SCASS: breaking into SCADA Systems Security

Nicola d'Ambrosio[a], Giulio Capodagli[a], Gaetano Perrone[a], Simon Pietro Romano[a]

[a] *University of Napoli Federico II, Department of Electrical Engineering and Information Technology, Via Claudio 21, 80125 Napoli, Italy*

Abstract

This paper explores the susceptibility of industrial control systems (ICS) to cybersecurity threats and addresses the inherent challenges in evaluating their security. Emphasizing the limitations of conventional vulnerability assessments and penetration testing within ICS, the study underscores the associated risks and cost constraints. To overcome these challenges, the authors introduce SCASS, an open-source, modular, and hybrid platform designed for building extensible cybersecurity testbeds in the ICS domain. SCASS leverages a unique combination of physical components sourced from ABB and virtual components running on a Raspberry Pi using Docker. The paper introduces an innovative approach to overcome constraints in container-based virtualization, ensuring seamless communication between physical and virtual systems. To showcase the platform's effectiveness, the authors replicate a physical testbed named "EPIC", providing the source code and presenting an advanced cybersecurity attack against the SCASS system.

This research aims to empower researchers in developing robust cybersecurity testbeds and offers support to enterprise companies seeking to replicate digital twins for security purposes. The open-source nature of SCASS encourages collaborative efforts to enhance ICS security assessment methodologies.

Keywords: ICS cybersecurity, SCADA systems, ICS cyber-range, ICS testbeds

1. Introduction

Cybersecurity incidents wield a significant impact on industrial systems, as evidenced by current market research on the state of operational technology and cybersecurity[1]. The integration of cybersecurity in operational technology (OT)

Email addresses: nicola.dambrosio2@unina.it (Nicola d'Ambrosio), g.capodagli@studenti.unina.it (Giulio Capodagli), gaetano.perrone@unina.it (Gaetano Perrone), spromano@unina.it (Simon Pietro Romano)

[1] 2023 State of Operational Technology and Cybersecurity, Fortinet, URL: https://www.fortinet.com/content/dam/fortinet/assets/reports/report-state-ot-cybersecurity.pdf, accessed 08-08-2023

environments is noted to be in an immature stage, posing escalating security risks with the advent of hybrid systems amalgamating cloud, IoT, and artificial intelligence into *Industrial Control Systems*. Despite the development of various methodologies and techniques to counteract these risks, the execution of offensive approaches like vulnerability assessments on ICS remains a challenge due to the potential for severe damage to the physical environment.

Security researchers address this challenge through the creation of digital twins, comprehensive simulations replicating the intricacies of an actual industrial control system. Various techniques, including virtualization, simulation, and physical reproduction, are employed for this purpose. Noteworthy literature, such as Kandasamy et al. (2021) [1], utilizes "hosted" virtualization to construct a complete digital twin of the "EPIC" physical testbed [2], emphasizing the benefits of container-based virtualization while acknowledging limitations in reproducing certain physical network attacks like *Address Resolution Protocol* (ARP) spoofing. This work contributes by demonstrating the resolution of such challenges through meticulous configuration of the container networking stack.

While acknowledging the benefits of full virtualization, cautious considerations are essential in replicating industrial control systems [3]. To address this, we propose a modular architecture facilitating the configuration of both physical and virtual components through a unified network, ensuring adherence to standard architectures defined in the ICS cybersecurity domain (Section 2.3).

This paper introduces the architecture of *SCADA Systems Security* (SCASS), a compact, hybrid, and lightweight environment designed for creating security testbeds for ICSs. Housed within a small cabinet, SCASS integrates physical components commonly employed in ICSs alongside a Raspberry Pi – a small, single-board computer capable of virtualizing the networking stack. Leveraging Linux container technology, SCASS establishes a lightweight and performant virtualization environment, simulating the components and network of ICSs. Through meticulous configuration of the networking stack, a communication link is established between the physical and virtual environments, enabling researchers and security experts to replicate complex and realistic scenarios.

The modular design of SCASS facilitates easy integration of external components, exemplified by a reproduction of the EPIC physical security testbed. The accompanying online availability of the source code positions it as a valuable resource for implementing cost-effective SCADA environments for both research and business purposes.

This paper is structured as follows. Section 2 lays the foundation with a comprehensive introduction to Industrial Control Systems, delving into the intricate components, protocols, and programming languages prevalent in this domain. Section 3 expands on this by providing an insightful overview of cybersecurity within the ICS domain. Meanwhile, Section 4 meticulously explores related works, offering detailed insights into both the EPIC testbed and its digital twin (EPICTWIN). The architecture of SCASS takes center stage in Section 5, where a thorough examination of its design and functionality is presented. Subsequently, Section 6 demonstrates the practical feasibility of leveraging SCASS to replicate the EPIC testbed, showcasing its adaptability and utility. In Section 7,

the paper transitions to a practical demonstration, illustrating the application of SCASS in executing an ICS attack scenario. This section provides a hands-on understanding of SCASS's capabilities in simulating real-world cybersecurity challenges. Section 8 draws the threads together, encapsulating the contributions made in this work and offering insightful suggestions for future extensions of the proposed methodology. This section serves as a comprehensive conclusion, summarizing the key findings and paving the way for further exploration in the realm of ICS cybersecurity.

2. Industrial Control Systems

An ICS is a computer-based system used to control, monitor, and operate industrial processes, such as those found in manufacturing, energy, and critical infrastructure fields. ICSs often include a combination of hardware, software, and networking components. Their architecture can be organized into three main groups of components that communicate via either a wired or a wireless network in order to exchange data and signals:

- *Field devices*: sensors and actuators that monitor and control industrial processes, such as temperature sensors, flow meters, and valves;
- *Control and acquisition system*: it is the central control system that receives input from the field devices and sends output to them, which includes the *Programmable Logic Controllers* and *Supervisory Control And Data Acquisition* (SCADA) systems that execute the control logic and manage the data flow between the field devices and the control center;
- *Control center*: every *Human Machine Interface* (HMI) and terminal that allows operators to monitor and control the industrial process. The control center typically includes computer systems and displays that provide real-time data and allow personnel to input commands and make changes to the control logic.

2.1. SCADA Systems

SCADA systems serve as crucial tools for remote monitoring and control of industrial processes and infrastructures. These systems collect data from field sensors and devices, transmitting it to a central control system for analysis and regulatory decision-making. Key components of a typical SCADA system include:

- *Programmable Logic Controllers*: specialized computers for industrial automation, designed to operate and control processes;
- *Remote Terminal Unit*s: industrial control devices facilitating remote monitoring and control of field devices, acting as intermediaries between field devices and the central control system;

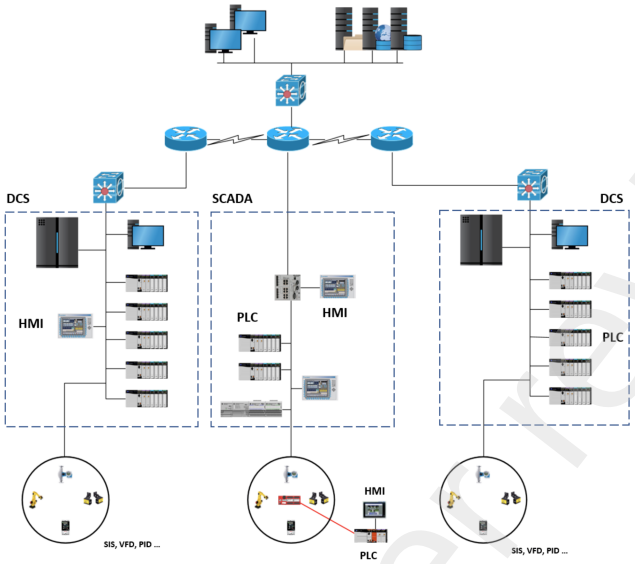

Fig. 1: The network infrastructure of an ICS (Ackermann, *Industrial Cybersecurity* [4]).

- *Human Machine Interfaces*: devices or software enabling human operators to interact with and control industrial machinery;
- Data Historians: dedicated to data storage, retrieval, analysis, and reporting for historical records and operational insights;
- Communication Networks: connecting *Remote Terminal Units* or PLCs to the central control system for seamless data exchange;
- Central Control System: the core of the SCADA architecture, processing data from remote locations and making control decisions.

This comprehensive system ensures effective and integrated industrial process monitoring and control, emphasizing efficiency and reliability in critical operational environments.

2.2. Communication protocols for industrial control systems

While *Operational Technology* (OT) networks share fundamental principles with *Information Technology* (IT) networks, subtle distinctions exist. OT networks, akin to IT networks, rely on the TCP/IP stack and common application-level protocols like *HyperText Transfer Protocol* (HTTP), *HyperText Transfer Protocol over Secure Socket Layer* (HTTPS), *Simple Mail Transfer Protocol* (SMTP), *File Transfer Protocol* (FTP), *Trivial File Transfer Protocol* (TFTP), *Service Message Block* (SMB), and *Simple Network Management Protocol* (SNMP), introducing vulnerabilities up to levels 1-3. However, over the decades, specific industrial protocols were designed to facilitate rapid, efficient,

and reliable communication between devices. Noteworthy protocols in this realm include *Distributed Network Protocol* (DNP3) [5], *Message Queue Telemetry Transport* (MQTT) [6], *Highway Addressable Remote Transducer* (HART) [7], *Process Field Bus* (ProFiBus) [8], and Profinet [9], each tailored to industrial needs with unique characteristics. Many of these protocols can operate natively on Ethernet, with some extensions facilitating TCP-level communications over time. An exemplar is Modbus [10], an application layer messaging protocol established in 1979. Recognized as a de facto standard for industrial automation, Modbus boasts reliability, ease of implementation, and openness. Employing a controller/worker architecture, Modbus operates on a request/reply model, cementing its status in industrial automation and process control.

Modbus is available in several versions:

- *ModbusRTU*: it is the original version of the network protocol and the most common Modbus implementation, which uses binary encoded data sent over serial lines (such as RS-485 or RS-232) with *Cyclic Redundancy Check* (CRC) error detection applied inside the payload;
- *Modbus ASCII*: makes use of ASCII characters over serial lines, performing CRC inside the payload;
- *Modbus/TCP*: messages are encapsulated in TCP/IP packets on an Ethernet network;
- *Modbus Plus*: it is a high-speed, proprietary to *Schneider Electric*, Modbus version, supporting *peer-to-peer* communications between multiple clients.

Other minor variants are *Modbus RTU/IP*, *Modbus over UDP*, *Pemex Modbus* (proprietary), *Enron Modbus* (proprietary). A Modbus *Protocol Data Unit* (PDU) consists of two fields:

- *Function Code*, a fixed-length field ranging from 1 to 127, specifying a particular service such as reading or writing a specific register, along with the amount of exchanged data;
- *Data*: the data pertinent to the service or an error response;

The PDU is encapsulated in a *Application Data Unit* (ADU), which is medium-dependent and generally composed of a header containing the slave address, a Modbus PDU and an error checksum. In essence, the structure of a Modbus packet ensures a standardized and orderly method of information exchange among Modbus devices.

Modbus/TCP, also known as Modbus Ethernet or Modbus TCP/IP, represents the variant of the Modbus protocol functioning on TCP, specifically port 502, across Ethernet networks. While Modbus PDUs retain their form, they are enveloped in TCP fragments for transmission on Ethernet networks, resulting in swifter and more reliable communication compared to serial versions. In Modbus/TCP, the ADU takes on the name *Modbus Application* (MBAP).

Differing from Modbus RTU and Modbus ASCII, MBAP excludes the Error Checksum trailer, which is typically appended at the data-link level. The MBAP header (Figure 2) encompasses:

- Transaction ID (2 bytes): a unique identifier for each request, echoed by the server to maintain request order;
- Protocol ID (2 bytes): set by the client, consistently equal to "00 00";
- Length (2 bytes): identifies the bytes in the subsequent message;
- Unit ID (1 byte): an identifier for worker devices sharing the same communication medium.

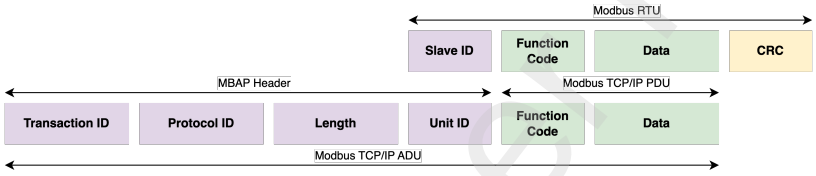

Fig. 2: Modbus RTU, Modbus TCP comparison.

2.2.1. Modbus vulnerabilities

Modbus stands out as a fast, efficient, and reliable network protocol in industrial automation. However, its extensive service history renders it intrinsically insecure, having been developed decades ago when security considerations were less prominent. Instances of security vulnerabilities and threats in Modbus-based systems have surfaced, lacking essential mechanisms for ensuring confidentiality, integrity, and availability in communications:

- Data exchanged between devices lack encryption, allowing messages to be easily intercepted, read, and altered by attackers, as the PDU checksum is not based on a cryptographic hash function;
- The protocol lacks an authentication mechanism, enabling attackers to manipulate worker devices by impersonating a controller device;
- Communication errors are challenging to detect due to the limited error protection provided by the checksum;
- Controller/worker architectures akin to Modbus expose a single point of failure, where the failure of a controller device affects the entire system's availability, making it susceptible to *Denial of Service* (DoS) attacks;
- Modbus lacks an anti-replay attack mechanism, making it unable to discard captured and resent Modbus/TCP packets.

Over time, various approaches have been introduced to address Modbus vulnerabilities. In 2008, an enhancement known as Modbus/TCP Security [11] was released, prioritizing security by operating over *Transport Layer Security* (TLS) [12], ensuring confidentiality and integrity through secure cipher suites. Authentication is achieved through signed certificates exchanged before each TLS session. Independent Modbus variants have also emerged, such as the one developed by Yi et al. [13], introducing authentication, confidentiality, and integrity through *Hash-Based Message Authentication Code* (HMAC) and *ShangMi 3* (SM3)-*ShangMi 4* (SM4) block ciphers. This implementation, efficient and lightweight, provides a standalone solution without requiring TLS connections and encapsulation, thereby minimizing protocol overhead.

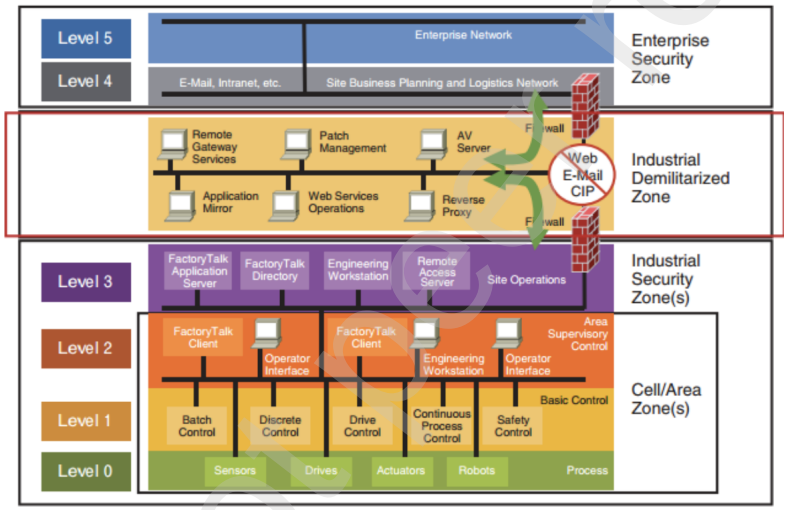

Fig. 3: Purdue Model. (Image Credit: Ackerman, *Industrial Cybersecurity [4]*)

2.3. The Purdue Model

Over time, numerous standards have emerged to establish a reference architecture for industrial control systems. A notable example is the Purdue Model, an integral component of the *Purdue Enterprise Reference Architecture* (PERA) [14], extensively covered in the ISA-99 standard by the *International Society of Automation* (ISA). PERA serves to compartmentalize an ICS infrastructure, delineating the OT network into five levels.

