Computers & Security 132 (2023) 103373

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

MOSTO: A toolkit to facilitate security auditing of ICS devices using Modbus/TCP

Ricardo J. Rodríguez[a,*], Stefano Marrone[b], Ibai Marcos[c], Giuseppe Porzio[b]

[a] Aragón Institute of Engineering Research (I3A) Universidad de Zaragoza, Spain
[b] Dpto. di Matematica e Fisica, University of Campania "Luigi Vanvitelli", Caserta, Italy
[c] Independent Researcher, Spain

A R T I C L E I N F O

Article history:
Received 26 January 2023
Revised 10 May 2023
Accepted 26 June 2023
Available online 27 June 2023

Keywords:
Industrial control systems
Penetration testing
Security auditing
Modbus

A B S T R A C T

The integration of the Internet into industrial plants has connected Industrial Control Systems (ICS) worldwide, resulting in an increase in the number of attack surfaces and the exposure of software and devices not originally intended for networking. In addition, the heterogeneity and technical obsolescence of ICS architectures, legacy hardware, and outdated software pose significant challenges. Since these systems control essential infrastructure such as power grids, water treatment plants, and transportation networks, security is of the utmost importance. Unfortunately, current methods for evaluating the security of ICS are often ad-hoc and difficult to formalize into a systematic evaluation methodology with predictable results. In this paper, we propose a practical method supported by a concrete toolkit for performing penetration testing in an industrial setting. The primary focus is on the Modbus/TCP protocol as the field control protocol. Our approach relies on a toolkit, named MOSTO, which is licensed under GNU GPL and enables auditors to assess the security of existing industrial control settings without interfering with ICS workflows. Furthermore, we present a model-driven framework that combines formal methods, testing techniques, and simulation to (formally) test security properties in ICS networks.

1. Introduction

Modern society is heavily dependent on large, heterogeneous, and complex software-intensive systems to support all kinds of daily activities. Industrial plants have opened their production and operation to the Internet, allowing remote control of their facilities for easier and faster management. As a result, they are more exposed to both accidental and malicious threats. As the complexity of these systems and the strength of attacks increase, cascading failures can disrupt the production of a system for several hours or days. In the worst case, the service discontinuity can lead to serious problems, from severe financial loss to death or injury.

During the "off-line" industrial era, Industrial Control Systems (ICS)[1] have been primarily protected against physical intrusion intended to cause physical attacks (damages to control devices) and/or logical attacks (for example, internal sabotage or compromise of operator workstations with malicious code). As the Internet has entered industrial plants, Supervisory Control And Data Acquisition (SCADA) systems have connected the world, increasing attack surfaces and exposing software and devices that were not initially designed to be networked (ENISA, 2013). According to the Purdue Enterprise Reference Architecture (Williams, 1994), networked control systems are mainly based on a Control Network (connecting Programmable Logic Controllers — PLCs) and a Supervisory Network, connecting SCADA servers, historical database management systems, and logging facilities, among other industrial network assets.

Recent reports show that the number of reported security incidents on ICS networks has increased in recent years. In the first half of 2022, Kaspersky ICS CERT reported that 31.8% of ICS computers worldwide had malicious objects that were blocked at least once (Kaspersky ICS CERT, 2022). Similarly, according to Dragos (Dragos, 2023), during 2022, cyberattacks against ICS increased by 87% in ransomware attacks on industrial organizations and by 35% in the number of ransomware groups targeting industrial control and operational technology systems.

Security is of the utmost importance, as these systems are used to control critical infrastructure such as power grids, water

* Corresponding author.
E-mail address: rjrodriguez@unizar.es (R.J. Rodríguez).

[1] In this paper, we use ICS interchangeably as a singular and plural acronym.

treatment plants, and transportation networks. Any compromise of these systems can have serious consequences ranging from financial loss to loss of human life. However, thousands of cybersecurity incidents have been reported in many industry sectors: chemical, healthcare, transportation, or food and agriculture, to name a few. A summary for the year 2012 can be found in Kozik and Choras (2013). Apart from the well-known case of the *Stuxnet* worm in 2011, which was a piece of malicious software specially designed to exploit Siemens PLCs in SCADA networks affecting in particular Iranian nuclear facilities (Farwell and Rohozinski, 2011; Langner, 2011), other threats targeting energy companies, research centers, or banking services, such as *GhostNet, Flame, DarkSeoul*, or *DragonFly*, have been made public over the years (Rauscher, 2013).

Recent attacks on critical infrastructure have highlighted the vulnerability of critical infrastructure to cyberattacks and the need for stronger security measures in ICS networks. For example, the 2015 cyberattack on Ukraine's power grid, carried out by hackers who gained access to control systems and disrupted the flow of electricity, left 230,000 people without power for several hours (Sullivan and Kamensky, 2017). Similarly, the 2021 ransomware attack on the Colonial Pipeline in the United States resulted in the shutdown of a major pipeline network that supplies fuel to the East Coast. The attack caused fuel shortages, price hikes, and disruptions to transportation and other critical services (Smith, 2022), again highlighting the need for robust cybersecurity measures to guard against such attacks.

In conclusion, the importance of security in industrial control systems cannot be overstated, as the potential consequences of a cyberattack on critical infrastructure are significant. Recent incidents have demonstrated the need for stronger security measures to safeguard these systems, regardless their industry.

We have examples of recent incidents of different industries: (i) on the energy industry: in February 2021, a cyberattack on a water treatment facility in Florida resulted in hackers gaining access to the plant's control systems and attempting to increase the level of sodium hydroxide in the water supply (Vera et al., 2021); (ii) on the manufacturing industry: in December 2020, the SolarWinds supply chain attack impacted multiple organizations, including some in the manufacturing industry (Peisert et al., 2021); (iii) on the water industry: in 2019, a ransomware attack on the city of Baltimore's computer systems affected the city's water billing system and resulted in a disruption to the city's water services (Gallagher, 2019); (iv) on the healthcare industry: in December 2020, the U.S. government warned of a cyber attack campaign targeting the healthcare industry (CISA, 2020); and (v) on the transportation industry: in June 2021, the Colonial Pipeline ransomware attack led to the shutdown of a major pipeline network that supplies fuel to the East Coast of the United States (Smith, 2022).

In these systems, security should be seen as a primary goal rather than added value: system designers and developers should include security assessments in the development cycle to measure how these disruptions are handled and to quantify, where possible, the impact suffered by the ICS under stress. Classical ICT security assessment methods must face the new challenges posed by this change in focus: from choosing the most appropriate set of basic security properties (i.e., triads as Confidentiality-Integrity-Availability, Safety-Reliability-Productivity, or People-Processes-Technology[2]) to obtaining and dealing with threats arising from overlapping of physical and cybernetics (Marrone et al., 2015), to a sensitivity adjustment in the evaluation of the impact of a successful attack on the safety, availability, and reputation of the owner of the plant (Knowles et al., 2015).

The evaluation of ICS systems is generally carried out with ad-hoc techniques that are difficult to formalize or frame in a systematic evaluation methodology to make them repeatable with predictable results. These achievements, paradoxically, would have a greater impact in the field due to the benefits they bring to the evaluation procedures established as mandatory by international safety and security standards. Some of these standards are, for example: IEC 62443 (International Electrotechnical Commissionn, 2009), which is a set of “horizontal standards” in the ICS landscape that explicitly addresses cybersecurity aspects by encouraging the definition of a systematic life cycle consisting of an *evaluate-implement-maintain* loop; and ISO-SAE-21434 (International Organization for Standardization, 2021), which is a vertical standard in the automotive domain based on a V-shaped model and explicitly addresses verification and validation activities. In general, the research community widely accepts that security cannot be properly handled outside of a systematic process, strongly fostering such processes in the case of critical systems that also cope with security issues (Lisova et al., 2019).

This paper presents an approach to audit and evaluate the security of computer-based control and supervision systems for industrial plants. The main focus of the work is on penetration testing, which is essential in security assessment procedures and is carried out by specialist people called *ethical hackers* (Wang and Yang, 2017). In particular, in this paper, we introduce a toolkit that helps an auditor assess the security of existing industrial control configurations using the Modbus protocol over TCP (Modbus/TCP). Our toolkit consists of two software artefacts that simulate Modbus/TCP master and slave devices, dubbed as `MOSTO` (MOdbus Simulator TOol). `MOSTO` allows an ethical hacker to recreate a real environment and test for potential security vulnerabilities and countermeasures without doing so in the production environment. We have also applied our toolkit in a real case study of a linear particle accelerator to demonstrate its applicability. In addition, we also present a model-driven framework that combines the use of formal methods, testing techniques, and simulation to demonstrate security properties in ICS networks.

Ethical hacking and penetration testing can play a crucial role in identifying vulnerabilities and evaluating the security of ICS systems. Using the same techniques as attackers, ethical hackers can identify vulnerabilities in ICS networks. Additionally, by attempting to exploit these vulnerabilities in a controlled and ethical manner, they can provide valuable insights into potential weaknesses that could be targeted by attackers. This can help organizations take proactive steps to address these vulnerabilities before they are exploited. In addition, they help evaluate security controls through penetration testing. By simulating an attack, ethical hackers can evaluate how well these controls can detect and prevent unauthorized access to ICS systems. Last, ethical hacking and penetration testing can be used to test disaster recovery plans and business continuity plans for ICS networks, as well as to assess compliance requirements for ICS security.

In summary, the contribution of this paper is two fold. First, we propose `MOSTO`, a tool to assess the security of Modbus-based ICS networks. This tool implements a fully configurable simulator that can function as a master or slave device and can be installed on real systems for auditing purposes. In addition, this tool can also be used as part of teaching specialized ICT communications security lessons or courses. Second, we present a model-driven framework that combines the use of formal methods, testing, and simulation techniques to test security properties in a real/simulated environment.

The paper is structured as follows: Section 2 introduces the Modbus protocol and its security. Section 3 is dedicated to our

[2] More details at https://cyberstartupobservatory.com/correctly-understanding-the-cyber-security-triads/.

