A Stealthy False Command Injection Attack on Modbus based SCADA Systems

Wael Alsabbagh[1,2], Samuel Amogbonjaye[2], Diego Urrego[2] and Peter Langendörfer[1,2]
[1] IHP – Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany
[2] Brandenburg University of Technology Cottbus-Senftenberg, Cottbus, Germany
{Alsabbagh, Langendoerfer}@ihp-microelectronics.com
{urregdie, amogbolu}@b-tu.de

***Abstract*—Modbus is a widely-used industrial protocol in Supervisory Control and Data Acquisition (SCADA) systems for different purposes such as controlling remote devices, monitoring physical processes, data acquisition, etc. Unfortunately, such a protocol lacks security means i.e., authentication, integrity, and confidentiality. This has exposed industrial plants using the Modbus protocol and made them attractive to malicious adversaries who could perform various kinds of cyber-attacks causing significant consequences as Stuxnet showed. In this paper, we exploit the insecurity of the Modbus protocol and perform a stealthy false command injection scenario concealing our injection from the SCADA operator. Our attack approach is comprised of two main phases: 1) Pre-attack phase (offline) where an attacker sniffs, collects and stores sufficient valid request-response pairs in a database, 2) Attack phase (online) where the attacker performs false command injection and conceals his injection by replaying a valid response from his database upon each request sent from the HMI user. Such a scenario is quite severe and might cause disastrous damages in SCADA systems and critical infrastructures if it is successfully implemented by malicious adversaries. Finally, we suggest some appropriate mitigation solutions to prevent such a serious threat.**

***Index Terms*— SCADA; PLCs; ICSs; Modbus Protocol; Cyber-attacks; Command Injection Attacks;**

I. Introduction

Supervisory Control and Data Acquisition (SCADA) systems are employed by millions of industries and plants to monitor and control critical physical processes such as oil and gas facilities, water treatment systems, nuclear plants, electrical power grids, etc. SCADA systems provide users with a fully automated control, as well as a remote access and service monitor. Typical SCADA systems consist of different industrial components e.g., Engineering Work Stations (EWSs), Human Machine Interfaces (HMIs), Programmable Logic Controllers (PLCs), Input/Output (I/O) modules, sensors, valves, motors, and others [1]. Due to the necessity of having a remote management in the critical infrastructures, SCADA systems are increasingly connected to Ethernet and Transmission Control Protocol/Internet Protocol (TCP/IP) based networks e.g., Internet, as well as Virtual Private Network (VPN)-based remote access to reduce maintenance costs [2]. Unfortunately, this connectivity brings its own risks and exposes millions of systems to cyber-attacks from the outer world that were not existing in the air-gapped era [3].

The security of SCADA systems has been recently a major focus of the cyber-security researchers and industrial engineers due to the critical role they play in any automation company. It is not a secret that many old SCADA components with no security measures are still operating in many critical plants for two major reasons. First, industrial devices have a long life-cycle (twenty years or longer) which results in not being security patched (up-to-date) for a while. Secondly, there may be legacy devices that are not compatible with newer security-improved protocols. Wherefore, we should expect that many insecure SCADA devices are placed in remote locations and linked to the outer world via Internet. Thus, if a skilled adversary gains access to a SCADA network, he can perform malicious attacks to disrupt the physical process that the target system controls, which eventually might cause serious damages as Stuxnet [4], BlackEnergy [5], Shamoon [6], Kemuri [7] and German Steel Mill [8] showed.

Along with the system-level security concerns, SCADA protocols such as Modbus, Distributed Network Protocol (DNP3), High Level Data Link Control (HDLC), International Electro Technical Commission (IEC) 60870, etc. are substantially vulnerable and lack fundamental security mechanisms [9]. All these protocols provide client/server communications between different SCADA devices connected on different buses or networks. The Modbus protocol [10] is believed to be the most common industrial protocol implemented by hundreds of vendors on thousands of device models to transfer digital/analog inputs/outputs and register data between the connected devices e.g., HMIs and PLCs. Despite that Modbus provides the industrial community with simplicity, applicability, and efficiency, it contains multiple vulnerabilities that have allowed attackers to exploit the insecurity of the protocol and conduct different attacks e.g., reconnaissance activity, command injection, data injection, access injection, etc. Hijacking the interconnection between PLC and HMI devices represents the most often used attack scenario targeting SCADA systems using the Modbus such the one occurred against Maroochy water breach [11].

In this work, we introduce a stealthy False Command Injection (FCI) attack approach based on integrating a database containing real Modbus request-response pairs between PLC and HMI devices. The database is created prior to the launch of our attack i.e., offline. In our approach, an attacker placed in

a man-in-the-middle (MITM) position interrupts the Modbus requests sent from the HMI dropping them from the network so they do not reach the PLC, compares them to the ones existing in his database, and then replies to the HMI with the expected responses. Meanwhile, he can inject the PLC with false commands i.e., sending malicious requests that alter inputs or outputs causing a dangerous behavior on his will. In other words, our approach effectively decouples the PLC from the HMI i.e., it generates two independent communication flows: one between the PLC and the attacker, and the other between the attacker and the HMI. This scenario is quite severe as the SCADA operator is tricked in a way that he is always shown fake views while the PLC is processing malicious commands sent by the attacker. For a practical implementation, we conducted our approach on a virtual SCADA system based on OpenPLC[1] and ScadaBR [2] software. This is due to logistic constraints and the difficulty of using real-world SCADA systems for research purposes. Finally we suggested some security countermeasures and mitigation solutions to prevent such a serious threat.

The rest of the paper is structured as follows. Section II discusses related works, while Section III provides a security overview of Modbus protocol. In section IV, we illustrate our attack approach, and show the implementation as well as the resulting evaluation in section V. Finally, we suggest some security countermeasures and appropriate mitigation solutions in Section VI, and conclude this paper in section VII.

