2021 22nd IEEE International Conference on Industrial Technology (ICIT) | 978-1-7281-5730-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICIT46573.2021.9453483

# A Stealth Program Injection Attack against S7-300 PLCs

Wael Alsabbagh[1,2] and Peter Langendörfer[1,2]
[1]*IHP – Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany*
[2]*Brandenburg University of Technology Cottbus-Senftenberg, Cottbus, Germany*
E-mail: (Alsabbagh,Langendoerfer)@ihp-microelectronics.com

***Abstract*—Industrial control systems (ICSs) consist of programmable logic controllers (PLCs) which communicate with an engineering station on one side, and control a certain physical process on the other side. Siemens PLCs, particularly S7-300 controllers, are widely used in industrial systems, and modern critical infrastructures heavily rely on them. But unfortunately, security features are largely absent in such devices or ignored/disabled because security is often at odds with operations. As a consequence of the already reported vulnerabilities, it is possible to leverage PLCs and perhaps even the corporate IT network. In this paper we show that S7-300 PLCs are vulnerable and demonstrate that exploiting the execution process of the logic program running in a PLC is feasible. We discuss a replay attack that compromises the password protected PLCs, then we show how to retrieve the Bytecode from the target and decompile the Bytecode to STL source code. Afterwards we present how to conduct a typical injection attack showing that even a very tiny modification in the code is sufficient to harm the target system. Finally we combine the replay attack with the injection approach to achieve a stronger attack – the stealth program injection attack – which can hide the previous modification by engaging a fake PLC, impersonating the real infected device. For real scenarios, we implemented all our attacks on a real industrial setting using S7-300 PLC. We eventually suggest mitigation approaches to secure systems against such threats.**

***Index Terms*—S7-300 PLCs, Injection Attack, Stealthy Attack, Replay Attack, Fake PLC;**

## I. Introduction

Programmable logic controllers (PLCs) in industrial control systems (ICSs) are directly connected to physical processes such as production lines, electrical power grids and other critical plants. They are equipped with control logic that defines how to monitor and control the behaviour of the processes. Thus, their safety, durability, and predictable response times are the primary design concerns. PLCs are offered by several vendors such as Siemens, Allen-Bradley, Mitsubishi, Schneider and Modicon. Each vendor has its own proprietary firmware, programming, communication protocols and maintenance software. However, the basic hardware and software architecture is similar, meaning that all PLCs contain variables, and logic to control their inputs and outputs. The PLC code is written on an engineering station in the vendor's control logic language. The control logic is then compiled into an executable format, and downloaded to the PLC. The operating PLCs are monitored and managed via dedicated machines running Human Machine Interface (HMI) software. Modern networked PLCs and engineering stations are communicating over TCP/IP, but the higher-level protocols in use are typically proprietary.

Siemens S7 PLCs in the Simatic family [1] are estimated to have over 30% in the worldwide PLC market [2]. Furthermore, the Simatic line of products includes the "Totally Integrated Automation Portal" (TIA), which functions as the engineering station. The TIA Portal and PLCs communicate over the S7 network protocol. Unfortunately, the majority of industrial controllers are not designed to be resilient against cyber-attacks. This means if a PLC is compromised, then the physical process controlled by the PLC is also compromised which eventually could lead to a disastrous incident. Stuxnet [3] is perhaps the most well-known attack on ICS. This malware used a windows PC to target Siemens S7-300 PLCs that are specifically connected with variable frequency drives. It infects the control logic of the PLCs to monitor the frequency of the attached motors, and only launches an attack if the frequency is within a certain normal range (i.e. 807 Hz and 1,210 Hz).

Our focus is to investigate the possibility of exploiting Siemens S7-300 PLCs by: First compromising the security of PLCs and altering the control logic program running. Second hiding the infected logic from the engineering software at the control center which can acquire the logic from the PLC remotely and reveal the infection of the control logic. Please note that compromising the ICS network is out of the scope of this work and can be achieved via typical attack vector in our IT world such as infected USB, vulnerable web server, etc. Finding PLCs connected directly to the Internet is an easy task using search engines such as Shodan, Censys etc. All our attack scenarios are network based, and can be successfully launched by any attacker with network access to the target PLC.

The rest of the paper is organized as follows. Section II discusses related work, while our experimental setup is presented in section III. We illustrate our attack scenarios in IV, and discuss the results as well as future work in V.

## II. Related Work

Since 2010, ICSs, in particular their configuration and monitoring interfaces, have become targets for attackers. Recent examples of cyber-attacks on ICS occurred in Ukraine [4], [5]. These attacks caused controlling electrical distribution and wide-spread blackouts. In 2014, the German federal office for information security announced a cyber-attack at an unnamed steel mill, where hackers manipulated and disrupted control systems to such a degree that a blast furnace could not be properly shot down, resulting in massive damage [6]. At black Hat USA 2015 Klick et al. [7] demonstrated injection of malware into the control logic of a Simatic S7-300 PLC, without the service disrupting. In a follow on work, Spenneberg et al [8] presented a PLC worm. The worm spreads internally from one PLC to other target PLCs. During the infection phase the worm scans the network for new targets (PLCs). Both of the previous two works [7], [8] could be easily detected by the ICS operator once he requests the current control logic the target PLCs have. A Ladder Logic Bomb malware written in ladder logic or one of the compatible languages was introduced in [9]. Such malware is inserted by an attacker into existing control logic on PLCs. Anyway, this scenario requires from an attacker to be familiar with the programming languages that the PLC is programmed with. A recent work presented a reverse engineering-attack called ICSREF [10], which can automatically generate malicious payloads against the target system, and does not require any prior knowledge of the ICS. [11] demonstrated common-mode failure attacks targeting an industrial system that consists of redundant modules for recovery purpose. These modules are commonly used in nuclear power plant settings. The authors used DLL hijacking to intercept and modify the command-37 packets sent between the engineering station and the PLC, and could cause all the modules to fail. But this attack was also detectable by the ICS operator. A group of researchers in [12] presents a remote attack on the control logic of PLC. They were able to infect the PLC and to hide the infection from the engineering software at the control center. They implemented their attack on Schneider Electric Modicon M221, and its vendor-supplied engineering software (SoMachine-Basic). Another work demonstrated a series of attacks targeting Siemens S7-1200 PLCs [13]. Their investigation involves attacks like session stealing, phantom PLC, cross connecting controllers and denial of S7 connections. Our work differs in that we run a more difficult attack to be detected at the control center, as well as we use S7-300 PLCs which are larger deployed in industrial systems.