In contrast, the *Converged Plantwide Ethernet* (CPwE) [15] focuses on presenting a standardized approach for deploying Ethernet-based networks within ICSs. Another significant contribution is Ackerman's Industrial Cybersecurity [4], which offers a comprehensive overview of modern architectures, typically organized into three zones and six levels (Figure 3):

1. *Enterprise zone*: this zone consists of the enterprise network and encompasses the site's business planning and logistics levels;

2. *Industrial Demilitarized zone*: this zone acts as a buffer between the enterprise and manufacturing zones, providing additional security controls and separation;
3. *Manufacturing zone*: this zone is further divided into the following levels:
   (a) Site operations level: this level includes functions such as data acquisition, data historian, and operator interfaces;
   (b) Area supervisory control level: it involves supervisory control functions that coordinate multiple units or areas;
   (c) Basic control level: this level comprises control functions for specific equipment or processes;
   (d) Process level: it represents the lowest level and involves direct control of physical processes.

These zones and levels provide a structured framework for organizing and securing an ICS architecture.

2.4. Programming languages for industrial control systems

Considerable attention has been dedicated by the *International Electrotechnical Commission* (IEC) to establishing a set of programming languages and interfaces, fostering program development and device configuration with a focus on portability and reusability. The culmination of these efforts is evident in IEC 61131-3 [16], the third version of IEC 61131, released in 1993, an international standard tailored for PLCs. This comprehensive standard encompasses five language models:

- Structured Text (ST): resembling high-level programming languages, ST facilitates the development of intricate algorithms;
- Function Block Diagram (FBD): representing system logic through graphical blocks and connections, FBD offers a visual approach to programming;
- Ladder Diagram (LD): this graphical programming language, commonly employed in industrial control systems, mirrors electrical ladder diagrams;
- Sequential Function Chart (SFC): allows the representation of complex control algorithms through steps, transitions, and actions;
- Instruction List (IL): a low-level, assembly-like language designed for efficient programming.

2.4.1. ABB Automation Builder

ABB Automation Builder serves as an Integrated Development Environment (IDE) for configuring and programming control systems on ABB automation products. Leveraging a robust toolset, Automation Builder enables the development, testing, and commissioning of automation solutions and communication networks. This IDE relies on CodeSys [17], a proprietary software platform compliant with the IEC 61131-3 standard programming languages, to support

the construction of automation applications. Automation Builder facilitates flexibility and compatibility by supporting the five standard programming languages mentioned above.

In particular, *Sequential Function Chart* (SFC) has become, since IEC 61131-3, a standard, while *Instruction List* (IL) has been deprecated in the same version. Applications are structured as sets of *Program Organizational Units*, ensuring modularity. This modular approach allows the creation of programs by combining objects using different IEC-61131 compliant languages. Additionally, IEC-61131 standardizes *variable types*, *tasks*, and *start modes*, fostering consistency in automation application development.

3. Industrial Control Systems and Cybersecurity

ICSs play a crucial role in managing critical infrastructures, making them susceptible to significant attack vectors. Moreover, the contemporary trend towards adopting cloud-based environments and leveraging cloud computing benefits escalates the risk of cyber attacks. Consequently, various techniques (often based on machine learning) have been implemented to detect attacks [18].

Given the impracticality of conducting security testing directly on SCADA systems, the development of formal methodologies becomes essential for defining cyber attacks on ICSs. This section aims to shed light on real cyber attacks against SCADA infrastructures and illustrate commonly employed methodologies for modeling cyber threats in ICSs. Section 7 will delve into demonstrating the application of these methodologies in modeling a real attack against SCASS. This demonstration involves the replication of the attack through an attacker machine with the capability to breach the internal system.

3.1. Relevant attacks on ICS infrastructures

ICSs often exhibit inherent vulnerabilities susceptible to security assaults. Process automation protocols and legacy devices, commonly lacking security-oriented features, render architectures exposed to a myriad of attacks. Individuals with strong motivations to inflict harm on organizations, companies, or institutions can exploit these vulnerabilities. This situation becomes particularly concerning in the realm of cyberwarfare tactics, where sensitive targets such as power plants, energy distribution systems, or air traffic control systems can face severe impacts. Malicious software injection becomes a potent method to disrupt operations within the infrastructure, potentially causing substantial economic losses or environmental damage.

In recent years, the landscape has witnessed numerous notable attacks directed at sensitive ICS targets, highlighting the urgency of addressing and fortifying the security posture of these critical systems.

3.1.1. Stuxnet

Stuxnet, discovered in 2010, stands out as the most infamous malware to impact PLCs, marking the first instance of malware designed to physically destroy

a military target. Likely developed by the United States and Israel, Stuxnet played a pivotal role in disrupting Iran's nuclear program by targeting Siemens SCADA systems, causing physical damage to uranium enrichment centrifuges at the Natanz Enrichment Complex and affecting over 100,000 systems globally. The unprecedented nature of Stuxnet's attack garnered immediate attention, leading to extensive studies by researchers [19] [20]. Its sophistication and the sheer audacity of its objectives underscore the evolving landscape of cyber threats, particularly those aimed at critical infrastructure.

3.1.2. Havex

Havex, unearthed in 2014, emerged as a *Remote Access Trojan* (RAT) associated with the toolkit of *Dragonfly*, a prominent Russian hacking group sponsored by the government [21]. Havex was meticulously crafted to target ICSs for data exfiltration, gathering network information from the OT zone and sensitive plant data. Notably, it was employed in cyber attacks against energy and manufacturing companies across Europe and the United States. The modus operandi of Havex involved data exfiltration by scanning the network using the *Open Platform Communications* (OPC) protocol [22], collecting valuable information, and transmitting it to an external Command and Control server that maintained communication with the compromised devices. Li et al. [23] have conducted an in-depth study on Havex, exploring the various mechanisms employed by the malware to propagate the infection.

3.1.3. Industroyer

Industroyer [24], also known as *CrashOverride*, emerged in 2017 as a creation of the likely state-sponsored hacking group Sandworm. This malware specifically targeted electrical networks, with the capability to manipulate circuit breakers. Industroyer is notably attributed to the blackout incident affecting approximately 20% of Kiev, the capital city of Ukraine. The architecture of Industroyer is modular, organized into distinct modules, each serving specific functions within the malware's operations, including the configuration of the payloads, persistence, command, and control.

3.1.4. Pipedream

Pipedream, also known as *Incontroller* [25] [26], hails from the *Chernovite Activity Group* and represents a sophisticated modular framework. Discovered recently, it serves as a versatile tool for damaging ICSs, adapting its functionality based on specific targets and environments. Pipedream is often described as a 'Swiss Army Knife' for attackers, primarily designed to target *Schneider Electric* and *Omron* PLCs. However, it exhibits flexibility by being capable of attacking various industrial technologies, including *CoDeSys*, *Modbus*, and OPC Unified Architecture (UA). The modular nature of Pipedream allows attaching additional modules to expand its capabilities, enabling attacks on devices from other vendors. The malware incorporates a virtual console designed to resemble a SCADA interface. This feature simplifies interactions for less skilled hackers,

empowering them to gather information about PLCs on the network, upload malicious software files, and manipulate device parameters with relative ease.

3.2. ICS Cybersecurity approaches

Numerous methodologies have been developed to model cybersecurity threats in the ICS domain [27]. However, the predominant approaches often involve integrating security considerations into established safety models [28].

Safety models are instrumental in evaluating factors that pose potential harm to systems, processes, or infrastructures across various domains such as transportation, manufacturing, and construction. The primary objective of these models is to pinpoint hazards, quantify associated risks in terms of likelihood and impact, and implement measures to mitigate or prevent adverse events, ensuring the protection of valuable assets. In systems focused on safety, failures can result in severe consequences, leading to significant harm to individuals, property (with substantial economic losses), and the environment.

3.2.1. STAMP and STPA

Systems Theoretic Accident Model and Process (STAMP) is an accident causality model developed by Leveson [29]. The foundational principle of STAMP is to conceptualize accidents as control problems rather than failure problems, with the goal of accident prevention by imposing constraints on the behavior and interactions of system components. STAMP finds extensive application across various domains, including air traffic control [30], nuclear power [31], automobiles, medical devices, and hospital safety.

Systems Theoretic Process Analysis (STPA) is a hazard analysis methodology that builds upon STAMP and is employed to study the dynamics of accidents. The hazard analysis using STPA involves several steps:

- Identifying accidents and hazards (*precondition*);
- Drawing the control structure (*precondition*);
- Identifying unsafe control actions;
- Identifying causal factors and creating scenarios.

The effectiveness of STPA surpasses that of traditional safety models like *Failure Tree Analysis* (FTA), *HAZard and OPerability analysis* (HAZOP), *Failure Mode and Effects Analysis* (FMEA), and *Event Tree Analysis* (ETA) in uncovering overlooked causes. Yan et al. [32] assert that STPA proves more efficient, especially in complex systems incorporating computer control. Many hazard analysis frameworks were established prior to the advent of computer control, leading to increased complexity, making STPA particularly valuable in such contexts.

Beyond systems engineering, STPA has demonstrated its versatility. Chen et al. [33] applied STPA to evaluate national responses to the COVID-19 pandemic, showcasing its adaptability across diverse fields. Furthermore, Castiglione et al. [34] introduced *Hazard Analysis of Smart Grid Infrastructures* (HA-Grid), leveraging STPA to identify attack paths for the target infrastructure.

3.2.2. STPA-Sec

Systems Theoretic Process Analysis for Security (STPA-Sec), an extension of the STPA safety model developed by Young and Leveson [35] [36], is tailored for the design and analysis of control systems from a security standpoint. This variant enhances the STAMP/STPA framework, specifically addressing security concerns. It introduces additional steps to identify and analyze security risks related to critical control processes essential for the system's safe and reliable operation.

Friedberg et al. [37] have also contributed to this area with the development of *STPA-SafeSec*, a unified process integrating safety and security analysis, offering a comprehensive approach to address both aspects.

The STPA-Sec process is primarily geared towards the early stages of the system engineering lifecycle, preceding solution development and implementation, to maximize its efficacy. The process involves the following key actions:

1. Application of Systems Engineering Foundations: the STPA-Sec process commences by clearly defining the purpose of the analysis within the framework of systems engineering foundations;
2. Identification of Types of Unsecure Control: going beyond the scope of STPA, STPA-Sec mandates the identification of unsecure control actions, broadening the focus from merely unsafe controls;
3. Identification, Elimination, or Control of Causes of Unsecure Control: the STPA-Sec process employs an information life cycle to track hazardous control actions. It identifies scenarios leading to unsecure control actions, extending the capabilities of STPA, which primarily centers around safe control actions. Furthermore, the process involves the generation of security scenarios through wargaming [38] to facilitate the selection of effective control strategies.

3.2.3. STRIDE

The STRIDE threat modeling methodology, developed by Microsoft [39], is a comprehensive approach to identifying and categorizing various security threats. The acronym STRIDE represents six threat categories:

- *Spoofing*: involves attackers impersonating someone or something else to gain unauthorized access or deceive the system;
- *Tampering*: encompasses unauthorized modification or alteration of data or systems, compromising their integrity;
- *Repudiation*: relates to the denial of performing a particular action or transaction, impacting non-repudiation requirements;
- *Information Disclosure*: involves unauthorized access to sensitive information, leading to a breach of confidentiality;

- *Denial of Service*: encompasses actions or attacks that disrupt or prevent legitimate users from accessing or using a system, resulting in a loss of availability;
- *Elevation of Privilege*: refers to an attacker gaining higher privileges or access rights than intended, enabling them to perform unauthorized actions or gain control over a system.

By considering these threat categories, organizations can gain a better understanding of potential security risks and develop appropriate countermeasures to mitigate them. The model takes into account the most common threats to a system, with each threat associated with a desirable property that it can compromise. STRIDE can be conducted in two ways: STRIDE-per-element, which focuses on each system component and is more complex, and STRIDE-per-interaction, which is more coarse-grained but can identify threats based on interactions between elements. The threat modeling process involves several key steps:

1. Asset Identification: identify the information and systems that need protection;
2. Threat Analysis: utilize the STRIDE mnemonic to systematically identify potential security threats for each identified asset;
3. Likelihood and Impact Assessment: evaluate the likelihood and impact of each identified threat to prioritize them for further action;
4. Countermeasures Development: develop security controls or countermeasures to effectively mitigate and prevent the identified threats.

STRIDE has proven to be a well-adopted methodology across various domains, including web applications, networking, and machine learning. Researchers, such as Khan et al. [40], have proposed approaches that combine the STAMP methodology with the STRIDE framework. Their studies demonstrate the effectiveness of using STRIDE for threat identification. Notably, STRIDE is recognized for its lightweight framework and the ability to provide more understandable results in the context of *Cyber Physical Systems*.

4. Related Works

The proposed framework encompasses various significant topics, including the application of digital twin systems in cybersecurity, cyber ranges, and the use of virtual environments to replicate SCADA systems. In this section, we explore pertinent related works on these subjects. We delve into the application of digital twins for ICS and security, outlining how researchers define simulated environments for such systems to address cybersecurity challenges. Additionally, we introduce a physical testbed called Electrical Power and Intelligent Control (EPIC) [2] and a project that replicates a virtualized twin, known as EPICTWIN [1].

4.1. Digital Twins

Digital Twin (DT) systems enable the creation of comprehensive digital representations of products that faithfully mirror the features of physical entities [41]. They offer a valuable approach to minimizing the costs associated with replicating a complete physical environment. For instance, Park et al. (2020) [42] propose a digital twin system to replicate a cyber-physical production system, encompassing services like production planning, automated execution, real-time monitoring, abnormal situation notification, and dynamic response. Zhang et al. (2018) [43] leverage digital twin technology to reproduce a smart manufacturing system.

In the realm of cybersecurity, digital twins are extensively adopted to model complex systems. Rojas et al. (2022) [44] implement a digital twin for intelligent metering systems in a smart home environment, demonstrating its utility in assessing vulnerabilities against cyber attacks.

Masi et al. (2023) advocate starting with the definition of an architectural representation of ICSs for critical infrastructure, known as the "Cybersecurity View", and constructing a digital twin used to model cybersecurity requirements, attacks, and derive countermeasures. Section 7 will demonstrate how cybersecurity scenarios against SCASS can be realized.

While DT offer numerous benefits, several authors emphasize the security risks associated with this technology [45, 46]. Attackers might exploit vulnerabilities to gain access to a blueprint of the infrastructure. Additionally, the simulated system may not perfectly emulate the real environment, making it challenging to replicate severe threats affecting the physical system. SCASS hybrid infrastructure addresses these concerns by integrating both virtual and physical components. This approach allows for defining the degree of realism in the SCASS system, striking a balance between reducing implementation costs, hardware, and software expenses while maintaining fidelity to the physical system. This hybrid model provides a practical trade-off that enhances security and feasibility.

4.2. Simulated environments for ICSs

Individuals and organizations design infrastructures to replicate real-world networks, systems, and technologies, enabling users to practice detecting and reacting to cyber threats like hacking attacks or data breaches. Cyber ranges are simulated environments serving various purposes, including training new employees, testing and evaluating new cybersecurity technologies, rehearsing incident response plans, researching and developing new cybersecurity techniques and strategies.