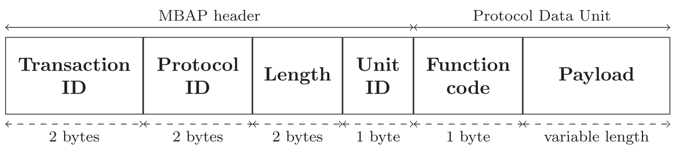


Fig. 1. Representation of a Modbus/TCP APDU.

MOSTO toolkit. Section 4 describes the linear particle accelerator lab where we tested our toolkit, as well as the experiments performed there. Section 5 frames the MOSTO toolkit within a broader model-driven methodology for ICS security assessment. Section 6 discusses related work. Finally, Section 7 ends the paper and points out future research.

2. The Modbus protocol and its security

Modbus (MODICON, 2012) is a well-established control protocol in ICS. Initially designed by Modicon in 1979 as a simple way to communicate control data between controllers and sensors using a serial interface (such as RS232 or RS485) port, it was widely adopted and deployed as a *de facto* standard in industrial automation. With the advent of connected factories, Modbus has also been used as a network protocol at the application level (e.g., the upper layer of the OSI model), being able to operate over different links, such as serial buses, routable networks over TCP/IP, or buses intercommunicated through an RS-485 communication link. Roughly speaking, Modbus is a request/response (also known as master/slave) protocol that assigns a unique address to each device intended to communicate. In this paper, we focus on the Modbus protocol variant intended to use Modbus messaging in a network environment using the TCP/IP protocols (commonly known as Modbus/TCP).

The Modbus/TCP specification defines an embedding of Modbus packets into TCP frames, assigning port 502 to listen and receive data. A Modbus/TCP frame includes the usual IP and TCP headers, followed by a Modbus/TCP-specific Application Protocol Data Unit (APDU), which consists of two parts: the Modbus-specific header (called the Modbus Application Protocol header, MBAP) and the payload (called PDU, Protocol Data Unit). Fig. 1 shows the fields of a Modbus/TCP APDU and its sizes. The payload varies in size, but is limited to 253 bytes to maintain compatibility with the Modbus serial protocol.

The MBAP header consists of four fields: *Transaction ID*, which is used to match transactions (note that multiple messages can be transmitted on the same TCP connection without waiting for a response); *Protocol Identifier*, which is always 0 for Modbus; *Length*, which provides the length of the remaining fields of the MBAP header and PDU (in bytes); *Unit ID*, used to identify a PLC located on a non-TCP network. This field is used for serial bridging when Modbus/TCP and serial Modbus coexist in the same network. The default values for this field are 00 or FF, which indicates that it should be ignored and repeated in the response.

Modbus defines three types of PDUs: request, response, and exception response. A PDU is made up of two fields: *Function code*, which defines the type of PDU; and the payload itself. For the Modbus exception response, the most significant bit of the function code is set to 1. Therefore, request and response function codes range from 1 to 127. There are three categories of function codes: public (standard), defined by user (vendor-specific functions), and reserved (used by legacy products). In particular, the public function codes are: 1–8, 11–12, 15–17, 20–24, 43. Data is represented in Modbus by four main tables, which are organized serially: discrete input, coils, input registers, and holding registers. Access is directed to a specific primary table based on the function code specified in the Modbus request.

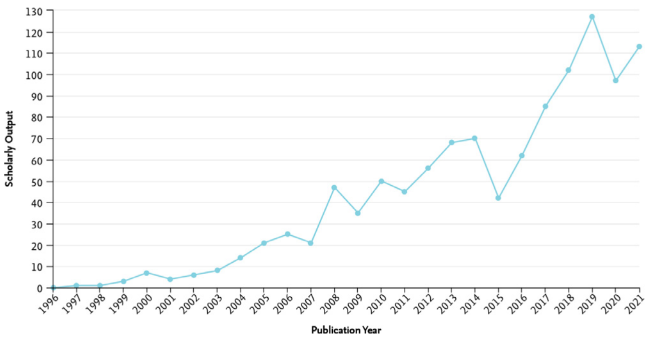


Fig. 2. Modbus paper over time.

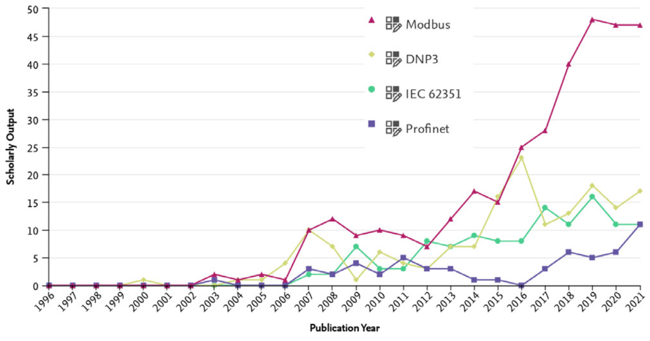


Fig. 3. Comparison of security concerns for control protocols.

Modbus security has been widely studied using testing (Chen et al., 2015), formal methods (Siddavatam et al., 2017), or approaches that combine both techniques (Nardone et al., 2016). In its native expression, Modbus lacks any security attribute (remember the triad Confidentiality-Integrity-Availability mentioned above): no authentication, no encryption, and no integrity at all are the main security concerns of this protocol. This lack of security awareness is motivated by the fact that the Modbus protocol was designed to operate in systems with isolated networks; surprisingly, insider attackers (for instance, disgruntled employees) were also not considered. Even if more secure variants of the original standard are available (MODICON, 2018), using such versions on legacy systems or on systems with low power/low power consumption requirements requires a lot of effort.

More than 40 years after its first definition, Modbus is a protocol that is in the spotlight when it comes to adapting traditional industry to the 21st century. To prove this claim, some analysis on the current state of the research has been done using the Scival tool[3]. Fig. 2 reports the number of the papers published over the years dealing with Modbus protocol. This figure shows that the interest of the scientific community has not diminished in these years.

In addition, we have compared its popularity among industrial protocols at the research level. Three other main protocols have been chosen for comparison: DNP-3, Profinet, and the IEC 62,351 smart grid related protocol. The comparison has been made by looking for the following sentences: *Modbus security, DNP3 security, "IEC 62351" security* and *Profinet security*. The results of the comparison (made on the number of the papers published over a period of time) are reported in Fig. 3. The figure demonstrates that also by comparing Modbus to these other three more recent conventional control protocols and narrowing down the searches to security concerns, the number of published articles on this protocol is growing over time. In our opinion, Modbus continues to inspire re-

[3] https://www.scival.com/home

search challenges in both the academic and industry communities due to its simplicity, high ubiquity, and legacy nature.

3. The MOSTO toolkit

This section introduces MOSTO, our toolkit for evaluating the security of Modbus-based ICS networks. MOSTO is primarily a simulator of the Modbus protocol developed with Python version 3, as we require our tool to be cross-platform. Furthermore, it makes use of Scapy, a Python library specially dedicated to manipulating and dissecting network packets. To promote our tool and foster more research in this area, we have released it under the GNU GPLv3 licence and is freely available in https://github.com/reverseame/MOSTO-pentesting-Modbus.

MOSTO can work as a master or slave device. As a master device, the tool allows the user to configure the bursts of Modbus messages to be sent, defining the number and order of types of Modbus messages; the time between consecutive transmissions; and a variance to randomize the time between subsequent transmissions. In addition, it also allows the user to configure different types of Modbus messages, as well as to define in detail each Modbus message to be transmitted (that is, each of the fields and values of the message). Since our tool is more focused on network communications than on the operation of the underlying controller (which is usually quite coupled with the specific manufacturer of the PLC), it makes it easy for the user to mimic any real device that works with the Modbus/TCP protocol.

In addition to simulating real systems for auditing purposes, this toolkit can also be used as part of teaching specialized ICT communications security lessons or courses. MOSTO is also scriptable, allowing it to be easily incorporated into existing workflows to automatically assess the security of network systems. In what follows, we detail the operation of MOSTO acting as a master device and as a slave device.

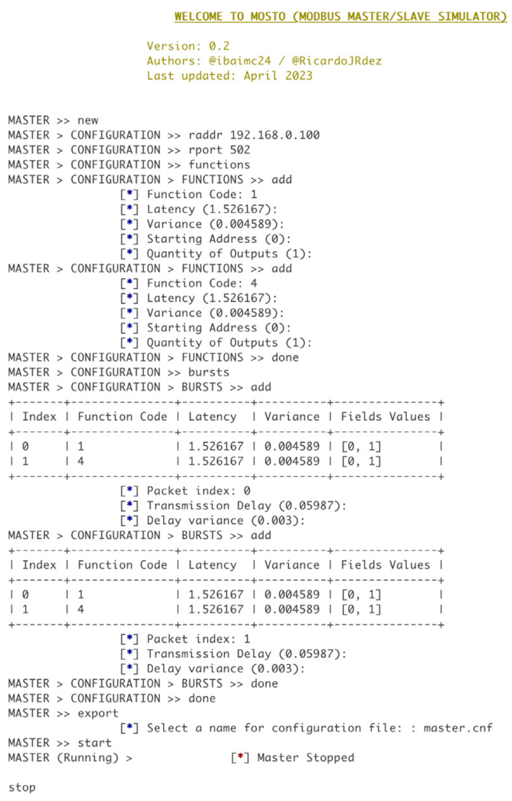
3.1. *MOSTO acting as a master device*

The *master-device* simulator workflow consists of two steps:

Configuration. Here, the user must specify how the communication should take place. Some parameters can be ignored (then taking default values) to facilitate configuration. In addition, the user can specify a random delay between transmissions and bursts of different types of messages to make the simulator as close to the real device as possible. As explained above, the user can also indicate which Modbus messages are to be sent, their corresponding fields and values, and the frequency of transmission.

Communication. Once the simulator is configured, the communication step can be started. Here, MOSTO performs the network communication accordingly to the configuration created in the previous step. In addition, our tool records the status of the communication to offer a visual aid to the user. This recording system also allows the user to easily perform checks and validations.

