2022

PLC Code Vulnerabilities and Attacks: Detection and Prevention

Abraham Serhane
University of Wollongong

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

PLC Code Vulnerabilities and Attacks: Detection and Prevention

Abraham Serhane

Supervisors:
Raad Raad
Willy Susilo

This thesis is presented as part of the requirement for the conferral of the degree:
Doctor of Philosophy

University of Wollongong

School of Electrical, Computer and Telecommunications Engineering

August 2022

To my family

Abstract

Programmable Logic Controllers (PLCs) play an important role in Industrial Control Systems (ICS), production lines, public infrastructure, and critical facilities. A compromised PLC would lead to devastating consequences that risk workplace safety, humans, environment, and associated systems. Because of their important role in ICS, more specifically PLC Based Systems (PLC-BS), PLCs have been targeted by various types of cyber-attacks. Many contributions have been dedicated to protecting ICS and exploring their vulnerabilities and threats, but little attention and progress have been made in enhancing the security of PLC code by utilizing internal PLC ladder logic code solutions. Mainly the contributions to protect and secure PLC-BS are related to external factors such as industrial networks, Supervisory Control And Data Acquisition Systems (SCADA), field devices, and servers. Focusing on those external factors would not be sufficient if adversaries gain access to a PLC since PLCs are insecure by design - do not have built-in self-defense features that could reduce or detect abnormalities or vulnerabilities within their running routines or codes. PLCs are defenseless against code exploitations and malicious code modifications. This research work focuses on exposing the vulnerabilities of PLC ladder logic code and provides countermeasure solutions to detect and prevent related code exploitation and vulnerabilities. Several test-bed experiments, using Rockwell PLCs, were conducted to deploy real-time attack models against PLC ladder logic code and provided countermeasure solutions to detect the associated threats and prevent them. The deployed attacks were successfully detected by the provided countermeasure solutions. These countermeasure techniques are novel, real-time PLC ladder logic code solutions that can be deployed to any PLC to enhance its code defense mechanism and enable it to detect and prevent code attacks and even bad code practices. The main novel contribution, among the provided countermeasure solutions, is the STC (Scan Time Code) technique. STC is a ladder logic code that was developed, deployed, and tested in several test-bed experiments to detect and prevent code abnormalities and threats. STC was able to detect and prevent a variety of real-time attack models against a PLC ladder logic code. STC was designed to capture and analyze the time a PLC spends in executing a specific routine or program per scan cycle to monitor any suspicious code modifications or behaviors. Any suspicious modifications or behaviors of PLC code within a particular routine would be detected by STC which in

return would stop and prevent further code execution and warn operators. In addition to detecting code modifications, the STC technique was used to detect any modification of the CPU time slice scheduling. Another countermeasure technique was PLC code that was used to detect and prevent the manipulation or deterioration of particular field devices. Moreover, several countermeasure PLC code techniques were proposed to expose the vulnerabilities of PLC alarms code where adversaries could find ways to launch cyber-attacks that could suppress (disable) or silence the alarms and critical faults of associated ICS devices monitored by PLCs. Suppressed alarms would not be reported to operators or promptly detected, resulting in devastating damage. All provided countermeasure solutions in this work were successfully tested and capable of detecting, preventing, or eliminating real-time attack scenarios. The results were analyzed and proved the validity of the provided countermeasure solutions. This research work, also, provides policies, recommendations, and general countermeasures to enhance the validity and security of PLC code. All the techniques provided in this work are applicable to be implemented and deployed to any PLC at no extra cost, additional resources, or complex integration. The techniques enhance the security of PLCs by building more defensive layers within their respective routines which in return would reduce financial losses, improve workplace safety, and protect human lives and the environment.

Acknowledgments

First and foremost, I would like to thank Professor Raad Raad for his continuous support and creative supervision. I would also like to thank Dr. Mohamad Raad for his contributions and constant support. A special thank you goes to my wife for being patient and supportive. A thank you to my parents who have been very supportive over the years. Lastly, I'd like to thank everyone I have had the pleasure to collaborate with in the course of this work.

Abraham Serhane

Certification

I, Abraham Serhane declare that this thesis submitted in fulfilment of the requirements for the conferral of the degree Doctor of Philosophy, in the School of Electrical, Computer and Telecommunications Engineering, from the University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

_____

Abraham Serhane

1st August 2022

List of Names or Abbreviations

AI: Artificial Intelligence.

ARP: Address Resolution Protocol.

CBA: Cost Benefit Analysis.

CPT: Compute instruction.

CPU: Central Processing Unit.

CT: Continuous Task.

CTU: Counter up instruction.

DCS: Distributed Control Systems.

DOS: Denial of Service.

DNET: DeviceNet.

DPI: Deep Packet Inspection.

GSV: Get System Value instruction.

GEQ: Greater than or Equal instruction.

GRT: Greater than instruction.

FBC: File Bit Comparison instruction.

FBD: Function Block Diagram.

FIFO: First in First Out instruction.

HMI: Human-Machine Interface.

IACS: Industrial Automation and Control Systems.

ICS: Industrial Control Systems.

IL: Instruction List.

IIoT: Industrial Internet of Things.

IL: Instruction List.

IoT: Internet of Things.

IT: Information Technology.

JMP: Jump instruction.

JSR: Jump to Subroutine instruction.

LD: Ladder logic Diagram.

LAN: Local Area Network.

LES: Less than instruction.

LEQ: Less than or equal instruction.

LBL: Label instruction.

MCR: Master Control Reset.

MITM: Man in The Middle.

MOV: Move instruction.

NOP: No Operation instruction.

OS: Operating System.

OT: Operational Technology.

OTE: Output Energized instruction.

OTL: Output Latch instruction.

OTU: Output Unlatch instruction.

PC: Personal Computers.

PLC-BS: PLC Based Systems.

PLC: Programmable Logic Controller.

RTO: Retentive Timer ON delay instruction.

RTOS: Real-time Operating System.

RTU: Remote Terminal Units.

SBR: Subroutine Instruction.

SCADA: Supervisory Control and Data Acquisition.

SFC: Sequential Function Chart.

SNMP: Simple Network Management Protocol.

SOTS: System Overhead Time Slice.

ST: Structured Text.

STC: Scan Time Code.

TIA: Totally Integrated Automation.

TOF: Timer OFF Delay instruction.

TON: Timer ON Delay instruction.

TS: Time Slice.

UCT: Unscheduled Communication Task.

WAN: Wide Area Network WAN.

XIC: Examine if Closed instruction.

XIO: Examine if Open instruction.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background

PLCs are widely used in ICS and automation including public facilities and infrastructure, such as oil refineries, power plants, nuclear plants, water treatment, and manufacturing factories. Because of their important role in nations' critical infrastructure, ICS, and manufacturing systems, PLCs have become a national security concern that need special attention and protection [1]. ICS are more prone than other systems to cyber-attacks due to their heterogenous nature, complex design, and the absence of overall common security framework. In 2011, as shown in Fig. 1.1, the number of SCADA attacks increased by 300%. The average number of ICS flaws increased by 5% every year after [2]. A study conducted by Kaspersky Lab, as shown in Fig. 1.2, shows that most of these PLC related attacks are either critical ones, 49%, or of medium risk, 42%. The report clearly indicates that only 85% of these known published vulnerabilities are fixed, but the rest are either partially fixed, can't be fixed, or not fixed at all [3]. According to Symantec, there were about 135 public vulnerabilities reported that are related to ICS/PLC-BS in 2015. While in 2014, only 35 ICS-related vulnerabilities were reported [4].

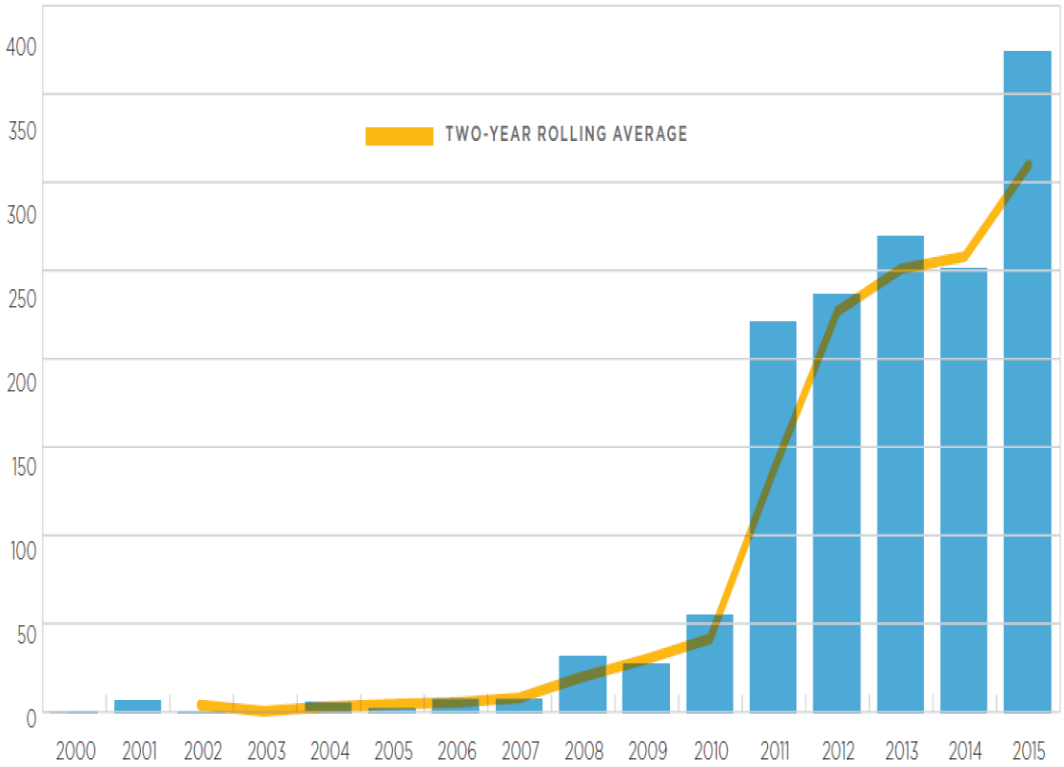


Fig. 1.1. SCADA Vulnerability Disclosures by Year [2].