## III. EXPERIMENTAL SET-UP

In this section, we describe our experimental set-up, starting with the process to be controlled and presenting the equipment used afterwards.

### A. The physical process to be controlled

In our experiments, we are using the following application example: there are two aquariums filled with water that is pumped from one to the other until a certain level is reached and then the pumping direction is inverted see figure 1. The PLC is connected to the engineering station (TIA Portal) via an Ethernet cable, and exchanging data over the network to control the water level in each aquarium. The control process in this set-up is cyclically running as follows:

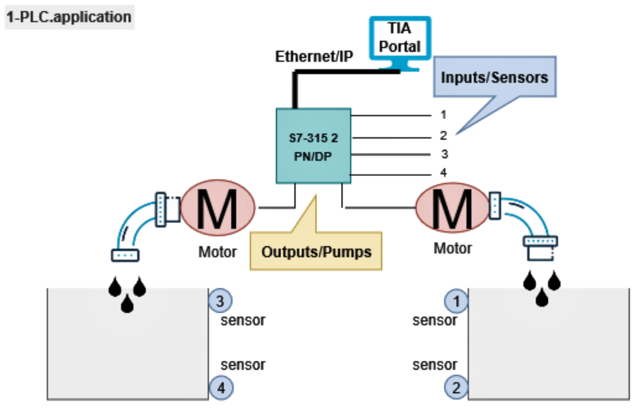


Fig. 1: Example application of our control process

The PLC reads the input signals coming from the sensors 1, 2, 3 and 4. The two upper sensors (Num. 1, 3) installed on both aquariums are reporting to the device when the aquariums are full of water, while the two lower sensors (Num. 2, 4) are reporting to the device when the aquariums are empty. Then the PLC powers the pumps on/off depending on the sensors' readings received.

### B. Hardware Equipment

In our testbed we have the following components: legitimate user, attacker machine, PLC, Communication processor, sensors and pumps which are described in detail in the following:

**1. Legitimate User -** It's a device that is connected to the PLC/CP using the TIA Portal software. Here, we use version 15.2[1] and Windows 7[2] as operating system.

**2. Attacker Machine -** it's a device that sneakily connects to the system without appropriate credentials. In our experiments, the attacker uses operating system LINUX Ubuntu 18.04.1 LTS[3] running on a Laptop[4].

**3. PLC S7-300 -** as mentioned before, we use Siemens products in our experiments and, particularly a CPU from the 300 family. The PLC used in this work is S7 315-2 PN/DP[5].

**4. Four capacitive proximity Sensors -** in our testbed, these are four sensors from Sick, Type CQ35-25NPP-KC16[6], with a sensing range of 25 mm and electrical wiring DC 4-wire.

[1]https://support.industry.siemens.com/cs/document/109752566/simatic-step-7-and-wincc-v15-trial-download-?dti=0& lc=en-US.

[2]https://www.microsoft.com/de-de/software-download/windows7.

[3]https://ubuntu.com/download/desktop.

[4]https://www.dell.com/support/home/de/de/debsdt1/productsupport/product/latitude-e6510/overview.

[5]https://support.industry.siemens.com/cs/pd/480032?pdti=td& dl=en& lc=en-WW.

[6]https://www.sick.com/de/en/proximity-sensors/capacitive-proximitysensors/cq/cq35-25npp-kc1/p/p244267.

**5. Two Pumps –** here, two DC-Runner 1.1 from Aqua Medic[7] with transparent pump housing 0-10 v connection for external control, maximum pumping output 1200 l/h and maximum pumping height: 1.5m.

### C. *Attacker Model and Attack Surface*

With regard to the types of attacks we consider, we assume that the attacker has already access to the network and is capable to send packets to the target via port number 102 at the S7 315-2PN/DP CPU. We also assume that the attacker has no TIA Portal software installed nor any prior knowledge about the actual process controlled by the PLC, how the PLC is connected, which communication protocol the PLC uses, or the logic program running on. In this work, the attack surface is a combination of device design and software implementation; more precisely, it is the implementation of the network stack, PLC specific protocol and PLC operating system.

## IV. Attack Description

As a consequence of the existing and already reported vulnerabilities, an attacker might carry out several attacks targeting industrial settings. Figure 2 shows an overview of the five attack scenarios that we perform in this work which consist of:

- Compromising the security of PLCs.
- Stealing control logic programs from PLCs.
- Decompiling the stolen byte code to STL source code.
- Infecting the control logic code.
- Hiding the ongoing injection attack from the ICS operator.

In the following we illustrate in detail all the five above-mentioned attack scenarios conducted against our example application given in section III.

### A. *Compromising the security measures of PLCs*

Siemens PLCs are normally password protected to prevent any unauthorized access and tampering of the logic programs running in their devices. Thus, it is not allowed to read/write from and to a controller without knowing the 8 characters password that a PLC is protected with. According to the best of our knowledge, most of the previous works mentioned two possibilities to bypass the password. Either by extracting the hash of the password and then pushing it back to the PLC (reply attack) [14], or using a representative list of "plain-text password, encoded-text password" pairs to brute-force each byte offline (brute-force attack) [17]. In this work, we also use a replay attack, but to remove the password that the PLC is set with, without any change in the current configuration of the target PLC.

A typical replay attack on the PLC consists of recording a sequence of packets related to a certain request/respond sent by the TIA Portal/PLC, and then pushing the captured/crafted packets back to the target at a later time without authorization.

[7]https://www.aquariumspecialty.com/aqua-medic-dc-runner-1-2-pump.html.

Technically, when a password is written to an S7-300 PLC, it is actually embedded in the SDB block which is defined by the static bytes: 0x3042, precisely in the block number 0000: 0x30303030 as shown in figure 3 [17]. Therefore, before any function or command is executed, the load process first checks SDB0 block (0x304230303030) to see if a password is already set. We have here two cases: 1) The PLC has no password and we can easily set a new password by sending an old captured load process sequence between the PLC and the TIA Portal which contains the setting of a new password. 2) The PLC has already one and we want either to update it with a new one, or to remove the password at all. In this work we are just interested in the second case i.e. when the PLC is already password protected.