II. Related Work

Modbus protocol is very simple, efficient and public free on one hand, But on the other hand it has many vulnerabilities that allows an adversary to perform reconnaissance activity or use arbitrary commands. Possible vulnerabilities in the Modbus specification and major implementations of the protocol were investigated by Hitsi [12]. Such weaknesses can be exploited to perform spoofing, replay, and flooding attacks. Morris et al. [13] illustrated theoretical data injection and Denial of Service (DoS) attacks against industrial equipment that relies on Modbus. Such attacks stem from the protocol's insufficient security measures for data integrity and availability. Morris, in a follow-up work [14], described and tested reconnaissance, response injection, command injection, and DoS attacks, and also elaborated on several standalone and stateful Intrusion Detection System (IDS) rules in an attempt to deter such incidents. Nardone et al. [15] formally analyzed and assessed the Modbus protocol in terms of the security features each variant provides. The work by Tsalis et al. [16] demonstrated that even in the presence of encryption, side-channel attacks might reveal information on Modbus protocol messages. Using a testbed comprising of virtual machines running on Linux, Parian et al. [17] detailed on two attacks, namely manipulation of packets via malware-infected hosts and classic MITM attacks i.e., Address Resolution Protocol (ARP) poisoning. Rosa et al. [2] showed the implementation of a set of attacks targeting a Hybrid Environment for Design and Validation (HEDVa). For a practical attack scenario, the authors built and configured a small testbed controlled by Modbus PLCs. As a part of their work, they conducted network reconnaissance, MITM attack, and finally injected PLCs with dangerous Read/Write (R/W) coils requests.

All the aforementioned works focused on confusing the physical processes controlled by exposed PLCs using the vulnerabilities of the Modbus protocol. But, the SCADA operator could detect and disclose these attacks easily as they can observe abnormal changes displayed on the HMIs. In our paper, we overcome this challenge and conceal our attack by sending the HMI fake views similar to the ones it expects to receive as illustrated in Section IV.

III. Modbus Protocol and Vulnerabilities

Modbus is an application layer messaging protocol located at the seventh level of the OSI[3] model. It provides a master/slave communication between devices connected on different buses and networks. Figure 1 depicts a typical SCADA communication where a data acquisition server or an HMI will run as a Modbus client component (master) and a PLC will run as its pair device, that is, a server component (slave).

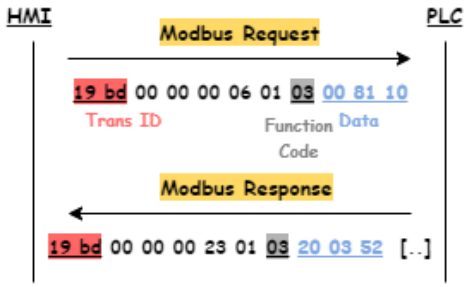

Fig. 1: Example of interaction between Modbus master and slave devices

The master device (HMI) sends a Modbus request to the connected slave device (PLC) to poll the data. The PLC replies upon the request with a Modbus response to the HMI. If the request is not correct, then the HMI sends exception response to the PLC. Figure 2 shows the architecture of a Modbus frame encapsulated over TCP/IP protocol. The Function code field determines the action required from the the PLC to do. Table I gives the details of some function codes and their corresponding actions. These function codes are the most frequently used in an interaction between PLCs and HMIs in SCADA systems.

The Modbus protocol itself lacks various security features which exposes it to cyber-attacks hijacking the Modbus communication between the connected devices and manipulating the frames to inject false commands/data into the PLC. However, in the following we list the most reported vulnerabilities in Modbus protocol as described in [19]–[21]:

[1]https://openplcproject.com/

[2]http://www.scadabr.com.br/

[3]https://www.fortinet.com/resources/cyberglossary/osi-model

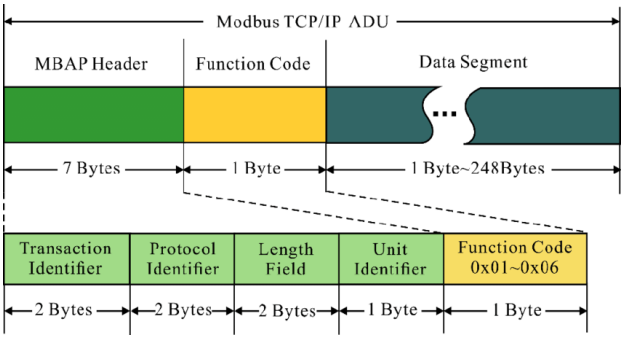

Fig. 2: Modbus TCP/IP frame format, adopted from [18]

Table I: Function codes and their corresponding actions

Function Code	Modbus Function
0x01	Read Coil Status
0x02	Read Discrete Input
0x03	Read Holding Registers
0x04	Read Input Registers
0x05	Write Single Coil
0x06	Write Single Holding Register
0x15	Write Multiple Coils
0x16	Write Multiple Holding Registers
0x17	Report Slave ID

- Integrity of Modbus frame is not verified by peer devices [22], [23]. The frame can be altered by an attacker and peer devices cannot reveal this manipulation.
- There is no facility for maintaining confidentiality of messages. The Modbus frames are transferred in plain text and any attacker placed in MITM position can sniff the packet and access the frame information.
- It does not support time-stamp for the frames. This is one of the critical problems because peer devices cannot know whether the received response is obtained for the recent or old request. Therefore, any manipulation may happen due to mismatch of real-time field values.
- Modbus is an open protocol and it had a simple frame format. Thus, A network analyzer tool usch as Wireshark[4] can be used by an attacker to retrieve the information from the network.

As a result of lacking the aforementioned security measures, Modbus is highly vulnerable to various cyber-attacks such as MITM attacks in the form of False Command Injection (FCI), False Access Injection (FAI), False Response Injection (FRI), Replay attacks and DoS attacks [24]–[27].

IV. Attack Description

Figure 3 shows a high-level overview of the attack scenario we perform to inject the PLC with false commands without being noticed by the HMI device. To this end, we need first to discover the network topology of the target system, then collect Modbus TCP/IP packets from the network traffic to create our database that eventually contains real request-response interaction pairs. However, these two steps are done prior to our injection attack. After collecting the needed pairs, we start our main attack by poisoning the ARP cache of the connected devices i.e., the HMI and PLC, and then inject the target PLC with false commands whilst we send the expected response packet upon each request to the HMI. This conceals our attack and the SCADA operator will be always shown fake views that he is expecting to see. In the following, we elaborate each attack step in detail.

A. Pre-Attack Phase (Offline)

Here, the attacker aims to get an overview of the network topology, open ports, connected devices, and communication protocols used in the target system. Then, he sniffs and collects real interactions between the HMI and PLC i.e., request-response pairs that both stations exchange over the Modbus TCP/IP frames.