Cyber ranges are extensively used in the field of ICS and are implemented on testbeds that provide the foundation for constructing attack scenarios. Ukwandu et al. [47] have proposed a taxonomy for cyber ranges, offering a comprehensive comparison of various virtual environments and testbeds. The taxonomy encompasses several categories such as teaming, types, econometrics, monitoring, management, recovery, attack type, and scenarios. Similarly, it covers testbeds with attributes like education, model, generation, execution, evaluation, management, types, post-incident, recovery, attacks, and trainee. The scientific literature

describes numerous testbeds simulating OT networks in physical, hybrid, or virtual structures. A trade-off is evident concerning the reality level: virtual testbeds are more elastic and capable of a broader range of simulations, while physical testbeds can be more effective in representing their real counterparts but are generally less customizable due to the need for physically adding or replacing components for reconfiguration.

Conti's survey [48] provides an extensive analysis of testbeds for industrial cybersecurity, categorizing over 50 systems based on parameters like sector, cost, and license. The survey highlights a significant number of testbeds developed in the smart grid field over the years, shedding light on the diverse landscape of industrial cybersecurity testbeds and their characteristics.

4.2.1. Physical Testbeds

Physical testbeds, exemplified by *EPIC* from the Singapore University of Technology and Design [49], offer a more realistic simulation of events but are often expensive and time-consuming to build. EPIC mimics a real-world power system, featuring real PLCs and communication through the IEC 61850 protocol. This physical testbed allows for renting and supports the implementation of various security assaults, including False Data Injection and power supply interruption, along with mitigation techniques. The Smart Grid Test Bed by the Idaho National Laboratory [50] stands out as the world's first full-scale replication of a smart grid within a physical testbed, demonstrating the scalability of such environments from smaller dimensions to real-world counterparts. In contrast, virtual testbeds, composed of virtual machines or containers, are more cost-effective as they don't require the deployment of real components. The survey emphasizes the trade-off between realism and customization in physical vs. virtual testbeds.

EPICTWIN [1] [51] serves as a digital twin for an existing plant, allowing the assessment and mitigation of threats and vulnerabilities within the testbed or the replicated plant. The digital twin is based on network protocols such as MQTT, IEC 61850 *Manufacturing Message Specification* (MMS), and *Generic Object Oriented Substation Events* (GOOSE). The approach described in the articles related to EPICTWIN proves valuable for generating twins of testbeds or plants for cybersecurity studies. SCASS adopts a similar electric layout as used by Kandasamy, with some variations. Both SCASS and EPICTWIN utilize Node-RED. However, SCASS uses Docker containers attached to a *macvlan* adapter, enabling the realization of an OT network shared by physical and virtual environments, facilitating level 3 attacks.

Maynard SCADA by Maynard et al. [52] is a scalable framework designed for deploying multiple virtual machines to create a SCADA network. This open-source framework, available under the GNU General Public License v3 [53], is intended for researchers. It leverages components like HMIs and RTUs created with OpenPLC [54] on VirtualBox virtual machines, configured through Vagrant. These instances communicate through OT network protocols. Although not explicitly mentioned, this framework could potentially be extended to support SCASS. For instance, deploying the SCASS network on cloud infrastructure using

Maynard SCADA could enhance scalability and offer a more realistic execution in hybrid cloud and on-premises environments. However, such extensions would introduce complexity, increase costs, and affect availability.

4.2.2. Hybrid Testbeds

Hybrid testbeds combine physical and virtual components, aiming to integrate the strengths of both. This approach involves tangible, real-world devices like PLCs alongside virtual counterparts for flexibility. The testbeds facilitate the creation of diverse scenarios for testing and experimentation, allowing for the simulation of various real-world situations, attacks, or events. The integration of both physical and virtual components ensures a realistic simulation within a controlled environment. Achieving the right balance between realism and flexibility is crucial in hybrid testbeds, where physical components contribute authenticity, and virtual devices provide adaptability and scalability.

KYPO4INDUSTRY is a training facility at Masaryk University, designed by Čeleda [55], aimed at teaching computer science students how to address cybersecurity threats in OT networks. It utilizes Raspberry Pi devices to simulate PLCs, allowing the automation of attack scenario generation through YAML files and Ansible configurations to deploy *Virtual Machines*. Despite its similar use of Raspberry Pi for simulating smart grid components, such as PLCs and *Intelligent Electronic Devices*, KYPO4INDUSTRY lacks source code availability, detailed networking information, and an example of an attack scenario. SCASS also employs Raspberry Pi for container execution to simulate smart grid components.

Koutsandria et al. [56] present a hybrid testbed that combines a physical system generated in Simulink with both physical and simulated PLCs communicating through OT network protocols like Modbus. They incorporate a network tap implemented on a Raspberry Pi for collecting traffic data and monitoring packet exchanges. While the source code is not available, they provide an attack scenario example that offers insights into reproducing various scenario attacks.

HYDRA from Roma Tre University [57] is an affordable open-source testbed emulating the behavior of a simple water distribution system. It features high modularity and flexibility, allowing easy attachment and detachment of components. Communication is implemented through Arduino boards using the Modbus protocol. However, there are no provided networking details, and it is not specifically focused on cyber attacks, lacking an example of an attack scenario.

The comparison in Table 1 summarizes the key features of these hybrid testbeds, highlighting the aspects where SCASS aligns or differs from them.

4.3. Electrical Power and Intelligent Control (EPIC)

The Electric Power and Intelligent Control (EPIC) testbed faithfully replicates a smart-grid architecture. The authors present this work to enable researchers to devise new cyber-attacks on smart-grid systems and evaluate the efficacy of defense mechanisms. Figure 5 illustrates the high-level architecture of EPIC,

	[55]	[56]	[57]	SCASS
Virtual elements	Containers	N	N	Containers
Source code availability	N	N	Y	Y
Extendibility	Y	Y	Y	Y
Networking details	N	N	N	Y
Attack scenario example reproduction	N	Y	N	Y
PLC programmability	Simulated	Simulated	Simulated	Physical

Table 1: Compare SCASS with other hybrid ICS testbeds

which comprises four stages: generation, transmission, smart home, and micro-grid. These stages are linked to a controller PLC, establishing connections to a SCADA system. A more intricate depiction is available in Figure 4.

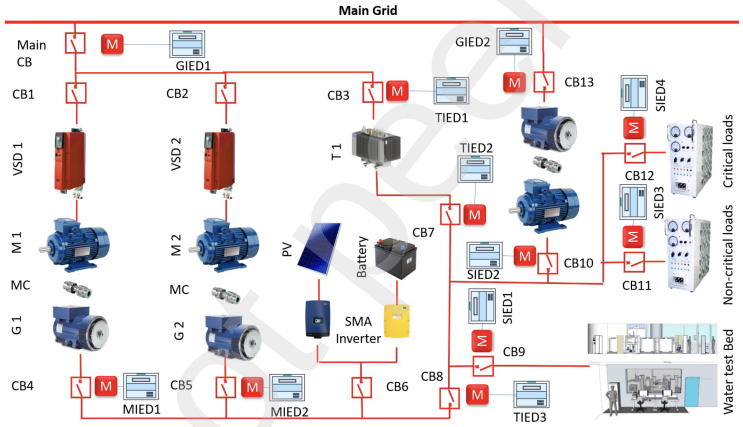

Fig. 4: *Electric layout in EPIC* architecture. Four areas can be distinguished: Generation, Micro-Grid, Transmission, Smart Home; prefixes match device names. (Image credit: *EPICTWIN* architecture by Kandasamy et al. [51] [1])

Within each stage, every generator is linked to a motor, and control over the motors is facilitated by Variable Frequency Drive (VFD) devices. The power line is capable of being disconnected from the main grid through Circuit Breakers, which are operated by an array of IEDs.

4.4. EPICTWIN

EPIC's architecture serves as an excellent model for reproducing cybersecurity scenarios, but its reliance on costly physical components presents challenges for experiment reproduction. To address these limitations, Kandasamy et al. (2022) [51] leverage digital twin technology to achieve full virtualization of the testbed. Physical components with electrical connections are simulated using MATLAB-SIMULINK in real mode. Meanwhile, PLCs, IEDs, and network

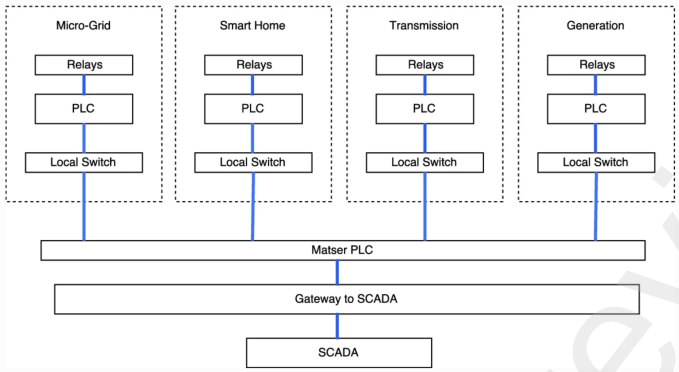

Fig. 5: EPIC Architecture

switches are virtualized through a Linux virtual machine. Specifically, the network is implemented using Open virtual switches deployed within the virtual machine. The SCADA system and all workflows connecting the components are constructed using Node-Red, a visual programming tool.

Leveraging container-based virtualization [58] for reproducing digital twins offers numerous advantages in terms of time and resource efficiency, as highlighted in Lin et al. (2021) [59]. In EPICTWIN, the authors suggest limitations in employing lightweight virtualization for replicating specific ICS cybersecurity attacks, like physical *Man In The Middle* (MITM). However, we demonstrate overcoming this constraint by utilizing Macvlan network drivers in the network interfaces of containers. Through this method, all virtual components operate on a compact Raspberry Pi embedded in the SCASS cabinet. Section 7 provides a practical demonstration of MITM attacks against the containerized SCASS architecture.

5. The SCASS architecture

This section focuses on the design of the SCASS target architecture. SCASS is implemented on a hybrid testbed, composed of physical and virtual components, which operates within a single cabinet. In particular, this cabinet houses a collection of industrial devices collaborating with virtual components to simulate a smart grid industrial environment. The cabinet itself was generously provided by COR, an Italian cyberwarfare joint military unit that sponsored the project. A hybrid architecture offers cost and size reduction benefits while maintaining a certain level of realism in simulations. This is achieved by incorporating physical components within the testbed, such as PLCs and RTUs. By having real physical devices executed in the testbed, the simulations can accurately mimic the behavior and interactions of physical industrial systems, enhancing the authenticity of the experiments. Furthermore, the presence of virtual devices reduces the implementation costs and allows more flexibility in terms of cyber attacks ICSs scenarios.

5.1. Physical components

The cabinet (Figure 6) includes the following physical components:

- IDS-409-1SFP Industrial Managed Switch by Perle Systems Inc.;
- Two UNO-PS/1AC/24DC/60W Power supply units by Phoenix Contact;
- Four PM554-TP-ETH PLCs by ABB Ltd (Figure 7);
- Raspberry PI4, equipped with a 64GB SD card by Samsung;
- C60N magneto thermic switch by Merlin Gerin (Schneider SA);
- 12HD7 monitor by Beetronic;
- Stabilized power supply by Mean Well.

The monitor is connected to the Raspberry Pi, which implements virtual elements and manages the virtual networks. The PM554-TP-ETH is a low to mid-end PLC that includes an AC500 e-Co CPU, an Industrial Ethernet port, an RS-485 serial port, and a set of six digital inputs and eight digital outputs. Additionally, accessories can be mounted to expand the capabilities of the device. PLCs can be configured through the ABB Automation Builder IDE described in Section 1. In particular, with Automation Builder and CodeSys, developers can configure and program the PM554-TP-ETH PLC, customize its functionality, and integrate it into larger automation systems efficiently. In fact, the base SCASS architecture can be arbitrarily extended by configuring the physical PLCs to control either virtualized or physical components and realize any testbed. The cabinet is made of metal and has DIN mountings.

5.2. Virtual Components

The Raspberry Pi runs a Docker daemon [60] facilitating the deployment of lightweight containers. It can establish virtual networks and container elements acting as either routers or PLC components, depending on the chosen Docker image.

We define different Docker images:

- "IED" is a Docker image implemented in Python, functioning as a Modbus server that awaits commands on the 501 TCP port. The main.py file manages the state refreshment of the "modbus_server", wherein the latter maintains a register that can assume values of 0 or 1, reflecting the circuit breaker state.
- "Router" refers to the router image used in EPIC replication. Two instances of the router element are employed: one connects the Controller PLC with the PLCs workers, and the other links the Controller PLC with the SCADA system.

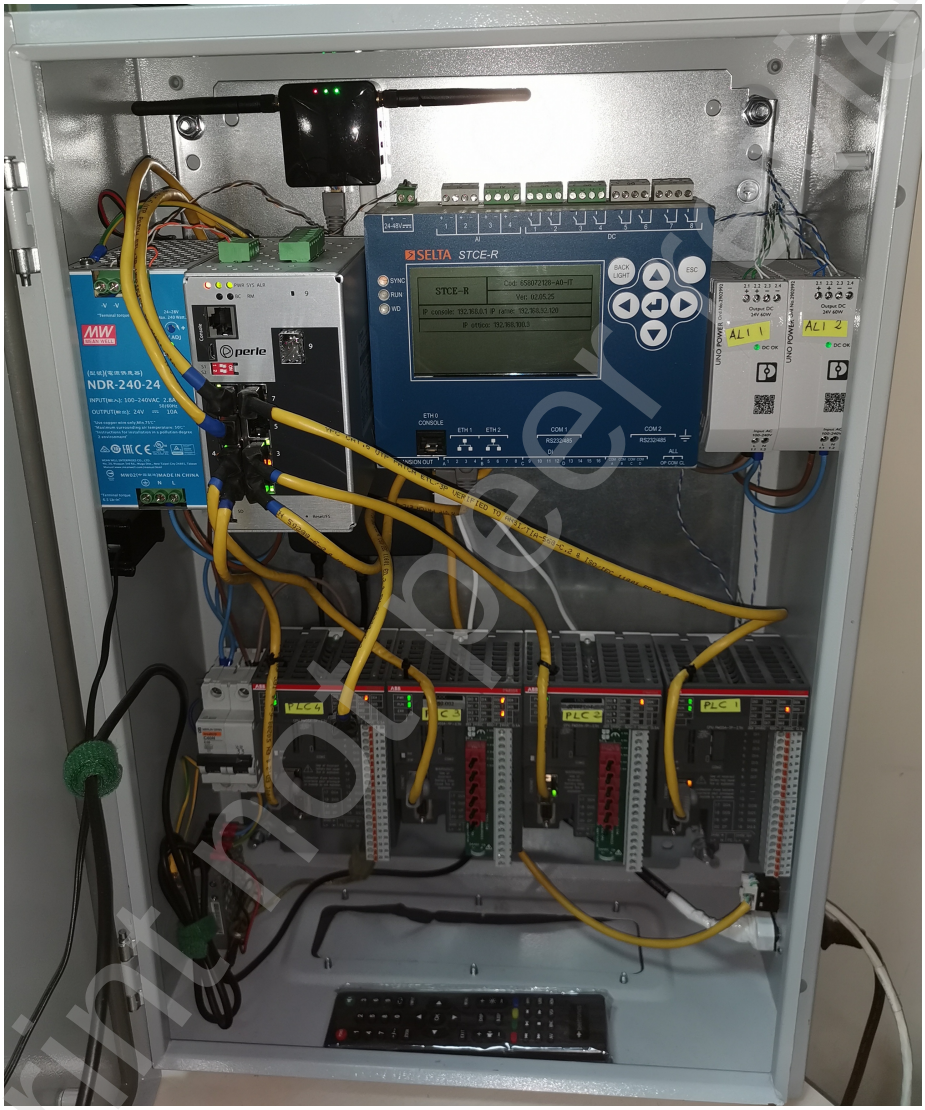

Fig. 6: The cabinet: the array of PLCs can be observed below; the Raspberry PI 4 is located under the industrial switch.