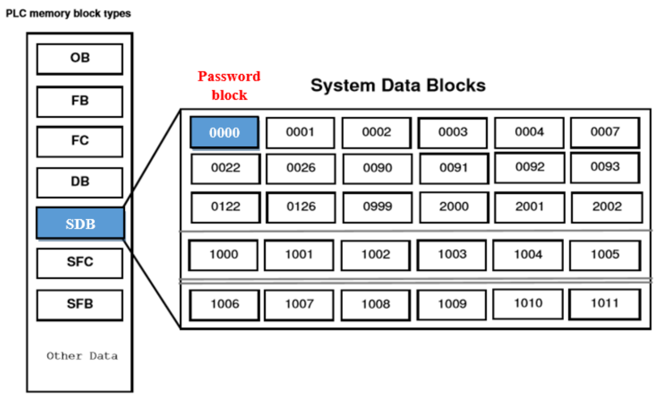


Fig. 3: S7-300 PLC memory structure

For this scenario the block SDB0 has a password and to update/remove the password, the old password should always be supplied by the user before any changes are done. Due to the fact that when the legitimate user updates the PLC with a new password, the PLC cannot overwrite the new password in the SDB0 directly. This means the PLC first needs to clear this block from its previous content, and then writes the new password into the block. This interesting finding triggered the idea that we can manipulate the setting of the password using the sequence of an old load process captured during updating an old password with a new one. For achieving this, we did the following: we first opened the TIA Portal, and changed the password in the configuration setting of the PLC. Then we downloaded the new configuration to the PLC providing the old password as a regular operation. In parallel, we recorded the entire load process between the TIA Portal and the PLC using Wireshark as a network sniffer. After recording the load process, we filtered the resulting packets keeping only the packets which are in charge of deleting the content of block SDB0 and ignoring the rest of packets which write the new password into SDB0. Similar to what Beresford presented in [14], we created our own load session by replacing the corresponding packets of updating a new password with only those needed to delete block SDB0, and eventually pushed our crafted packets sequence back to the target PLC as a new

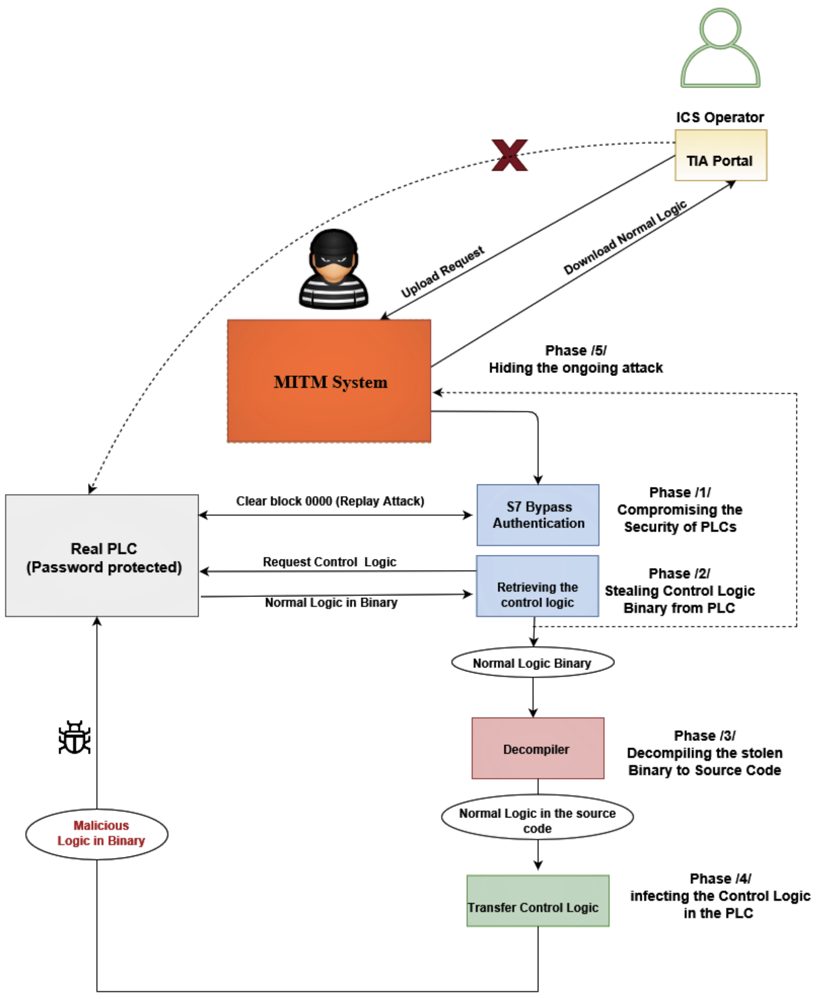

ICS Operator
TIA Portal
Upload Request
Download Normal Logic
MITM System
Phase /5/
Hiding the ongoing attack
Real PLC
(Password protected)
Clear block 0000 (Replay Attack)
S7 Bypass Authentication
Phase /1/
Compromising the Security of PLCs
Request Control Logic
Normal Logic in Binary
Retrieving the control logic
Phase /2/
Stealing Control Logic Binary from PLC
Normal Logic Binary
Decompiler
Phase /3/
Decompiling the stolen Binary to Source Code
Normal Logic in the source code
Malicious Logic in Binary
Transfer Control Logic
Phase /4/
infecting the Control Logic in the PLC

Fig. 2: High-level overview of our attack scenarios

load process. After the replay attack is done, we found that the PLC got updated and is no longer password protected. Algorithm 1 describes the core of our python script used to perform this attack. Our script is based on Scapy features and not only used to bypass the authentication of the PLC, but also to perform different replay attacks by replacing the corresponding captured/crafted packet for the attack presented in the next subsections.

## B. Stealing the Bytecode from the PLC

After the security measures are compromised, we communicate with the exposed PLC using a Python-snap7 library, and request the control logic running in the target device. This step is easy to be done by using the function full_upload (type, block number) from the Python-snap7 library. For our example application, we managed successfully to upload the program running in the PLC on the attacker machine by replacing the above-mentioned function's parameters with the corresponding block name and number i.e. we set the parameters on OB and 1 respectively.