Fig. 4 shows a snapshot of a simulated Modbus master device with two functions (READ COILS and READ INPUT REGISTERS) with default values suggested by MOSTO. The configuration can be exported directly to a file as a binary object, which can be imported into another run.



```
                    WELCOME TO MOSTO (MODBUS MASTER/SLAVE SIMULATOR)

                  Version: 0.2
                  Authors: @ibaimc24 / @RicardoJRdez
                  Last updated: April 2023

MASTER >> new
MASTER > CONFIGURATION >> raddr 192.168.0.100
MASTER > CONFIGURATION >> rport 502
MASTER > CONFIGURATION >> functions
MASTER > CONFIGURATION > FUNCTIONS >> add
                [*] Function Code: 1
                [*] Latency (1.526167):
                [*] Variance (0.004589):
                [*] Starting Address (0):
                [*] Quantity of Outputs (1):
MASTER > CONFIGURATION > FUNCTIONS >> add
                [*] Function Code: 4
                [*] Latency (1.526167):
                [*] Variance (0.004589):
                [*] Starting Address (0):
                [*] Quantity of Outputs (1):
MASTER > CONFIGURATION > FUNCTIONS >> done
MASTER > CONFIGURATION >> bursts
MASTER > CONFIGURATION > BURSTS >> add
+-------+---------------+----------+----------+---------------+
| Index | Function Code | Latency  | Variance | Fields Values |
+-------+---------------+----------+----------+---------------+
| 0     | 1             | 1.526167 | 0.004589 | [0, 1]        |
| 1     | 4             | 1.526167 | 0.004589 | [0, 1]        |
+-------+---------------+----------+----------+---------------+
                [*] Packet index: 0
                [*] Transmission Delay (0.05987):
                [*] Delay variance (0.003):
MASTER > CONFIGURATION > BURSTS >> add
+-------+---------------+----------+----------+---------------+
| Index | Function Code | Latency  | Variance | Fields Values |
+-------+---------------+----------+----------+---------------+
| 0     | 1             | 1.526167 | 0.004589 | [0, 1]        |
| 1     | 4             | 1.526167 | 0.004589 | [0, 1]        |
+-------+---------------+----------+----------+---------------+
                [*] Packet index: 1
                [*] Transmission Delay (0.05987):
                [*] Delay variance (0.003):
MASTER > CONFIGURATION > BURSTS >> done
MASTER > CONFIGURATION >> done
MASTER >> export
                [*] Select a name for configuration file: : master.cnf
MASTER >> start
MASTER (Running) >               [*] Master Stopped

stop
```

Fig. 4. A running of MOSTO to simulate Modbus master device with two functions (ReadCoils and ReadInputRegisters) with default values.

3.2. *MOSTO acting as a slave device*

When MOSTO acts as a slave device, it allows the user to configure different Modbus functions. In addition, it can communicate with more than one master device (that is, it works concurrently) and the simulated data is unique in each master-slave communication pair. The execution of a simulated slave device with MOSTO is transparent to the real master device, which cannot discern whether it is a real or simulated device. As future work, we aim to extend this simulator to act as a honeypot device in ICS networks integrating it with existing devices such as Energy Box (Mlot et al., 2022).

Internally, the slave simulator follows a component design, separated in three parts following the principle of separation of duties: the (simulated) PLC data content and its management, the network communication, and the simulation of the mechanics of the physical process controlled by the (simulated) PLC.

The *slave-device* simulator workflow also consists of two steps:

Configuration. Here, the user configures the Modbus/TCP slave device to specify network parameters, data types, and Modbus protocol-specific parameters (such as which functions are supported and which memory addresses are allocated). In a real Modbus/TCP slave device, the controller memory is used to control every sensor or actuator connected to the device. This memory is limited to 65,535 addresses (restricted by 16-bit address space) due to the Modbus protocol requirements. Controllers typically store data in logical tables, dividing the addressing equally. Therefore, each table is addressed from 0 to 10,000 (0000 to 270E, in hexadecimal notation). However, the Modbus protocol itself allows addressing and thus the valid addressing ranges from 0 to 65,535 (0000 to FFFF). Permissions to access each address in simulated device's logical table are also defined in the configuration.

Communication. Once the simulator has been configured, it enters the listening state. Like a real controller, the simulator accepts a series of TCP connections that are processed

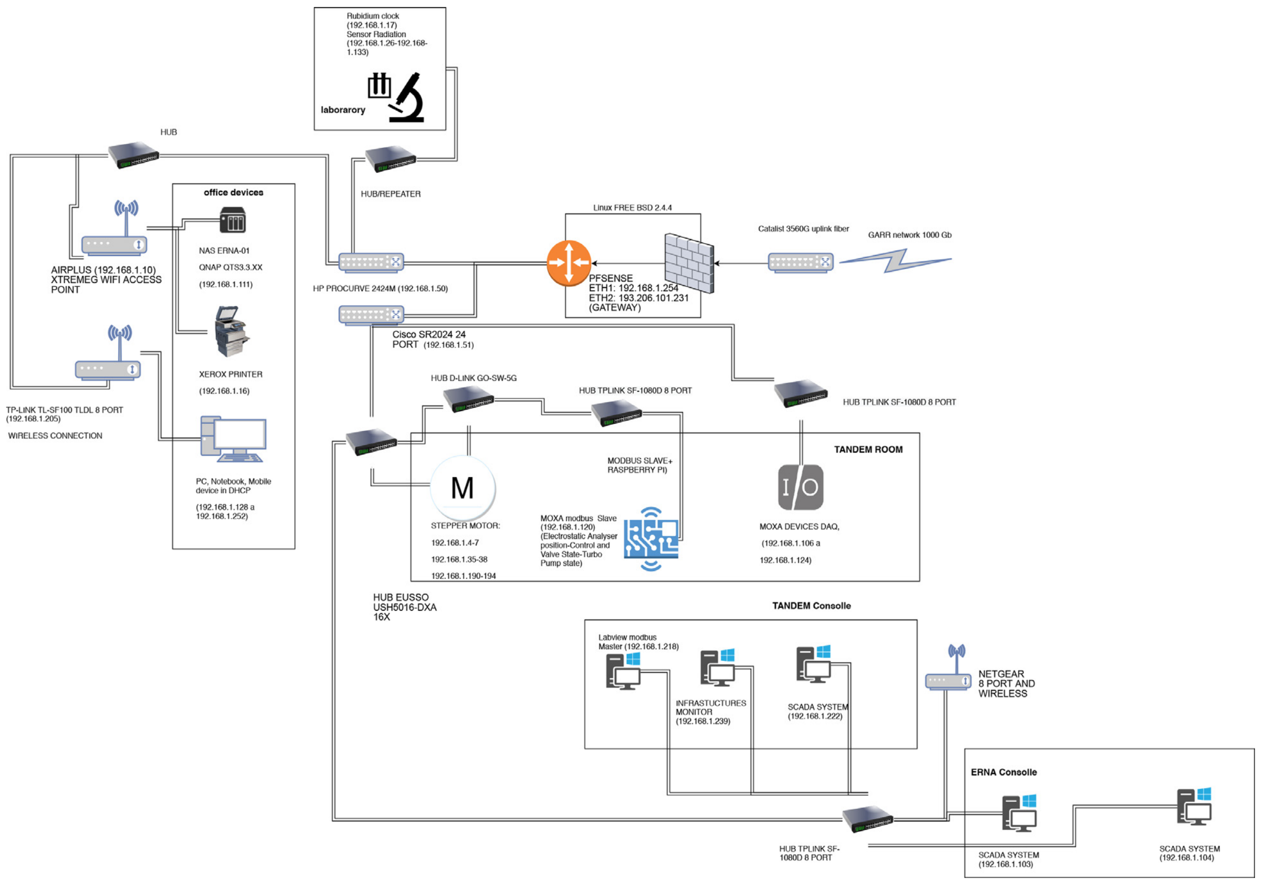

Rubidium clock
(192.168.1.17)
Sensor Radiation
(192.168.1.26-192.168-1.133)
laborarory
HUB
HUB/REPEATER
office devices
AIRPLUS (192.168.1.10) XTREMEG WIFI ACCESS POINT
NAS ERNA-01
QNAP QTS3.3.XX
(192.168.1.111)
XEROX PRINTER
(192.168.1.16)
PC, Notebook, Mobile device in DHCP
(192.168.1.128 a 192.168.1.252)
TP-LINK TL-SF100 TLDL 8 PORT (192.168.1.205)
WIRELESS CONNECTION
HP PROCURVE 2424M (192.168.1.50)
Cisco SR2024 24 PORT (192.168.1.51)
Linux FREE BSD 2.4.4
PFSENSE
ETH1: 192.168.1.254
ETH2: 193.206.101.231
(GATEWAY)
Catalist 3560G uplink fiber
GARR network 1000 Gb
HUB D-LINK GO-SW-5G
HUB TPLINK SF-1080D 8 PORT
HUB TPLINK SF-1080D 8 PORT
TANDEM ROOM
MODBUS SLAVE+ RASPBERRY PI)
M
STEPPER MOTOR:
192.168.1.4-7
192.168.1.35-38
192.168.1.190-194
MOXA modbus Slave (192.168.1.120) (Electrostatic Analyser position-Control and Valve State-Turbo Pump state)
I/O
MOXA DEVICES DAQ,
(192.168.1.106 a
192.168.1.124)
HUB EUSSO USH5016-DXA 16X
TANDEM Consolle
Labview modbus Master (192.168.1.218)
INFRASTUCTURES MONITOR (192.168.1.239)
SCADA SYSTEM (192.168.1.222)
NETGEAR 8 PORT AND WIRELESS
ERNA Consolle
SCADA SYSTEM (192.168.1.103)
SCADA SYSTEM (192.168.1.104)
HUB TPLINK SF-1080D 8 PORT

Fig. 5. Network diagram of the CIRCE laboratory.

as Modbus/TCP master-slave communications. Once the connection is established, the simulator waits for the Modbus messages to dissect and process them individually. The simulator's response to the message depends on the configuration previously made and the type of message received. If the requested function is undefined, the simulator returns a Modbus exception message to indicate the master that such a function is unavailable. Otherwise, it checks if the requested data follows the protocol specifications and if the requested memory addresses are in range. If all checks pass, the simulator executes the requested function (changing the state of the simulated physical process, if necessary) and generates an appropriate response.