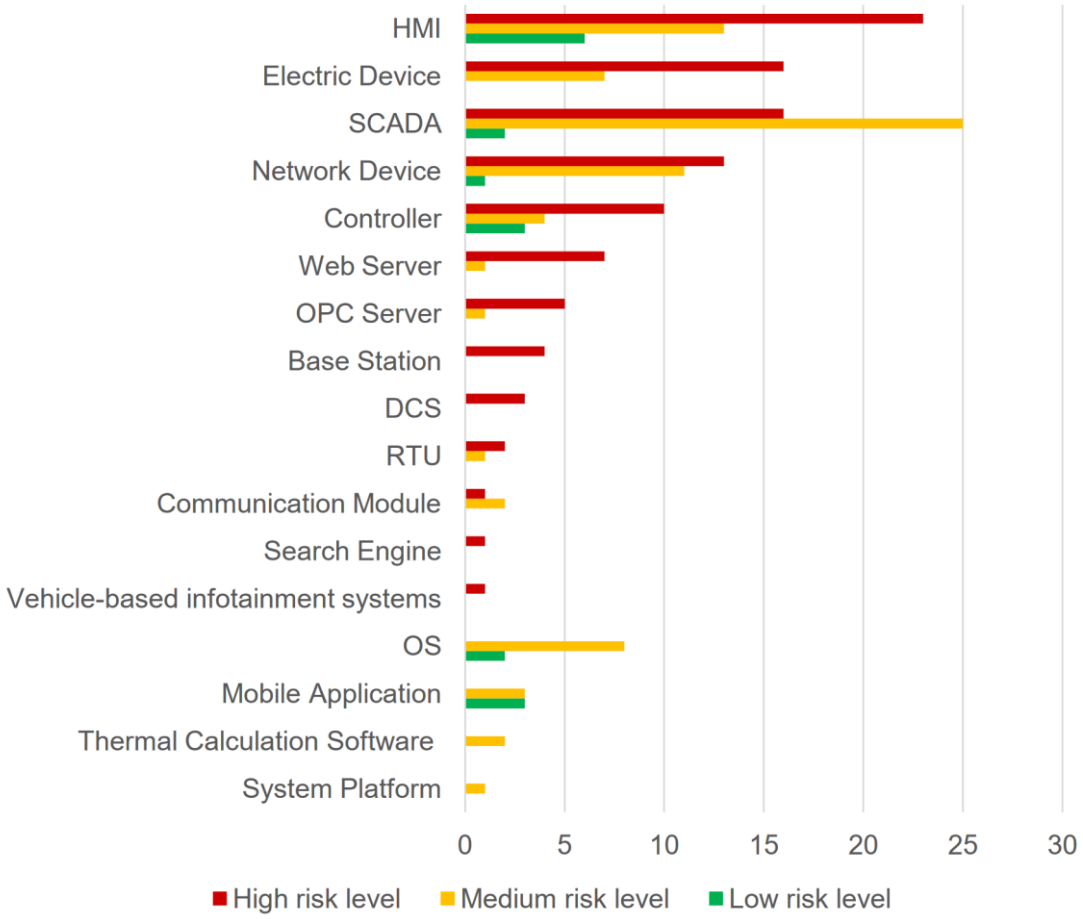


Fig. 1.2. SCADA's and other components vulnerabilities [3].

In 2017, Kaspersky Lab's Industrial Control Systems Cyber Emergency Response Team reported that the percentage of ICS computers and associated devices that were attacked by cyber-attacks increased from 36.6% in the first half of 2017 to 41.2% in the first half of 2018 [5]. A survey, conducted by the Kaspersky Lab team, shows that about 70% of the surveyed companies were worried about ICS cyber-threats, considering attacks against their ICS systems highly possible [6]. The survey shows that more than 52% felt the need to provide more resources for more enhanced secure ICS. With the advancement and development of Internet of Things (IoT) and Industry 4.0 approaching, where ICS are more interconnected and remotely accessible, cyber security of ICS's - including PLCs - has become a hot issue. In 2019, as shown in Fig. 1.3, a survey conducted by CS2AI showed that many organizations had numerous types of their control systems, including PLCs and HMIs, directly connected to the internet; were remotely accessible [7].

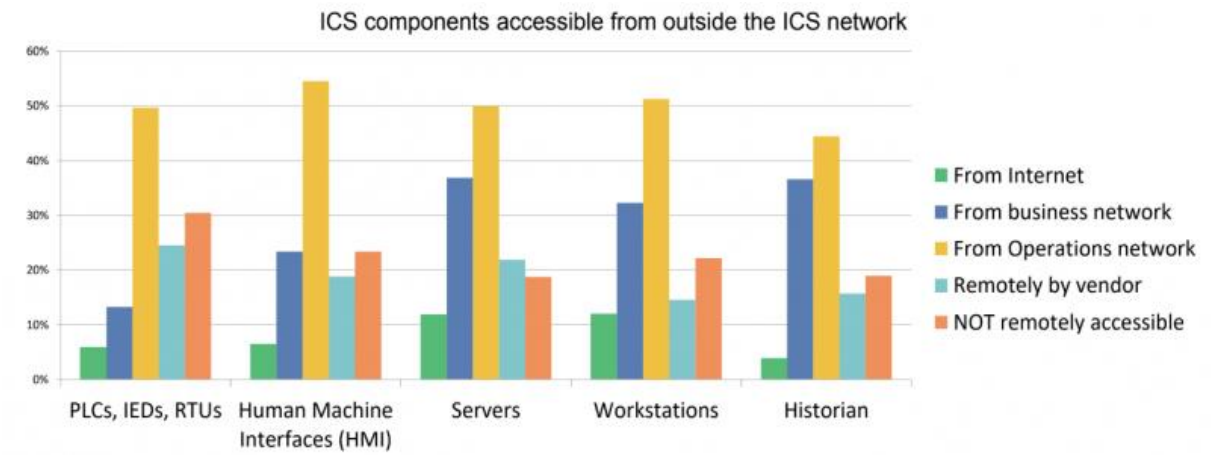


Fig. 1.3. PLCs HMIs, and other SCADA components connected directly to internet [7].

Many cyber security research studies on ICS have been published but they are relatively inadequate compared to those of IoT and Information Technology (IT). And they are even more insufficient when compared to PLCs in addressing PLC code.

PLCs by design have no defense mechanisms. PLCs do not have the built-in features to detect malicious code or suspicious code modifications. PLCs are defenseless blind real-time executors of any syntax or code elements as long as the code is legitimate and does not create any compiling errors. Enhancing the security and the validity of PLC code is crucial even though the overall PLC based systems might be secure and well protected. It is important to not just try to prevent adversaries from accessing PLC code, but also ensure that if they do, the damage to the code would be as minimal as possible, detected, and reported. Adding more techniques to secure and validate the PLC code would not just impede and slow down attackers but would also find inadvertent wrong or bad code practices.

Nevertheless, it has been proven that relying on old methods such as airgap techniques and securing the accessibility to a PLC is no longer sufficient and effective since adversaries, hackers or disgruntled workers, could find a backdoor to access the code and modify it, causing failures and critical consequences to PLC based systems. For instance, an inside attack by disgruntled worker targeted the alarms code of the Maroochy wastewater system in Australia polluting residential areas and water with about 260,000 gallons of raw sewage; causing serious environmental damage [8]. Another example that attacked an airgaped public facility was the Stuxnet worm [9] [10]. Stuxnet gained access to the nuclear power plant computers via USB [11]. Once Stuxnet gained access to the computers starting spreading out and infected PLCs, Huma Machine Interface (HMI) devices, and related devies; causing significant physical damage to centrigues [12].

1.2 Contributions

The main contribution of this research work is to enhance the security of the PLC code by equipping PLCs with ladder logic code techniques that could make the running code more secure, self-aware of code threats, and capable of detecting and preventing certain malicious attacks and code exploitations. One of the main and novel solutions proposed in this work was providing a unique ladder logic code technique, STC, that was able to expose certain vulnerabilities in the PLC code. STC was able to detect suspicious code modifications and behaviors and prevent further attacks.

Contrary to the claim in [13] that claims scan time of a certain PLC program not applicable in detecting code abnormalities, the STC technique, introduced in this work, was able to detect code behaviors based on the scan time of a designated PLC code or routine.

The provided techniques are very straightforward code techniques that could be easily implemented and deployed to any running code without overloading the system with external resources and complex solutions.

This research work exposed the vulnerabilities of the ladder logic programs that are executed within the PLC, which is responsible for controlling and monitoring SCADA or ICS devices and operations. It also provides countermeasure solutions to mitigate or eliminate certain vulnerabilities and prevents related attacks.

1.2.1 Attack Models

This work addressed several attack models with different scenarios. The attacks were deployed to a real-time PLC code testbed. The generated attack models were able to embed malicious code within the running PLC code, affecting its behavior, manipulating its intended operations, and altering critical values. A variety of attack scenarios were applied to exploit and compromise PLC code are listed as follows:

- PLC alarms code related.
- Suppressing alarms of ICS monitored and controlled by PLC code.
- Delaying alarms to evade alerting staff and operators and create more damage to associated devices.
- Faking alarms to evade or fool staff and operators into assuming all alarms are valid and real-time based.
- Inhibiting the status of certain alarms: always ON (active) or OFF (inactive).

- Tampering with PLC ladder logic code.
- Tampering with the behavior and diagnostics of physical devices of drives, robots, pins, clamps, etc. by suppressing or modifying real-time status or feedback.
- Tampering with critical preset values of instructions, such as timers and counters.
- Delaying and overloading the execution time of certain routines and overall program without being detected or noticed.
- Deleting potion PLC code (a couple of rungs or lines of codes) without interrupting the running program or triggering any faults.
- Deleting certain instructions within a rung without interrupting the running code or triggering and faults.
- Skipping several critical code lines (rungs) while the code was running code. The attack did not trigger any interruption or faults.
- Tampering with field device behavior and code setup. The attack was not detected or noticed by the PLC or operators.
- Modifying the execution time allowance scheduled by the CPU by changing the system overhead time slice of the PLC. That drastically affected the scan time of certain routines but was not detected by the PLC.

1.2.2 Countermeasure Solutions:

Several countermeasure solutions were developed and applied to PLC code to make it more self-aware of any code abnormalities and undesirable behavior. The proposed solutions enabled the PLC to detect the introduced attacks models and prevent their malicious consequences. Those countermeasures were successfully developed, deployed, and tested. They were able to prevent and detect all the attack models during real-time implementation and testing. The research provides an analysis of the countermeasures and the attacks using real-time trends that captured the behavior of the code during the designated duration.

The countermeasure solutions or techniques are summarized as follows:

- Scan Time Code (STC) technique: One of the major and novel techniques provided in this work was the utilization of scan time of a particular ladder logic code to analyze and monitor its behavior and suspicious modifications. The average scan time of a particular routine was studied and analyzed under normal and attack-free situations. That average was used as a reference base where it was compared to any ongoing future scanning. If the scan time of a particular routine changed, that would be an indicator of

a suspicious routine modification. So, whenever a scan time discrepancy between the average and the monitored one was found, the PLC stopped further scanning and warned operators. STC was used to detect and prevent the following attack based on the changes of the scan time of a routine:

- Detecting an increase in the scan time indicated that additional code elements were embedded to the routine.
- Detecting a significant increase in the scan time indicated that finite loops were embedded to overload the scan cycle.
- Detecting a drastic increase in the scan time indicated that the system overhead time slice was modified.
- Detecting a decrease in the scan time indicated that code elements were deleted.
- Detecting a significant decrease in scan time indicated that the routine is skipped using instruction such as jumpers, JMP.
- Countermeasures to detect any malicious attacks against PLC alarms code.
- Detected if alarms were faked.
- Detected if alarms were valid and not suppressed.
- Detected if alarms code was tampered with.
- Detected if alarms were inactive or their statuses were inhibited.
- Countermeasures to detect malicious attacks against field devices behaviors. The countermeasure validated the physical behavior of field devices with respect to designated actual running ladder logic code. For instance, a timer was used to verify that a pin is not taking extra time during extending or retracting its rod. The following is a list of vulnerabilities were detected and prevented code:
    - Swapping Statuses
    - Swapping Outputs
    - Embedding Always ON status
    - Embedding Always OFF status
    - Deleting Outputs
    - Delaying status
    - Delaying Outputs
    - General recommendations and guidelines to enhance the security of PLC code and reduce vulnerabilities.