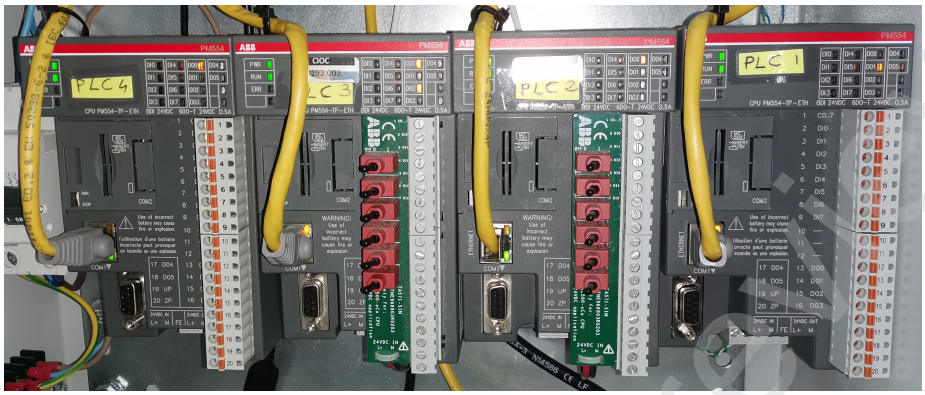

Fig. 7: The PLCs array.

- The "SCADA-System" container simulates a SCADA system. It is not utilized for more complex scenarios, as the SCADA system can be replicated with virtual machines outside the SCASS cabinet to reproduce enterprise cyber-attack scenarios.

The logic of virtual components is realized through Node-RED, a flow-based programming tool commonly used to model industrial processes [61] [62] [63]. Through Node-RED, it is possible to execute logic blocks that simulate the ICS process. Virtual components are implemented using Python programs, while the communication through components is implemented with the Pymodbus library [64]. Namely, each virtual PLC acts as a "Modbus server" and listens for incoming connections.

5.3. SCASS and the Purdue model

Figure 8 shows the relationship between the SCASS system and the Purdue model described in Section 1. The SCASS cabinet is connected to external systems via the OUT line, as described in the previous section. When properly equipped with physical components, the SCASS cabinet can simulate the first three levels of the Purdue model. Levels two and three can also be simulated through external components that cover the industrial DMZ, levels four and five. The OUT line signifies the physical networking connection between the SCASS cabinet and external systems, as will be further explained in Section 5.4. This setup enables the implementation of complex attack scenarios.

5.4. Networking

The interconnection of SCASS components is facilitated by the physical switch, as illustrated in Figure 9, representing the data link layer of the network communication. The IDS-404-1SFP Industrial Managed Switch is responsible for managing the physical communication, interconnecting all the physical PLCs in the cabinet.

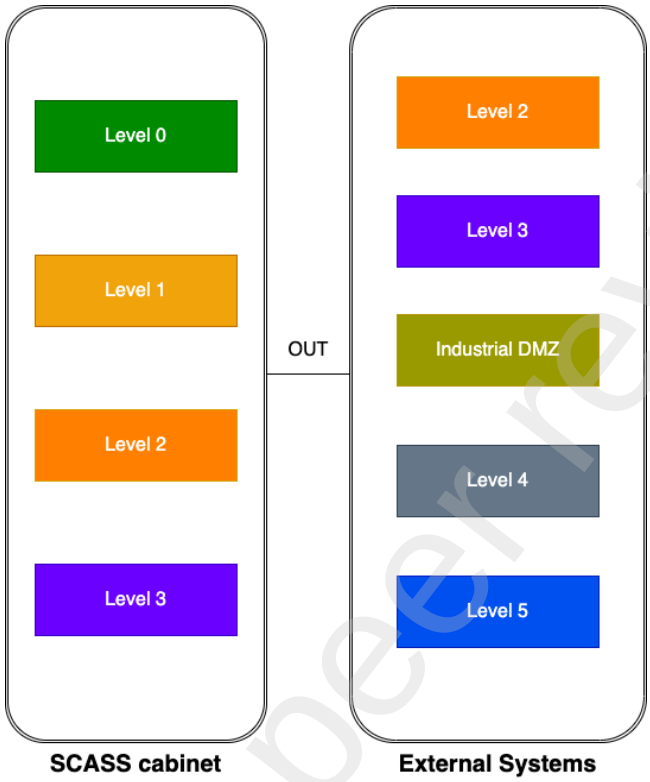

Fig. 8: SCASS and the Purdue model

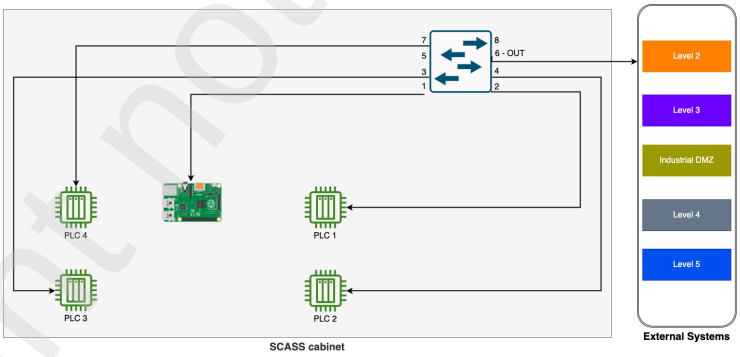

Fig. 9: SCASS networking - level 2

Each PLC is connected to the switch, establishing a connection with the physical network interface of the Raspberry Pi. The network layer is orchestrated by virtual containers running on the Raspberry Pi. VLANs are configured within the physical switch, with each physical PLC residing on a specific VLAN. An

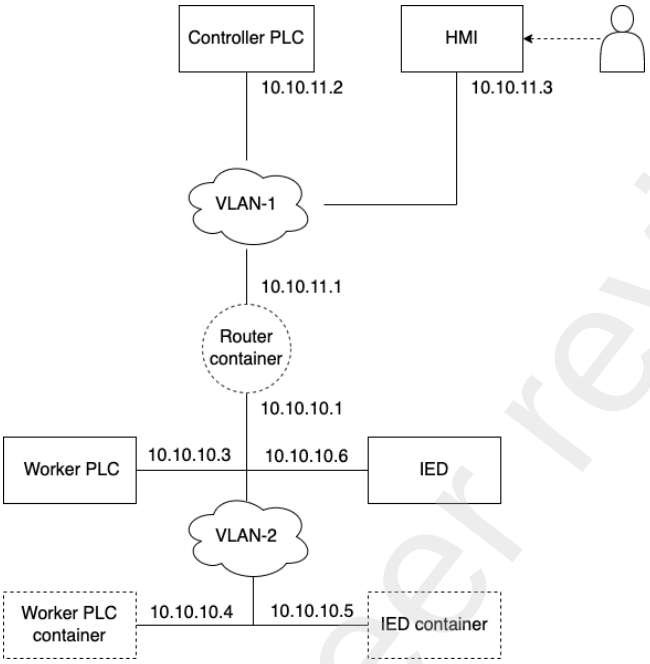

Fig. 10: SCASS networking layer example

external switch line, labeled as the "OUT" line, connects to the internal physical switch, extending the connectivity externally. This configuration allows the creation of heterogeneous networks, enabling the incorporation of various elements such as virtual servers emulating the SCADA system, physically connected components, etc.

The networking layer is implemented using the Docker networking stack, where Docker engine manages VLANs, and the Macvlan network driver assigns each Docker container an IP address associated with a specific VLAN. This approach facilitates the establishment of a unified networking stack, seamlessly interconnecting physical and virtual components. Certain containers serve as default gateways, enabling the construction of intricate infrastructures comprising multiple networks.

Figure 10 provides an illustrative example of the networking configuration that interconnects both virtual and physical components. In this example, the SCASS switch is configured with two VLANs, realizing a controller-worker architecture commonly adopted in SCADA systems.

The HMI element is under the control of a human operator and resides in the same network as the controller PLC, specifically in VLAN-1. The controller PLC is responsible for issuing commands to the worker PLCs situated within the internal network, denoted as VLAN-2. The internal PLCs manage the internal IEDs, which can be either containerized or physical. The HMI can be positioned within the cabinet, facilitated by a Docker container, or it may be an external

machine linked to the internal switch and engaged in communication through the implemented VLANs. All communication transpires through the Modbus protocol.

5.4.1. Containers and cyber-attack scenarios

The Raspberry Pi, being a compact embedded device, benefits from an efficient virtualization environment, making container-based virtualization an ideal choice. While container-based virtualization offers numerous advantages, it does present certain challenges. Notably, lightweight virtualization techniques may struggle to replicate specific attack scenarios involving Windows machines or exploiting vulnerabilities in the Linux kernel. One significant challenge highlighted by Kandasamy et al. (2021) [1] is the difficulty in simulating Man-in-the-Middle (MITM) attacks using containers, particularly those involving the spoofing of physical addresses to the data link layer. MITM attacks are crucial for replicating cyber-attacks against ICSs, as they often leverage MITM techniques at the data link layer. To address this issue, we leverage the Macvlan network driver in Docker, a solution we previously employed in our work [65]. This feature allows the creation of a shared network that connects physical and virtual components. The Macvlan network driver provides containers with a virtual Network Interface Controller (NIC) that emulates a direct connection to the physical network. By integrating concepts from our previous work, we establish a networking stack that seamlessly connects physical and lightweight virtual elements.

On the Raspberry Pi, virtual networks and elements are generated through Docker virtualization. Containers, configured with the macvlan network driver, acquire a pseudo-physical MAC address and an IP address that is visible to each element connected to the internal switch.

6. Reproducing the EPIC architecture through SCASS

To showcase the system's adaptability, we configure SCASS to replicate the Electric Power and Intelligent Control (EPIC) testbed [2] using a methodology akin to that proposed by Kandasamy et al. (2021) [1]. We outline the recreated infrastructure and subsequently highlight key distinctions from our approach. The fundamental idea behind the design is to establish a foundational structure that can be duplicated in a branch layout, as illustrated in Figure 11. In this configuration, distinct devices are assigned specific responsibilities.

Each branch represents an independent power line that operates separately from the others. It consists of a virtual device acting as an IED, a physical PLC responsible for the control loop, and a virtual PLC that serves as the operator terminal device. The virtual PLC activates or deactivates specific system branches and collects data from the connected IED-generator components. An additional PLC is configured as a HMI for testing purposes. The devices in the primary control system are configured to communicate using Modbus messages. Additionally, ABB PLCs support using *User Datagram Protocol*

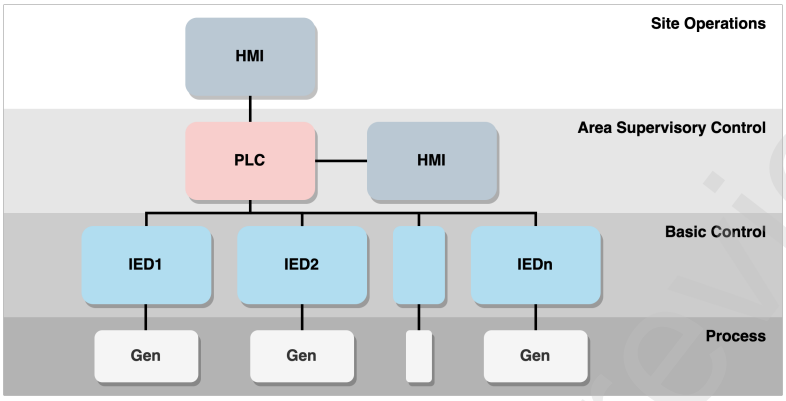

Fig. 11: SCASS target architecture: every basic control branch can be duplicated.

(UDP) for synchronizing device memory areas. This enables efficient and reliable data exchange between the devices, enhancing the overall functionality and coordination of the control system. A modular approach has been adopted to facilitate the replication of power lines. In the physical PLCs, a combination of *Functional Block Diagram* (FBD) and *Structured Text* (ST) code blocks has been utilized to construct intricate functional units. Each functional unit manages communication with every deployed IED-generator branch and the HMI. In this setup, each IED-Generator block corresponds to a Node-RED instance within a Docker container. This design choice allows for easy system scalability since replicating power lines can be performed by just deploying multiple Node-RED instances and duplicating the necessary functional blocks within the PLC firmware. By employing modularity and leveraging the flexibility of Node-RED within Docker containers, the replication and expansion of power lines can be accomplished effortlessly. This approach streamlines the deployment process and enhances the scalability of the system. The operator terminal is configured to perform the following functions:

- activate/deactivate IED devices by sending messages to the physical PLC;
- receive information about the state of power line failures;
- trigger the shutdown of a specific generator if its failure state persists for a specific duration;
- gather operational data from the generators.

The area supervisory control device is responsible for the following:

- maintaining information about the failure states of the generators;
- relaying messages between the area supervisory control device and the IEDs;

- performing monitoring and control actions to maintain a steady state in the generators;
- sending periodic updates regarding operational data collected from the power lines to an HMI device.

The IEDs are configured to:

- measure the energy produced by the generators;
- activate the circuit breaker when the energy production exceeds a predefined threshold;
- update the failure state for the relevant power line.

These configurations facilitate seamless communication, monitoring, and control within the system, ensuring efficient management of power lines and their associated generators.

6.1. The virtual environment

The virtual components are created using the "Infrastructure as Code (IaC)" methodology [66]. Specifically, a "docker-compose"[2] file is employed to define the virtual components. In the compose file, the following aspects are specified:

- one IED for the generation phase;
- three IEDs for the transmission phase;
- four IEDs for the smart-home phase;
- two IEDs for micro-grid phase.

6.2. EPIC Networking

The networking is planned based on the concepts elucidated in Section 5.4. Six VLANs are defined in the physical switch and mapped to virtual networks generated through the Docker engine. Physical elements connect to tagged VLAN ports, while virtual components acquire the designated VLAN through the Docker configuration. Specifically, the Raspberry Pi is configured to access all VLANs in trunk mode, enabling a generic configuration that links virtual and physical devices. Virtual networks have 28-bit subnet masks and are organized as follows:

- "scass-microgrid": 192.168.72.16-192.168.72.31
- "scass-smarthome": 192.168.72.32-192.168.72.47
- "scass-trasmission": 192.168.72.48-192.168.72.63

[2] https://docs.docker.com/compose/

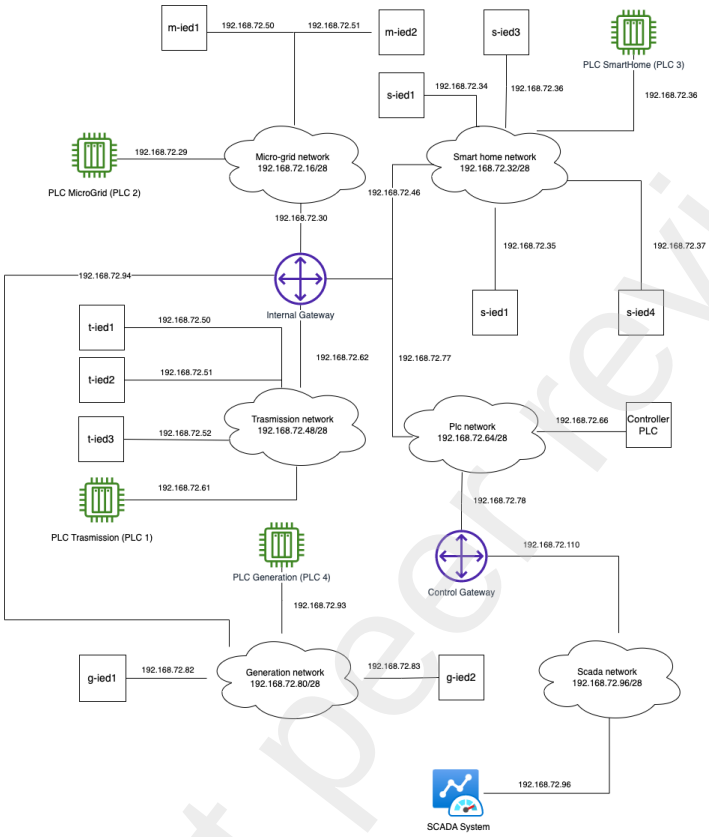

Fig. 12: EPIC networking through SCASS

- "scass-plc": 192.168.72.64-192.168.72.79
- "scass-generation": 192.168.72.80-192.168.72.95
- "scass-scada": 192.168.72.96-192.168.72.127

The network architecture is reported in Figure 12. Two router containers serve as default gateways for the multiple VLANs defined in the physical switch. The internal gateway segregates the four networks, enabling the realization of the four stages in the EPIC architecture. Each network features a physical PLC controlling a set of virtual IEDs simulating circuit breakers. The Control Gateway separates the SCADA environment from the other networks, facilitating communication between the SCADA system and the Controller PLC. Figure 13 illustrates the communication flows between components, utilizing a controller-worker paradigm. The SCASS system issues commands to the containerized Controller PLC, which oversees physical PLCs. Each PLC manages multiple virtual IEDs, emulating the EPIC architecture.