4. Case study: The CIRCE laboratory

In this section, we show the benefits of using MOSTO through a case study. Specifically, we have carried out several experiments in a real scientific laboratory. We first introduce the network infrastructure of this lab and then detail our experiments.

4.1. Network infrastructure of the CIRCE laboratory

The *Centre for Isotopic Research on Cultural and Environmental heritage* (CIRCE) of the University of Campania "Luigi Vanvitelli" was created in 2005 within the action POR Campania - 2000/2006 Misura 3.16, and funded by the European Union and the Campania Region (Italy). Its activity is mainly focused on basic nuclear physics and nuclear astrophysics, as well as industrial applications. The structure of the CIRCE laboratory network is shown in Fig. 5.

The entire lab is connected to the GARR network (i.e., the Italian high speed backbone connecting universities and research centers) via a general router and a FreeBSD machine acting as a firewall. This machine is connected to two switches (an HP Procurve 2424M and a Cisco SR2024) that are responsible, respectively, for managing the traffic of radiation laboratories and office devices and TANDEM control rooms (that is, SCADA servers and operator workstations). All these parts are segmented, following the principle of network segmentation. This last segment of the network also contains the Modbus master and slave devices. Connectivity is also available through secure Wi-Fi hotspots.

4.2. Experimental settings

We have used a network LAN tap to collect network traffic. A network LAN tap is a piece of hardware designed to monitor network traffic in depth. In particular, we have used the Throwing Star LAN Tap from Great Scott Gadgets. This hardware allows us to intercept network traffic and divert the flow in and flow out to additional network sockets. In particular, we have connected this device in place of the Modbus/TCP slave device of the CIRCE network that manages the electrostatic analyzer and the pelletron accelerator, as shown in Fig. 6.

Since the input and output communication streams are separated, as a monitoring device we need a device that has two Ethernet ports or use two separate monitoring devices. For the sake of simplicity (and low cost), we opted for the second option. Therefore, we have used two RaspberryPi 3B devices running a Debian 9.3 OS on ARM Quad-Core Cortex A7 900MHZ processor and 1GB RAM memory as monitoring devices. Let us remark that using resource-constrained hardware like the RaspberryPi 3B (presumably with little memory to support ingress/egress buffers) was sufficient for our experimental setup, as no network frames were

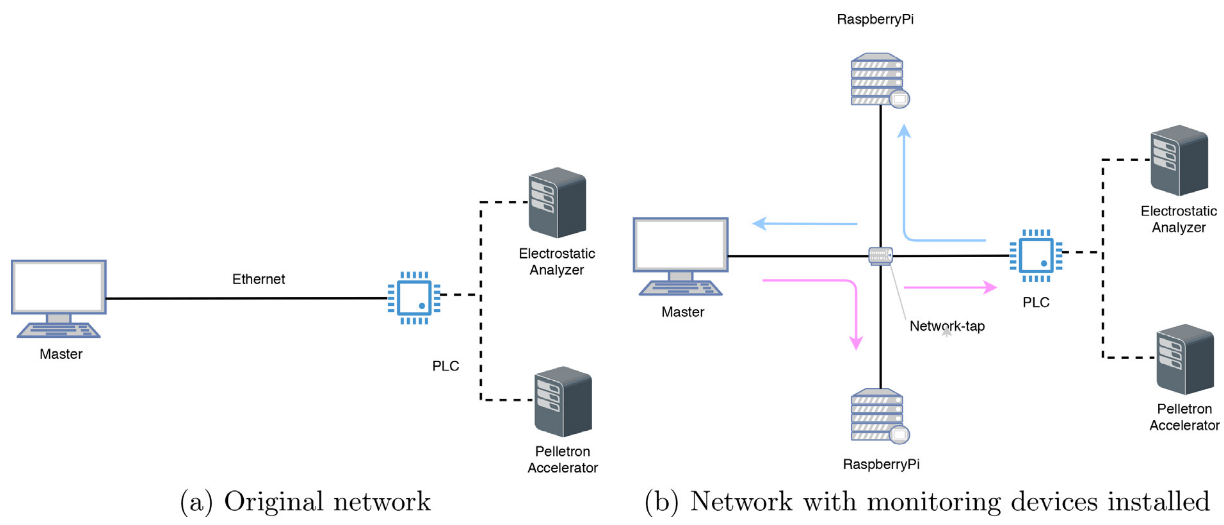


Fig. 6. Network diagram of the CIRCE laboratory with monitoring devices.

lost and the observed network load was not actually very high. In fact, the main bottleneck of our experimental setup was in the external storage of the memory card, which forced us to visit at least twice a month the CIRCE lab to collect the experimental data, as discussed below.

As software for capturing network traffic, we use `tcpdump`, a well-known and easy-to-use command-line packet analyzer. This tool captures network traffic and stores it as a PCAP (packet capture) format file. Due to limited memory storage capacity on RaspberryPi 3B devices, approximately every two weeks we have visited the CIRCE lab to extract the collected data from the monitoring devices and continue the monitoring process. Fig. 7 shows our monitoring system installed in the CIRCE laboratory.

After a non-continuous period of six months (due to scheduled short stops or technical problems at CIRCE), we stopped the monitoring system. After merging and sanitizing the PCAP files of the input and output communication channels (remember that the network tap has two connectors to separate each stream), we got PCAP files ranging from 5 to 7 gigabytes in total. These PCAP files represent a complete, sequential view of the communications captured by our system. Since the analysis of these files exhausts the resources of our personal computers, we divide them into smaller parts (but keeping the sequential view) to make them manageable for analysis.

4.3. Experimental findings and lessons learned

We detected Modbus/TCP traffic as well as other types of traffic in the captured PCAP files. This fact was not unknown in advance, since the CIRCE technicians already warned us about this issue. On the contrary, it served to indicate that the capture system and analysis workflow was working as expected. However, this problem also shows that the network where the Modbus/TCP devices were operating was not segmented correctly. From a security point of view, this not only facilitates attacks such as eavesdropping and fingerprinting, but also makes it more difficult to detect attacks based on exhausting network bandwidth (such as Denial-of-Service attacks (Mahjabin et al., 2017)).

In our experiments we found 152,773,862 Modbus messages (49.84%) and 153,754,209 non-Modbus messages (50.16%), that is, the network traffic was divided approximately equally between Modbus and non-Modbus messages. In particular, the non-Modbus messages were related to the ICMP, DNS, HTTP, and TLS/SSL protocols. Considering that carrying out activities in a timely manner is critical for industrial control systems, network congestion can negatively affect system performance. Therefore, isolating the Modbus network from other networks would significantly improve its per-

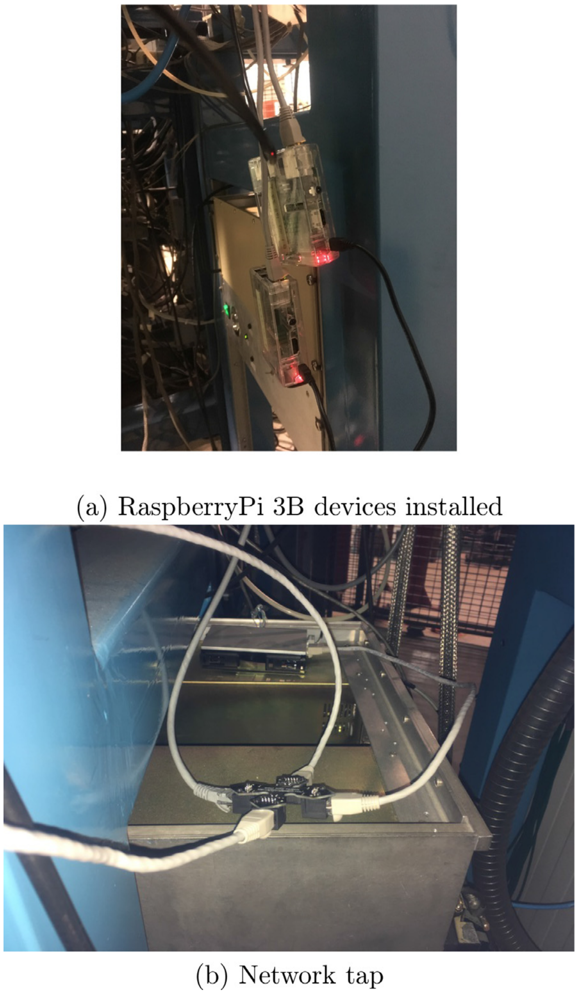
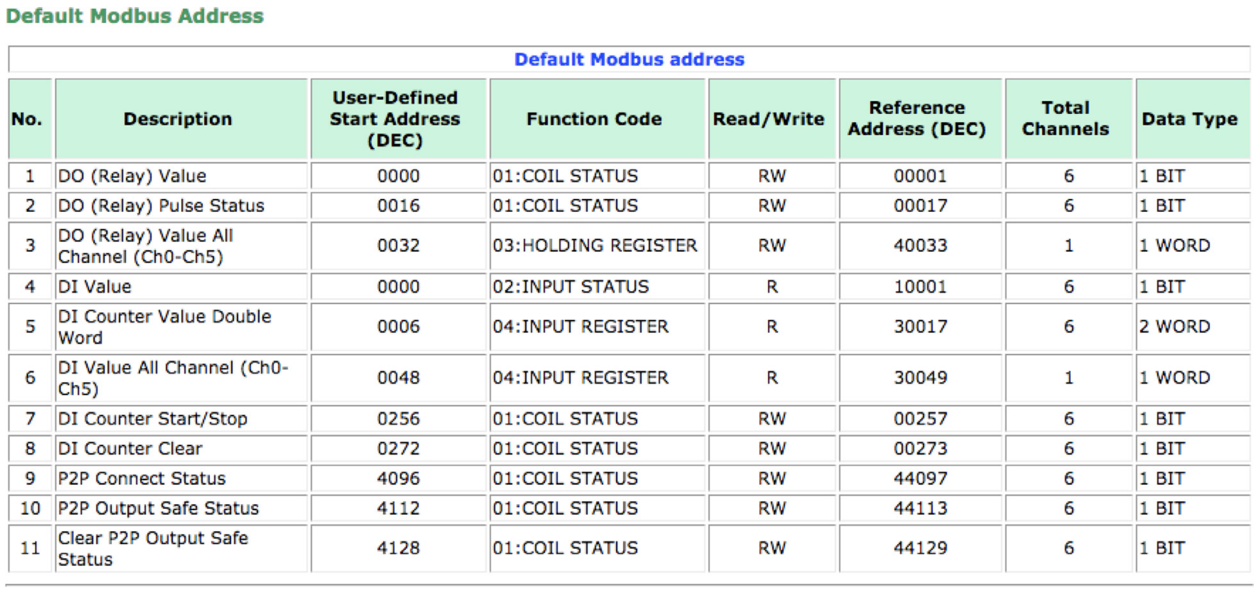


Fig. 7. Monitoring system installed in the CIRCE laboratory.