1.2.3 Thesis Outline

Chapter 2 presents a background of PLCs and related technologies including ICS components such as HMIs, PLCs, Networks, and RTUs. It presents, also, an overview of PLC major components and functionalities.

Chapter 3 presents a literature review of research works of major ICS cyber incidents is presented, as well, in addition to recent statistical reports related to ICS vulnerabilities and attacks. A special section is dedicated to vulnerabilities and threats that are directly related to PLC, including the operating systems of PLCs, PLC code, and associated I/O.

Chapter 4 presents a contributed, published work that goes over ladder logic code vulnerabilities and bad code practice within PLCs.

Chapter 5 presents a contributed, published work that is an extension of the published research work presented in Chapter 3. The chapter provides literature of PLC-BS threats and vulnerabilities. It proposes practical guidelines for good code practicing and general security recommendations.

Chapter 6 presents a contributed, published work about a novel solution to detect and prevent PLC alarms suppression or manipulation. It demonstrates the vulnerabilities in PLC alarms code and the introduced real-time countermeasures that were introduced to detect and prevent associated code exploitations and attacks.

Chapter 7 presents a contributed, published work that proposes novel solutions with real-time techniques to detect and prevent code abnormalities and suspicious code modifications. The chapter explains a real-time conducted test bed that was developed based on PLC ladder logic code, where the test bed was used, also, to develop attack models against the running code. In this chapter, a detailed explanation about the attacks that were able to exploit code vulnerabilities without being detected and without stopping any running code, though the attacks caused major code modifications, is provided. The chapter provides details about the introduced novel real-time countermeasures against such attacks. It explains the role of developed STC code in detecting and preventing code attacks, code abnormalities, and physical abnormal behaviors.

Chapter 8 presents the conclusions extracted from this work and potential future work to extend what has been presented in this thesis.

1.2.4 Publications

A. Serhane, R. Raad, and W. Susilo, M. Raad, "PLC Code-Level Vulnerabilities," in 2018 International Conference on Computer and Applications (ICCA), 2018.

A. Serhane, R. Raad, and W. Susilo, M. Raad, "Programmable logic Controllers Based systems (PLC-BS): vulnerabilities and threats," Springer Nature Appl. Sci., vol. 1, 2019.

A. Serhane, R. Raad, and W. Susilo, M. Raad, "Applied methods to detect and prevent vulnerabilities within PLC alarms code," Springer Nature Appl. Sci. 4, 127,2022.

A. Serhane, R. Raad, and W. Susilo, M. Raad, “Equipping PLC code with Self-Detection Mechanisms to Detect and Prevent Vulnerabilities and Threats.” Submitted to International Journal of Information Security, Springer Nature.

Chapter 2 Overview of PLC-BS: Technologies

2.1 Introdution

PLC-BS are hardware components or machinery that are mainly monitored and controlled by PLCs. PLC-BS consists of PLCs and other associated industrial components such as: sensors, field devices, HMIs, I/O modules, industrial networks, etc.

The term “PLC-BS” is used in this work to ensure that other non-PLC driven systems are excluded from this research. It is often wrongly assumed that any SCADA or industrial system is driven by PLCs, but that is not always the case. Industrial automated systems might rely on PLCs or other components such as PCs or relays. Systems like ICS, SCADA, or OT may or may not include PLCs in certain areas or processes, as shown Fig. 2.1. PLC-BS could be integrated with SCADA systems to provide better visualization and analysis of the collected and logged data. PLC-BS do not depend on SCADA for their functionalities.

Another inaccurate naming convention is when ICS and SCADA terms are used interchangeably. ICS is a more accurate and general term to be used while referring to industrial automated systems. ICS is a general and collective term that is used to refer to varieties of control systems that include different types of PCs, field devices, I/O modules, networks, HMIs, supervisory devices, and controls - such as PLCs or DCS.

ICS provide means to monitor and control operations and processes through SCADA systems, DCS systems, or others. So, ICS is more general and “all-encompassing” term than SCADA in certain situations.

A PLC-BS are a major subset of ICS. PLC-BS could be standalone systems or could be integrated with other industrial systems, such as SCADA systems or Industrial Automation and Control Systems (IACS) as shown in Fig. 2.1.

All PLC based systems including their industrial computing components, devices, networks, and operations are categorized under Operational Technology (OT) framework. So, PLC-BS, whether integrated with SCADA or not, are a subset of ICS. And ICS are a subset of OT.

Overall, PLCs are a leading technology in controlling and monitoring automated processes or operations in industrial automation systems, manufacturing systems, production lines, infrastructure, and even in Distributed Control Systems (DCS).

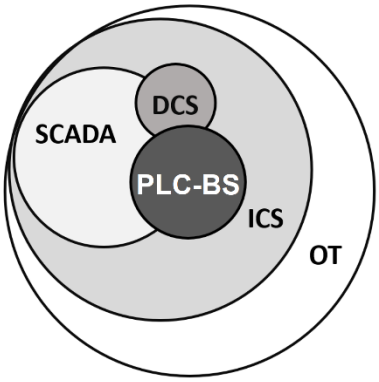


Fig. 2.1. PLC is a common major component among automated systems.

2.2 ICS Components

ICS and SCADA terms have been used interchangeably in research works, but ICS is a more accurate term to be used while referring to industrial automated systems. ICS or ICAs is a general and collective term that is used to refer to different types of automated industrial and manufacturing systems that include variety of field devices controls - such as PLCs or PCs, and machinery components. ICS monitor and control operations and processes through variety of means such as PLCs, PCs, DCS systems, etc. On the other hand, SCADA are integrated applications that are used as visual aid to collect, monitor, analyze data, alarms, tasks, and events of associated components. SCADA could be integrated to ICS architecture for better efficiency and monitoring.

2.2.1 DCS

DCS are process oriented and state driven control systems that might contain one or more computers. DCS are known as computer or PLC based systems that receive input signal to act upon and process tasks. Unlike SCADA, DCS systems are not built to gather information and log monitored data or events. DCs are not used in large geographical areas but rather at a single local site.

2.2.2 SCADA

SCADA systems, nowadays, are used to monitor, log, and control operations, processes, and associate devices of automated systems at a supervisory level. A SCADA system consist of several critical components as shown in Fig. 2.2. SCADAs are used to control, locally or remotely, large-scale processes and operations covering many devices several sites over a large distance.

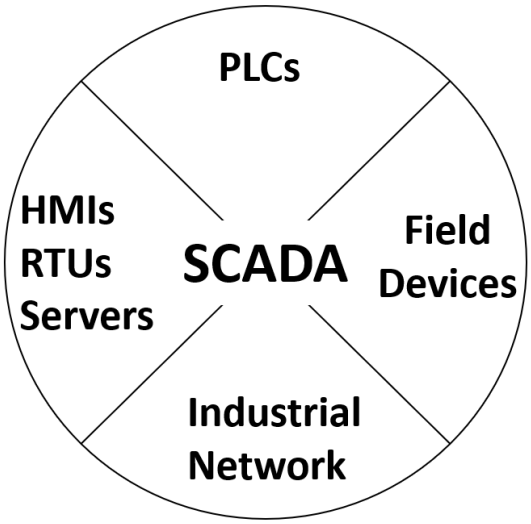


Fig. 2.2. PLC is a common major component in a typical SCADA system.

SCADA systems are used in a variety of automated systems and critical sectors such as:

- Smart buildings.
- Smart cities.
- Oil and Gas sectors.
- Energy sectors.
- Water and Wastewater treatment and distribution.
- Manufacturing and Industrial systems
- Petrochemical facilities
- Infrastructure facilities.

Generally, SCADA systems are composed of the following:

- **HMIs:** Are application-based devices that visually display data on customized visual panels or PC based Graphical User Interface (GUIs) to aid operators in monitoring and manually controlling ICS associated devices and operations. With the help of PLCs, HMIs are the main resource to check system and devices statuses, alarms and faults, manual and semi-manual tasks. HMIs could be used to log information about the controlled devices and offer statistical tools and charts. Such server-based HMIs could be used to utilize supervisory control software, PLC design software, and supervisory monitoring tools. HMIs can be used locally or remotely based on the types used.

- **RTUs:** Remote terminal units (RTUs) are less dominant than PLCs. They are microprocessor-controlled field devices that interface with SCADA or DCS systems and associated devices.
- **Networks**: Are used to carry signal among ICS devices. Networks could be EthernetI/P, Modbus, PROFINET, PROFIBUS, DeviceNet, ControlNet, EtherCAT, etc.
- **Field devices**: Are monitored and controlled by a PLC through designated network and I/O modules. Field devices could be solely inputs that report the status of a device to the PLC such as limit switches or solely outputs such as typical actuators. Other devices could be more complex and interact with a PLC in two ways such drives, encoders, etc.
- **PLCs**: are industrial grade computing controllers that are capable of being programmed to monitor and control automated processes and operations. More details are provided in Section 2.3.

2.3 PLC Fundamentals

As special industrial grade computers, PLCs are used to monitor and control functions, operations, processes, and other devices to automate systems. PLCs eliminated much of the hardwiring and relay control circuits; reducing costs and troubleshooting time. PLCs are rigid and reliable, real-time computers that continuously capture all incoming inputs (data from sensors), execute code, and update outputs (such as energizing actuators or coils).

2.3.1 Why PLCs?

- **Fast response:** since they are built to be fast and dedicated computers to do specific tasks. Being equipped with special industrial grade I/O modules and interfaces made monitoring and controlling much faster than regular PCs and microcontrollers.
- **Easier to program:** PLCs can be programmed using simple languages, like ladder logic. A PLC routine or program can be easily edited while the code is running; no need for recompiling the whole file and downloading it.
- **Reliable:** PLCs are proven to run for years and execute their programs without any issues once are properly compiled and downloaded.
- **Cost effective:** They were designed to replace relays and associated hardwiring. Nowadays, PLCs are being used to replace relays circuits including safety relays and control circuits.

- **Better communications:** PLCs use a variety of communication protocols and I/O interfaces. A PLC is designed to communicate to field devices associated with its controlled automated system: HMIs, sensors, limit switches, drives, actuators, other PLCs, etc. A PLC, for instance, that is Ethernet based can handle thousands of inputs and outputs.
- **Easier to troubleshoot:** By having all the inputs of field devices reported to a PLC and all controlling outputs transmitted from it, a PLC can track the status of every field device, control its associated outputs, and communicate that to HMIs and operators.
- **Rugged:** PLCs are built to tolerate tough environments and harsh industrial conditions.