1) Network Reconnaissance: Discovering the network is the first step that an attacker needs to do, meant to collect information/data about all the components of the SCADA environment, to identify the network topology, hosts and services. For instance, industrial devices such as PLCs and HMIs are identified by IP and Media Access Control (MAC) addresses, operating system versions and a set of services. Thus, to obtain these addresses and information we used the NMAP[5] Port scanner that identifies the Modbus protocol on the network. Figure 4 shows the scanning process where synchronize (SYN) packets are sent from the attacker machine over the network.

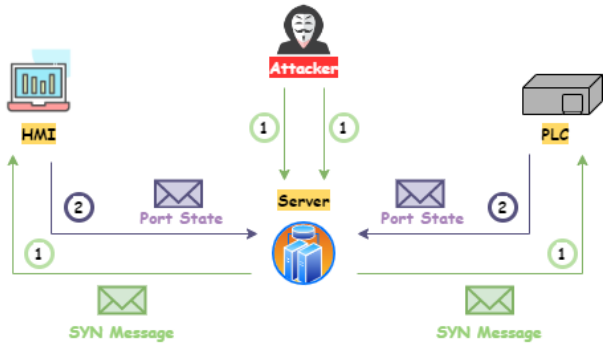

Fig. 4: Network Reconnaissance Attack

Using this technique, SYN packets can scan thousands of ports per second due to the fact that the TCP connection is not fully established (half-open communication). Therefore, it is difficult to detect by default network rules.

2) Sniffing and Collecting Data: NMAP tool provides the attacker only a perspective on the target system from the network point-of-view. Meaning that, it does not provide process-level information, which is required to implement sophisticated attacks. Thus, in the next step the attacker listens to

[4]https://www.wireshark.org/

[5]https://nmap.org/

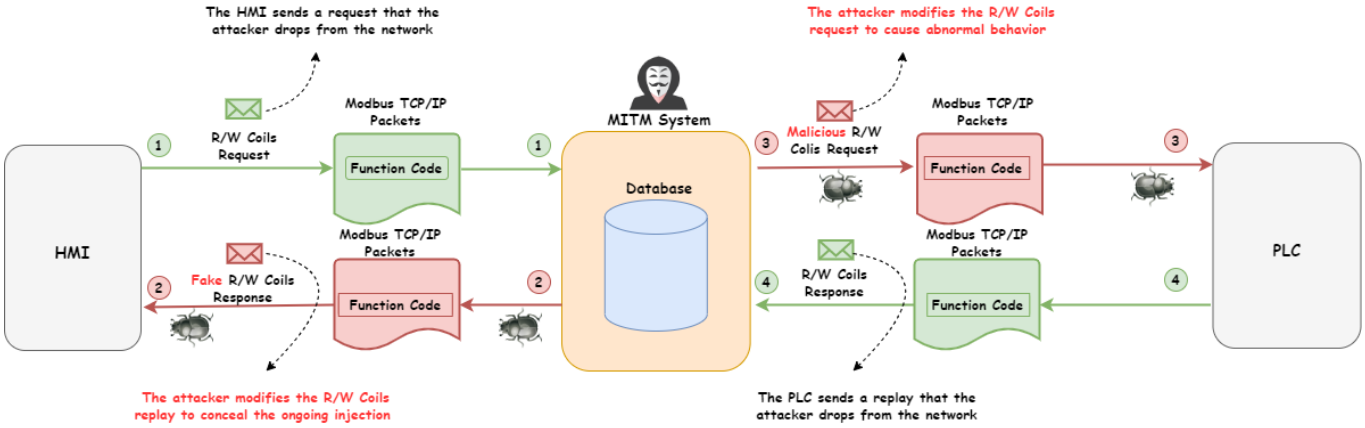

Fig. 3: High-level Overview of our Attack Approach

the network traffic and captures each Modbus TCP/IP request frame sent from the HMI alongside with its corresponding Modbus response(s) from the PLC. To this end, we first run a network analyzer software e.g., Wiershark. Our investigations showed that each request and its corresponding response(s) share the same Transaction Identifier (*TID*), *Unit ID* (which indicates how many frames the response from the PLC is comprised of), and *Function Code*. For this, encapsulated Modbus protocol messages can be extracted and grouped into request-response pairs based on those three parameters. Figure 5 shows an example of a Modbus interaction between PLC and HMI devices where the response from the PLC consists of two frames, and all the frames (request and response) have the same values: *0x19bd* in *TID*, *0x02* in *Unit ID*, and *0x03* in *Function Code*.

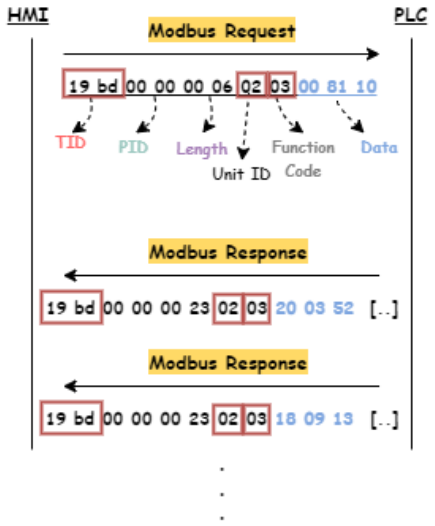

Fig. 5: Example of a Modbus request-response interaction between PLC and HMI

Based on those parameters, an attacker can easily extract and pair the Modbus packets as request-response frames. Moreover, he can analyze the packets deeper and gather more detailed information about each Modbus register effected the others. To quicken the comparison process during our injection, all the duplicate pairs are eliminated as depicted in figure 6. Please note that duplicated messages can exist if there is a periodical status check between the PLC and HMI. Finally, to collect a sufficient number of request-response pairs, the sniffing process should last for a reasonably long period of time e.g., in this work, we sniff the network for approx. 30 minutes. For our virtual SCADA system presented in figure 9, we managed successfully to create a database containing 18 request-response pairs. It is worth mentioning that pairing the captured Modbus frames in our database into request-response frames, helps the attacker to win the strict race condition that the HMI and PLC must meet before he replies his forged Modbus response frame to the HMI.

