, by providing a graphical user interface where the results are stored and can be easily accessed by the customers. That should be the expansion of the existing cloud application.
Furthermore, the experience should be simplified on the device side also, to not have the customer run the tool on a terminal, but through another user interface accessible on the device along with the existing settings.

One of the requirements given by the laws is periodic scanning: although it is not been implemented in the scanning tool itself, it can be already achieved by taking advantage of a local cron job that runs it periodically, or by performing further changes to be able to remotely schedule the scanning from the cloud application.

It has been a great experience to work on this project and to be able to contribute to the security of industrial devices. I am grateful to Corvina and Exor for the opportunity to work on this project from scratch.

Bibliography

[1] 5 Trends in Computer Science Research - Top Universities. https://www.topuniversities.com/courses/computer-science-information-systems/5-trends-computer-science-research. accessed 18/10/2024.

[2] What is the Agile methodology? https://www.atlassian.com/agile. accessed 02/10/2024.

[3] Test driven development. https://developer.ibm.com/articles/5-steps-of-test-driven-development. accessed 24/11/2024.

[4] What is the CIA Triad? https://www.fortinet.com/resources/cyberglossary/cia-triad. accessed 23/09/2024.

[5] European Commission. Proposal for a Regulation of the European Parliament and of the Council on horizontal cybersecurity requirements for products with digital elements and amending Regulation (EU) 2019/1020. *Cyber Resilience Act - Impact assessment Part 1*, 2022. https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act-impact-assessment.

[6] EU Cyber Resilience Act. https://digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act. accessed 26/09/2024.

[7] European Cyber Resilience Act (CRA). https://www.european-cyber-resilience-act.com. accessed 26/09/2024.

[8] CVE security vulnerability database. https://www.cvedetails.com. accessed 26/09/2024.

[9] Common Vulnerability Scoring System version 4.0 - Specification Document. https://www.first.org/cvss/v4-0/cvss-v40-specification.pdf. accessed 28/10/2024.

[10] NVD - Vulnerability Metrics. https://nvd.nist.gov/vuln-metrics/cvss. accessed 23/09/2024.

[11] Microsoft Digital Defense Report: 600 million cyberattacks per day around the globe. https://news.microsoft.com/en-cee/2024/11/29/microsoft-digital-defense-report-600-million-cyberattacks-per-day-around-the-globe. accessed 30/11/2024.

[12] Cybersecurity in Manufacturing: Key Compliance and Protection Tips. https://www.exorint.com/exor-innovation-blog/cybersecurity-and-the-manufacturing-industry. accessed 24/09/2024.

[13] Go cgo compiler. https://pkg.go.dev/cmd/cgo. accessed 29/10/2024.

[14] List of all valid GOOS/GOARCH combinations. https://pkg.go.dev/internal/platform#pkg-variables. accessed 30/09/2024.

[15] Why We Write Everything in Go - Bitly blog. https://bitly.is/whywewriteeverythingingo. accessed 30/09/2024.

[16] The Go Programming Language. https://go.dev. accessed 30/09/2024.

[17] Go (programming language) - Wikipedia. https://en.wikipedia.org/wiki/Go_(programming_language). accessed 30/09/2024.

[18] Go valid ldflags. https://pkg.go.dev/cmd/link#hdr-Command_Line. accessed 29/10/2024.

[19] What does the w flag mean when passed in via the ldflags option to the go command? - Stack Overflow. https://stackoverflow.com/a/22276273/13203376. accessed 29/10/2024.

[20] Go encoding/pem package. https://pkg.go.dev/encoding/pem. accessed 31/10/2024.

[21] Go standard library. https://pkg.go.dev/std. accessed 31/10/2024.

[22] Go testing. https://pkg.go.dev/testing. accessed 24/11/2024.

[23] Go crypto/x509 package. https://pkg.go.dev/crypto/x509. accessed 31/10/2024.

[24] Go trimpath argument. https://pkg.go.dev/cmd/go. accessed 29/10/2024.

[25] Go valid GOOS/GOARCH combinations. https://pkg.go.dev/internal/platform#pkg-variables. accessed 29/10/2024.

[26] Zerolog benchmarks. https://github.com/rs/zerolog?tab=readme-ov-file#benchmarks. accessed 21/10/2024.

[27] Industrial Cybersecurity getting straight to the point: IACS Vs ICS what is correct? https://www.linkedin.com/pulse/industrial-cybersecurity-getting-strait-point-iacs-vs-sabino-costa-1e. accessed 26/09/2024.

[28] IEC: International Electrotechnical Commission. https://www.iec.ch/who-we-are. accessed 25/09/2024.

[29] IXON - Practical Guide for IEC 62443. https://www.ixon.cloud/it/securitytools. accessed 11/04/2024.

[30] Understanding IEC 62443. https://www.iec.ch/blog/understanding-iec-62443. accessed 24/09/2024.

[31] The Power of IEC 62443: Safeguarding Industrial Automation by Mohamed Wassef O. https://www.linkedin.com/pulse/power-iec-62443-safeguarding-industrial-automation-othmani-gsrde. accessed 24/09/2024.

[32] ISO: International Organization for Standardization. https://www.iso.org/iso-name-and-logo.html. accessed 25/09/2024.

[33] ISO/IEC 27001:2022. https://www.iso.org/standard/27001. accessed 25/09/2024.

[34] OT Security vs. IT Security: Comparative Analysis. https://www.fortinet.com/resources/cyberglossary/it-vs-ot-cybersecurity. accessed 23/09/2024.

[35] What Is the Difference Between IT and OT? https://www.paloaltonetworks.com/cyberpedia/it-vs-ot. accessed 23/09/2024.

[36] Kubernetes Overview. https://kubernetes.io/docs/concepts/overview. accessed 04/10/2024.

[37] Alessandro Marchetto. Security Testing course slides, A.Y. 2023/24.

[38] What are microservices? https://microservices.io. accessed 04/10/2024.

[39] Directive on measures for a high common level of cybersecurity across the Union (NIS2 Directive) - FAQs. https://digital-strategy.ec.europa.eu/en/faqs/directive-measures-high-common-level-cybersecurity-across-union-nis2-directive-faqs. accessed 25/09/2024.

[40] What Is OpenAPI? https://swagger.io/docs/specification/v3_0/about. accessed 30/09/2024.

[41] What is a REST API? https://www.redhat.com/en/topics/api/what-is-a-rest-api. accessed 30/09/2024.

[42] Tristan Richardson and John R. Levine. The Remote Framebuffer Protocol. RFC 6143, March 2011.

[43] What is scrum? https://www.atlassian.com/agile/scrum. accessed 02/10/2024.

[44] Scrum Epics and Stories. https://www.atlassian.com/agile/project-management/epics-stories-themes. accessed 03/10/2024.

[45] What is Virtualization? https://aws.amazon.com/what-is/virtualization. accessed 04/10/2024.

[46] What is HMI or Human Machine Interface? https://www.exorint.com/exor-innovation-blog/what-is-hmi-human-machine-interface-and-do-you-make-or-buy-it. accessed 24/09/2024.

[47] What is Industry 4.0? https://www.ibm.com/topics/industry-4-0. accessed 24/09/2024.

[48] The Yocto Project. https://www.yoctoproject.org. accessed 01/10/2024.