2.3.2 Components and Parts associated with PLCs

The following is a list of major components of a PLC and associated components that can be also seen in Fig. 2.3:

- Power supply
- Memory unit: stores instructions, data, and the PLC code.
- Central Processing Unit (CPU).
- Operating System (OS): It is used to manage and control the hardware resources and functionalities within a PLC and other peripheral devices. PLCs have RTOS (Real Time OS) that supports real time multi-threaded applications with managed processes and deterministic behaviors within designated time constraints. A good example of PLC OS is VxWorks [14]. Unlike regular microcontrollers, PLCs consist of firmware (OS) which make them vulnerable to attacks and threats.
- Communication port: used for communication to HMIs, other PLCs, field devices, etc.
- I/O module: It is a module that provides input and output (Digital and/or analog). Input/Output Modules (I/O Modules) act as mediators between the processor and the input/output devices. The input modules receive signals from switches or sensors and send them to the processor. The output modules send the processor signals to the control devices like relays, drives, motors, etc. I/O modules could be built-in within the PLC or standalone ones. Modules can be added to the PLC Chassis or access remotely from other locations.

- Programming software: a software designer that is a PC based to design and write the code of the PLC. Once the code is developed, it should be downloaded to its designated PLC. A good example would be Logix Designer for Rockwell or Totally Integrated Automation (TIA) for Siemens.
- Communication Software: It is used to facilitate communication among PLCs, PCs, and related devices. A good example would be RSLinx sotware. RSlinx facilitates monitoring, editing, or downloading the designed code to the PLC.

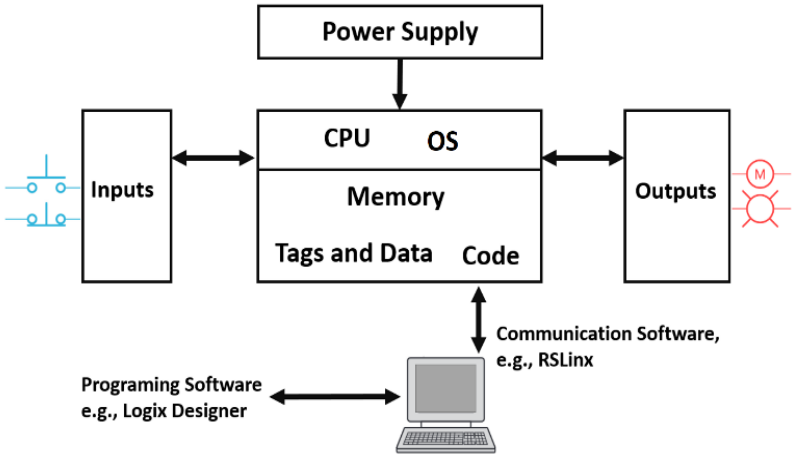


Fig. 2.3. PLC Components

2.3.3 PLC Code Design

To design, develop, or edit PLC code a PC based software design is required, such as TIA portal for Siemens or Studio 5000 (Logix Designer) for Rockwell [15]. In addition to code design software, some PLCs vendors require data communication software, such as RSLinx, to enable users downloading the code from a PC to a PLC where the running code within the PLC can be monitored or modified. PLC programing software, such as Studio 500, allows users to organize and structure the code, schedule events (interrupts), tasks, scan options, etc.
Typically, PLC code is a continuous real-time code that controls, processes, and monitors all the parameters, inputs, outputs, and other decisions needed to run any automated or manually associated devices or systems. Therefore, the code must be highly reliable with real-time data availability and high integrity to make prompt and precise logical calls and decisions. Unlike some other high-level languages, ladder logic code is accessible and editable at any time even when the PLC is running without stopping or restarting the whole PLC ladder logic program.

The PLC code can be designed in several IEC 61131-3 approved standard programming languages [16]:

- Ladder Logic Diagram (LD or LLD) is the most used language. It is a depiction of instructions, symbols, arranged in rungs mimicking hardwire schematics, see Fig. 2.4.
- Function Block Diagram (FBD): interconnected graphical blocks represent process flow and parameters, see Fig. 2.5.
- Sequential Function Chart (SFC): interconnected graphical blocks represent process flow and parameters, see Fig. 2.6.
- Structured Text (ST): a high-level language that resembles “C” or “Pascal”. It is a textual language rather than graphical as shown in Fig. 2.7.
- Instruction List (IL): a low-level language uses mnemonic instructions; it resembles assembly. Mostly used in old PLCs.

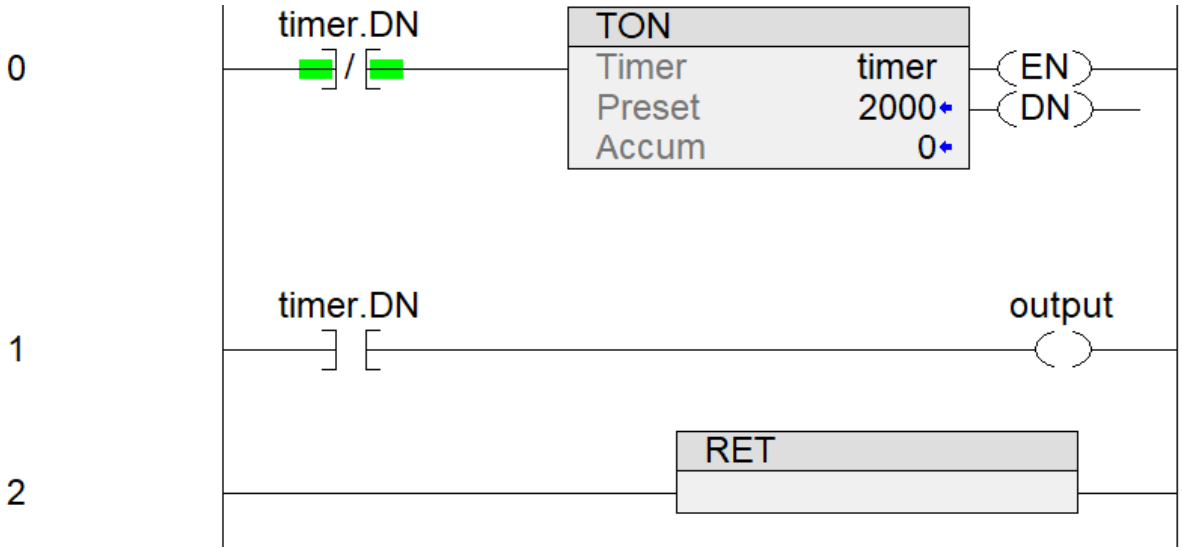


Fig. 2.4. Ladder Logic Diagram

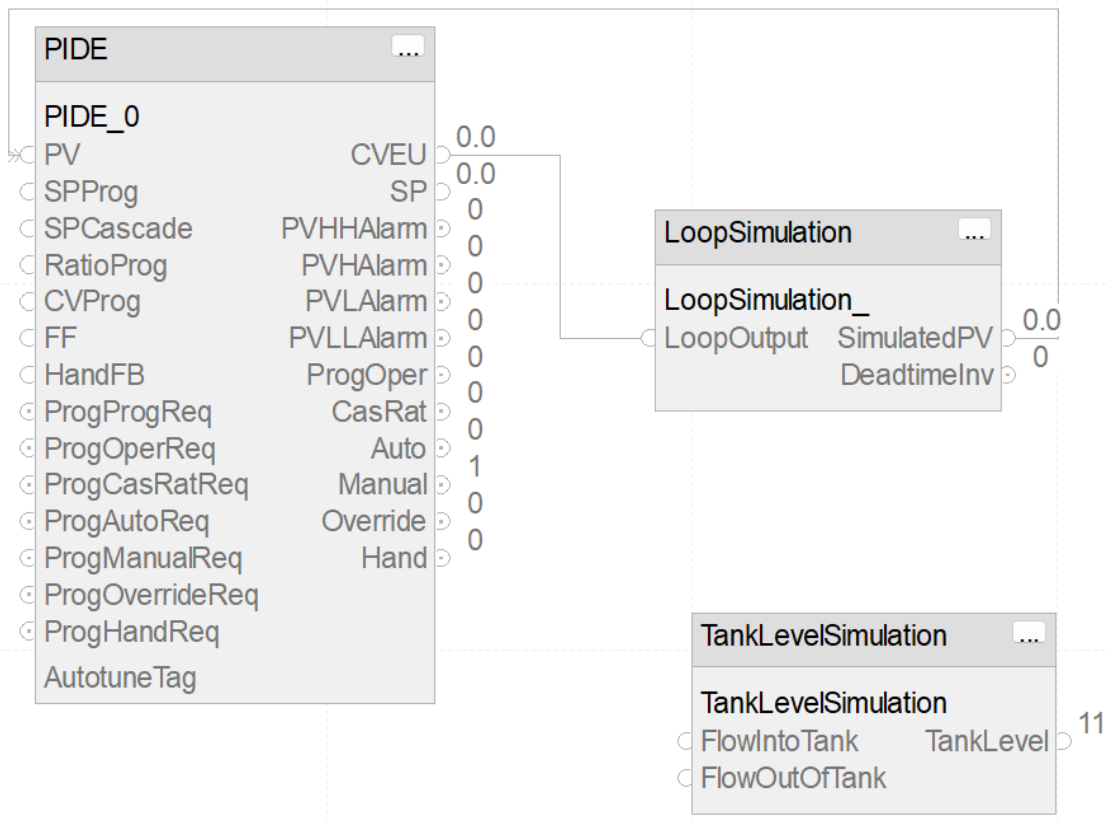


Fig. 2.5. Function Block Diagram.

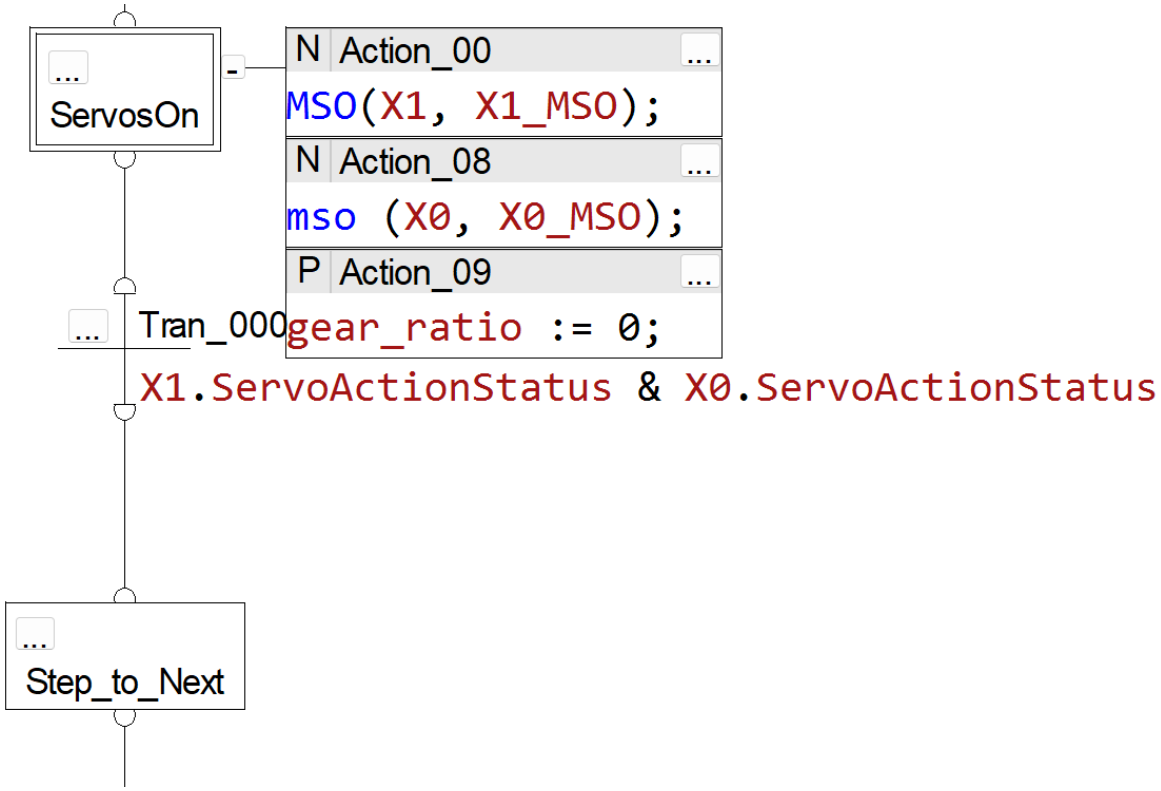


Fig. 2.6. SFC Diagram.

```
If timer.DN & State = 5 then (* Wait for timer to be done *)
                             (* then counter to 0 *)
    State := 0;
    counter := 0;
end_if;
```

Fig. 2.7. Structured text code.