Default Modbus Address

Default Modbus address							
No.	Description	User-Defined Start Address (DEC)	Function Code	Read/Write	Reference Address (DEC)	Total Channels	Data Type
1	DO (Relay) Value	0000	01:COIL STATUS	RW	00001	6	1 BIT
2	DO (Relay) Pulse Status	0016	01:COIL STATUS	RW	00017	6	1 BIT
3	DO (Relay) Value All Channel (Ch0-Ch5)	0032	03:HOLDING REGISTER	RW	40033	1	1 WORD
4	DI Value	0000	02:INPUT STATUS	R	10001	6	1 BIT
5	DI Counter Value Double Word	0006	04:INPUT REGISTER	R	30017	6	2 WORD
6	DI Value All Channel (Ch0-Ch5)	0048	04:INPUT REGISTER	R	30049	1	1 WORD
7	DI Counter Start/Stop	0256	01:COIL STATUS	RW	00257	6	1 BIT
8	DI Counter Clear	0272	01:COIL STATUS	RW	00273	6	1 BIT
9	P2P Connect Status	4096	01:COIL STATUS	RW	44097	6	1 BIT
10	P2P Output Safe Status	4112	01:COIL STATUS	RW	44113	6	1 BIT
11	Clear P2P Output Safe Status	4128	01:COIL STATUS	RW	44129	6	1 BIT

Fig. 8. Modbus/TCP slave device configuration (as provided by the device configuration webpage).

formance. This problem can be easily solved by using VLANs. Additionally, our analysis reported close to 70 different network devices using common Internet protocols in the same network segment. We also found that some of these devices also have open ports and outdated software. Thus, these devices are potential targets for an attacker looking for vulnerabilities in order to intrude into the system via the Internet.

Regarding Modbus traffic, only the messages related to function 1 (`ReadCoils`), function 2 (`ReadDiscreteInputs`), and function 15 (`WriteMultipleCoils`) are used, the latter being the less frequent. Also, Modbus message bursts consist of consecutive `ReadCoils` and `ReadDiscreteInputs`, rarely ending with a `WriteMultipleCoils`. Most of the packets are the same length (10, 12, or 14 bytes, respectively).

Since the Modbus message header size is known by the standard, an attacker can find out the amount, type, and structure of data that is requested during communication by passively sniffing the network. Furthermore, the privacy of the Modbus slave device is compromised as an attacker can inject Modbus messages to request different data sizes from the slave device and then discover the full configuration and behavior of the controller. We have verified this statement experimentally by injecting specially crafted Modbus messages into the network after revision and approval of lab technicians. These crafted Modbus messages allowed us to know the structure and data content of the Modbus/TCP slave device, without altering the state of the system.

Fig. 8 shows the configuration of the Modbus/TCP slave device, obtained through the web page that provides the Modbus/TCP slave device for its configuration. The results of our experiment is in Listing 1. As shown, the structure inferred by traffic injection matches the device configuration perfectly.

Finally, we found that the Modbus/TCP master device was not using the *Transaction ID* field that allows to identify the type of message (see Section 2). We believe this issue is caused by an implementation error of the Modbus protocol on the master device. Normally this field is simply copied by the slave device when it sends the response and therefore the master device can perform the request and response message pairing very easily. However, we note that the master device always sets this message field to zero and therefore, both master and slave devices are vulnerable to replay attacks. A replay attack is a network attack in which a valid data transmission is intentionally replayed or delayed.

```
SUPPORTED FUNCTION CODES
     [ 1, 2, 3, 4, 5, 6, 8, 15, 16 ]
     COILS
          From 0 to 5
          From 16 to 21
          From 256 to 261
          From 272 to 277
          From 4096 to 4101
          From 4112 to 4117
          From 4128 to 4133
     DISCRETE INPUTS
          From 0 to 5
     HOLDING REGISTERS
          From 32 to 32
     INPUT REGISTERS
          From 16 to 27
          From 48 to 48
```

Listing 1. Inferred configuration of the Modbus/TCP slave device.

As part of a responsible disclosure process, we have communicated our findings to CIRCE lab technicians (one of the authors of this paper) and helped them establish appropriate security countermeasures to protect the system against each issue found during our investigation.

4.4. Simulating the CIRCE laboratory with MOSTO

As a last experiment, we recreated the CIRCE lab system with MOSTO. In the simulated system we can analyze Modbus protocol vulnerabilities and perform attacks, as well as test solutions, without disturbing the behavior of the real system.

We have created three virtual machines using Virtual Box 5.2 to reproduce the test environment on a single host machine. *Machine 1* runs Ubuntu Server 16.04 and runs MOSTO acting as a Modbus slave device. As the master device, *Machine 2* runs Windows 10 Education and runs MOSTO acting as a Modbus master device. *Ma-*

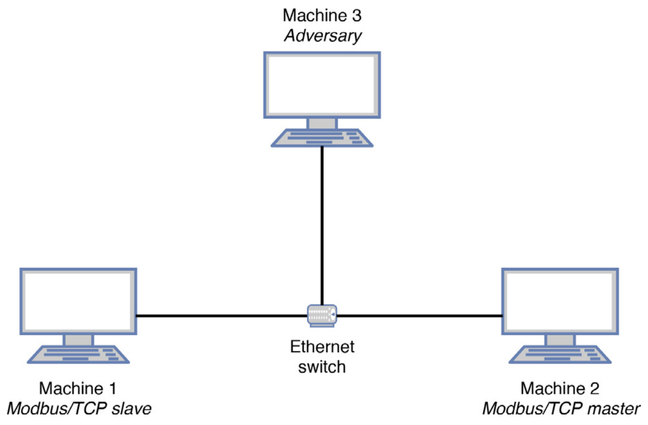
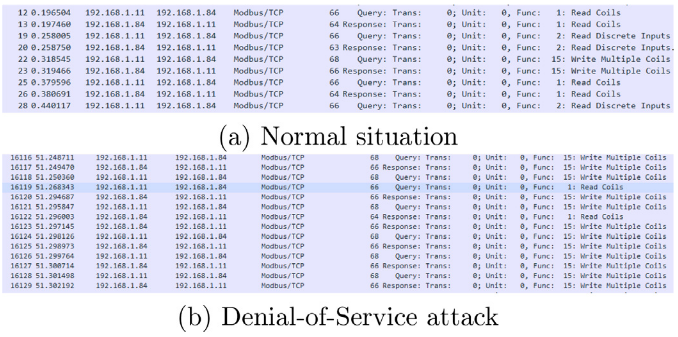


Fig. 9. Recreation of the CIRCE laboratory.



```
   12 0.196504   192.168.1.11   192.168.1.84    Modbus/TCP          66     Query: Trans:     0; Unit:    0, Func:   1: Read Coils
   13 0.197460   192.168.1.84   192.168.1.11    Modbus/TCP          64 Response: Trans:     0; Unit:    0, Func:   1: Read Coils
   19 0.258005   192.168.1.11   192.168.1.84    Modbus/TCP          66     Query: Trans:     0; Unit:    0, Func:   2: Read Discrete Inputs
   20 0.258750   192.168.1.84   192.168.1.11    Modbus/TCP          63 Response: Trans:     0; Unit:    0, Func:   2: Read Discrete Inputs.
   22 0.318545   192.168.1.11   192.168.1.84    Modbus/TCP          68     Query: Trans:     0; Unit:    0, Func:  15: Write Multiple Coils
   23 0.319466   192.168.1.84   192.168.1.11    Modbus/TCP          66 Response: Trans:     0; Unit:    0, Func:  15: Write Multiple Coils
   25 0.379596   192.168.1.11   192.168.1.84    Modbus/TCP          66     Query: Trans:     0; Unit:    0, Func:   1: Read Coils
   26 0.380691   192.168.1.84   192.168.1.11    Modbus/TCP          64 Response: Trans:     0; Unit:    0, Func:   1: Read Coils
   28 0.440117   192.168.1.11   192.168.1.84    Modbus/TCP          66     Query: Trans:     0; Unit:    0, Func:   2: Read Discrete Inputs
```

(a) Normal situation

```
 16116 51.248711    192.168.1.11    192.168.1.84    Modbus/TCP           68     Query: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16117 51.249470    192.168.1.84    192.168.1.11    Modbus/TCP           66 Response: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16118 51.250360    192.168.1.11    192.168.1.84    Modbus/TCP           68     Query: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16119 51.268343    192.168.1.11    192.168.1.84    Modbus/TCP           66     Query: Trans:    0; Unit:   0, Func:   1: Read Coils
 16120 51.294687    192.168.1.84    192.168.1.11    Modbus/TCP           66 Response: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16121 51.295847    192.168.1.11    192.168.1.84    Modbus/TCP           68     Query: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16122 51.296003    192.168.1.84    192.168.1.11    Modbus/TCP           64 Response: Trans:    0; Unit:   0, Func:   1: Read Coils
 16123 51.297145    192.168.1.84    192.168.1.11    Modbus/TCP           66 Response: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16124 51.298126    192.168.1.11    192.168.1.84    Modbus/TCP           68     Query: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16125 51.298973    192.168.1.84    192.168.1.11    Modbus/TCP           66 Response: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16126 51.299764    192.168.1.11    192.168.1.84    Modbus/TCP           68     Query: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16127 51.300714    192.168.1.84    192.168.1.11    Modbus/TCP           66 Response: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16128 51.301498    192.168.1.11    192.168.1.84    Modbus/TCP           68     Query: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
 16129 51.302192    192.168.1.84    192.168.1.11    Modbus/TCP           66 Response: Trans:    0; Unit:   0, Func:  15: Write Multiple Coils
```

(b) Denial-of-Service attack

Fig. 10. Communication traces under different circumstances.

chine 3 has been added to the network as a malicious agent, and runs Kali Linux 17.2. This machine is used to inject Modbus traffic to *Machine 1*. A description of our experimental system setup is shown in Fig. 9.