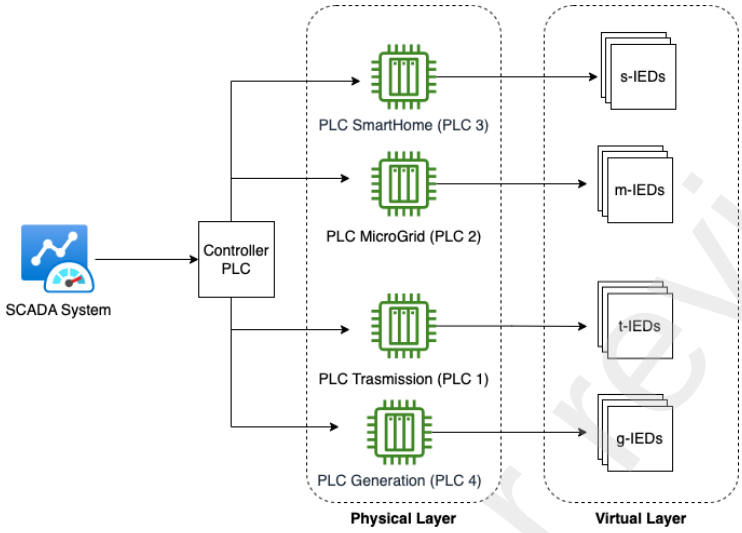

Fig. 13: EPIC components' flows

Communications within the system rely on the Modbus protocol. The physical PLCs are programmed using CodeSys, whereas the virtual elements are created using Python modules that implement Modbus servers, as detailed in Section 5.

6.3. Differences with EPICTWIN

The SCASS architecture differs from the EPICTWIN testbed in various aspects. Table 2 provides a comparison of the EPICTWIN and SCASS implementations.

EPIC testbed's element	**EPICTWIN**	**SCASS Implementation**
Network switch	Ubuntu VM (OVS + GRE)	Docker container in Raspberry Pi
PLC controller	Ubuntu VM	Physical (except for Controller PLC, not present in ET)
IED controllers	Ubuntu VM	Docker containers in Raspberry Pi
AMI controllers	Docker	Not implemented
SCADA	node-red process in Ubuntu VM	node-red process in Windows VM

Table 2: EPICTWIN - SCASS comparison

As evident, our implementation extensively employs container-based virtualization for networking and controllers. In EPICTWIN, the authors acknowledge

the advantages of Docker virtualization but face limitations when using containers to fully replicate a network stack environment capable of reproducing MITM attacks. As mentioned earlier, we utilize the Docker Macvlan network driver to directly connect the container to the physical network, fully virtualizing the network stack and connecting real PLCs with virtual components in the Raspberry Pi.

This approach allows us to leverage the benefits of container-based virtualization, including performance, scalability, and portability, to reproduce the networking stack. The networking stack runs on a small Raspberry Pi, and the entire infrastructure is contained in the compact cabinet shown in Figure 6.

We do not emulate the AMI controllers since we do not simulate physical processes. SCASS employs physical PLCs that can connect to physical devices, offering a faithful replication of a physical environment. A significant distinction in our EPIC replication is the inclusion of the Controller PLC. Our system replicates the typical "controller-workers" architecture presented in the EPIC testbed and illustrated in Figure 14.

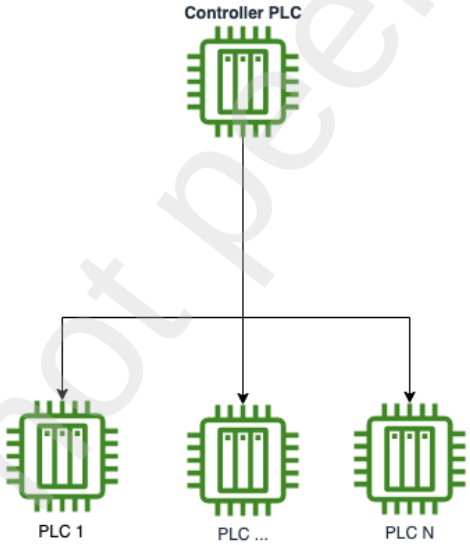

Fig. 14: Controller PLC and worker PLCs

In the EPICTWIN architecture, the "Controller PLC" does not seem to be present. The clarity regarding the presence or absence of such element in the cited architecture may require further clarification.

The distinction between the two architectures is depicted in Figure 15. EPICTWIN employs Openvswitch technology for virtualization, while our approach utilizes the Docker networking layer, configured to align with the physical switch containing multiple VLANs. This choice in networking technologies highlights the flexibility and adaptability of virtualization solutions across different implementations.

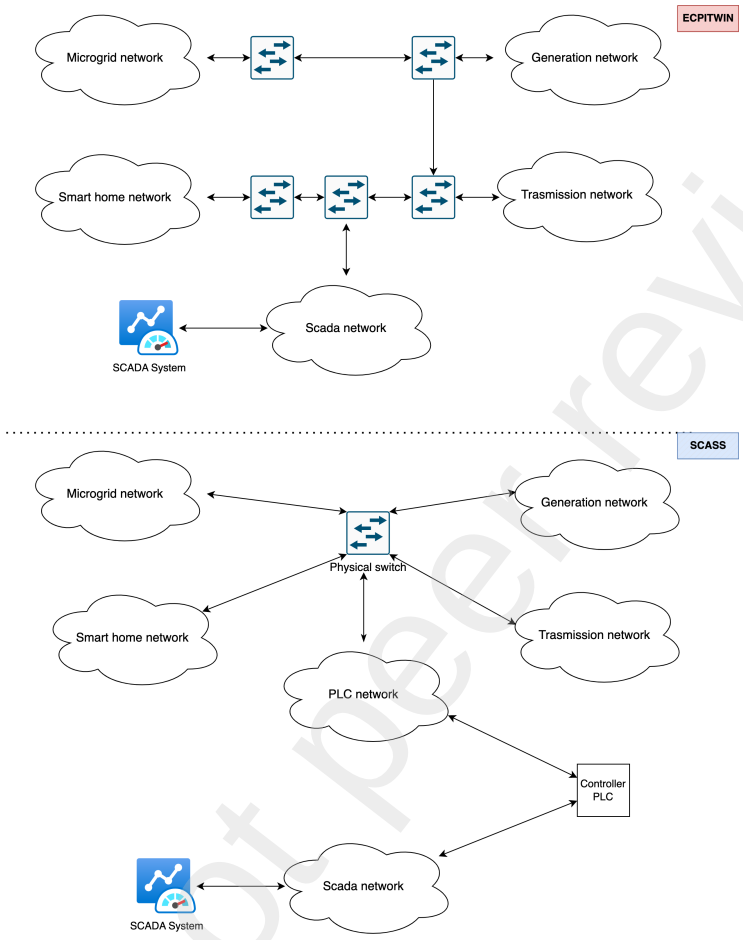

Fig. 15: EPICTWIN versus SCASS networking

The implementation details of the EPICTWIN infrastructure are regrettably not publicly available. However, to facilitate the replication and extension of the SCASS architecture within the community, we have released the source code, encompassing both real and virtual PLCs. The source code for the EPIC replication is openly accessible on GitHub[3]. This allows security researchers to enhance the foundational EPIC infrastructure for replicating more intricate scenarios.

[3]https://github.com/NS-unina/SCASS

7. Breaking into SCASS System Security

In this section, we demonstrate the capability of the SCASS system to replicate realistic scenarios in SCADA systems. This involves the entire process, from identifying an attack path for the target infrastructure to executing attack operations.

7.1. Threat analysis

A threat analysis utilizing the STAMP/STPA method has been conducted to identify vulnerabilities within SCASS. As a result, two successful attacks have been generated against the infrastructure: False Data Injection (FDI) and Denial of Service (DoS). Specifically, the DoS attack targeted the area supervisory control device, compromising its functionality and availability. The threat analysis operations performed are based on a variation of the method presented by Castiglione et al. [34]. The original method, utilized to model threats on the EPICTWIN testbed, has been adapted and applied in the current research to analyze and assess threats in the SCASS infrastructure. By leveraging the insights and framework provided by Castiglione et al., this study aims to effectively identify and address potential security vulnerabilities and threats within the SCASS system. The STAMP model for SCASS can be established by considering the components of the target infrastructure. In this model, the control chain of the system is represented as a sequence of associations, as illustrated in Figure 9. These associations are established based on the specific responsibilities assigned to each individual component.

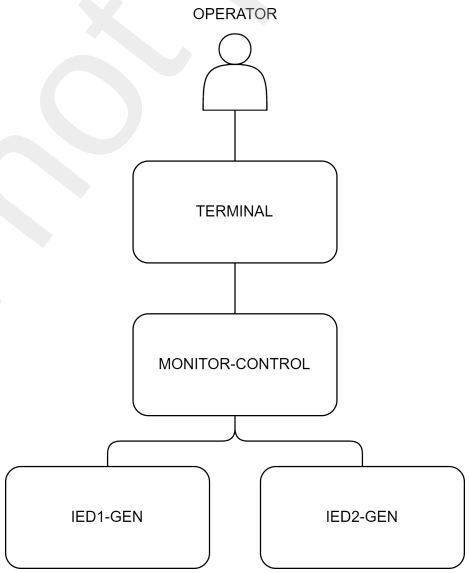

Fig. 16: Entities associations to identify the control chain in SCASS.

By structuring the model in this manner, the relationships and interactions between the components within the SCASS system can be effectively captured and analyzed. The STAMP model enables a comprehensive understanding of how accidents or security threats can propagate throughout the system, allowing for the identification of potential vulnerabilities and the development of appropriate countermeasures. Once the control chain has been identified, the next step involves executing the hazard analysis using STPA. This analysis consists of a series of steps, with the initial step being the identification of accidents and hazards.

Step 1: identify Accidents and Hazards

The first step entails identifying potential accidents and hazards within the target architecture. Accidents refer to unintended and undesired events that can lead to adverse consequences, while hazards are conditions or situations that can contribute to the occurrence of accidents. Leveson [67] provides a definition that can serve as a guiding principle for identifying accidents and hazards in this context.

> *"[...] An undesired and unplanned event that results in a loss, including a loss of human life or human injury, property damage, environmental pollution, mission loss, financial loss, etc."*
>
> *(N. Leveson, Engineering a Safer World)*

It is crucial to thoroughly examine the system, considering various scenarios and potential points of failure, in order to identify possible accidents and associated hazards. By effectively identifying accidents and hazards, the analysis can progress to subsequent steps, facilitating a comprehensive understanding of potential risks and vulnerabilities within the system. The accident list is as follows:

- A_1: uncontrolled generator overload (with damage);
- A_2: undesired generator shutdown.

A hazard is

> *"[...] A system state or a set of conditions that together with a worst-case set of environmental conditions will lead to an accident (loss)"*
>
> *(N. Leveson, Engineering a Safer World)*

A list of hazards is given below:

- H_1: TERMINAL not synchronized with physical process;
- H_2: MONITOR-CONTROL not synchronized with physical process;
- H_3: unsafe configuration of MONITOR-CONTROL;
- H_4: unsafe configuration of IED1;
- H_5: unsafe configuration of IED2.

Step 2: define the control structure

In this step, Control Actions and Feedback will be specified, starting from the STAMP model. The diagram can be extended by specifying control actions and feedbacks for the system (Figure 17).

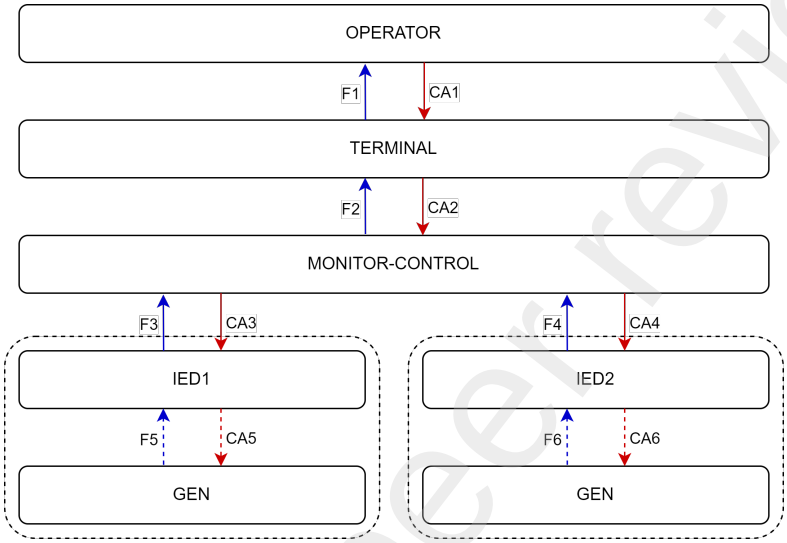

Fig. 17: STAMP model, identifying *Control Actions* and *Feedback*.

The Control Actions for the system are:

- CA_1: send generators start/shutdown, IEDs start/shutdown signals;
- CA_2: relay generators start/shutdown, IEDs start/shutdown signals;
- CA_3: apply generators start/shutdown, increase/decrease signals, IED1 start/shutdown signals;
- CA_4: apply generators start/shutdown, increase/decrease signals, IED2 start/shutdown signals.

The Feedbacks are:

- F_1: receive measurements from generators, state from IEDs;
- F_2: relay measurements from generators, state from IEDs;
- F_3: IED1 state, measurements from the generator controlled by IED1;
- F_4: IED2 state, measurements from the generator controlled by IED2.

Step 3: identify Hazardous Control Actions

The Hazardous Control Actions leading to accidents A1, A2 are listed below:

- HCA_1: CA_1 is applied with incorrect parameters, closing the generator controlled by IED1 -> H_3, H_4;
- HCA_2: CA_1 is applied with incorrect parameters, closing the generator controlled by IED2 -> H_3, H_5.

HCA1 and HCA2 were identified as the closest controls to the process. The hierarchy in the STAMP model can be climbed to identify the remaining hazardous controls.

- HCA_3: CA_2 is applied with incorrect parameters, shutting down the generator -> H_3, H_4, H_5;
- HCA_4: F_2 is applied with incorrect parameters, giving incorrect feedback to the operator -> H_1;
- HCA_5: F_3 is applied with incorrect parameters, giving incorrect feedback to the area supervisory control device -> H_1, H_2;
- HCA_6: F_4 is applied with incorrect parameters, giving incorrect feedback to the area supervisory control device -> H_1, H_2;
- HCA_7: CA_3 is applied with incorrect parameters, causing the generator to deviate from steady state -> H_4;
- HCA_8: CA_4 is applied with incorrect parameters, causing the generator to deviate from steady state -> H_5.

Step 4: associate attack steps with Hazardous Control Actions

In this phase, attack steps will be associated with HCAs. This operation leads to the definition of a list of attack steps. An attack step is a tuple $a_i = (t, e, v)_i$ where:

- t is the type of threat;
- e is the targeted control action or feedback;
- v is the value injected.