2.3.4 Structures and Organization of PLC Code Software

To program or monitor a PLC ladder logic code, a particular code designer software, which is PC based, must be used. Code designer software (programming software) would enable users to create a PLC project and develop PLC programs to be deployed or downloaded to a PLC controller. Code designer software would enable developers to perform real-time monitoring and modification of any code running within a PLC. So, understanding the structure and the organization of a code designer software would help in understanding the PLC controller's scanning and execution of the code.

Though PLCs' software shares similar basic functionalities and navigations, this chapter focuses on Rockwell PLCs and Studio 5000 software [15], Logix Designer version 33, because it was used in the conducted experiments, and because Rockwell is one of the major dominant shares of the worldwide PLC market [15]. Logix Designer software allows only one PLC controller to be configured and handled per project. A Logix Designer project is a PC based program that stores all related controller's codes and information in one file, including any related code's instructions, tags, and I/O module configurations.

The main components of the project file, as shown in Fig. 2.8, are summarized as follows:

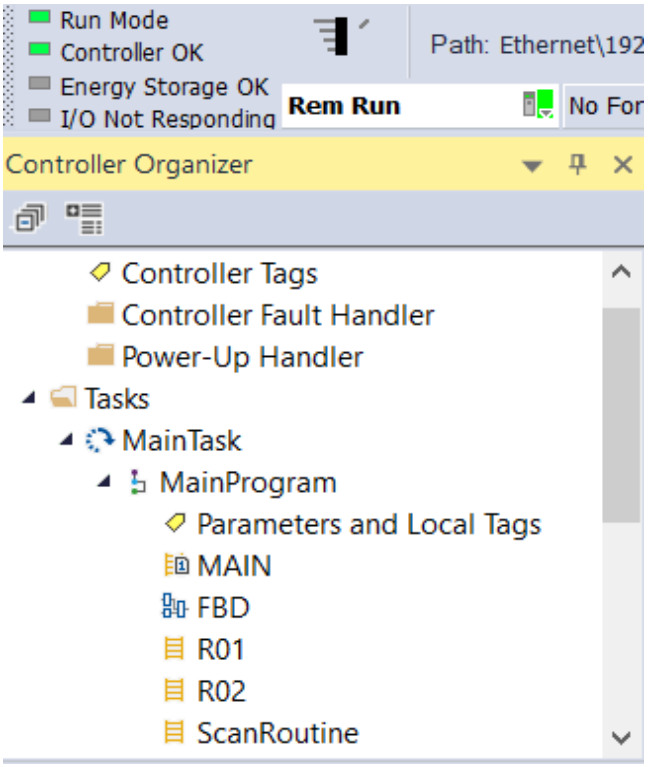


Fig. 2.8. PLC project organizer structure.

2.3.4.1 Tasks

A task is a collection of programs that are executed at any given or scheduled time. Whenever a task is executed, the related programs within the task are executed in the order listed or scheduled. Any tasks can be configured and set as follow:

- Event tasks: These function as interrupts that get triggered by a specific event that occurs. An event runs once and then returns control to where the prior task left off.
- Periodic tasks: get executed at specific, designated time intervals for a fixed duration of time. When triggered, a periodic task interrupts all other tasks with a lower priority until its operation is done and then allows the other tasks to resume where they left off. Periodic tasks are faster than the normal scan tasks.
- Continuous task: The continuous task is widely used in programming PLCs because it is the only place where routines or programs run continuously. Each time the listed routines finish a full scan, they restart running immediately, as shown in Fig. 2.9. In a project, only one continuous task is allowed. The continuous task has the lowest priority and runs in the background. Any non-allocated CPU time not used by other background tasks is used by the continuous task to run its listed programs.

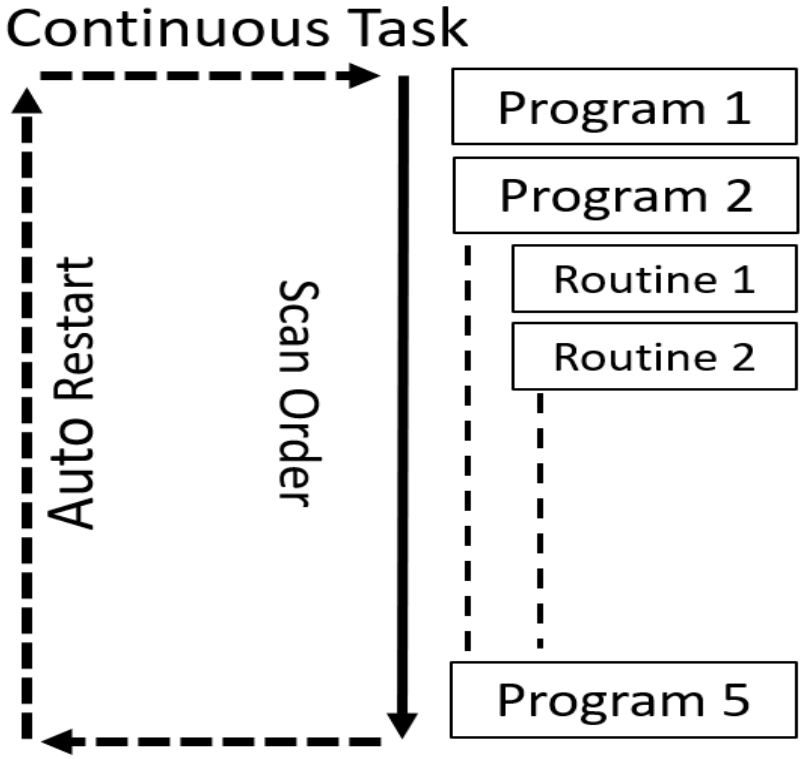


Fig. 2.9. A continuous Task behavior.

2.3.4.2 Programs

A project task handles several programs where each program can consist of several routines. In any created task, only one program runs at once. The way the routines are listed within the program specifies the order in which a program executes. For instance, the “Main Task”, as shown previously in Fig. 2.8, consists of several routines such as: Main, FBD, R01, R02, and ScanRoutine. In this example and according to the listed order, the Main Program is scheduled to execute first, R01 second, R02 third, and ScanRoutine last. Programs that are not assigned to a task are unscheduled.

2.3.4.3 Routines

Routines, within a program, are the code files that contain sequences of operations using logic elements and instructions. A routine can be created in any IEC 61131-3 standard programming languages [16]: LD, SFC, FBD, or ST. A project can handle several types of routines. A routine can be specified as one of the following types:

- A main routine: is the first routine to be executed within its specified program. In a main routine, other subroutines can be called to be scanned or executed.
- A subroutine: is one that is called by the main routine or another routine.
- Fault routine: is used in case a major fault occurs. Typically, when a PLC controller detects a major fault, it looks for a fault routine to execute; if the fault routine does not exist, the controller would be faulted and shuts down.

2.3.4.4 Rungs

In a ladder logic diagram code, each routine consists of a set of rungs or lines of code. Each rung can have several inputs, outputs, or other programming instructions.

2.3.4.5 Tags

Tags are text-based and meaningful names that are used to refer to the memory locations where data or values are stored. There are two scopes for tags: program (local) scope or controller scope. Program tags are defined and assigned locally within a program scope database which makes them only accessible by local routines that belong to the same program. For instance, “Parameters and Local Tags” shown in Fig. 2.10 are considered local tags that belong to the “MainProgram” scope. Any routines listed outside “MainProgram” cannot access them.
A controller might have several duplicated local tags that carry the same name if each belongs to a different program scope. On the other hand, controller tags are global tags that are accessible by any routines within a PLC controller. For example, “Controller Tags” shown in Fig. 2.10 are considered global ones which means they must be uniquely and globally declared and not duplicated.

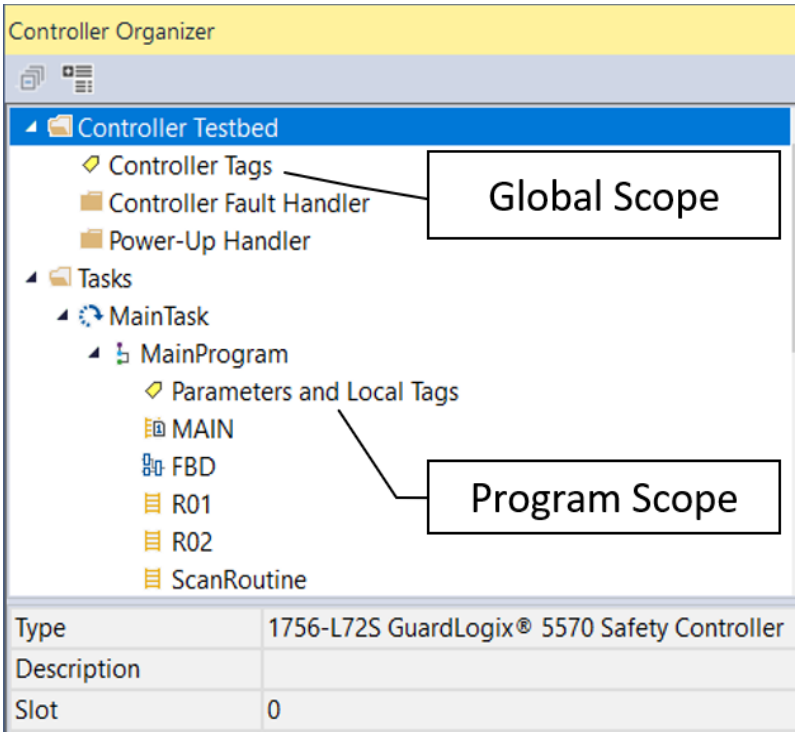


Fig. 2.10. Tags' scope.