## C. Decompiling the Bytecode to STL Code

In the next step, the Bytecode set and the corresponding STL instruction set of the user program running in the PLC need to be identified. For achieving that, we applied an offline division method to extract all the instructions used in our program one by one as follows. We opened the TIA Portal software, and programmed our target PLC with a certain code consisting of 10 times the same instruction. Here, we used the instruction NOP 0 which has no effect on the program. After that we downloaded this code to our PLC and recorded the packets containing the Bytecode which is the representation of 10 NOP 0 instructions as shown in figure 4a. We could identify that each NOP 0 instruction is represented as OxF000 in the Bytecode. Afterwards we opened the normal program (used in our example application) in the TIA Portal software and inserted NOP 0 before and after each instruction, and then downloaded the new program to the PLC. We recorded the packets that contain this new Bytecode, and identified each Hex-byte representing each instruction as shown in figure 4b. After extracting all instructions with the corresponding Hex-

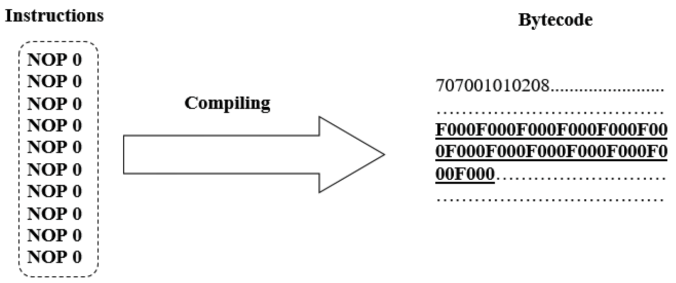
**Algorithm 1** Replay attack based on captured packets using Scapy

```
Function Replay (Pcapfile, Ethernet_interface, SrcIP, SrcPort):
    RecvSeqNum = 0
    SYN = True
    for packet in rdPcap (Pcapfile) do
        IP = packet[IP]
        TCP = packet[TCP]
        delete IP.checksum
        IP.src = SrcIP
        IP.port = SrcPort
        if TCP.flags == Ack or TCP.flags == RSTACK then
            TCP.ack = RecvSeqNum+1
            if SYN or TCP.flags == RSTACK then
                Sendp(packet, iface=Ethernet_interface)
                SYN = False
                Continue
            end if
        end if
        Recv = Srp1(packet, iface=Ethernet_interface)
        RecvSeqNum = rcv[TCP].seq
    end for
end Function
```

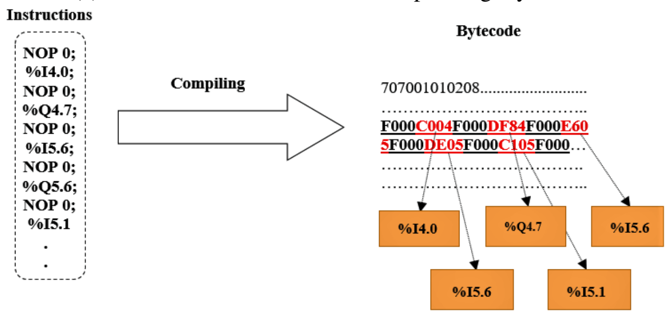
Bytes, we created a small Mapping Database of pairs: Hex-Bytes to their corresponding STL instructions, and used this mapping Database to convert the original machine Bytecode to its STL source code online. However, although our mapping database is very limited to the instructions used in our program, this method could be developed to map all Hex-Bytes to their STL instructions for any logic control program. Figure 5 presents the online mapping process, which takes the content of the code block (OB1) as input and utilizes the created Mapping-Database to retrieve the STL logic program, while figure 6 shows the S7 upload message captured by the attacker machine and the corresponding STL instruction on the TIA Portal station.

## *D. Infecting the control logic code*

After a successful decompiling, an attacker now has sufficient knowledge of the control process running in the PLC, and all needed to corrupt the system is as easy as replacing one or more instructions by new ones. In our case, our program has two outputs (2 pumps) and modifying any switch/sensor status that a pump reads will corrupt the physical process and make the system work incorrectly e.g. for aquarium.1 in our example application, swapping the low sensor switch %I4.4 (Normal-opened) with the high sensor switch %I4.3 (Normal-closed) will confuse the process as pump.1 turns off and on before the water reaches the required levels. This could lead to a physical damage in a real industrial system. Figure 7 shows that the user and the attacker bytecodes captured by Wireshark vary in only four bytes. Please note that in this scenario, the infected code size remains the same as the original one as we just swapped instructions without adding any new ones i.e. the code size was not increased as well as the cycle time remains the same when executing the attacker code.



(a) NOP instruction and the corresponding Bytecode



(b) Inserting NOP instructions in the Bytecode

Fig. 4: An example of NOP division method

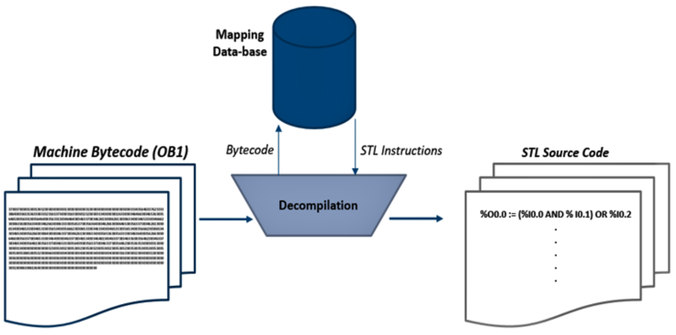
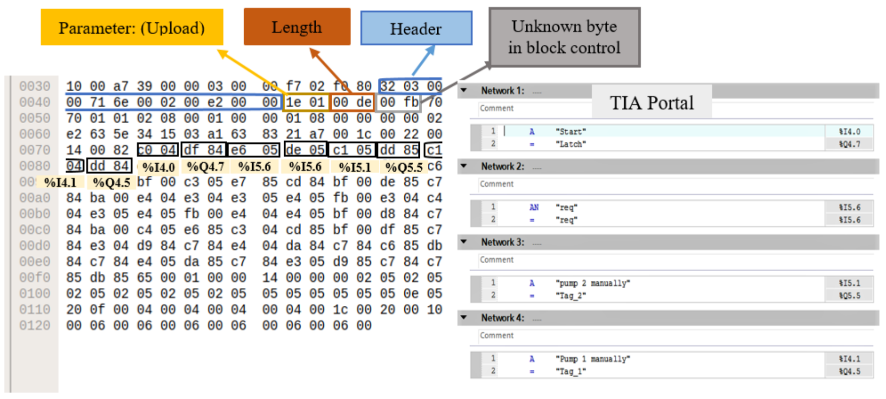