Values can be injected through forgery; the element can be a Control Action or a Feedback and therefore, respectively, a command or a measurement. Possible attack steps against SCASS can be grouped in a table (Table 3: Attack steps and related HCAs) and associated with HCAs.

These attack steps can be considered as possible attack goals for an attack path.

Attack step	Description	HCAs
a_1	$(forgery, CA_1, cmd)$	*1, 2*
a_2	$(forgery, CA_2, cmd)$	*3*
a_3	$(forgery, CA_3, cmd)$	*7*
a_4	$(forgery, CA_4, cmd)$	*8*
a_5	$(forgery, F_1, \bar{v})$	*4*
a_6	$(forgery, F_2, \bar{v})$	*5*
a_7	$(forgery, F_3, \bar{v})$	*6*

Table 3: Attack steps and related *Hazardous Control Actions*.

7.2. Deriving the attack graph with MulVal

The attack graph for SCASS (Figure 11) can be derived using *MulVAL* [68], a logic-based enterprise network security analyzer. MulVAL requires a rule file and an input file.

- the *rule* file (*interaction_ rules_ def.P*) contains the rules to generate the attack graph;
- the *input* file (*scass.pl*) describes elements in the network and their vulnerabilities as well.

Rules can be defined starting from primitives. The following is a group of custom primitives or the exploit preconditions.

```
primitive(controlsFlow(_host, _flow)).
primitive(inSubnet(_host, _subnet)).
primitive(transportsFlow(_host, _flow)).
primitive(protocol(_flow, _encrypted)).
primitive(isGateway(_host)).
primitive(controlFlow(_source, _destination, _flow)).
primitive(feedbackFlow(_source, _destination, _flow)).
primitive(l2Discovery(_host, _protocol)).
```

The following is a list of custom derivatives or the exploits postconditions.

```
derived(canTamper(_flow)).
derived(canForgeSegment(_flow)).
derived(arpPoisoning(_host1, _host2)).
derived(lossVisibility(_flow)).
derived(netVisibility(_flow)).
```

The following is a rule description.

```
interaction_rule(
   (canForgeSegment(Flow) :-
    arpPoisoning(H, K),
    protocol(Flow, plaintext),
    controlFlow(H, K, Flow)),
   rule_desc('Segment forged', 'certain')).
```

As an example, `canForgeSegment` can occur when the attacker has poisoned H and K, the protocol is not ciphered and there is a control flow logic between the hosts. The attacker location and the attacker goals can be specified in the input file.

```
attackGoal(canForgeSegment(feedbackAction2)).
attackGoal(canForgeSegment(controlAction3)).

attackGoal(lossVisibility(feedbackAction3)).

attackerLocated(scassNet).
hacl(_,_,_,_).
hacl(X,Y,_,_):-
 inSubnet(X,S),
 inSubnet(Y,S).
```

The attacker's goal is to forge or modify segments on the area supervisory control device and the operator terminal, allowing them to potentially manipulate control processes and gather unauthorized access to critical information. Additionally, the attacker aims to ensure that any feedback generated by the IEDs remains hidden or altered, effectively concealing their malicious activities. By considering the presence of the attacker within the OT network, it is crucial to thoroughly evaluate and fortify the network's security measures to prevent unauthorized access, detect any suspicious activities, and mitigate potential risks. The following lines of code define devices in an OT network — `scassNet` — and specify how ARP is the protocol used to execute level 2 discovery.

```
inSubnet(terminal, scassNet).
l2Discovery(terminal, arp).
inSubnet(monitorControl, scassNet).
l2Discovery(monitorControl, arp).
inSubnet(gIed1, scassNet).
l2Discovery(gIed1, arp).
inSubnet(gIed2, scassNet).
l2Discovery(gIed2, arp).
inSubnet(vulnHost, scassNet).
```

The following lines of code enable the specification of control actions and feedback within a control flow in the Operator Terminal (OT) network, along with the corresponding protocol used. In this example, CA_2, CA_3, F_2 and F_3 have been modeled as interactions:

```
controlFlow(terminal, monitorControl, controlAction2).
feedbackFlow(monitorControl, terminal, feedbackAction2).
protocol(controlAction2, plaintext).
protocol(feedbackAction2, plaintext).

controlFlow(monitorControl, gIed2, controlAction3).
feedbackFlow(gIed2, monitorControl, feedbackAction3).
protocol(feedbackAction3, plaintext).
protocol(feedbackAction3, unauthenticated).
```

The protocols used to send commands and feedback are specified as plaintext. The command to generate a graph with MulVAL is (Figure 18):

```
graph_gen.sh scass.pl -v -p -r interaction_rules_def.P
```

The attack path can be represented by connecting leaves, which signify preconditions, with a root or the attack goal. In this context, the leaves represent the necessary conditions for the attack, while the root represents the desired attack goal. Each step towards the attack goal corresponds to a successful exploitation of vulnerabilities, which may have a certain probability of execution.

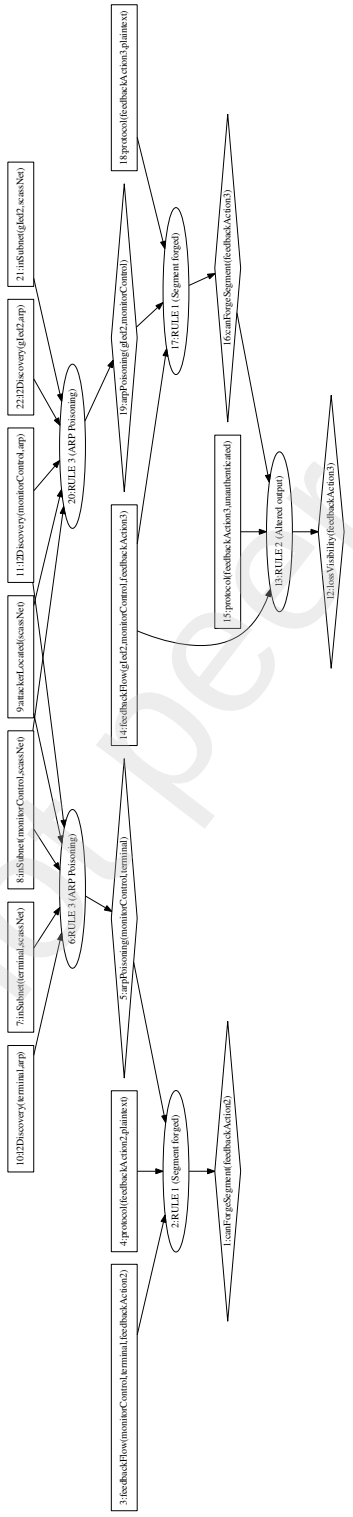

Fig. 18: The SCASS attack graph.

7.2.1. Server Impersonation

On the left side of the tree diagram, a Server Impersonation attack is depicted, aimed at providing false feedback to the terminal operator. The preconditions for this attack include:

- the attacker must be present within the OT network, where the terminal operator and the area supervisory control devices reside;
- the devices involved must support ARP.

To forge feedback to the terminal, the following preconditions must be met:

- the ARP *Poisoning* attack has been successfully executed against the terminal operator and the area supervisory control device;
- there is a data flow between the terminal operator and the supervisory area device;
- the messaging protocol does not incorporate encryption.

By outlining these preconditions and the attack path, it becomes possible to analyze the specific vulnerabilities that may be exploited and the necessary conditions required for the successful execution of the attack.

7.2.2. False Data Injection

The tree on the right models a False Data Injection (FDI) attack. To successfully execute the attack on SCASS, the attacker needs to satisfy specific preconditions within the OT network, where the IEDs and the area supervisory control device are located. The preconditions for the attack include:

- the attacker must have access to the OT network;
- the devices within the network must support Address Resolution Protocol (ARP).

To forge messages to the area supervisory control device, the following preconditions need to be met:

- the ARP Poisoning attack must have been successfully executed;
- a data flow should exist between the IED and the area supervisory control device;
- the protocol being used does not employ encryption.

To disrupt the legitimate feedback from reaching the area supervisory control device, the following preconditions must be fulfilled:

- a data flow between the IED and the area supervisory control device must be present;

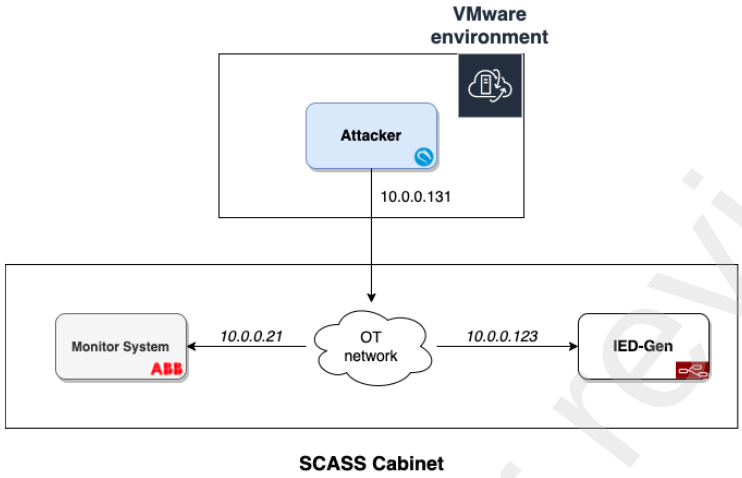

Fig. 19: Deployed Architecture for attacking SCASS

- the attacker should have the ability to forge feedback messages to the area supervisory control device;
- the protocol being utilized lacks authentication mechanisms.

MulVAL, indeed, provides a method to exploit these preconditions and attack SCASS successfully. By understanding and addressing these vulnerabilities, appropriate countermeasures can be implemented to enhance the security and resilience of the SCASS system.

7.3. Attacking SCASS

The approach employed for attacking SCASS focuses on altering the operations of one or more generators within the target architecture using a TCP injection attack. By forging fragments, the attacker can disrupt existing connections and manipulate the monitor-control cycle by injecting false Modbus messages that contain fabricated measurements. It is important to note that all attacks discussed herein assume that the attacker has already compromised a device within the Operational Technology OT zone. The details of how the device was compromised are not within the scope of this work. Additionally, prior to launching the attacks, the attacker performs a discovery phase to identify the functioning devices within the plant. The architecture implemented to replicate an ICS cybersecurity attack is depicted in Figure 19. We configure the SCASS networking with a 24-bit network (i.e., 10.0.0.0/24). The setup includes a physical Supervisory Area Control Device (Monitor System) and a virtual Intelligent Electronic Device (IED) simulating the measurements obtained from a remote sensor.

To conduct the attacks, a Kali Linux [69] VM instance is utilized (the "Attacker" rectangle in Figure 19), running in bridge mode on VMWare Workstation

Player (Windows 11). Kali Linux, maintained by Offensive Security, is a Debian-based operating system primarily used for penetration testing and cybersecurity tasks. It comes with a comprehensive set of preinstalled security tools and software packages, making it a well-rounded solution for security testing and ethical hacking purposes. Scanning activities, such as conducting a simple ping scan, can be performed using tools like nmap (as shown in Figure 20).

```
nmap -sP 10.0.0.0/24
```

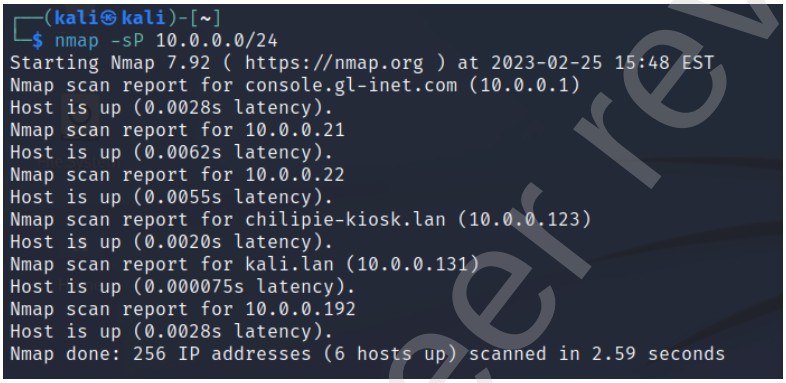

Fig. 20: Ping scanning operations within the SCASS network, which leads to the discovery of two PLCs in the OT network.

In summary, the approach described involves leveraging a Kali Linux VM instance with its built-in security tools to execute TCP injection attacks and manipulate the operations of generators within the target architecture.

Scapy [70] is a widely used network packet manipulation tool based on Python. It provides functionalities for capturing, analyzing, and sending network packets across various layers of the *Open Systems Interconnection* (OSI) model. Scapy can function as a standalone application, but its services can also be imported into Python scripts or programs. In the context of this study, Scapy has been utilized to execute a *Transmission Control Protocol* (TCP) Injection attack for the *False Data Injection* (FDI) attack. Scapy enables the proper forging and transmission of packets to execute the FDI attack.

FDI attacks are carried out by executing a TCP Injection attack within the communication channel between the supervisory control area device and one of the IEDs responsible for controlling the generators. The objective of the attack is to manipulate the data flow, specifically introducing false measurements into the physical PLC, thereby causing incorrect commands to be generated. Prior to launching the attack on the target infrastructure, the attacker is required to listen to the conversations between the devices to gather information and devise an effective assault strategy. To achieve this, the attacker will perform a MITM attack, intercepting and eavesdropping the communications between the devices. Once the necessary information has been gathered, the attacker will initiate a Scapy script to exploit a preexisting TCP connection and inject

false data into the target system. By introducing this manipulated data, the attack is executed, generating erroneous commands within the physical PLC. This sequence of events outlines conducting an FDI attack. It emphasizes the importance of intercepting and manipulating communication channels and utilizing tools like Scapy to introduce false data, ultimately compromising the integrity and reliability of the control system. The first step involves conducting a MITM attack using ARP spoofing and Modbus server impersonation. ARP spoofing is a commonly employed technique in MITM attacks, utilizing the ARP mechanism used by devices to relate network and physical addresses. In this attack, the attacker sends Gratuitous ARP Replies to the parties involved in a communication, poisoning their ARP cache. This leads to associating the attacker's MAC address with the IP address of the victim, allowing the attacker to intercept and read the victim's messages. Various tools can be utilized to automate ARP spoofing operations, such as arpspoof [71] (included in the dsniff suite available in Kali Linux), Ettercap [72], and Bettercap [73]. Alternatively, Python scripts using Scapy packages can also be written to create level 2 spoofers. The supervisory area control device serves as a relay for operational data to the operator terminal. The operator terminal periodically polls the physical PLC to retrieve information regarding failures and energy production. To prevent the injected data from being received by the HMI and potentially alerting plant operators to a malfunction, the attacker can instantiate a TCP proxy. This proxy can redirect Modbus Read Requests intended for the supervisory area control device to a local Modbus server instance. The local Modbus server is initialized with a set of registers containing plausible values for the plant (Figure 23). Creating a rogue Modbus server (Figure 21) can be easily accomplished using Py-Modbus[4], a Python library specifically designed for Modbus communication and manipulation. By employing this setup, the attacker can intercept Modbus Read Requests and redirect them to their own Modbus server, ensuring that the operator terminal receives plausible values and remains unaware of the injected data. This allows the attacker to maintain control and prevent detection while manipulating the operational data within the system.

The second step involves packet sniffing, which becomes possible due to the successful execution of the MITM attack. During this phase, the attacker aims to understand how communications are conducted within the plant, with a focus on discerning control actions. Tools like Wireshark, TShark, and tcpdump can be utilized to capture and analyze network traffic in real-time. These tools allow the attacker to inspect and study the packets exchanged between the devices.

By examining the captured traffic, the attacker may notice (Figure 22) a periodic exchange of values between the IP addresses 10.0.0.123 (the IED) and 10.0.0.21 (the supervisory area control device). The attacker may infer that the data sent by the IED represents some form of measurement, while the data flowing in the opposite direction may correspond to commands issued by the physical PLC to the IED, aimed at maintaining the steady state of the generator.