B. Attack Phase (Online)

At the end of the previous stage, an attacker has the Modbus request-response frames that are frequently exchanged between the HMI and PLC. So, he can start his major attack by first placing himself between the HMI and PLC (MITM position). This step is done by using the well-known ARP Poisoning approach. Then all messages will go through the attacker's machine who drops the received request frame from the network, compares it to the ones existing in the database and finally responses to the HMI with the expected correct response accordingly. In the mean time, the attacker sends the PLC a malicious request frame e.g., R/W coils request, and drops also the original response sent from the PLC to the HMI. In the following, we illustrate this phase in detail.

1) ARP Poisoning (MITM) Approach: The concept of an ARP poisoning attack comprises of two parts: an ARP spoofing and a communication hijacking. In the first stage, an attacker manipulates the ARP cache of both PLC and HMI devices by broadcasting malicious and forged "is-at" ARP messages over the network as depicted in figure 7. This technique forces both devices to send the packets through the

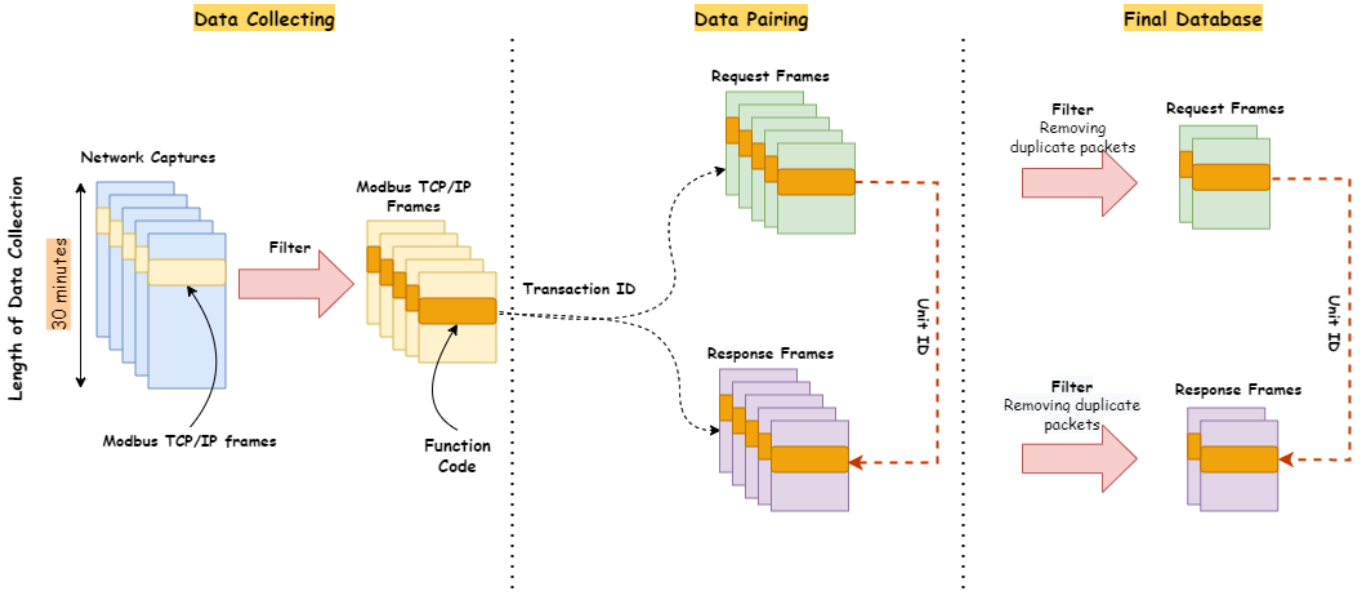

Fig. 6: Scheme of creating our database

attacker MAC address, and requires from the attacker to only know both IP and MAC addresses of the victims (e.g., HMI and PLC) which are already obtained in the early steps of the pre-attack phase.

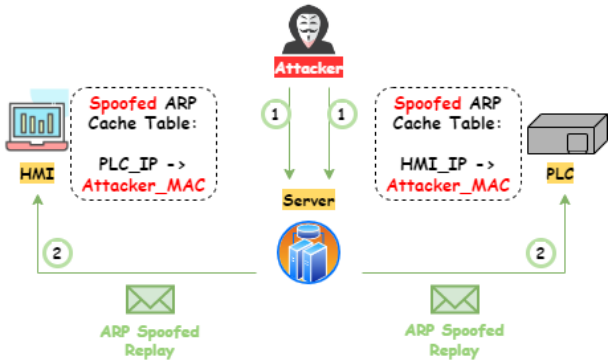

Fig. 7: ARP Poisoning Attack

As soon as the ARP cache of each victim is spoofed, the traffic gets redirected through the attacker's machine. At this point, the attacker is capable of reading all the Modbus messages transmitted between the HMI and PLC, and then forwarding them to the final destination respectively (interception attack), or actively change them before pushing them back to the network (modification attack). Please note that the HMI may generate a realistic state update, while decoupling HMI-PLC interactions. For this purpose, the adversary should reply to each Modbus request in real-time. Moreover, TCP session hijacking requires from the attacker to maintain the integrity of the TCP connection e.g., appropriate TCP sequence numbers to prevent losing the connection.

2) Stealthy Command Injection Attack: Figure 8 presents the full-chain of our injection scenario. When a Modbus request frame is sent from the HMI to the PLC, the attacker intercepts this frame, drops it from the network, compares the frame to the request frames in his database, and finally computes the corresponding response(s) accordingly. Meanwhile, he sends a forged Modbus request frame (e.g., R/W coils request) to the PLC impersonating the HMI. This approach is quite severe as the SCADA operator is tricked in a way that he is always shown fake views while the PLC is processing malicious commands sent by the attacker.