Tag types can be based on physical addresses of I/O modules or can be memory-based defined ones with text-based description, see Fig. 2.11. Data types of tags can be configured as arrays, user-defined structures, and predefined structures (timers, controls, counters, etc.). Tags can be defined based on any of the following IEC 61131-3 atomic data types:

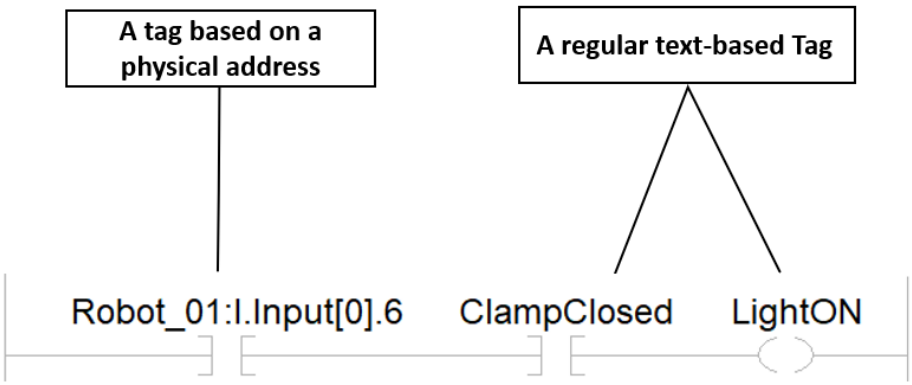


Fig. 2.11. Variety of defined tags within ladder logic code.

- **BOOL:** a Boolean tag is a one bit that holds exclusively one of two possible states or values: a “1” or a “0”. A Boolean tag can be defined as a stand-alone or can be an element of predefined structure such as a BOOL array that combines multiple bits.
- **SINT:** A tag Single Integer (SINT) of data type stores 8 bits of information.
- **INT:** A tag of Integer (INT) data type stores 16 bits of information.
- **DINT**: A tag of Double Integer (DINT) data type stores 32 bits (i.e., Word) of information.
- **LINT**: A tag of Long Integers (LINT) data type stores 64 bits of information; double words.
- **REAL**: A tag of “REAL” data type stores signed 32-bit floating-point decimal values.

For Logix 5000 controllers, DINT data types use less memory and execute faster than other data types [17]. Also, a tag with direct reference element within an array (such as Array[20]) uses less memory and executes faster than an indirect reference one (such as Array[Index_Array]).

2.3.5 PLC Code Scanning and Execution

Once the design of the code is done, it should be downloaded to the PLC. The PLC compiles the file and verifies it. If all are valid and the PLC is back to “RUN” mode, the controller starts scanning the code of all routines in a continuous matter, unless specified otherwise. a PLC continuously monitors, in real-time, all related physical I/Os, internal tag input, and code elements, and it updates outputs accordingly, which is called a PLC scan cycle.

Most of the controllers continuously scans each routine one at a time. It starts scanning or executing from the first top rung going from left to right all the way to the bottom of the routine. In order for a controller to execute and update any available output instructions or code elements (such as additions, counters, timers, etc.), all rung conditions, such inputs or comparative instructions, in the associate rung must be “TRUE”. If all rung conditions are not “TRUE”, then the controller stops scanning the rest of the rung, i.e., it doesn’t update any outputs, and moves on to scan the next available rung.

The time a PLC spends in executing all the code within a routine during a scan cycle is called scan time.

2.3.6 PLC Network Architecture

Data communication among PLCs and field devices can be established via industrial communication networks. Those networks have to be reliable, real-time, and accurate to handle data monitoring and data controllability among various devices. Originally industrial networks started as point-to-point communication link, limited and straight forward. A good example of that is a serial communications link. A transmitted signal (carried from the PLC to other devices or actuators i.e., an output) or received signal (sent to the PLC from a sensor i.e., an input) – is connected via point-to-point connection or terminal cards. Although it is limited, a point-to-point serial communications link is less vulnerable to security threats. However, with the advancement of technology and emerged needs, industrial networks evolved to complex levels such as Local Area Network (LAN) and Wide Area Network (WAN). Some industrial networks can handle diagnostics and power up interconnected related devices in addition to carrying signals from/to PLC via I/O modules or scanners like DeviceNet. Field devices, for instance, like I/O devices or modules have become more networked and software (firmware) based. I/O module devices are now connected, locally, or remotely, to industrial networks to get better modularity, reduced cost, less wiring – as shown in Fig. 2.12 and Fig. 2.13 - easy and quick installation or maintenance, and good diagnostics [18] [19].

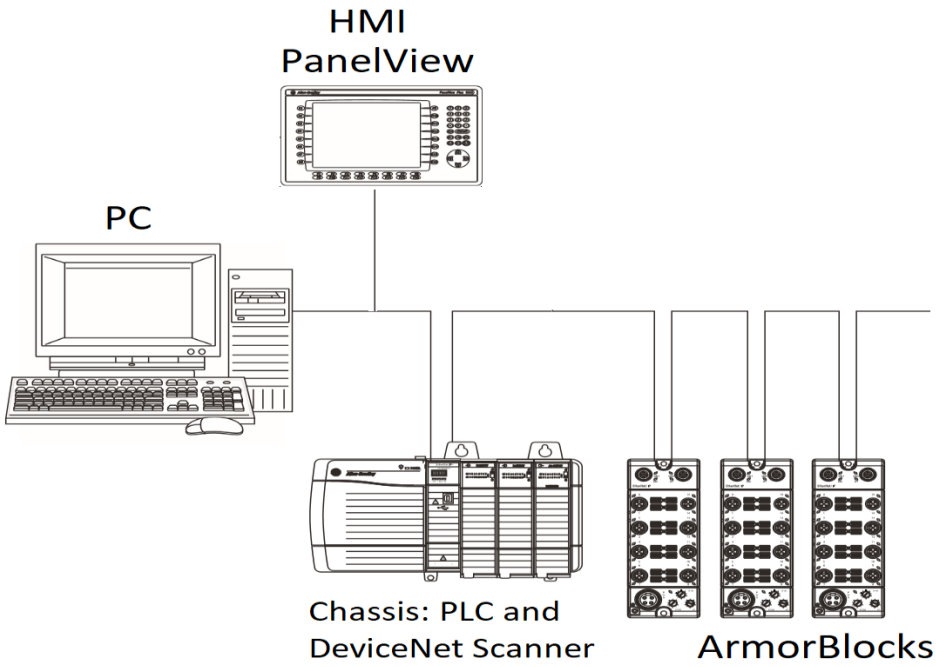


Fig. 2.12. DeviceNet Bus with Scanner and ArmorBlocks [19].

With that advancement of industrial communication networks, vulnerability to cyber attacks or threats increases. Generally, industrial networks consist of I/O devices, Scanners' modules, and network cables. A PLC communicates to other sensors, actuators, or devices via one of the above networks. DeviceNet, which is widely used would be a good example of such advancements. DeviceNet consists mainly of the following modules: DNET Scanner, ArmorBlock/ArmorPoint, and Flex I/O [19], as shown in Fig. 2.14. Each module has its own firmware that could be vulnerable to security threats.

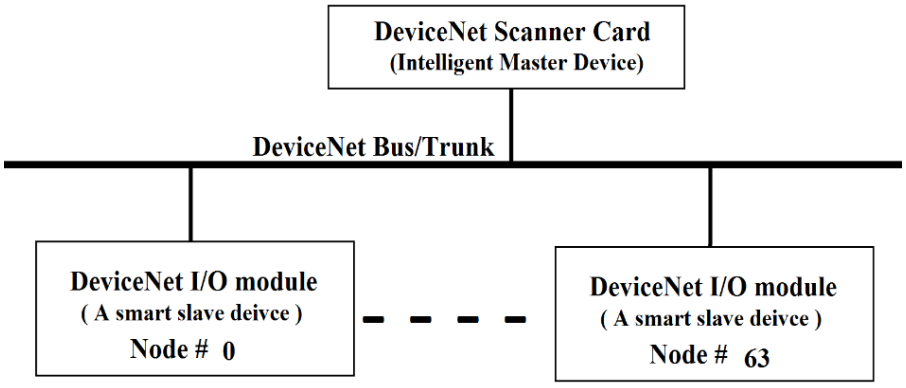


Fig. 2.13. DeviceNet Architecture [19].

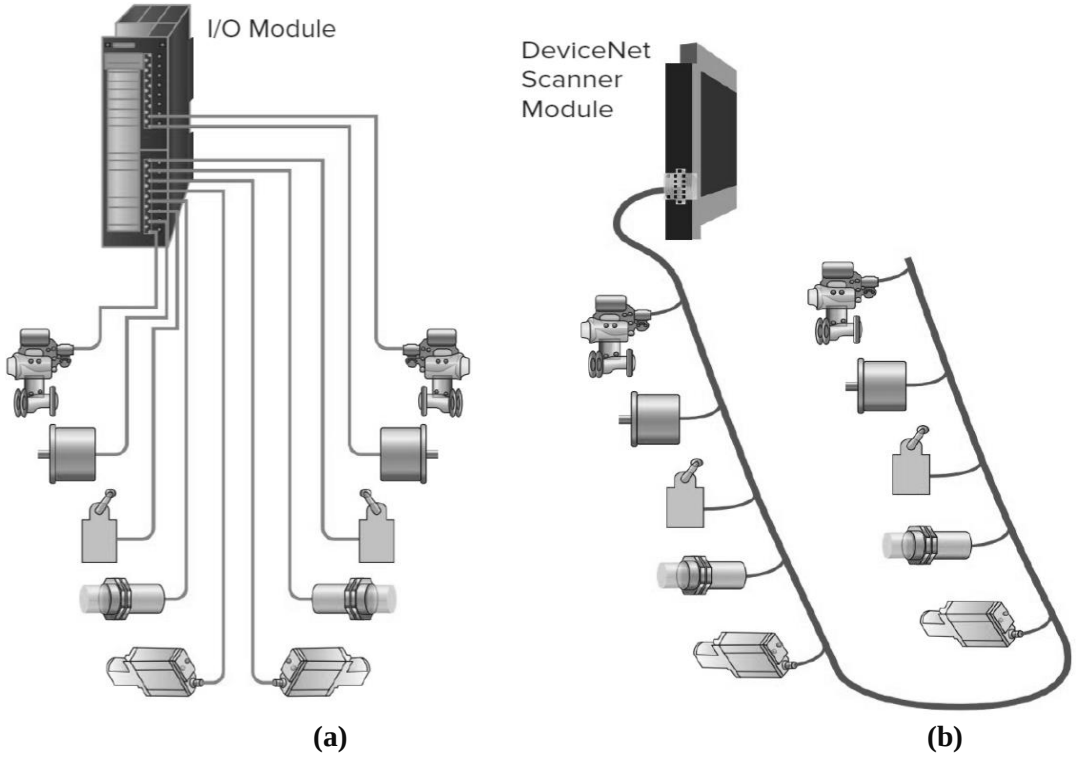


Fig. 2.14. (a) and (b) shows DeviceNet Bus that reduces wiring and costs [19].

Major advanced PLCs, presently, can talk to all systems and devices, including safety devices, via industrial networks - such as Ethernet/IP. GuardLogix5570 PLC, for instance can talk to all safety devise and regular field ones via Ethernet/IP, as shown in Fig. 2.15.