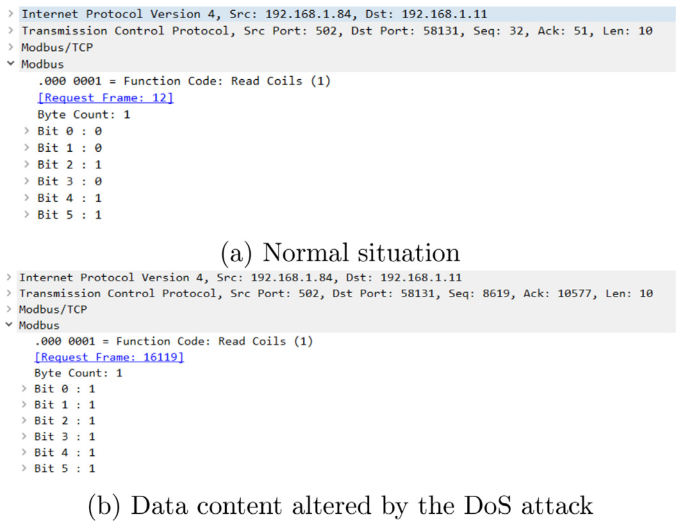
The results of our experimentation show that the traffic injection into the simulated system obtained the complete Modbus data structure of the slave device. Thus, we have managed to recreate a real system with our set of tools.

4.5. Denial of service attacks

As Modbus protocol does not have any kind of security mechanisms, an attacker can force a Modbus slave device into unwanted states continuously. Here, we have developed a Python script to send a large number of Modbus messages to modify the state of the real Modbus slave device, while exhausting its hardware resources. We performed this experiment when the CIRCE laboratory was undergoing maintenance to avoid disruption of normal operation. The goal of this script, which is also publicly available in our GitHub repository[4], is to perform a Denial of Service (DoS) attack on the slave device.

Fig. 10 shows a snapshot of the network flow captured in normal operation and under the DoS attack. Similarly, Fig. 11 shows the data content in both situations. Note that the number of messages exchanged increases in the second scenario, as well as the content of the data has changed. Similarly, the delay between consecutive queries is less in the second scenario, thus choking the device's resources and exhausting it due to the large number of messages to be processed.

[4] See https://github.com/reverseame/MOSTO-pentesting-Modbus.



```
> Internet Protocol Version 4, Src: 192.168.1.84, Dst: 192.168.1.11
> Transmission Control Protocol, Src Port: 502, Dst Port: 58131, Seq: 32, Ack: 51, Len: 10
> Modbus/TCP
v Modbus
    .000 0001 = Function Code: Read Coils (1)
    [Request Frame: 12]
    Byte Count: 1
  > Bit 0 : 0
  > Bit 1 : 0
  > Bit 2 : 1
  > Bit 3 : 0
  > Bit 4 : 1
  > Bit 5 : 1
```

(a) Normal situation

```
> Internet Protocol Version 4, Src: 192.168.1.84, Dst: 192.168.1.11
> Transmission Control Protocol, Src Port: 502, Dst Port: 58131, Seq: 8619, Ack: 10577, Len: 10
> Modbus/TCP
v Modbus
    .000 0001 = Function Code: Read Coils (1)
    [Request Frame: 16119]
    Byte Count: 1
  > Bit 0 : 1
  > Bit 1 : 1
  > Bit 2 : 1
  > Bit 3 : 1
  > Bit 4 : 1
  > Bit 5 : 1
```

(b) Data content altered by the DoS attack

Fig. 11. Data content under different circumstances.

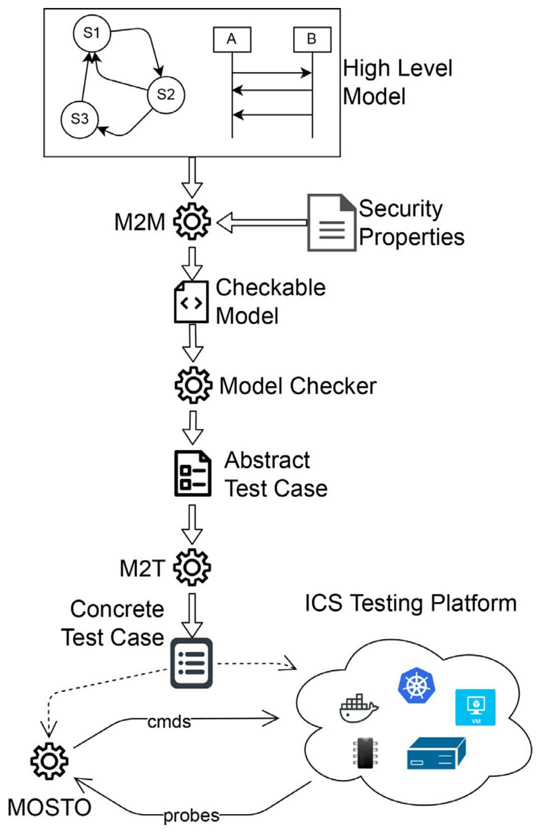


Fig. 12. The model-driven security framework.

5. Towards a model-driven security framework

This section introduces a model-driven framework that combines the use of formal methods, testing techniques, and simulation to demonstrate security properties in ICS networks. The experimental scenario carried out in the CIRCE laboratory and shown above is only a first ad-hoc experience of applying MOSTO in a specific setting. However, MOSTO is part of a broader investigation whose overview is shown in Fig. 12.

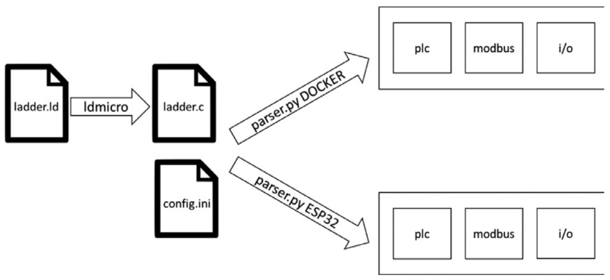


Fig. 13. From Ladder to ESP32/Docker.

We envision a framework that unites a model-driven approach, driven through formal methods with test and simulation techniques, to test security properties in a real/simulated environment.

Starting from a *high-level model* of the ICS under test and its behavior, a model-to-model transformation (*M2M*) is responsible for generating a formal model (*Checkable Model*) that — regarding an adequate translation of one or more *security properties* — is ready to be used as input for an off-the-shelf model checker. The model checker generates *Abstract Test Cases* (ATCs) that represent the sequences of steps capable of demonstrating that the security properties are not met. The research in Nardone et al. (2016) describes the part of this approach where such test sequences are generated through Dynamic State Machines (Nardone et al., 2015), the Promela language (Mikk et al., 1998), and the SPIN model checker (Holzmann, 2004).

The stepping of the ATCs into concrete executable sequences is handled by a Model-to-Text (M2T) transformation, which adapts the abstract sequences of steps to the specific simulation/test environment. In this last case, MOSTO plays the role of test engine, executing the specific test cases and exchanging commands and probes with the system under test. To be flexible enough to adapt to different system configurations, an ICS test platform can be used considering real-world PLCs, PLCs simulated through the use of microcontrollers such as ESP32 devices, and software-defined environments where SCADA systems can be virtualized and/or PLC software containerized.

Let us highlight the relationship between our methodology and the concept of Digital Twins (DT), which is receiving increasing attention from the academic and practitioner communities. The ultimate goal of a vulnerability assessment (regardless of its intent, whether malicious by attackers or ethical by auditors) is to obtain new and unknown information about the system. To this aim, DT are a powerful tool to create a simulation of a specific system, capable of also evaluating possible countermeasures through modeling and simulation. The proposed approach is aimed at building a DT system by combining the explicit knowledge manually embedded in the high level model and the knowledge obtained by auditing the ICS under study using MOSTO. Additional discussion is given in Section 7, which also addresses vulnerability testing and Artificial Intelligence approaches.

5.1. Simulating a ladder program

For specificity, design and implementation tips on how to deal efficiently with the flexibility of the environment are reported in this paper. In particular, the problem of simulating a PLC is discussed, proposing a solution for further investigation. The goal is to generate executable code for both ESP32 and Docker container to emulate a PLC. Fig. 13 proposes a workflow of tools and artefacts that transform a program expressed in Ladder Diagrams (LD) into both executable code.

The workflow is based on `ldmicro`, an open source toolset for transforming an LD program for the PIC16 and AVR architectures[5]. The `ladder.ld` program is edited and used to generate the `ladder.c` file. This C file structures the code with a main loop that implements the classical control loop (that is, read input reading, apply control logic, and write output). Another tool (called `Ladder2X`) is then used to parse it and deploy this code on specific platforms. The information needed by the Ladder2X tool is reported in a `config.ini` file. Useful information ranges from IP addresses to scheduling information and Modbus-specific data. An example of this file is shown in Listing 2.

```
[timing]
MODBUS_WELCOME_TASK_PERIOD_MS = 100
MODBUS_CONNECTION_TASK_PERIOD_MS = 100
PLC_TASK_PERIOD_MS = 100

[network]
MY_IP = 192.168.1.1
GATEWAY_COMMA_SEPARATED = 192,168,8,1
SUBNET_COMMA_SEPARATED = 255,255,255,0
SSID = "MY-WIFI"
PASSWORD = "MY-PASSWORD"

[modbus]
MODBUS_PORT = 502
MODBUS_BUFF_SIZE = 100
MAX_DISCRETE_INPUT = 100
MAX_COILS = 100
MAX_HOLD_REGS = 100
MAX_INP_REGS = 100
INTERNAL_RELAYS_OFFSET = 20
```

Listing 2. Example of a configuration file for the `Ladder2X` tool.