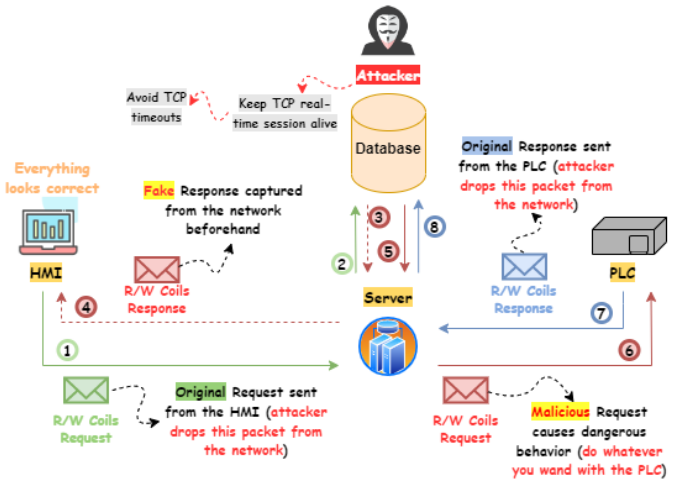

Fig. 8: Stealthy Command Injection Attack Scenario

This approach has a challenge. If an attacker stops his ARP poisoning attack i.e., he stops sending fake response messages to the HMI, and the HMI requires the PLC registers' values upon a Modbus request frame, the PLC responses reporting the current PLC status i.e., the modified inputs and outputs. Therefore, the SCADA operator can reveal that the system is

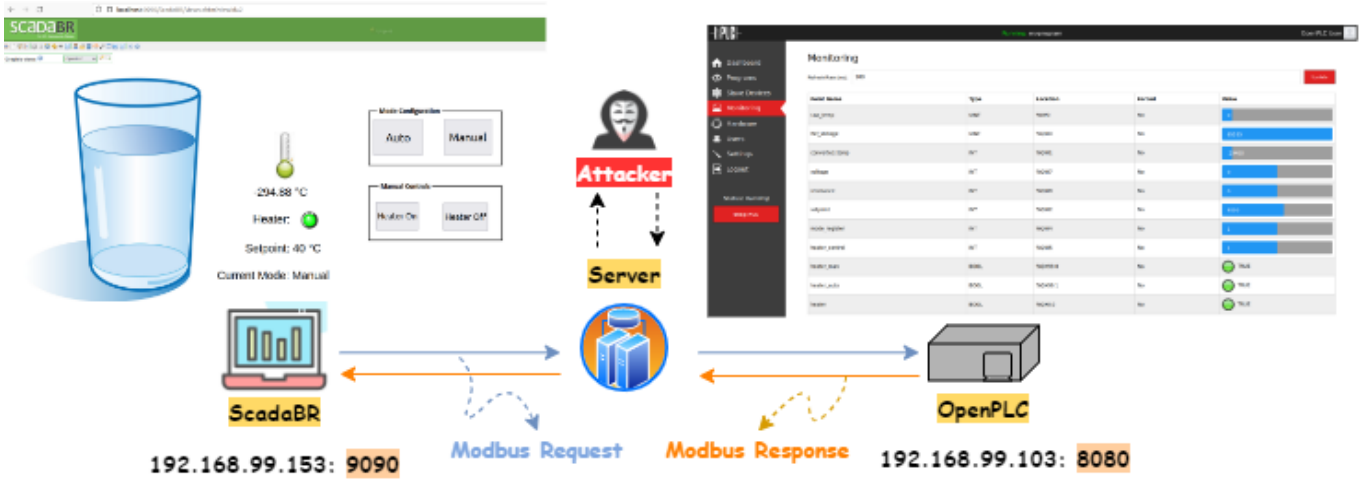

Fig. 9: Virtual SCADA system based on OpenPLC and ScadaBR software

operating abnormally. To overcome this challenge and make our attack even severer than it is, the attacker should re-initiate all the registers to the values stored prior to his attack. To this end, he needs to read all the register values before the attack, and writes these values again to the PLC before he closes the TCP communication with the devices. This restores the previously system state after stopping the attack and the PLC will report to the HMI the last view prior to the injection.

V. Implementation and Evaluation

A. Experimental Settings

*1) **Lab Setup**:* We evaluate our attack approach on a virtual SCADA system based on OpenPLC and ScadaBR software as shown in figure 9. The given virtual system represents a water tank heater experiment. It aims at keeping the temperature of a water tank at a certain value e.g. 40 C. Meaning that, if the temperature goes lower than 40 C, the corresponding sensor (Input) reports to the PLC, and the PLC responds by sending a control command to the heater (Output) to switch it ON. The heater remains ON until the temperature is again as high as the configured set-point. This process works in two configuration modes: Auto and Manual. The interaction between both OpenPLC and ScadaBR is handled using Modbus protocol over TCP/IP where ScadaBR is the master device and OpenPLC is the slave device. The control logic program is developed using the OpenPLC Editor in one of the five high-level programming languages defined in IEC-61131 [28]. The PLC Program is then compiled to an ST file before being uploaded to the OpenPLC.

*2) **Attacker Model**:* We assume that an attacker has access to the level-3 network of the Purdue Model [6]. This assumption is based on real world SCADA attacks e.g., TRITON [29] and BalckEnergy attack [5] that got access to the control center via a typical IT attack vector such as infected USB stick and social engineering attack. After the level-3 network access, an attacker can make use of software and libraries to communicate with the target PLC over the network. Since these assumptions have been reported to hold true in reports on real world attacks, we are convinced that our attack is a realistic one.

[6]https://www.goingstealthy.com/the-ics-prude-model/

B. Attack Implementation

After placing the attacker in MITM position between the ScadaBR and OpenPLC, he first reads all the current values that are stored in the PLC's memory registers e.g., coils, inputs, etc. Figure 10,11,12,13,14 and 15 show all the request-response frames exchanged between the attacker and the OpenPLC to obtain these values. To inject the PLC with different false commands, we developed a simple python script that sends crafted Modbus request frames to the target IP address on the open port 502. Table II shows the format of the frames we used to attack the given virtual system in figure 9. For instance, if an attacker aims at turning the heater OFF in the Auto mode, the following frame should be sent to the PLC: *0x010600040000*. Our results showed that we could successfully modify different inputs, outputs and data stored in the PLC registers as shown in figure 16.