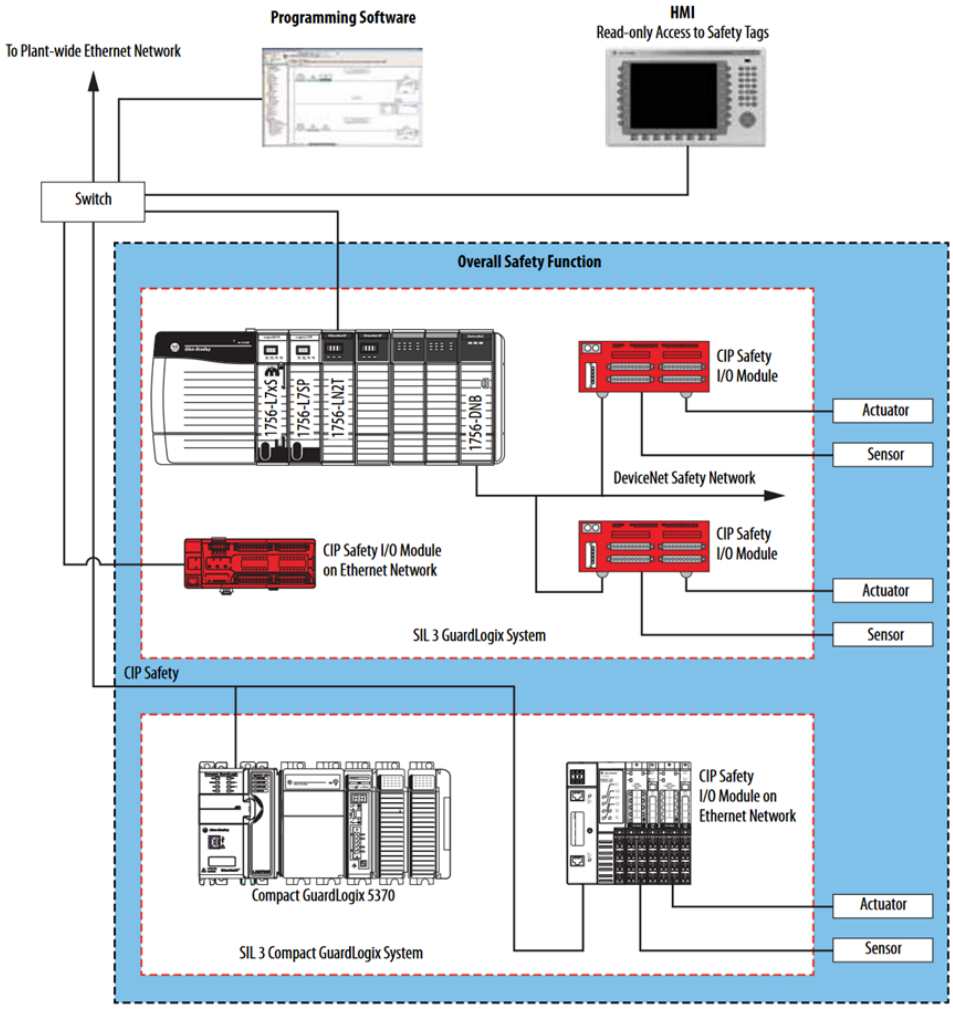


Fig. 2.15. PLC Architecture of Rockwell GuardLogix [17].

2.4 Conclusion

This chapter covered the background of PLCs and associated technology, including all major components. It presented a good overview of PLCs and associated components. A special section was introduced that discussed fundamentals of PLCs, their functionalities, and related hardware.

Chapter 3 Literature Review: ICS and PLC-BS Vulnerabilities and Threats

3.1 Introdution

Cyber-attacks and threats are not limited to PC anymore. Attacks could utilize vulnerabilities in any associated components of ICS including PLCs. The attacks could be directly designed to infect PLCs, or they could indirectly do serious damage to any PLC-BS components.

3.2 Overview: ICS and PLC-BS Threats

When attacks are launched against ICS systems, all associated components, including PLC-BS, must be of a concern. Attacks could utilize ICS vulnerabilities, to gain access to PLC-BS components such as industrial networks, field devices, SCADA servers, and HMIs. Attacks that infect Windows computers, for instance, would be a threat to PLCs as well since most of the advanced, modern, HMIs are PC or server based. PC based HMIs are frequently used to access, monitor, and edit PLC code through the installed PLC code design software, like Studio 5000 software. The "WannaCry" malware, that infected windows computers, was able to find its way to ICS sector [20]. "WannaCry" was able to use some computers as a bridge to access and compromise well protected industrial networks, as shown Fig. 3.1. Wannacry infected several industrial computers in different sectors, see Fig. 3.2.

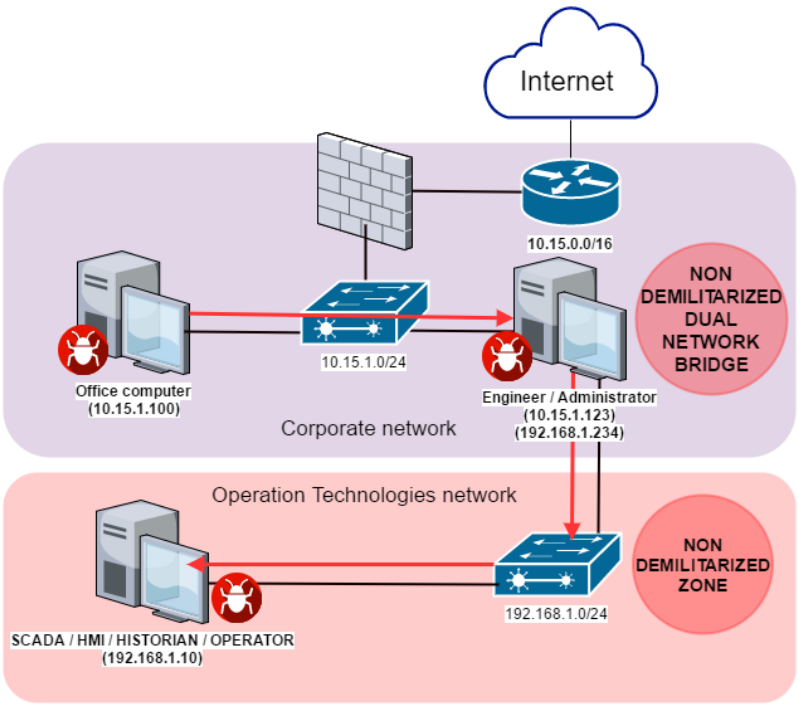


Fig. 3.1. WannaCry penetrating to ICS networks [20].

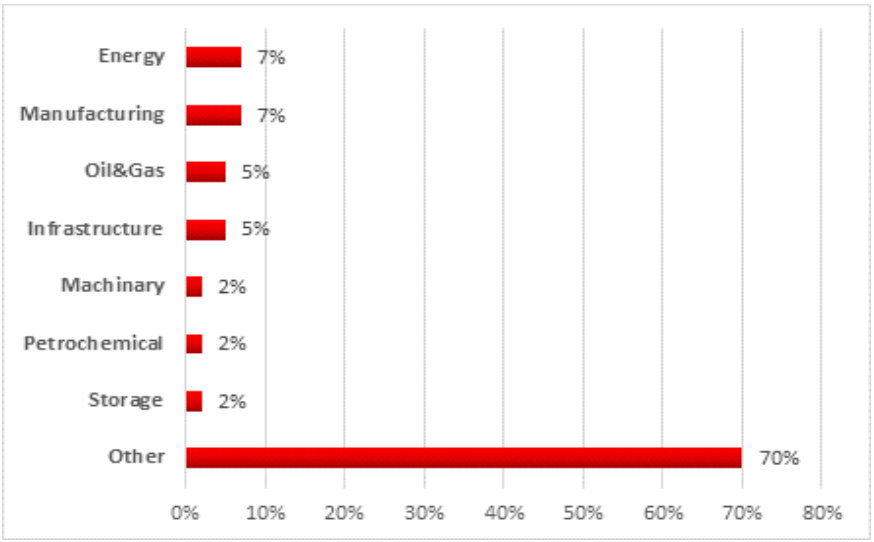


Fig. 3.2. Breakdown of ICS sectors infected with WannaCry [20].

According to Kaspersky ICS CERT the attacks against industrial infrastructure computers could affect SCADA servers, OPC, HMIs, networks, historians, and workstations [21].

ICS attacks have been globally targeted by adversaries and on the rise across several industrial sectors are on the rise [22], especially after being remotely accessible and connected to the Internet [5]. The concerns about cyber threats are increasing since they are not limited to manufacturing systems but also targeting more critical infrastructure sectors [23]. For instance, the US Department of Homeland Security responded to 25 water cyber-security incidents in 2015 [24]. According to Kaspersky ICS CERT, a surge in ransom attacks infected industrial companies as well as other organizations and some of the cases had serious consequences [25] [26]. The percentage of attacks against ICS increased by 0.4 p.p. more than that reported in 2020 [21]. A Kaspersky Lab study, published in 2019, shows the percentage of identified vulnerabilities of each PLC-BS component, as shown in Fig. 3.3.

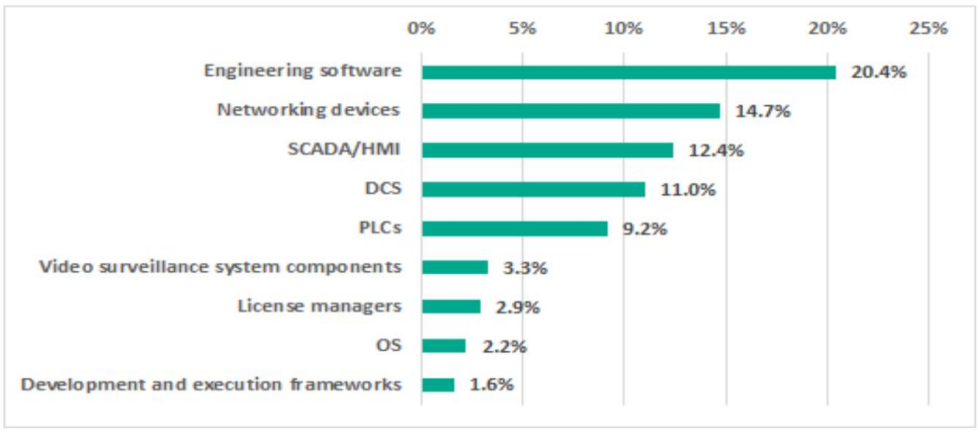


Fig. 3.3. Percentage of vulnerabilities identified among PLC-BS components [27].

The percentage of vulnerabilities of each component is identified as follows [27]:

- Associated Engineering software, mainly windows OS and third-party software, were exploited. Hackers utilize Windows based programs to initialize attacks on other ICS components.
- Networking was 15% because more Ethernet based systems are becoming widely used. That increases the vulnerabilities within communication, especially if it is remotely accessible.
- HMIs and DCS vulnerabilities are increased due to the increase of remote access and the use of Windows based platforms or architecture.
- PLCs' threats are increasing due to the increased need for remote access, more attractive to adversaries looking for creating damage, and affected by ICS malware.

The published report shows that the number of identified vulnerabilities among PLC-BS components was increased to 509 in 2019 compared to 415 in 2018 and 322 in 2017 [64]. More than half of those vulnerabilities were identified as high or critical risk level [64].