`Ladder2X` is a program developed in Python that takes a Ladder program as input, parses it, and based on the target device, generates the code to be deployed to it. In particular, the output of `Ladder2X` is a file that contains:

1. a *plcCycle()* method to be called cyclically, which performs the PLC task;
2. a *plcSetup()* method used to set the pin mode, in case the target device is a microcontroller;
3. a read and a write function for each input and output pin, containing both the call to *digitalRead()* and *digitalWrite()* of the IO module and reading or writing to the corresponding Modbus array;
4. a read and a write function for each internal state variable, containing the read or write in the corresponding Modbus data structure.

It is important to indicate that the code generated by the `Ladder2X` must be integrated in both platforms (ESP32 and Docker) with some third-party libraries (for instance, Modbus communication libraries, digital IO, etc.). Another valuable point of discussion is related to a shared data layer among all PLCs, due to the need to keep the information sent to all PLCs about changes of the virtualized physical inputs consistent. The details of this data layer, called `Virtualized Field`, are out of the scope of this paper.

6. Related work

As indicated in the survey published in Volkova et al. (2019), simulation remains key to enable security analysis of industrial systems. In this section, we review some works close to ours in the field of Modbus/TCP simulators. We first distinguish between works focused on the simulation of Modbus/TCP slave devices and

[5] https://cq.cx/ladder.pl

Modbus/TCP masters and then works on PLC simulation, machine and deep learning approaches, and other types of testing.

Simulation of Modbus/TCP slave devices. Almost all PLC manufacturers provide PLC simulation software. However, these programs are licensed and only to demonstrate how the PLC of a specific brand works. In this regard, it is worth mentioning `ModbusPal` (ModbusPal, 2020), an open source Java-based tool that only simulates Modbus/TCP slave devices and offers a user-friendly interface. It is mainly used for educational purposes and the characteristics of the simulated Modbus/TCP slave are configured dynamically. Although this feature is interesting, it is not really accurate compared to a real PLC, so it may be counterproductive to use this tool for security testing. On the contrary, MOSTO allows more detailed configuration, which allows a more accurate simulation of a real PLC.

Simulation of Modbus/TCP master devices. Modbus/TCP master device simulators are also used primarily for educational or controller testing purposes. In this regard, it is worth mentioning `qModMaster` (DomoticX, 2020), which is an open source Qt-based implementation of a ModBus master application that allows a user to communicate with an IP address and a port number, defining each field of a Modbus message explicitly. Unlike MOSTO, `qModMaster` does not allow the user to define the burst, the periodicity, or the delay between consecutive messages. That is, the tool simulates the transmission of a single message instead of a communication (more than one message).

A similar tool is `ModbusTester` (Modbus Tester, 2020), a simple utility that allows reading from a register and writing to a register of Modbus devices. Its graphical interface is very simple and intuitive, and it allows monitoring several Modbus slaves and/or data areas at the same time. However, it has the same limitations as `qModMaster`. These tools are useful for validating and unit testing of Modbus devices, but unlike our MOSTO, they cannot simulate Modbus communications properly.

Other related works. The OpenPLC project[6] is complementary to the work presented in this paper. OpenPLC presents many important features with respect the state of the art in PLC simulation, as described in Alves and Morris (2018).

Other works not mentioned here have more limitations, such as monolithic design, single platform tool, or lack of scalability. MOSTO overcomes these limitations by following a component design approach and being a cross-platform tool. To allow easy scalability, our simulation tool is based on the tool given in Kobayashi et al. (2007), which uses the Scapy Python module for Modbus/TCP packet manipulation. In addition, our simulator is scriptable and can therefore be easily integrated into existing security network analysis workflows.

Recently, the use of machine learning and deep learning approaches in industrial security is generating more and more interest from the scientific community. Some recent examples of these works are Anton et al. (2018); Antón et al. (2019); Das and Morris (2018). They clearly show the need for PLC simulators and PLC-based systems to enable such approaches. This is due to the data hunger of these approaches.

It is also worth mentioning the work in Voyiatzis et al. (2015), in which the authors introduced a Modbus/TCP fuzzer tool. This tool is used to empirically demonstrate bugs and vulnerabilities in eight implementations of the Modbus protocol. Our tool is complementary to theirs and can be integrated to improve the security assessment of Modbus devices.

Regarding the DT and the evaluation and control of cybersecurity in ICS, this topic is explored in several research papers. In Pokhrel et al. (2020), a systematic literature review of the use of DT in the prediction of security incidents is introduced. Masi et al. (2023) focus on critical infrastructures and cybersecurity-oriented DT construction. Similarly, Gehrmann and Gunnarsson (2020) report a solid architecture to synchronize the DT and the twinned system, explicitly addressing throughput and latency requirements. Finally, the issue of “attacks against and from Digital Twins” among the open areas of research is explicitly addressed in Faleiro et al. (2022).

7. Conclusions and future work

The security of ICS must be a primary goal to guarantee the safety of people. However, ICS security assessment is typically carried out with ad-hoc techniques that are difficult to formalize or to frame into a systematic assessment methodology to be repeatable with predictable results. Therefore, auditing and evaluating the security of these systems becomes a difficult and daunting task.

To facilitate this task, in this paper we have presented MOSTO, a Python-based Modbus/TCP simulation tool that allows us to recreate real systems. Our tool allows an auditor to assess the system security of existing industrial control systems without interfering in the course of work of the system under assessment. MOSTO has been released under GNU GPLv3 licenses for the sake of reproducibility and to foster research in this area. As case study, we have chosen the CIRCE linear particle accelerator lab of the Department of Mathematics and Physics of the University of Campania “Luigi Vanvitelli”, located in Italy. As a result of our security assessment with MOSTO, the lab’s technical staff understood the potential vulnerabilities the system was exposing, allowing them to take appropriate action and devise countermeasures to achieve a safer and more secure system. Finally, we present a modeling, analysis and simulation framework where MOSTO can be integrated. Using model transformation methods, this framework enables security testers and engineers to address very specific and ad-hoc security issues that broad spectrum solutions fail to address or detect.

As future work, our goal is to extend the simulator by testing it on real-world devices such as commercial PLCs and PLC emulators as OpenPLC. An additional research step will also include a deeper comparative analysis with other approaches for automatic analysis and simulation of the Modbus/TCP protocol. We also aim to fully build the modeling framework we envision, testing and validating it with real case studies.

Beyond the completion of the work reported in this paper, further research activities need to be discussed. In fact, computer security research focuses on obtaining information from a system, primarily through the use of penetration testing and scanning techniques and, more recently, feeding complex Machine Learning (ML) algorithms to infer knowledge of a system. This aspect is not explicitly considered in this paper and, in particular, in the model-driven methodology described in Section 5. Future research activities will consider these aspects (i.e., ML-based vulnerability scanning and anomaly detection) and will integrate them in the approach here proposed.

As a preliminary discussion on this topic, such an integration might consider defining some *a priori* model fragments (both state machines and sequence diagrams) as some kind of model library elements. In this case, the vulnerability analysis of the ICS software, as well as the use of ML-based classifiers, could detect some predefined error-prone behaviors in the system. Such model fragments could be then instantiated and used to enrich the high-level model in order to generate more realistic test cases. This future line of research will also facilitate the construction of a cybersecurity-oriented DT for the ICS system under audit.

[6] www.openplcproject.com

Declaration of Competing Interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ricardo J. Rodriguez reports financial support was provided by Spain Ministry of Science and Innovation. Ricardo J. Rodriguez reports financial support was provided by Government of Aragón. Stefano Marrone reports financial support was provided by University of Campania Luigi Vanvitelli.

CRediT authorship contribution statement

Ricardo J. Rodríguez: Conceptualization, Methodology, Investigation, Writing – original draft, Writing – review & editing, Supervision. **Stefano Marrone:** Methodology, Investigation, Resources, Data curation, Visualization, Writing – original draft, Writing – review & editing. **Ibai Marcos:** Software, Validation, Formal analysis, Data curation. **Giuseppe Porzio:** Investigation, Resources, Writing – original draft.

Data availability

The data that has been used is confidential.

Acknowledgments

The research of Ricardo J. Rodríguez was supported by the grant TED2021-131115A-I00 funded by MCIN/AEI/ 10.13039/501100011033 and by European Union NextGenerationEU/PRTR, and by the University, Industry and Innovation Department of the Aragonese Government under Programa de Proyectos Estratégicos de Grupos de Investigación (DisCo research group, ref. T21-23R). The research of Stefano Marrone is supported by the DARWINIST (aDversarial scenArios geneRation With dIgital twiNs In induSTry) project funded by Universitá della Campania “Luigi Vanvitelli”.

References

Alves, T., Morris, T., 2018. OpenPLC: An IEC 61,1313 compliant open source industrial controller for cyber security research. Computers & Security 78, 364–379. doi:10.1016/j.cose.2018.07.007.

Anton, S.D., Kanoor, S., Fraunholz, D., Schotten, H.D., 2018. Evaluation of machine learning-based anomaly detection algorithms on an industrial Modbus/TCP data set. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. Association for Computing Machinery, New York, NY, USA, pp. 1–9. doi:10.1145/3230833.3232818.

Antón, S.D.D., Sinha, S., Schotten, H.D., 2019. Anomaly-based Intrusion Detection in Industrial Data with SVM and Random Forests. In: Begusic, D., Rozic, N., Radic, J., Saric, M. (Eds.), 2019 International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2019, Split, Croatia, September 19–21, 2019. IEEE, pp. 1–6. doi:10.23919/SOFTCOM.2019.8903672.

Chen, B., Pattanaik, N., Goulart, A., Butler-purry, K.L., Kundur, D., 2015. Implementing attacks for Modbus/TCP protocol in a real-time cyber physical system test bed. In: 2015 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR), pp. 1–6.

CISA, 2020. Ransomware Activity Targeting the Healthcare and Public Health Sector. Accessed on April 24, 2023.,[Online; https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-302a].