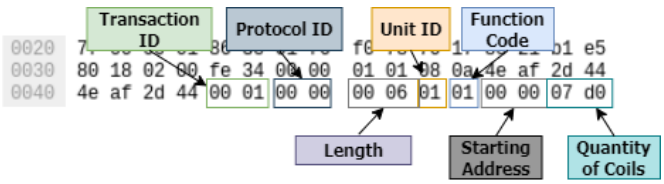

Fig. 10: Request frame sent from the attacker to the OpenPLC - Read all PLC coils

To conceal our injection from the SCADA operator, we developed an attacking tool that sniffs all the Modbus request frames from the network, compares them to the ones in our database, and finally responses with the appropriate response(s) to the HMI. Algorithm 1 depicts the main core of

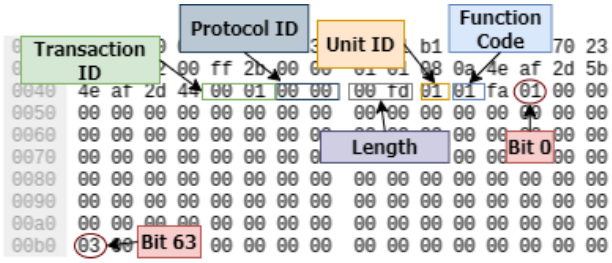

Fig. 11: Response frame from the OpenPLC - Only two coils have values: Bit 0 and 63

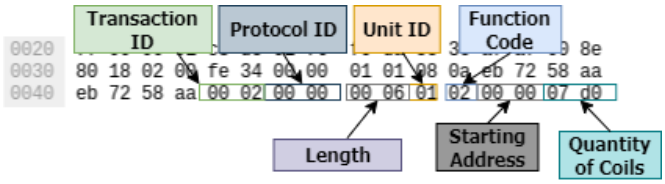

Fig. 12: Request frame from the attacker to the OpenPLC - Read all discrete registers

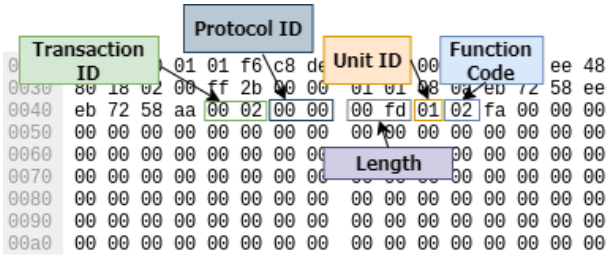

Fig. 13: Response frame from the OpenPLC - All the bits have the value of null

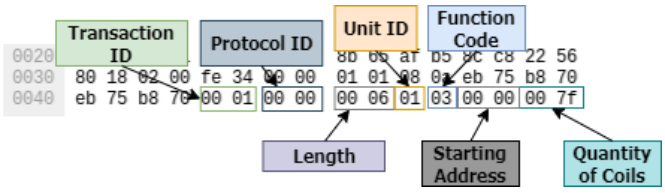

Fig. 14: Request frame from the attacker to the OpenPLC - Read all PLC holding registers

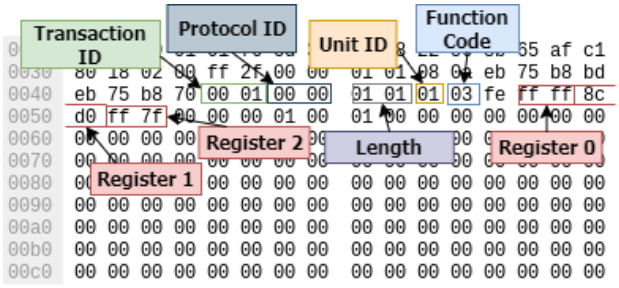

Fig. 15: Response frame from the OpenPLC - Only three registers have values in the PLC memory: Reg. 1,2 and 3

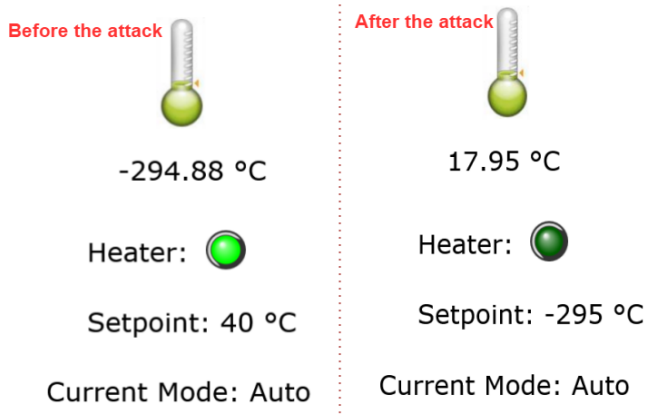

Fig. 16: The HMI monitor before and after the attack

the python script we used to design our attacking tool. After launching our tool, if the ScadaBR sends a Modbus request (e.g., write single register) to the OpenPLC, our MITM system intercepts this frame, compares it with the ones existing in our database, precisely with the request frames and finally replies to the ScadaBR by sending the corresponding response frame(s) based on the *Transaction ID*, *Unit ID*, and *Function Code*. It is worth mentioning that the attacker still needs to drop the original request from the network to avoid updating the PLC's registers. However, this is an easy task as the attacker only needs to not complete the full-cycle of the MITM attack i.e., he does not forward the frame to the final destination (PLC).

Algorithm 1 FCI Attack based on the Database Approach

```
Function inject (iface=eno, src_port)
packet = sniff (iface = eno, timeout = cfg_sniff_time)
save_pcap (sniff.pcap)
for pcap in rdpcap (save_pcap) do
    src_id = pcap [1:6], dest_id = pcap [7:12], mbus_pkt = filter_mbus(pcap)
    for pkt in mbus_pkt() do
        trans_id = pkt [1:2], Protocol_id = pkt[3:4], Length = pkt[5:6], Unit_id = pkt[7], function = pkt[8:9], start_address = pkt[10:11], data = pkt[12: ]
        if (src_ip = ScadaBR_src_ip & dest_ip = plc_ip) then
            for p in rdpcap (response_pcap) do
                if trans_id == p[1:2] & unit_id == p[7] & function == p[8:9] then
                    fgd_pkt = p[1:]) break
                end if
                P = P+1
            end for
        end if
        pkt = pkt + 1
    end for
    pcap = pcap + 1
end for
while time_slot() do
    sendp(iface, fgd_pkt , src_ip, port)
end while
END Function
```

Table II: Modbus frames and their corresponding actions

Action	Modbus frame			
Turning the Heater ON (Manual Mode)	0x01	0x06	0x00 0x05	0x00 0x01
Turning the Heater OFF (Manual Mode)	0x01	0x06	0x00 0x05	0x00 0x00
Turning the Heater ON (Auto Mode)	0x01	0x06	0x00 0x04	0x00 0x01
Turning the Heater OFF (Auto Mode)	0x01	0x06	0x00 0x04	0x00 0x00
Setting a new temperature	0x01	0x06	0x00 0x01	0x2f 0xff
Setting a new set-point	0x01	0x06	0x00 0x02	0x2f 0xff

VI. Security Countermeasures

Our experiments presented in this paper showed that there is no security in the Modbus protocol. Therefore, if attackers could access a Modbus device on a network they would be able to read/write whatever and whenever they want. Based on this fact, many industrial engineers implemented firewalls between the internet server and the control network to protect their systems i.e., all the Modbus devices are placed behind the firewalls see figure 17.