[4] `https://pymodbus.readthedocs.io`

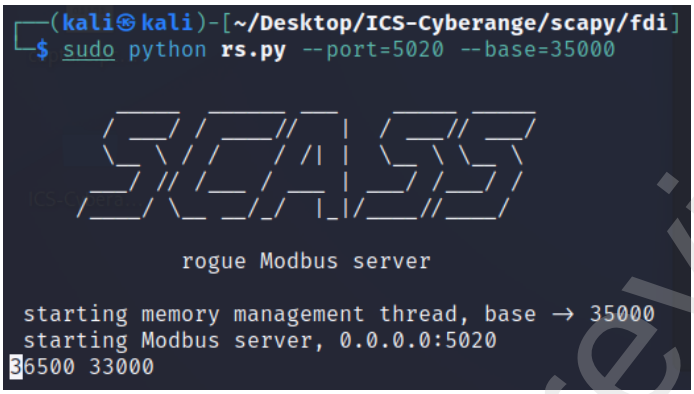

Fig. 21: Rogue Modbus server executing on the attacker's machine.

To better understand the control logic employed in the plant, the attacker can isolate the message sequence using appropriate filters. By focusing on the TCP fragments exchanged between the components, the attacker can gain insights into the control law governing the plant's operation. By leveraging packet sniffing techniques and appropriate tools, the attacker can observe the communication patterns and deduce the control actions taking place within the plant. This information lays the foundation for further analysis and manipulation of the control system.

The third step involves executing TCP Injection to introduce false measurements into the physical PLC, thereby triggering the generation of incorrect values. TCP Injection is the process of hijacking an established TCP session between two parties and injecting data into the communication stream. By modifying the contents of TCP fragments, the attacker gains control over the network session and can manipulate the data being transmitted. Each TCP fragment includes a Sequence (SEQ) number that serves to uniquely identify the sent data and an ACK number that confirms receipt and indicates the next expected sequence number, ensuring reliable and ordered data transfer. During TCP Injection, the attacker leverages their knowledge of the control law and the TCP session between the components involved. By injecting manipulated TCP fragments into the communication stream, the attacker can introduce false measurements into the physical PLC, thereby influencing the generation of incorrect values. By exploiting the vulnerabilities in the TCP communication protocol, the attacker effectively disrupts the control process and achieves their objective of manipulating the operations of the physical PLC. The process of TCP Injection involves capturing and analyzing SEQ and ACK numbers from legitimate TCP fragments exchanged between two parties in a conversation. The attacker then utilizes these numbers proactively to craft and send manipulated packets to the target system (as shown in Figure 23).

In this scenario, the attacker can prepare a Scapy program that performs the following actions:

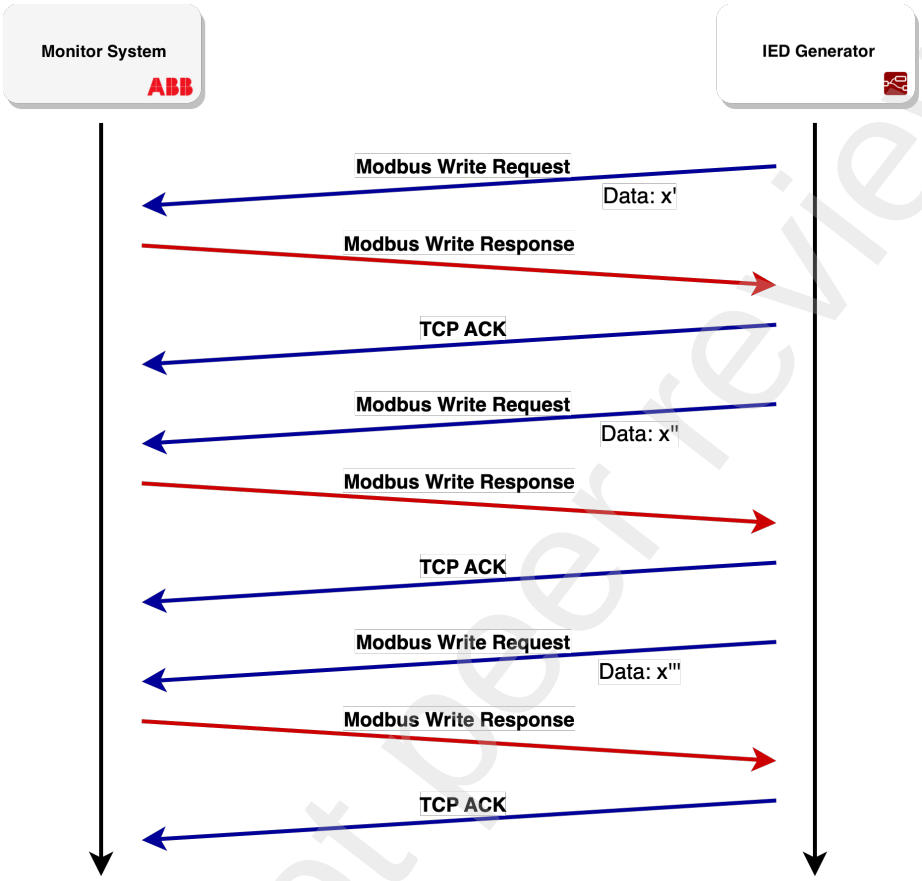

Fig. 22: Modbus conversation between the *Area Supervisory Control* device and *IED1*. Every arrow is a TCP fragment.

1. capturing packets in groups;
2. inspecting each packet in the group to identify Modbus Write Requests sent to the IED. The TCP payload of the packet is compared to a specific string to evaluate its content;
3. searching for the corresponding ACK fragment within the packet group. If found, the SEQ and ACK numbers from this fragment are copied. If not found, the capture process is restarted;
4. generating a Modbus Write Request to be sent to the physical PLC, utilizing the sniffed SEQ and ACK numbers at the TCP level;
5. restarting the packet capture process every time a forged packet is sent within the OT network.

As a result, the monitor-control cycle is altered. The physical PLC within the target architecture receives legitimate Modbus messages but generates commands

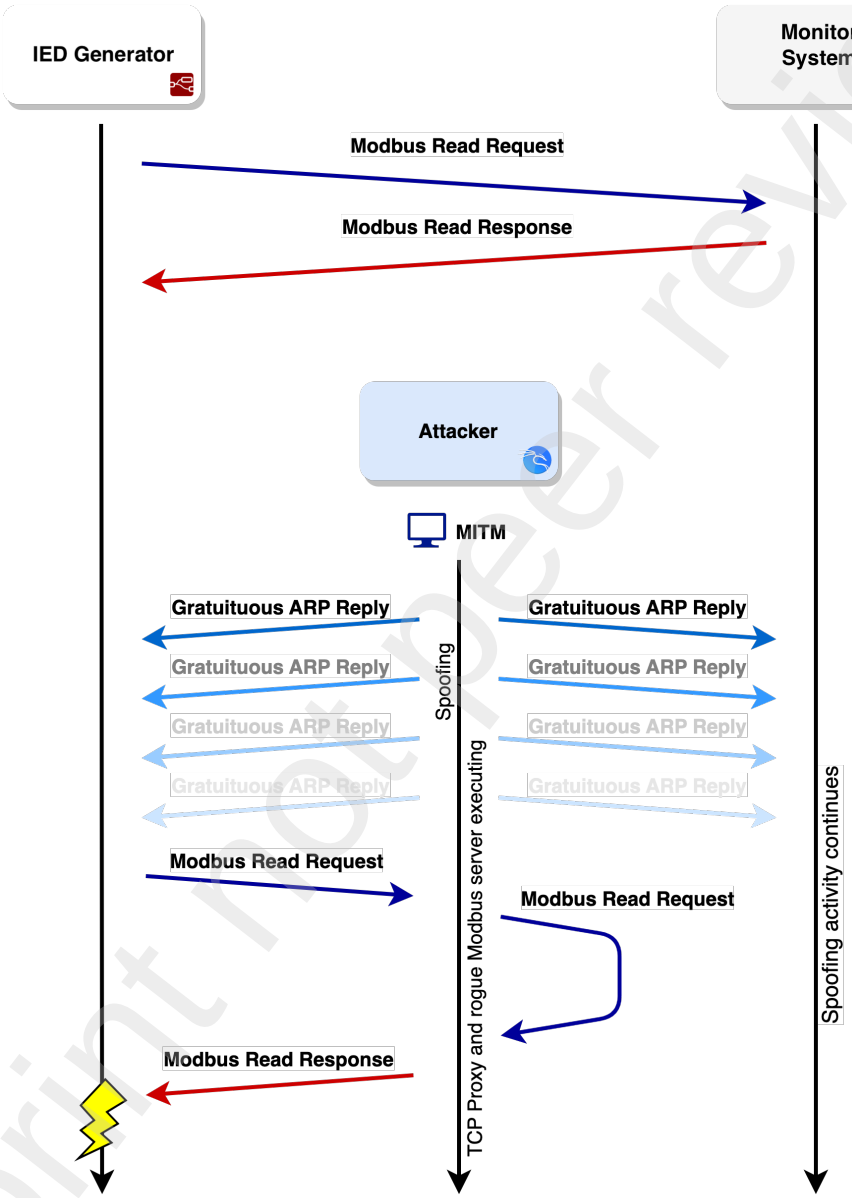

Fig. 23: Modbus server impersonation.

based on false measurements. Consequently, the PLC receives commands to decrease power generation until the Scapy script remains active.

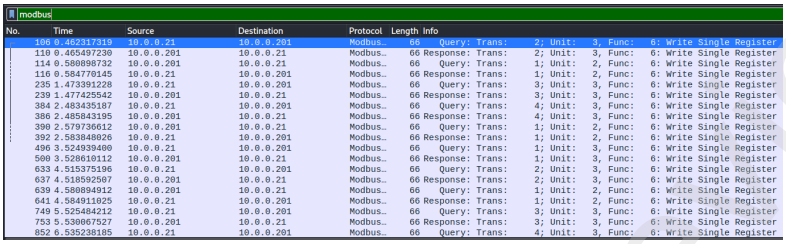

Fig. 24: Modbus packets exchanged between the *Area Supervisory Control* device and *IED1*.

8. Conclusions

This work has introduced SCASS as a robust and versatile architecture for replicating heterogeneous security testbeds. In our future endeavors, we aspire to enhance the capabilities of the testbed by scaling the reproduction of a complete grid within the cabinet, utilizing every available physical device. Additionally, we aim to broaden the scope of the target architecture by incorporating other commonly used protocols in OT systems, such as MQTT or IEC 61850. We plan to diversify the range of both physical and virtual devices, exploring alternatives like IFTTT and n8n.io, and incorporating libraries such as PyModbusTCP and OpenPLC.

In the context of modeling cybersecurity threats in the ICS domain, various methodologies and approaches have been developed [27]. Notably, security contextualization of well-established safety models [28, 29, 32, 35, 36] has been widely adopted. In our upcoming work, we aim to practically apply such methodologies within SCASS to identify comprehensive attack graphs.

Moreover, the SCASS environment stands to benefit from the integration of cyber-range functionalities, including scoring or monitoring systems. The inclusion of these features would facilitate the reproduction of red-team versus blue-team cyber simulation scenarios, offering a holistic testing ground for cybersecurity measures. The continuous evolution and refinement of SCASS align with the dynamic landscape of cybersecurity, ensuring its relevance and effectiveness in addressing emerging threats and challenges in the ICS domain.

CRediT authorship contribution statement

Nicola d'Ambrosio: Methodology, Software, Validation, Formal analysis, Visualization, Writing - Review & Editing. **Giulio Capodagli**: Conceptualization, Writing - Original Draft, Methodology, Software, Validation, Formal analysis, Visualization, Writing - Review & Editing. **Gaetano Perrone**: Conceptualization, Writing - Original Draft, Methodology, Software, Validation,

Formal analysis, Visualization, Writing - Review & Editing. **Simon Pietro Romano**: Project administration, Supervision, Methodology, Writing - Original Draft, Writing - Review & Editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

The development of SCASS has been carried out in collaboration with "Comando per le Operazioni in Rete" (COR), the main Italian military interforce unit specializing in cyber defense and cyber security operations. COR has provided the necessary hardware required for the project's implementation.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] N. K. Kandasamy, S. Venugopalan, T. K. Wong, L. J. Nicholas, Epictwin: An electric power digital twin for cyber security testing, research and education (2021). arXiv:arXiv:2105.04260.

[2] S. Adepu, N. K. Kandasamy, A. Mathur, EPIC: An electric power testbed for research and training in cyber physical systems security, in: Computer Security, Springer International Publishing, 2019, pp. 37–52. doi:10.1007/978-3-030-12786-2_3.
URL https://doi.org/10.1007/978-3-030-12786-2_3

[3] D. Howard, The digital twin: Virtual validation in electronics development and design, in: 2019 Pan Pacific Microelectronics Symposium (Pan Pacific), 2019, pp. 1–9. doi:10.23919/PanPacific.2019.8696712.

[4] P. Ackerman, Industrial Cybersecurity, Packt Publishing Ltd., 2017.

[5] IEEE standard for electric power systems communications-distributed network protocol (DNP3) (2023). doi:10.1109/ieeestd.2012.6327578.
URL https://doi.org/10.1109/ieeestd.2012.6327578

[6] ISO/IEC 20922:2016(en) Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1 (Preview), https://www.iso.org/obp/ui/#iso:std:iso-iec:20922:ed-1:v1:en (2023).

[7] FieldComm Group site, https://www.fieldcommgroup.org/ (2023).

[8] PROFIBUS Technology and Application - System Description, https://www.profibus.com/download/profibus-technology-and-application-system-description (2023).

[9] PROFINET System Description, http://us.profinet.com/wp-content/uploads/2012/11/PROFINET_SystemDescription_ENG_2014_web.pdf (2023).

[10] Modbus Specification and Implementation guide, https://modbus.org/docs/Modbus_over_serial_line_V1.pdf (2023).

[11] Modbus/TCP Security Protocol Specification, https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf (2023).

[12] The Transport Layer Security (TLS) Protocol Version 1.3 RFC, https://datatracker.ietf.org/doc/html/rfc8446 (2023).

[13] F. Yi, L. Zhang, S. Yang, D. Zhao, A security-enhanced modbus TCP protocol and authorized access mechanism, in: 2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC), IEEE, 2021, pp. 61–67. doi:10.1109/dsc53577.2021.00016.
URL https://doi.org/10.1109/dsc53577.2021.00016

[14] J. Schekkerman, How to Survive in the Jungle of Enterprise Architecture Framework: Creating or Choosing an Enterprise Architecture Framework, Trafford, 2003.

[15] Rockwell, Cisco, Converged plantwide ethernet (cpwe) design and implementation guide, Tech. rep., Cisco (2011).

[16] IEC 61131-3:2013, https://webstore.iec.ch/publication/4552 (2023).

[17] CODESTS, Codesys development system, https://help.codesys.com/webapp/_cds_f_development_system_introduction;product=codesys;version=3.5.17.0, online; 27-Jul-2023 (2021).

[18] D. Bhamare, M. Zolanvari, A. Erbad, R. Jain, K. Khan, N. Meskin, Cybersecurity for industrial control systems: A survey, Computers and Security 89 (2020) 101677. doi:https://doi.org/10.1016/j.cose.2019.101677.
URL https://www.sciencedirect.com/science/article/pii/S0167404819302172

[19] R. Langner, Stuxnet: Dissecting a cyberwarfare weapon, IEEE Security and Privacy Magazine 9 (3) (2011) 49–51. doi:10.1109/msp.2011.67.
URL https://doi.org/10.1109/msp.2011.67

[20] D. P. Fidler, Was stuxnet an act of war? decoding a cyberattack, IEEE Security and Privacy Magazine 9 (4) (2011) 56–59. doi:10.1109/msp.2011.96.
URL https://doi.org/10.1109/msp.2011.96

[21] İ. Kara, M. Aydos, The ghost in the system: technical analysis of remote access trojan, International Journal on Information Technologies & Security 11 (1) (2019) 73–84.