Fig. 5: The mapping process of the machine Bytecode to STL source code

It's worth mentioning that an attacker might skip the decompiling process and just replaces the original machine code with a totally new one even without knowing the program that the PLC runs. This holds true just in case there are no security means implemented which check changes in the code size, cycle time, etc. However, our method can cope with such protection mechanisms but on other hand the ICS operator can easily detect this attack if he requests and compares the online code run in the infected PLC with the offline code that he has on the TIA Portal.

## *E. Advanced Stealthy Injection Attack*

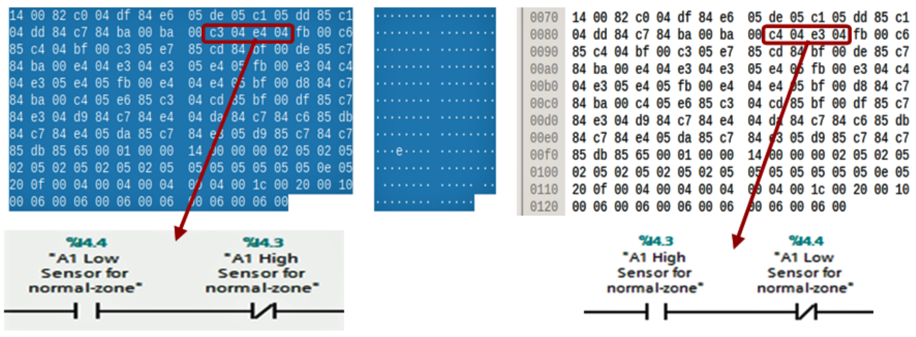
To overcome the challenge of keeping an ongoing injection attack hidden, we present a new method based on replacing the infected PLC with a fake one displayed for the ICS operator. Technically, the TIA Portal provides the user with the ability to compare the online code running in the remote PLC with



```
0030  10 00 a7 39 00 00 03 00  00 f7 02 f0 80 32 03 00
0040  00 71 6e 00 02 00 e2 00  00 1e 01 00 de 00 fb 70
0050  70 01 01 02 08 00 01 00  00 01 08 00 00 00 00 02
0060  e2 63 5e 34 15 03 a1 63  83 21 a7 00 1c 00 22 00
0070  14 00 82 c0 04 df 84 e6  05 de 05 c1 05 dd 85 c1
0080  04 dd 84                                      c6
00                   bf 00 c3 05 e7  85 cd 84 bf 00 de 85 c7
00a0  84 ba 00 e4 04 e3 04 e3  05 e4 05 fb 00 e3 04 c4
00b0  04 e3 05 e4 05 fb 00 e4  04 e4 05 bf 00 d8 84 c7
00c0  84 ba 00 c4 05 e6 85 c3  04 cd 85 bf 00 df 85 c7
00d0  84 e3 04 d9 84 c7 84 e4  04 da 84 c7 84 c6 85 db
00e0  84 c7 84 e4 05 da 85 c7  84 e3 05 d9 85 c7 84 c7
00f0  85 db 85 65 00 01 00 00  14 00 00 00 02 05 02 05
0100  02 05 02 05 02 05 02 05  05 05 05 05 05 05 0e 05
0110  20 0f 00 04 00 04 00 04  00 04 00 1c 00 20 00 10
0120  00 06 00 06 00 06 00 06  00 06 00 06 00
```

```
Network 1: .....
Comment
1   A    "Start"              %I4.0
2   =    "Latch"              %Q4.7
Network 2: .....
Comment
1   AN   "req"                %I5.6
2   =    "req"                %I5.6
Network 3: .....
Comment
1   A    "pump 2 manually"    %I5.1
2   =    "Tag_2"              %Q5.5
Network 4: .....
Comment
1   A    "Pump 1 manually"    %I4.1
2   =    "Tag_1"              %Q4.5
```

Fig. 6: S7 Upload message: Mapping the Bytecode with STL instructions


%I4.4
"A1 Low Sensor for normal-zone"
%I4.3
"A1 High Sensor for normal-zone"
%I4.3
"A1 High Sensor for normal-zone"
%I4.4
"A1 Low Sensor for normal-zone"

```
14 00 82 c0 04 df 84 e6  05 de 05 c1 05 dd 85 c1
04 dd 84 c7 84 ba 00 ba  00 c3 04 e4 04 fb 00 c6
85 c4 04 bf 00 c3 05 e7  85 cd 84 bf 00 de 85 c7
84 ba 00 e4 04 e3 04 e3  05 e4 05 fb 00 e3 04 c4
04 e3 05 e4 05 fb 00 e4  04 e4 05 bf 00 d8 84 c7
84 ba 00 c4 05 e6 85 c3  04 cd 85 bf 00 df 85 c7
84 e3 04 d9 84 c7 84 e4  04 da 84 c7 84 c6 85 db
84 c7 84 e4 05 da 85 c7  84 e3 05 d9 85 c7 84 c7
85 db 85 65 00 01 00 00  14 00 00 00 02 05 02 05
02 05 02 05 02 05 02 05  05 05 05 05 05 05 0e 05
20 0f 00 04 00 04 00 04  00 04 00 1c 00 20 00 10
00 06 00 06 00 06 00 06  00 06 00 06 00
```

```
0070  14 00 82 c0 04 df 84 e6  05 de 05 c1 05 dd 85 c1
0080  04 dd 84 c7 84 ba 00 ba  00 c4 04 e3 04 fb 00 c6
0090  85 c4 04 bf 00 c3 05 e7  85 cd 84 bf 00 de 85 c7
00a0  84 ba 00 e4 04 e3 04 e3  05 e4 05 fb 00 e3 04 c4
00b0  04 e3 05 e4 05 fb 00 e4  04 e4 05 bf 00 d8 84 c7
00c0  84 ba 00 c4 05 e6 85 c3  04 cd 85 bf 00 df 85 c7
00d0  84 e3 04 d9 84 c7 84 e4  04 da 84 c7 84 c6 85 db
00e0  84 c7 84 e4 05 da 85 c7  84 e3 05 d9 85 c7 84 c7
00f0  85 db 85 65 00 01 00 00  14 00 00 00 02 05 02 05
0100  02 05 02 05 02 05 02 05  05 05 05 05 05 05 0e 05
0110  20 0f 00 04 00 04 00 04  00 04 00 1c 00 20 00 10
0120  00 06 00 06 00 06 00 06  00 06 00 06 00
```