Das, R., Morris, T., 2018. Machine learning and cyber security. In: Proceedings of the 2017 International Conference on Computer, Electrical & Communication Engineering (ICCECE), pp. 1–7. doi:10.1109/ICCECE.2017.8526232.

DomoticX, 2020. Modbus Software – qModMaster (Windows). Accessed on January 1, 2020., [Online; http://domoticx.com/modbus-software-qmodmaster-windows/].

Dragos, 2023. ICS/OT CYBERSECURITY – YEAR IN REVIEW 2022. Accessed on April 24, 2023., [Online; https://hub.dragos.com/ics-cybersecurity-year-in-review-2022].

ENISA, 2013. Can we learn from SCADA security incidents? Technical Report. European Union Agency for Cybersecurity.

Faleiro, R., Pan, L., Pokhrel, S.R., Doss, R., 2022. Digital Twin for Cybersecurity: Towards Enhancing Cyber Resilience. In: Xiang, W., Han, F., Phan, T.K. (Eds.), Broadband Communications, Networks, and Systems. Springer International Publishing, Cham, pp. 57–76.

Farwell, J.P., Rohozinski, R., 2011. StuxNet and the future of cyber war. Survival (Lond) 53 (1), 23–40. doi:10.1080/00396338.2011.555586.

Gallagher, S., 2019. “RobbinHood” ransomware takes down Baltimore City government networks. Accessed on April 24, 2023.,[Online; https://arstechnica.com/information-technology/2019/05/baltimore-city-government-hit-by-robbinhood-ransomware/].

Gehrmann, C., Gunnarsson, M., 2020. A digital twin based industrial automation and control system security architecture. IEEE Trans. Ind. Inf. 16 (1), 669–680. doi:10.1109/TII.2019.2938885.

Holzmann, G.J., 2004. The SPIN Model Checker: Primer and Reference Manual, Vol. 1003. Addison-Wesley Reading.

International Electrotechnical Commissionn, 2009. IEC TS 62443-1-1:2009 Industrial communication networks - Network and system security - Part 1-1: Terminology, concepts and models. Accessed on May 2, 2023., [Online; https://webstore.iec.ch/publication/7029].

International Organization for Standardization, 2021. ISO/SAE FDIS 21434 Road Vehicles-Cybersecurity Engineering. Accessed on May 2, 2023., [Online; https://www.iso.org/standard/70918.html].

Kaspersky ICS CERT, 2022. Threat landscape for industrial automation systems. Statistics for H1 2022. Accessed on April 24, 2023., [Online; https://ics-cert.kaspersky.com/publications/reports/2022/09/08/threat-landscape-for-industrial-automation-systems-statistics-for-h1-2022/].

Knowles, W., Prince, D., Hutchison, D., Disso, J., Jones, K., 2015. A survey of cyber security management in industrial control systems. Int. J. Crit. Infrastruct. Prot. 9, 52–80. doi:10.1016/j.ijcip.2015.02.002.

Kobayashi, T.H., Batista, A.B., Brito, A.M., Pires, P.S.M., 2007. Using a packet manipulation tool for security analysis of industrial network protocols. In: 2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007), pp. 744–747. doi:10.1109/EFTA.2007.4416847.

Kozik, R., Choras, M., 2013. Current cyber security threats and challenges in critical infrastructures protection. In: Proceedings of the 2nd International Conference on Informatics and Applications (ICIA), pp. 93–97. doi:10.1109/ICoIA.2013.6650236.

Langner, R., 2011. StuxNet: dissecting a cyberwarfare weapon. IEEE Security & Privacy 9 (3), 49–51. doi:10.1109/MSP.2011.67.

Lisova, E., Šljivo, I., ČauŠević, A., 2019. Safety and security co-analyses: a systematic literature review. IEEE Syst. J. 13 (3), 2189–2200. doi:10.1109/JSYST.2018.2881017.

Mahjabin, T., Xiao, Y., Sun, G., Jiang, W., 2017. A survey of distributed denial-of-service attack, prevention, and mitigation techniques. Int. J. Distrib. Sens. Netw. 13 (12). doi:10.1177/1550147717741463. 1550147717741463

Marrone, S., Rodríguez, R., Nardone, R., Flammini, F., Vittorini, V., 2015. On synergies of cyber and physical security modelling in vulnerability assessment of railway systems. Comput. Electr. Eng. 47, 275–285. doi:10.1016/j.compeleceng.2015.07.011.

Masi, M., Sellitto, G., Aranha, H., Pavleska, T., 2023. Securing critical infrastructures with a cybersecurity digital twin. Software and Systems Modeling 22 (2), 689–707. doi:10.1007/s10270-022-01075-0.

Mikk, E., Lakhnech, Y., Siegel, M., Holzmann, G.J., 1998. Implementing statecharts in PROMELA/SPIN. In: Proceedings of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, pp. 90–101.

Mlot, E.D.G., Saldana, J., Rodríguez, R.J., 2022. Towards a testbed for critical industrial systems: SunSpec protocol on DER systems as a case study. In: Proceedings of the 27th International Conference on Emerging Technologies and Factory Automation. IEEE, pp. 1–4. doi:10.1109/ETFA52439.2022.9921522.

Modbus Tester, 2020. Modbus Tester. Accessed on January 1, 2020., [Online; http://www.modbus.pl/Modbus%20Tester_MODBUS.PL.html].

ModbusPal, 2020. ModbusPal - Java MODBUS simulator. Accessed on January 1, 2020., [Online; http://modbuspal.sourceforge.net/].

MODICON, 2012. MODBUS Application Protocol Specification v1.1b3. Technical Report. MODICON, Inc., Industrial Automation Systems.

MODICON, 2018. MODBUS TCP Security v2.1. Technical Report. MODICON, Inc., Industrial Automation Systems.

Nardone, R., Gentile, U., Peron, A., Benerecetti, M., Vittorini, V., Marrone, S., De Guglielmo, R., Mazzocca, N., Velardi, L., 2015. Dynamic State Machines for Formalizing Railway Control System Specifications. In: Artho, C., Ölveczky, P.C. (Eds.), Formal Techniques for Safety-Critical Systems. Springer International Publishing, Cham, pp. 93–109.

Nardone, R., Rodríguez, R.J., Marrone, S., 2016. Formal security assessment of Modbus protocol. In: Proceedings of the 11th International Conference for Internet Technology and Secured Transactions. IEEE, pp. 142–147. doi:10.1109/ICITST.2016.7856685.

Peisert, S., Schneier, B., Okhravi, H., Massacci, F., Benzel, T., Landwehr, C., Mannan, M., Mirkovic, J., Prakash, A., Michael, J., 2021. Perspectives on the solarwinds incident. IEEE Security & Privacy 19 (02), 7–13. doi:10.1109/MSEC.2021.3051235.

Pokhrel, A., Katta, V., Colomo-Palacios, R., 2020. Digital twin for cybersecurity incident prediction: a multivocal literature review. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. Association for Computing Machinery, New York, NY, USA, pp. 671–678. doi:10.1145/3387940.3392199.

Rauscher, K., 2013. Writing the rules of cyberwar. IEEE Spectr 50 (12), 30–32. doi:10.1109/MSPEC.2013.6676992.

Siddavatam, I., Parekh, S., Shah, T., Kazi, F., 2017. Testing and validation of modbus/TCP protocol for secure SCADA communication in CPS using formal methods. Scalable Computing: Practice and Experience 18 (4), 313–330. doi:10.12694/scpe.v18i4.1331.

Smith, S., 2022. Out of Gas: A Deep Dive Into the Colonial Pipeline Cyberattack. 10.4135/9781529605679.

Sullivan, J.E., Kamensky, D., 2017. How cyber-attacks in Ukraine show the vulnerability of the u.s. power grid. The Electricity Journal 30 (3), 30–35. doi:10.1016/j.tej.2017.02.006.

Vera, A., Lynch, J., Carrega, C., 2021. Someone tried to poison a Florida city by hacking into the water treatment system, sheriff says. Accessed on April 24, 2023., [Online; https://edition.cnn.com/2021/02/08/us/oldsmar-florida-hack-water-poison/index.html].

Volkova, A., Niedermeier, M., Basmadjian, R., De Meer, H., 2019. Security challenges in control network protocols: asurvey. IEEE Commun. Surv. Tutorials 21 (1), 619–639. doi:10.1109/COMST.2018.2872114.

Voyiatzis, A.G., Katsigiannis, K., Koubias, S., 2015. A Modbus/TCP Fuzzer for testing internetworked industrial systems. In: 2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA), pp. 1–6. doi:10.1109/ETFA.2015.7301400.

Wang, Y., Yang, J., 2017. Ethical hacking and network defense: choose your best network vulnerability scanning tool. In: 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA), pp. 110–113.

Williams, T., 1994. The Purdue enterprise reference architecture. Comput. Ind. 24 (2–3), 141–158. doi:10.1016/0166-3615(94)90017-5.

Ricardo J. Rodríguez is an Associate Professor at the University of Zaragoza, Spain. He received his Ph.D. in computer science and system engineering from the University of Zaragoza. He was also a Visiting Professor in the University of Campania "Luigi Vanvitelli" (Italy) during two three-month periods in 2016 and in 2018, and in the TU Ilmenau (Germany) during a three-month period in 2017. His research interests include survivability analysis, program binary analysis, and contactless cards security. He has been involved in reviewing tasks for international conferences and journals.

Stefano Marrone is an Associate Professor in Computer Engineering at the Universitá della Campania "Luigi Vanvitelli". His interests include the definition of model driven processes for the design and the analysis of transportation control systems, complex communication networks and critical infrastructures and in the use of formal methods to critical systems design. He is involved in research projects with both academic and industrial partners. He authored more than 50 papers appeared in international proceedings of conferences and journals.

Ibai Marcos received a B.Sc. in Telecommunications Technology and Services Engineering from the University of Zaragoza in 2018. His research interests include network security, communication protocols, and forensics.

Giuseppe Porzio is a Ph.D. student at the Universitá della Campania "Luigi Vanvitelli". His research interests include network security and industrial critical systems.