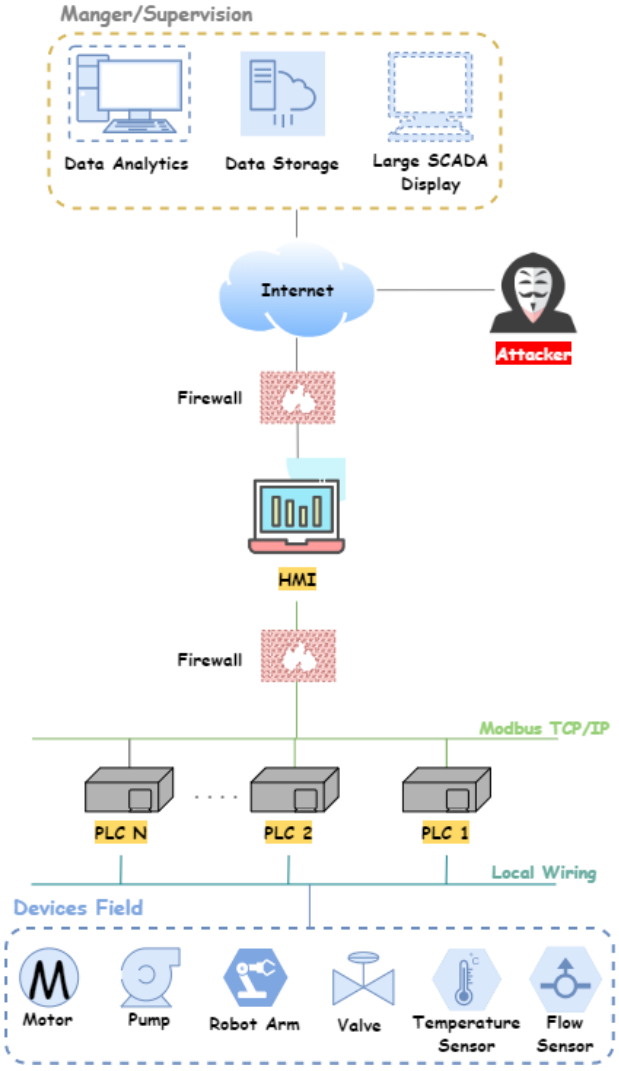

Fig. 17: SCADA system architecture using firewalls

This method splits Modbus devices from the internet but if any server behind the firewall is authorized to access the modbus devices through the firewall there is a vulnerability. Therefore, implementing firewalls alone without any additional security measures failed partly to prevent cyber-attacks. The advanced firewall presented in [30] would be a more reasonable protection method. The authors designed an industrial-specific firewall based on the Modbus protocol. Their firewall combines security policies with Deep Packet Inspection (DPI). An alternative solution would be using the modified version of the modbus protocol introduced in [31]. This new protocol version implements anti-replay techniques and authentication mechanisms that validate each packet received at modbus devices. Another appropriate solution would be the one introduced in [32] which deploys security functions in the messaging stack prior to the transmission. The authors used AES [33], RSA [34], or SHA-2 [35] algorithms to encrypt the Modbus packet while a secret key is exchanged between the master and the salve using a separate secure channel.

All the aforementioned security methods are reasonable solutions to secure Modbus based SCADA systems against our injection attack or similar scenarios if they are implemented.

VII. Conclusion and Future Work

This paper presented a false command injection (FCI) attack scenario against Modbus based SCADA systems, where an external adversary exploited insecurities of the Modbus protocol and injected the target PLC with malicious commands. To make our attack more challenging, we involved a database containing real request-response interaction pairs which helps the adversary to always reply to the HMI with the expected responses. This could conceal our attack from the operator, and decouples the PLC from the HMI i.e., the operator will not notice any abnormal behavior on the control site. Our attack scenario is quite severe in case it is conducted against real-world SCADA systems, and the consequences could be even disastrous if the targets are critical infrastructures or nuclear plants. To secure Modbus based SCADA systems, plants, and industrial environments we suggested some security countermeasures that assist in mitigating/detecting our attack or similar scenarios if they are applied.

In respect of securing SCADA systems, the Modbus organization released a newer Modbus protocol variant namely, Modbus Transport Layer Security (TLS[7]) running on the port

[7]https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf

802. This advanced protocol adds security specifications that the traditional Modbus protocol lacks (e.g., authentication and messages-integrity mechanisms) to prevent cyber-attacks such as DoS, MITM, replay attacks, etc. The Modbus TLS is still not well analyzed by the researchers from the security point-of-view. Thus, we aim in the future to investigate this developed protocol against our attack approach as the Modbus organization claimed that it is more resilient against cyber-attacks and even secure by an additional security layer between the server and client devices. Therefore, investigating the security of such a protocol will be more challenging and complex.

REFERENCES

[1] A. S. Aragó, E. R. Martínez and S. S. Clares, "SCADA Laboratory and Test-bed as a Service for Critical Infrastructure Protection," In Proceedings of the 2nd International Symposium on ICS & SCADA Cyber Security Research, St Pölten, Austria, 11–12 September 2014. DOI: 10.14236/EWIC/ICS-CSR2014.4.

[2] L. Rosa, T. Cruz, P. Simões, E. Monteiro and L. Lev, "Attacking SCADA systems: A practical perspective," 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2017, pp. 741-746, doi: 10.23919/INM.2017.7987369.

[3] W. Alsabbagh and P. Langendöerfer, "A New Injection Threat on S7-1500 PLCs - Disrupting the Physical Process Offline," in IEEE Open Journal of the Industrial Electronics Society, vol. 3, pp. 146-162, 2022, doi: 10.1109/OJIES.2022.3151528.

[4] N. Falliere, "Exploring Stuxnet's PLC infection process," in Virus Bul letin Covering Global Threat Landscape Conf., Sep. 2010.

[5] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the cyber-attack on the Ukrainian power grid," Technical report, SANS E-ISAC, March 18 2016. Available at: https://ics.sans.org/media/ESAC_SANS_Ukraine_DUC_5. pdf.