(a) The User Bytecode

(b) The Attacker Bytecode

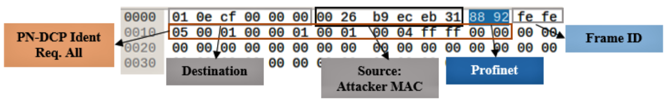
Fig. 7: Modifying the PLC's Bytecode

the offline one that the user has at the control center. This reveals any infection of the control logic. The main goal of our approach is to hinder the operator from uploading the actual infected code from the remote PLC by redirecting its connection to our fake PLC, which sends the uninfected version that we want the user to see. This hides our ongoing injection and achieves a full stealth attack. This scenario can be realized by using our MITM system (see figure 2) which is initiated after the attacker penetrated the ICS network and can send/receive messages to/from the TIA Portal/PLC remotely.

*1) **Impersonating a real PLC:*** An interesting fact that we found in our investigation is that the ICS operator can be tricked by connecting him to a fake PLC (Attacker machine) impersonating the real remote device. This is because of the existing vulnerability of PN-DCP protocol (Data link layer). The PN-DCP protocol is basically used for discovering devices or configuring devices' names, IP addresses, etc. TIA Portal requests all accessible devices in the network by broadcasting a certain packet called "identify all", and all S7 PLCs available will replay with a certain respond packet called "identify ok". The payload of the respond packet sent by each PLC contains all details of the device e.g. the name, IP address, vendor's name, subnets, etc. In this work, we aim at blocking the TIA Portal from reaching the remote PLC and connecting it to a fake PLC instead. This is done as follows: we recorded both "identify all" and "identify ok" packets between the TIA Portal and the remote PLC from during an old session, then modified the respond packet of the real PLC replacing the IP address of the PLC 192.168.0.1 with the IP address of our fake PLC 192.168.0.3.

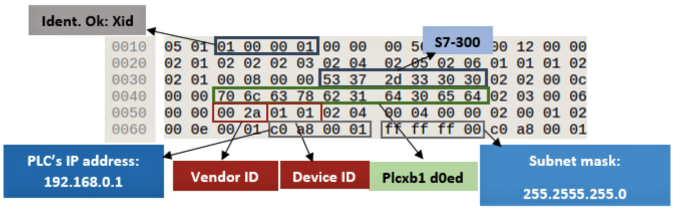
For a real scenario, we first implement our MITM system between the TIA Portal and the target PLC, so all the packets transferred between the two remote stations will go first through our MITM system and are then redirected depending on the attacker's ARP table (ARP Poisoning attack). Afterwards our MITM system sends an "identify all" packet through the network (see figure 8a), and the remote PLC responses by sending the "identify ok" packet back to the attacker as shown in figure 8b. Once the TIA Portal sends a new "identify all" packet through the network in an attempt to connect with the remote PLC, our MITM system listens to the network, identifies the request and drops the packet to prevent the real PLC from responding to this request, and then sends our crafted "identify ok" packet back to the TIA Portal as shown in figure 8c. Thus, the TIA Portal registers our fake PLC as the real PLC located at 192.168.0.3. Please note that the only difference between the real and crafted "identify ok" packets is the IP addresses.

The next step the legitimate user takes after finding an accessible PLC (which is in fact our fake PLC), he attempts



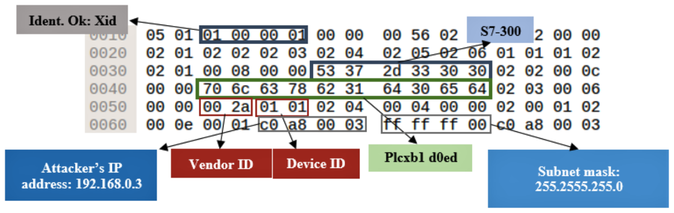
```
0000  01 0e cf 00 00 00 00 26  b9 ec eb 31 88 92 fe fe
0010  05 00 01 00 00 01 00 01  00 04 ff ff 00 00 00 00
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030              00 00
```

(a) Identify all request message from the attacker to the PLC



```
0010  05 01 01 00 00 01 00 00  00 5        00 12 00 00
0020  02 01 02 02 02 03 02 04  02 05 02 06 01 01 01 02
0030  02 01 00 08 00 00 53 37  2d 33 30 30 02 02 00 0c
0040  00 00 70 6c 63 78 62 31  64 30 65 64 02 03 00 06
0050  00 00 00 2a 01 01 02 04  00 04 00 00 02 00 01 02
0060  00 0e 00 01 c0 a8 00 01  ff ff ff 00 c0 a8 00 01
```

(b) Identify ok respond message sent from the PLC to the attacker

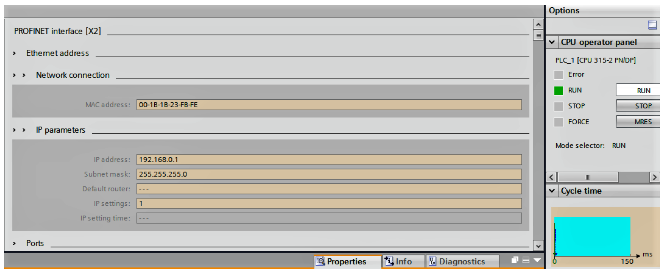


```
      05 01 01 00 00 01 00 00  00 56 02          2 00 00
0020  02 01 02 02 02 03 02 04  02 05 02 06 01 01 01 02
0030  02 01 00 08 00 00 53 37  2d 33 30 30 02 02 00 0c
0040  00 00 70 6c 63 78 62 31  64 30 65 64 02 03 00 06
0050  00 00 00 2a 01 01 02 04  00 04 00 00 02 00 01 02
0060  00 0e 00 01 c0 a8 00 03  ff ff ff 00 c0 a8 00 03
```