[22] O. Foundation, What is opc, https://opcfoundation.org/about/what-is-opc/, online; 27-Jul-2023 (2013).

[23] M. Li, Y. Huang, H. Pan, Research on attack mechanism of network intrusion in industrial control system, in: 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), IEEE, 2019. doi:10.1109/iaeac47372.2019.8997670.
URL https://doi.org/10.1109/iaeac47372.2019.8997670

[24] Wired, Russia's Sandworm Hackers Attempted a Third Blackout in Ukraine, https://www.wired.com/story/sandworm-russia-ukraine-blackout-gru/ (2023).

[25] Drago, CHERNOVITE's PIPEDREAM Malware Targeting Industrial Control Systems (ICS), https://www.dragos.com/blog/industry-news/chernovite-pipedream-malware-targeting-industrial-control-systems/ (2023).

[26] Wired, Fed's uncover a 'Swiss Army Knife' for Hacking Industrial Control Systems, https://www.wired.com/story/pipedream-ics-malware/ (2023).

[27] D. Bhamare, M. Zolanvari, A. Erbad, R. Jain, K. Khan, N. Meskin, Cybersecurity for industrial control systems: A survey, Computers and Security 89 (2020) 101677. doi:10.1016/j.cose.2019.101677.
URL https://doi.org/10.1016/j.cose.2019.101677

[28] S. Kriaa, L. Pietre-Cambacedes, M. Bouissou, Y. Halgand, A survey of approaches combining safety and security for industrial control systems, Reliability Engineering and System Safety 139 (2015) 156–178. doi:10.1016/j.ress.2015.02.008.
URL https://doi.org/10.1016/j.ress.2015.02.008

[29] N. Leveson, A new accident model for engineering safer systems, Safety Science 42 (4) (2004) 237–270. doi:10.1016/s0925-7535(03)00047-x.
URL https://doi.org/10.1016/s0925-7535(03)00047-x

[30] H. Niu, C. Ma, C. Wang, P. Han, Hazard analysis of traffic collision avoidance system based on STAMP model, in: 2018 IEEE International Conference on Progress in Informatics and Computing (PIC), IEEE, 2018. doi:10.1109/pic.2018.8706283.
URL https://doi.org/10.1109/pic.2018.8706283

[31] S. H. Lee, S.-M. Shin, J. S. Hwang, J. Park, Operational vulnerability identification procedure for nuclear facilities using STAMP/STPA, IEEE

Access 8 (2020) 166034–166046. doi:10.1109/access.2020.3021741.
URL https://doi.org/10.1109/access.2020.3021741

[32] F. Yan, T. Tang, H. Yan, Scenario based STPA analysis in automated urban guided transport system, in: 2016 IEEE International Conference on Intelligent Rail Transportation (ICIRT), IEEE, 2016, pp. 425–431. doi:10.1109/icirt.2016.7588764.
URL https://doi.org/10.1109/icirt.2016.7588764

[33] S. Chen, S. Khastgir, P. Jennings, Analyzing national responses to COVID-19 pandemic using STPA, Safety Science 138 (2021) 105195. doi:10.1016/j.ssci.2021.105195.
URL https://doi.org/10.1016/j.ssci.2021.105195

[34] L. M. Castiglione, Z. Hau, P. Get, K. T. Co, L. Munoz-Gonzalez, F. Teng, E. Lupu, HA-grid: Security aware hazard analysis for smart grids, in: 2022 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), IEEE, 2022, pp. 446–452. doi:10.1109/smartgridcomm52983.2022.9961003.
URL https://doi.org/10.1109/smartgridcomm52983.2022.9961003

[35] W. Young, N. Leveson, Systems thinking for safety and security, in: Proceedings of the 29th Annual Computer Security Applications Conference, ACSAC '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 1–8. doi:10.1145/2523649.2530277.
URL https://doi.org/10.1145/2523649.2530277

[36] W. Young, N. G. Leveson, An integrated approach to safety and security based on systems theory, Communications of the ACM 57 (2) (2014) 31–35. doi:10.1145/2556938.
URL https://doi.org/10.1145/2556938

[37] I. Friedberg, K. McLaughlin, P. Smith, D. Laverty, S. Sezer, STPA-SafeSec: Safety and security analysis for cyber-physical systems, Journal of Information Security and Applications 34 (2017) 183–196. doi:10.1016/j.jisa.2016.05.008.
URL https://doi.org/10.1016/j.jisa.2016.05.008

[38] E. Colbert, D. Sullivan, A. Kott, Cyber-physical war gaming.

[39] S. Hernan, S. Lambert, T. Ostwald, A. Shostack, Threat modeling-uncover security design flaws using the stride approach, MSDN Magazine-Louisville (2006) 68–75.

[40] R. Khan, K. McLaughlin, D. Laverty, S. Sezer, STRIDE-based threat modeling for cyber-physical systems, in: 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), IEEE, 2017. doi:10.1109/isgteurope.2017.8260283.
URL https://doi.org/10.1109/isgteurope.2017.8260283

[41] S. Haag, R. Anderl, Digital twin – proof of concept, Manufacturing Letters 15 (2018) 64–66. doi:10.1016/j.mfglet.2018.02.006.
URL https://doi.org/10.1016/j.mfglet.2018.02.006

[42] K. T. Park, J. Lee, H.-J. Kim, S. D. Noh, Digital twin-based cyber physical production system architectural framework for personalized production, The International Journal of Advanced Manufacturing Technology 106 (5) (2020) 1787–1810. doi:10.1007/s00170-019-04653-7.
URL https://doi.org/10.1007/s00170-019-04653-7

[43] H. Zhang, G. Zhang, Q. Yan, Digital twin-driven cyber-physical production system towards smart shop-floor, Journal of Ambient Intelligence and Humanized Computing 10 (11) (2018) 4439–4453. doi:10.1007/s12652-018-1125-4.
URL https://doi.org/10.1007/s12652-018-1125-4

[44] J. C. Olivares-Rojas, E. Reyes-Archundia, J. A. Gutiérrez-Gnecchi, I. Molina-Moreno, J. Cerda-Jacobo, A. Méndez-Patiño, Towards cybersecurity of the smart grid using digital twins, IEEE Internet Computing 26 (3) (2022) 52–57. doi:10.1109/MIC.2021.3063674.

[45] D. Holmes, M. Papathanasaki, L. Maglaras, M. A. Ferrag, S. Nepal, H. Janicke, Digital twins and cyber security – solution or challenge?, in: 2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), 2021, pp. 1–8. doi:10.1109/SEEDA-CECNSM53056.2021.9566277.

[46] S. R. Mark Hearn, Cybersecurity considerations for digital twin, Tech. rep., Industrial Internet Consortium (2019).

[47] E. Ukwandu, M. A. B. Farah, H. Hindy, D. Brosset, D. Kavallieros, R. Atkinson, C. Tachtatzis, M. Bures, I. Andonovic, X. Bellekens, A review of cyber-ranges and test-beds: Current and future trends (2020). arXiv:arXiv:2010.06850.

[48] M. Conti, D. Donadel, F. Turrin, A survey on industrial control system testbeds and datasets for security research, IEEE Communications Surveys and Tutorials 23 (4) (2021) 2248–2294. doi:10.1109/comst.2021.3094360.
URL https://doi.org/10.1109/comst.2021.3094360

[49] ElectricPower and Intelligent Control (EPIC) Testbed, https://itrust.sutd.edu.sg/itrust-labs-home/itrust-labs_epic (2021).

[50] Idaho National Laboratory (INL) Smart Grid Test Bed Revision 2, https://www.energy.gov/sites/prod/files/2017/10/f38/CX-270322.pdf (2017).

[51] N. K. Kandasamy, S. Venugopalan, T. K. Wong, N. J. Leu, An electric power digital twin for cyber security testing, research and education, Computers and Electrical Engineering 101 (2022) 108061. doi:10.1016/j.compeleceng.2022.108061.
URL https://doi.org/10.1016/j.compeleceng.2022.108061

[52] P. Maynard, K. McLaughlin, S. Sezer, An open framework for deploying experimental SCADA testbed networks, in: Electronic Workshops in Computing, BCS Learning & Development, 2018, pp. 92–101. doi:10.14236/ewic/ics2018.11.
URL https://doi.org/10.14236/ewic/ics2018.11

[53] P. Maynard, Peter Maynard's GitHub, https://github.com/PMaynard/ICS-TestBed-Framework (2018).

[54] OpenPLC site, https://openplcproject.com/ (2023).

[55] P. Čeleda, J. Vykopal, V. Švábenský, K. Slavíček, KYPO4industry, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, ACM, 2020, p. 1026–1032. doi:10.1145/3328778.3366908.
URL https://doi.org/10.1145/3328778.3366908

[56] G. Koutsandria, R. Gentz, M. Jamei, A. Scaglione, S. Peisert, C. McParland, A real-time testbed environment for cyber-physical security on the power grid, in: Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy, ACM, 2015, p. 67–78. doi:10.1145/2808705.2808707.
URL https://doi.org/10.1145/2808705.2808707

[57] G. Bernieri, F. D. Moro, L. Faramondi, F. Pascucci, A testbed for integrated fault diagnosis and cyber security investigation, in: 2016 International Conference on Control, Decision and Information Technologies (CoDIT), IEEE, 2016, pp. 454–459. doi:10.1109/codit.2016.7593605.
URL https://doi.org/10.1109/codit.2016.7593605

[58] M. J. Scheepers, Virtualization and containerization of application infrastructure : A comparison, Tech. rep., Thijs Ai (2014).
URL https://thijs.ai/papers/scheepers-virtualization-containerization.pdf

[59] T. Y. Lin, G. Shi, C. Yang, Y. Zhang, J. Wang, Z. Jia, L. Guo, Y. Xiao, Z. Wei, S. Lan, Efficient container virtualization-based digital twin simulation of smart industrial systems, Journal of Cleaner Production 281 (2021) 124443. doi:10.1016/j.jclepro.2020.124443.
URL https://doi.org/10.1016/j.jclepro.2020.124443

[60] C. Boettiger, An introduction to docker for reproducible research, ACM SIGOPS Operating Systems Review 49 (1) (2015) 71–79.

[61] K. Ferencz, J. Domokos, Using node-red platform in an industrial environment, XXXV. Jubileumi Kandó Konferencia, Budapest (2019) 52–63.

[62] M. Tabaa, B. Chouri, S. Saadaoui, K. Alami, Industrial communication based on modbus and node-red, Procedia Computer Science 130 (2018) 583–588, the 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops. doi:https://doi.org/10.1016/j.procs.2018.04.107.
URL https://www.sciencedirect.com/science/article/pii/S1877050918304691

[63] A. Nicolae, A. Korodi, Node-red and opc ua based lightweight and low-cost historian with application in the water industry, in: 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), 2018, pp. 1012–1017. doi:10.1109/INDIN.2018.8471928.

[64] K. Mehta, R. Joshi, S. Kulkarni, B. Soni, Implementation of plc based software prototype for 45.6 mhz, 100 kw, icrh dac using epics control system, International Journal of Scientific and Engineering Research 6.

[65] F. Caturano, G. Perrone, S. P. Romano, Capturing flags in a dynamically deployed microservices-based heterogeneous environment, in: 2020 Principles, Systems and Applications of IP Telecommunications (IPTComm), 2020, pp. 1–7. doi:10.1109/IPTComm50535.2020.9261519.

[66] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, D. A. Tamburri, Devops: Introducing infrastructure-as-code, in: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), 2017, pp. 497–498. doi:10.1109/ICSE-C.2017.162.

[67] N. Leveson, Engineering a Safer World, The MIT Press, 2016.

[68] MulVal, a multi stage Vulnerability Analysis tool, https://github.com/risksense/mulval/ (2023).

[69] Kali Linux home page, https://www.kali.org/ (2023).

[70] Scapy Project, https://scapy.net/ (2023).

[71] arpspoof Linux man page, https://linux.die.net/man/8/arpspoof (2023).

[72] Home page of the Ettercap Project, https://www.ettercap-project.org/ (2023).

[73] Bettercap home page, https://www.bettercap.org/ (2023).

Acronyms

ADU *Application Data Unit*. 5

ARP *Address Resolution Protocol*. 2, 36, 38, 41

CPS *Cyber Physical System*. 13

CPwE *Converged Plantwide Ethernet*. 7

CRC *Cyclic Redundancy Check*. 5

DNP3 *Distributed Network Protocol*. 5

DoS *Denial of Service*. 6

DT *Digital Twin*. 14

ETA *Event Tree Analysis*. 11

FBD *Functional Block Diagram*. 25

FDI *False Data Injection*. 40, 41

FMEA *Failure Mode and Effects Analysis*. 11

FTA *Failure Tree Analysis*. 11

FTP *File Transfer Protocol*. 4

GOOSE *Generic Object Oriented Substation Events*. 15

HA-Grid *Hazard Analysis of Smart Grid Infrastructures*. 11

HART *Highway Addressable Remote Transducer*. 5

HAZOP *HAZard and OPerability analysis*. 11

HCA *Hazardous Control Action*. 35

HMAC *Hash-Based Message Authentication Code*. 7

HMI *Human Machine Interface*. 3, 15, 23, 24, 25, 26, 41

HTTP *HyperText Transfer Protocol*. 4

HTTPS *HyperText Transfer Protocol over Secure Socket Layer*. 4

ICS *Industrial Control System*. 2, 3, 4, 7, 8, 9, 10, 11, 13, 14, 18, 21, 24

IDE Integrated Development Environment. 8, 19

IEC *International Electrotechnical Commission*. 8, 9

IED *Intelligent Electronic Device*. 16, 17, 23, 24, 25, 26, 27, 33, 36, 40

IL *Instruction List*. 9

ISA *International Society of Automation*. 7

IT *Information Technology*. 4

MBAP *Modbus Application*. 5, 6

MITM *Man In The Middle*. 18, 29, 40, 41

MMS *Manufacturing Message Specification*. 15

MQTT *Message Queue Telemetry Transport*. 5, 15

OPC *Open Platform Communications*. 10

OSI *Open Systems Interconnection*. 40

OT *Operational Technology*. 4, 7, 10, 15, 16, 36, 38, 39, 40

PDU *Protocol Data Unit*. 5, 6

PERA *Purdue Enterprise Reference Architecture*. 7

PLC *Programmable Logic Controller (PLC)*. 3, 4, 8, 9, 10, 11, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 40, 41, 43, 45

POU *Program Organizational Unit*. 9

ProFiBus *Process Field Bus*. 5

RAT *Remote Access Trojan*. 10

RTU *Remote Terminal Unit*. 4, 15, 18

SCADA *Supervisory Control And Data Acquisition*. 3, 4, 9, 10, 13, 15

SCASS *SCADA Systems Security*. 2, 3, 9, 14

SFC *Sequential Function Chart*. 9

SM3 *ShangMi 3*. 7

SM4 *ShangMi 4*. 7

SMB *Service Message Block*. 4

SMTP *Simple Mail Transfer Protocol*. 4

SNMP *Simple Network Management Protocol*. 4

ST *Structured Text*. 25

STAMP *Systems Theoretic Accident Model and Process*. 11, 12, 31, 33

STPA *Systems Theoretic Process Analysis*. 11, 12, 32

STPA-Sec *Systems Theoretic Process Analysis for Security*. 12

TCP *Transmission Control Protocol*. 40, 41

TFTP *Trivial File Transfer Protocol*. 4

TLS *Transport Layer Security*. 7

UA Unified Architecture. 10

UDP *User Datagram Protocol*. 24

VM *Virtual Machine*. 16, 39