[6] Z. Dehlawi and N. Abokhodair, "Saudi Arabia's response to cyber conflict: A case study of the Shamoon malware incident," 2013 IEEE International Conference on Intelligence and Security Informatics, 2013, pp. 73-75, doi: 10.1109/ISI.2013.6578789.

[7] Verizon. (2016) Data breach digest. scenarios from the field. [Online]. Available at: https://www.ndia.org/-/media/sites/ndia/meetings-and-events/divisions/cfam/past-events/2016-august/cfam-forum-slides—verizon-data-breach-digest.ashx?mod=article_inline.

[8] R. M. Lee, M. J. Assante, and T. Conway, "German steel mill cyber attack," Industrial Control Systems, vol. 30, p. 62, 2014.

[9] M. A. Teixeira, T. Salman, M. Zolanvari, R. Jain, N. Meskin, Nader and M. Samaka, "SCADA System Testbed for Cybersecurity Research Using Machine Learning Approach," In Future Internet journal, vol. 10, 2018. DOI: 10.3390/fi10080076.

[10] Modicon Inc., Modicon Modbus Protocol Reference Guide - PI-MBUS-300 Rev. June 1996.

[11] J. Slay, and M. Miller, "Lessons learned from the maroochywater breach," Critical Infrastructure Protection, volume253/2007, pages 73–82. Springer Boston, November 2007.

[12] P. Huitsing, R. Chandia, M. Papa, and S. Shenoi, "Attack taxonomies for the modbus protocols," International Journal of Critical Infrastructure Protection, vol. 1, pp. 37–44, 2008.

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

[14] W. Gao and T. H. Morris, "On cyber attacks and signature based intrusion detection for modbus based industrial control systems," Journal of Digital Forensics, Security and Law, vol. 9, no. 1, p. 3, 2014.

[15] R. Nardone, R. J. Rodríguez, and S. Marrone, "Formal security assessment of modbus protocol," in 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST). IEEE, 2016, pp. 142–147.

[16] N. Tsalis, G. Stergiopoulos, E. Bitsikas, D. Gritzalis, and T. K. Apostolopoulos, "Side channel attacks over encrypted tcp/ip modbus reveal functionality leaks." in ICETE (2), 2018, pp. 219–229.

[17] C. Parian, T. Guldimann, and S. Bhatia, "Fooling the master: Exploiting weaknesses in the modbus protocol," Procedia Computer Science, vol. 171, pp. 2453–2458, 2020.

[18] Q. Bai, B. Jin, D. Wang, Y. Wang and X. Liu, "Compact Modbus TCP/IP protocol for data acquisition systems based on limited hardware resources," Journal of Instrumentation. 13(04):T04004-T04004. DOI: 10.1088/1748-0221/13/04/T04004.

[19] R. Nardone, R. J. Rodr´ıguez and S. Marrone, "Formal security assessment of Modbus protocol," in Proceedings of the 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST), pp. 142–147, IEEE, Barcelona, Spain, December 2016.

[20] A. Volkova, M. Niedermeier, R. Basmadjian, and H. D. Meer, "Security challenges in control network protocols: a survey," IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 619–639, 2019.

[21] L. Rosa, M. Freitas, S. Mazo, E. Monteiro, T. Cruz, and P. Simões, "A comprehensive security analysis of a SCADA protocol: from OSINT to mitigation," IEEE Access, vol. 7, Article ID 42156, 2019.

[22] A. M. Abdul and S. Umar, "Data integrity and security [DIS] based protocol for cognitive radio ad hoc networks," Indonesian Journal of Electrical Engineering and Computer Science, vol. 5, no. 1, pp. 187–195, 2017.

[23] K. Rambabu and N. Venkatram, "Contemporary affirmation of security and intrusion handling strategies of internet of things in recent literature," Journal of 9eoretical and Applied Information Technology, vol. 96, no. 9, pp. 2729–2744, 2018.

[24] S. Bhatia, N. Kush, C. Djamaludin, A. Akande, and E. Foo, "Practical modbus flooding attack and detection," in Proceedings of the Twelfth Australasian Information Security Conference (AISC 2014)[Conferences in Research and Practice in Information Technology, Volume 149], pp. 57–65, Australian Computer Society, Inc., Auckland, New Zealand, January 2014.

[25] A. M. Abdul and S. Umar, "Attacks of denial-of-service on networks layer of OSI model and maintaining of security," Indonesian Journal of Electrical Engineering and Computer Science, vol. 5, no. 1, pp. 181–186, 2017.

[26] L. Rajesh and P. Satyanarayana, "Detecting flooding attacks in communication protocol of industrial control systems," International Journal of Advanced Computer Science and Applications, vol. 11, no. 1, 2020.

[27] B. Chen, N. Pattanaik, A. Goulart, L. Karen, B. Purry, and D. Kundur, "Implementing Attacks for Modbus/TCP Protocol in a Real-Time Cyber Physical System Test bed," in Proceedings of the 2015 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR), pp. 1–6, IEEE, Charleston, SC, USA, May 2015.

[28] Tiegelkamp, M.; John, K. IEC 61131-3: Programming industrial automation systems. Springer, 1995.

[29] Attackers Deploy New ICS Attack Framework "TRITON," and Cause Operational Disruption to Critical Infrastructure. Accessed: Apr. 12, 2021. [Online]. Available: https://www.fireeye.com/blog/threat-research/2017/12/attackers-deploy-new-ics-attack-framework-triton.html.

[30] W. Shang, Q. Qiao, M. Wan, and P. Zeng, "Design and implementation of industrial firewall for modbus/TCP," JcP, vol. 11, no. 5, pp. 432– 438, 2016.

[31] I. N. Fovino, A. Carcano, M. Masera, and A. Trombetta, "Design and implementation of a secure modbus protocol," in International conference on critical infrastructure protection. Springer, 2009, pp. 83–96.

[32] A. Shahzad, M. Lee, Y.-K. Lee, S. Kim, N. Xiong, J.-Y. Choi, and Y. Cho, "Real time MODBUS transmissions and cryptography security designs and enhancements of protocol sensitive information," Symmetry, vol. 7, no. 3, pp. 1176–1210, 2015.

[33] J. Daemen and V. Rijmen, The design of Rijndael: AES-the advanced encryption standard. Springer Science & Business Media, 2013.

[34] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126, 1978.

[35] "Secure Hash Algorithm 2," National Institute of Standards and Technology (NIST), Standard, 2002.