Ident. Ok: Xid
S7-300
Attacker's IP address: 192.168.0.3
Vendor ID
Device ID
Plcxb1 d0ed
Subnet mask: 255.2555.255.0

(c) identify ok respond message sent from the attacker to TIA Portal

Fig. 8: PN-DCP protocol messages



PROFINET interface [X2]

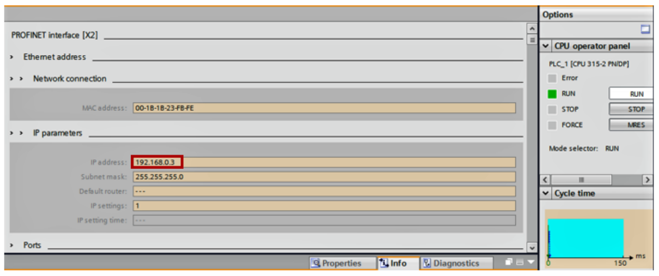
MAC address: 00-1B-1B-23-FB-FE

IP address: 192.168.0.1
Subnet mask: 255.255.255.0
Default router: ---
IP settings: 1
IP setting time: ---

PLC_1 [CPU 315-2 PN/DP]

Mode selector: RUN

(a) CPU's online diagnostic before the attack



PROFINET interface [X2]

MAC address: 00-1B-1B-23-FB-FE

IP address: 192.168.0.3
Subnet mask: 255.255.255.0
Default router: ---
IP settings: 1
IP setting time: ---

PLC_1 [CPU 315-2 PN/DP]

Mode selector: RUN

(b) CPU's online diagnostic after the attack

Fig. 9: Different IP addresses shown in TIA Portal

to establish an online session by initiating a TCP connection. Our MITM system will redirect the request to our fake PLC's IP address and establishes a TCP connection with the TIA Portal. After a successful connection, we need to keep the session between the TIA Portal and our fake PLC alive. Our investigation to solve this problem showed that S7-300 PLCs keep the online session with TIA Portal alive by exchanging four specific packets[8] along the time the user is online. Meaning that we only need to keep sending the two PLC's respond packets (S7Comm [response] & TCP [ACK]) which are sufficient to keep our connection alive with the TIA Portal.

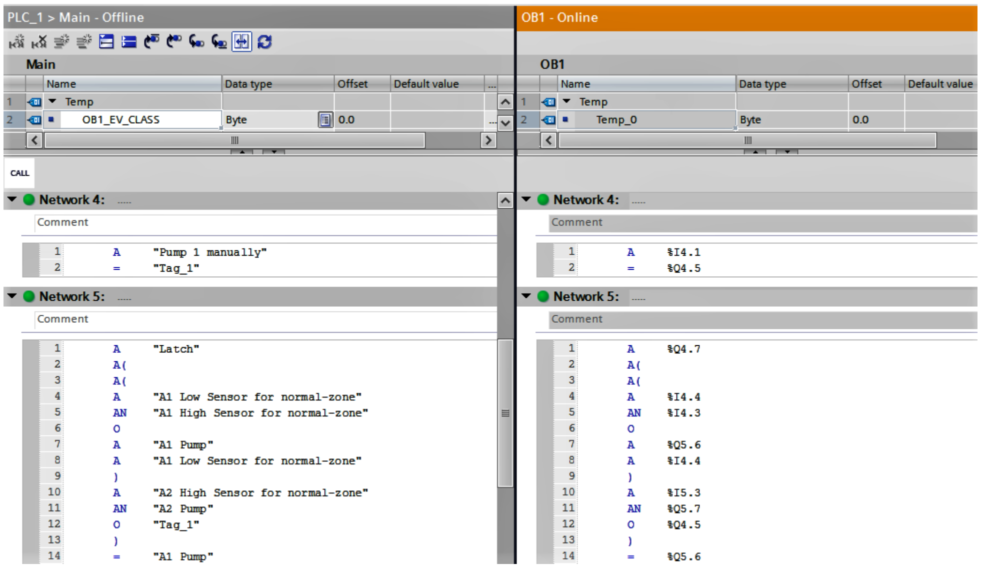
The ICS operator could potentially spot the abnormality in case he checks the differences in the IP address between the real PLC and the fake PLC. As shown in figure 9, there are differences in the IP addresses between the real PLC and our fake PLC that are displayed in the TIA Portal. But in normal operation this impersonating might go undetected as the IP address is only shown if the operator explicitly checks details of the Profinet interface, which is not required during an ongoing operation.

2) ***Transferring the Original logic to the TIA Portal:*** As we mentioned earlier once the ICS operator suspects that the remote PLC runs a different logic program than that it should run, he will request the current logic program (online program) and compare it with the one that he has on the TIA Portal (offline program). This allows him to detect any potential infection/modification. To trick this security check out, we recorded an old upload session between the TIA Portal and the remote PLC, and then modified the packets captured during the security check by the original code instead of the infected code. For a real scenario, once the ICS operator sends a new upload request over the network, our MITM system drops this request and sends our crafted load sequence back using the python script presented in algorithm 1. We managed successfully to upload the original code from the attacker machine to the TIA Portal while the PLC runs another code. This stealthy attack could cause significant harm in the relevant ICSs as the offline and online programs are matching fully and the engineer will not detect our ongoing injection attack unless he checks the IP address of the connected device in the Profinet interface. Figure 10 shows that both the offline and online programs are identical during our ongoing injection attack.

[8]1. S7Comm: POSCTR:[Request]. 2. S7Comm POSCTR: [Response]. 3. COTP TPDU (0). 4. TCP [ACK]

## V. DISCUSSION AND FUTURE WORK

In this paper, we presented an advanced stealthy attack scenario, for infecting the control logic including vulnerability exploitation, decompilation, injection, and concealment of the infection via a fake PLC approach. For a practical implementation we performed all our attack scenarios on real hardware/software used in industrial settings. We found that the ICS operator is able to detect this attack only if he checks the IP address of the connected device, taking into account that he needs to check the details of the Profinet interface, which is normally not required during an ongoing control operation. However, to mitigate the effect of such attacks we highly suggest some countermeasures to our attack such as protection and detecting of control logic. The first step to protect our systems from various sorts of attacks is to improve the isolation from other networks [15], combining this with standard security practices [16], and even defence-in-depth security in the control systems. In addition, a digital signature should be employed not only to the firmware as most of the


PLC_1 > Main - Offline

Main

|  | Name | Data type | Offset | Default value |
|---|---|---|---|---|
| 1 | Temp |  |  |  |
| 2 | OB1_EV_CLASS | Byte | 0.0 |  |

CALL

Network 4: .....

Comment

```
A      "Pump 1 manually"
=      "Tag_1"
```

Network 5: .....

Comment

```
A      "Latch"
A(
A(
A      "A1 Low Sensor for normal-zone"
AN     "A1 High Sensor for normal-zone"
O
A      "A1 Pump"
A      "A1 Low Sensor for normal-zone"
)
A      "A2 High Sensor for normal-zone"
AN     "A2 Pump"
O      "Tag_1"
)
=      "A1 Pump"
```

OB1 - Online

OB1

|  | Name | Data type | Offset | Default value |
|---|---|---|---|---|
| 1 | Temp |  |  |  |
| 2 | Temp_0 | Byte | 0.0 |  |

Network 4: .....

Comment

```
A      %I4.1
=      %Q4.5
```

Network 5: .....

Comment

```
A      %Q4.7
A(
A(
A      %I4.4
AN     %I4.3
O
A      %Q5.6
A      %I4.4
)
A      %I5.3
AN     %Q5.7
O      %Q4.5
)
=      %Q5.6
```

Fig. 10: Online and Offline control logic comparison shown in TIA Portal

PLC vendors do but also to the control logic. Furthermore, a mechanism to check the protocol header which contains information about the type of the payload is also recommended as a solution to detect and block any potential unauthorized transfer of the control logic. Finally, Siemens provides their devices with Multi-Point Interface (MPI) to communicate with other devices from SIMATIC family. The MPI protocol is also used as a programming protocol, and allows the TIA Portal to upload/download the control logic to/from the PLCs. This protocol is so far not supported by any network sniffer and still provides a more secure communication between the control center and the remote devices. This helps to prevent attackers from snooping which in turn improves security as listening and capturing packets transferred over the network is the main base for attackers to perform most of the attacks against ICSs. The exploits in this paper are efficient but not at all complicated as S7-300 PLCs still use the old version of S7 protocol which lacks of security mechanisms compared to the newer version (S7communication Plus) that the modern S7 PLCs e.g. S7-1200/1500 PLCs use. So in our future work we will investigate if our stealthy attack can be run successfully against the modern S7 PLCs. We are aware of the fact that this will be more challenging as S7comm plus supports improved security implementing anti-replay mechanisms and integrity checks.

## References

[1] Siemens. Modular PLC controllers SIMATIC S7, 2014. Available at: http://www.automation.siemens.com/ mcms/programmable-logic-controller/en/simatic-s7-controller.

[2] Electrical engineering Blog. The top most used PLC systems around the world. Electrical installation & energy efficiency, May 2013. Available at: http://engineering.electrical-equipment.org/electrical-distribution/the-top-most-used-plc-systems-around-the-world.html.

[3] N. Falliere, Exploring Stuxnet's PLC infection process, Sept. 2010.

[4] 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/E-SAC_SANS_Ukraine_DUC_5. pdf

[5] G. liang, S. R. Weller, J. Zhao, F. Luo, and Z.Y. Dong, "The 2015 Ukraine blackout: Implications for false data injection attacks," IEEE Transactions on Power Systems, 2016, doi: 10.1109/TPWRS.2016.2631891

[6] T. De Maizière, "Die Lage Der IT-Sicherheit in Deutschland 2014," The German Federal Office for Information Security,2014. Avaliable at: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/ Lagebericht2014.pdf

[7] J. Klick, S. Lau, D. Marzin, J. Malchow, and V. Roth, "Internet-facing PLCs-a new back orifice," in Black Hat USA 2015, Las Vegas, USA, 2015.

[8] A. Spenneberg, M. Brüggemann, and H. Schwartke, "PLC-blaster: A worm living solely in the PLC," in Black Hat Asia, Marina Bay Sands, Singapore, 2016.

[9] N. Govil, A. Agrawal, N. O. Tippenhauer, "On Ladder Logic Bombs in Industrial Control Systems," January, 2018, dio: 10.1007/978-3-319-72817-9_8.

[10] A. Keliris, and M. Maniatakos, "ICSREF: A framework for automated reverse engineering of industrial control systems binaries," in 26th Annual Network and Distributed System Security Symposium, NDSS 2019.

[11] B. Lim, D. Chen, Y. An, Z. Kalbarczyk, and R. Iyer, "Attack induced common-mode failures on plc-based safety system in a nuclear power plant: Practical experience report," in 2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC), pages 205–210.

[12] K. Sushma, A. Nehal, Y. Hyunguk, and A. Irfan, "CLIK on PLCs! Attacking Control Logic with Decompilation and Virtual PLC," 2019, dio: 10.14722/bar.2019.23xxx.

[13] H. Hui, and K. Mclaughlin, "Investigating Current PLC Security Issues Regarding Siemens S7 Communications and TIA Portal," 5th International Symposium for ICS & SCADA Cyber Security Research 2018 (ICS-CSR 2018), pages: 66-72, dio: 10.14236/ewic/ICS2018.8.

[14] D. Beresford, "Exploiting Siemens Simatic S7 PLCs," Black Hat USA, 2011.

[15] M. Stouffer, V. Pillitteri, "Guide to industrial control systems (ics) security," NIST special publication, 2015.

[16] "Framework for improving critical infrastructure cybersecurity version 1.1," National Institute of Standards and Technology, Tech. Rep., 2018, Available at: https://doi.org/10.6028/NIST.CSWP.04162018.

[17] H. Wardak, S. Zhioua and A. Almulhem, "PLC access control: a security analysis," 2016 World Congress on Industrial Control Systems Security (WCICSS), London, 2016, pp. 1-6, doi: 10.1109/WCICSS.2016.7882935.