3.3 History of Major ICS Cyber Incidents

The following is a summary of major documented attacks against ICS since 2000:

Table 1: List of ICS Cyber Incidents

Year	Name	Details
2000	Maroochy	Adversary disabled alarms and made the pumps release more than 250,000 gallons of untreated sewage damaging wildlife and rivers [8] [28].
2008	Turkey cyber-intrusion	Turkey Pipeline explosion has been attributed to ICS failure or intrusion [29] [30] [31].
2010	Stuxnet	Infected PCs, networks, WinCC HMIs, Step 7 PLC, and created damages to nuclear centrifuges [32] [33] [34] [35].
2010	Night Dragon	Attacked Energy and petrochemical companies [36].
2011	Duqu	Data-stealing malware [37] [38].
2011	Flame	Data-stealing malware [38] [39].
2011	Gauss	Collected and stole data from compromised devices [40].

2011	Gas Pipeline Intrusion	Attacked 23 U.S. natural gas pipeline [41].
2012	W32.DistTrack “Shamoon”	Targeted Saudi Aramco and RasGas: it was information-stealing malware with destructive capabilities [42].
2013	New York Dam	Attempts to manipulate SCADA equipment [43].
2013	Havex (Oldrea)	Gathered server information of ICS and enumerated OPC tags [44].
2014	German Steel Mill	Caused infrastructural damage and multiple control system failures [45] [46].
2014	BlackEnergy	Compromised many ICSs and targeted HMIs [47] [48].
2014	Crouching Yeti Dragonfly Energetic Bear	Infected more than 2,800 victims including 101 organizations [49].
2015	Ukraine Power Grid	Cyber-attack on Ukrainian power grids and other institutions [50] [51].
2016	Kemuri	Manipulated The water utility’s SCADA platform including 100s of PLCs that control valves [52].
2016	Vairant of Shamoon	Affected Saudi civil aviation agency and other organizations [53].
2016	CrashOverride	Attacked Ukrainian power grids including manipulating SCADA systems. It was more severely than the attack occurred in 2015 [54] [15].
2017	NotPetya	One of the most destructive cyber-attacks against critical infrastructure that caused billions of dollars in damage across many countries [55] [56] [57].
2017	WannaCry	Attacked numerous ICS -various manufacturing companies, oil refineries and energy sectors - through enterprises’ local networks [20].
2017	Dragonfly 2	A destructive and intelligent information gathering tool that attacked energy sector and other organizations [58] [59].
2017	HATMAN TRITON TRISIS	Targeted ICS systems; specifically, Triconex safety PLCs by Schneider Electric’s [60] [61] [62].
2018	SPECTRE Meltdown	Stole information, credentials, and launched botnet. Industrial PCs, networks, SCADA servers, etc. Reported products: ABB, Cisco, Yokogawa, Siemens, and Schneider Electric [63] [64].
2018	SamSam MSIL/Samas	A Ransome targeted critical infrastructure and industries [65].
2019	Attacks by Xenotime	Targeted electric utilities and energy sectors in the United States and the Asia-Pacific region [66] [67].
2020	PoetRAT	Targeted SCADAs of Azerbaijan energy sectors [68]

2020	RagnarLocker	A ransomware that targeted critical infrastructure sectors and organizations [69] [70].
2021	Colonial Pipeline Cyberattack	A ransomware that stole data and disrupted pipeline operations including HMIs [71] [72].
2021	Ghost ZuCaNo	Attacked SCADA systems of several water sectors in USA [23].
2022	Log4j Exploitation	Exploited vulnerabilities in the Apache Log4j library including associated SCADA devices [73] [74].

3.4 Recent Statistics on ICS Threats

According to Kaspersky Lab [75], the percentage of blocked malicious attacks tried to attack ICS computers in 2021 increased from 38.6% to 39.6%, i.e., increased by 1 p.p. – see Fig. 3.4. More data of the blocked threats are presented as shown in Fig. 3.5. The percentage of the blocked malicious attacks targeted oil and gas sectors rose by +3.5 p.p. during the second half of 2021, as shown in Fig. 3.6. The major threat resources of attacks were internet based, removable devices, and emails, as shown in Fig. 3.7. The sources of attacks vary from one country to another. For instance, in Australia the percentage of attacks coming from email clients were higher than that of removable devices, see Fig. 3.8. The types of the blocked threats of ICS computers are shown in Fig. 3.9 [75]. And the types of OT industries that were targeted by attacks, in 2021, are shown in Fig. 3.10 [76].

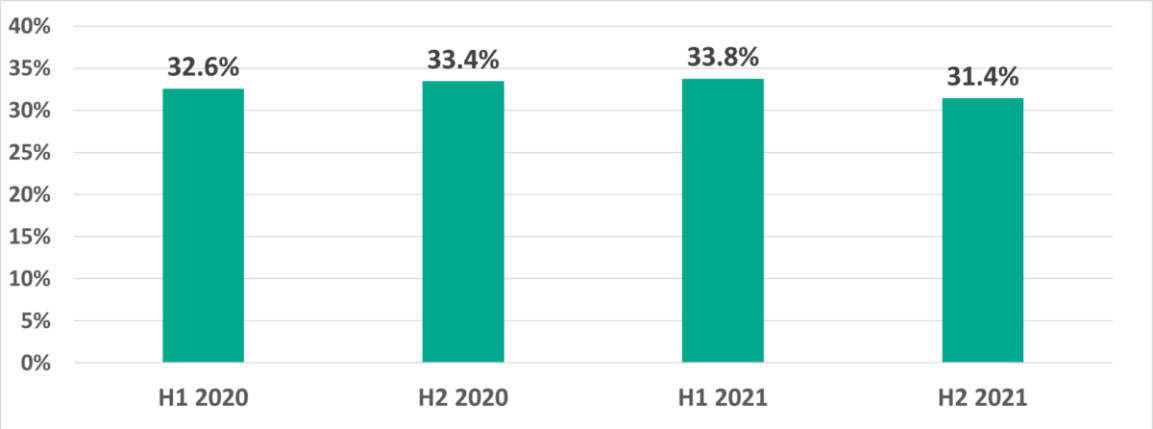


Fig. 3.4. Percentage of blocked malicious attacks against ICS [75].

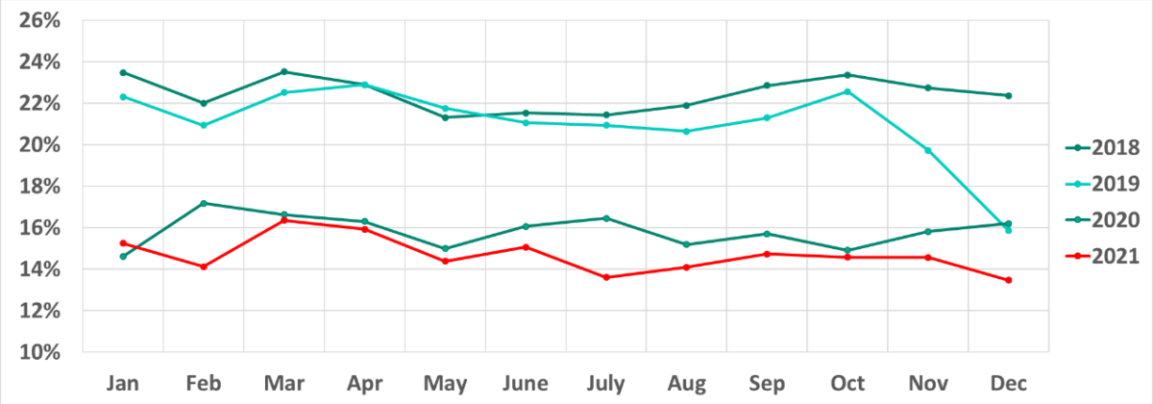


Fig. 3.5. Percentage of blocked malicious attacks against ICS through 2018-2021 [75].

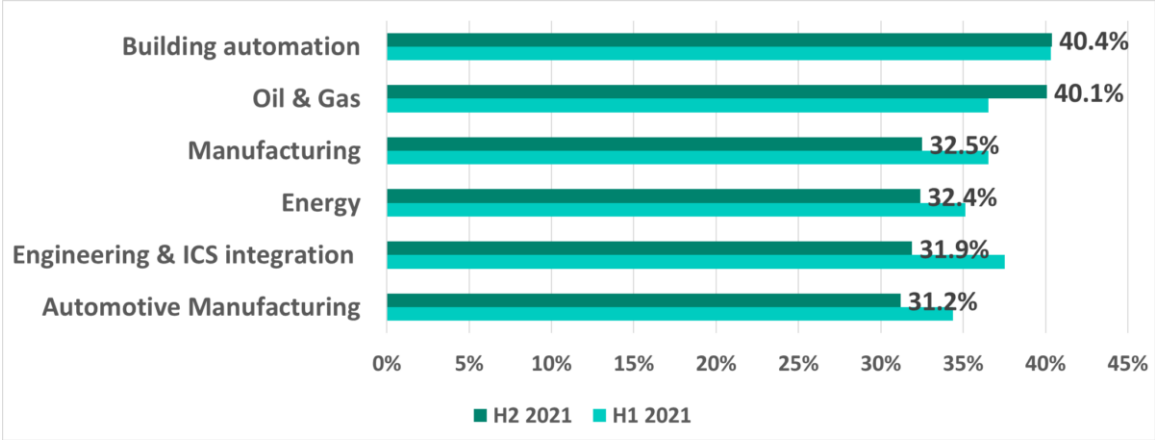


Fig. 3.6. Percentage of blocked malicious attacks against different PLC-BS sectors [75].

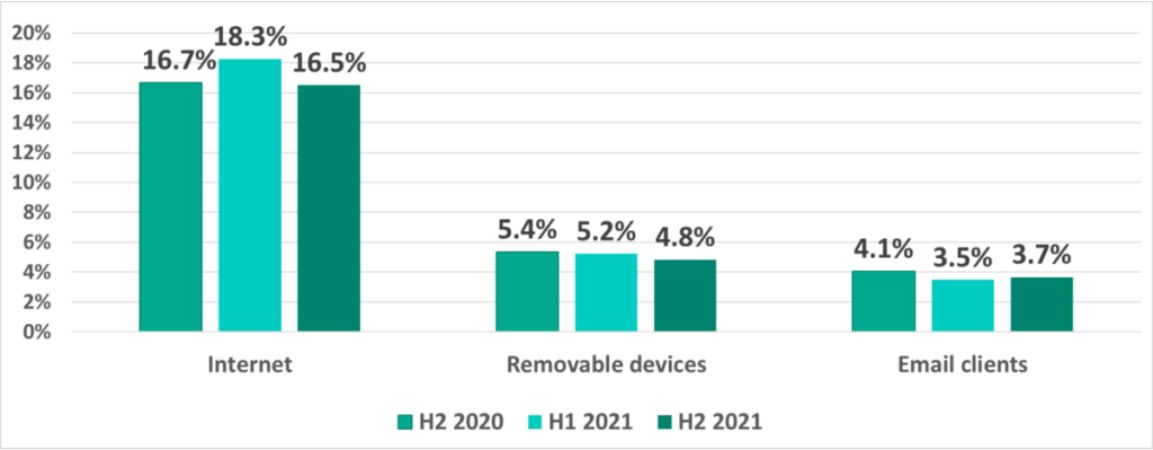


Fig. 3.7. Percentage of major threat sources [75].

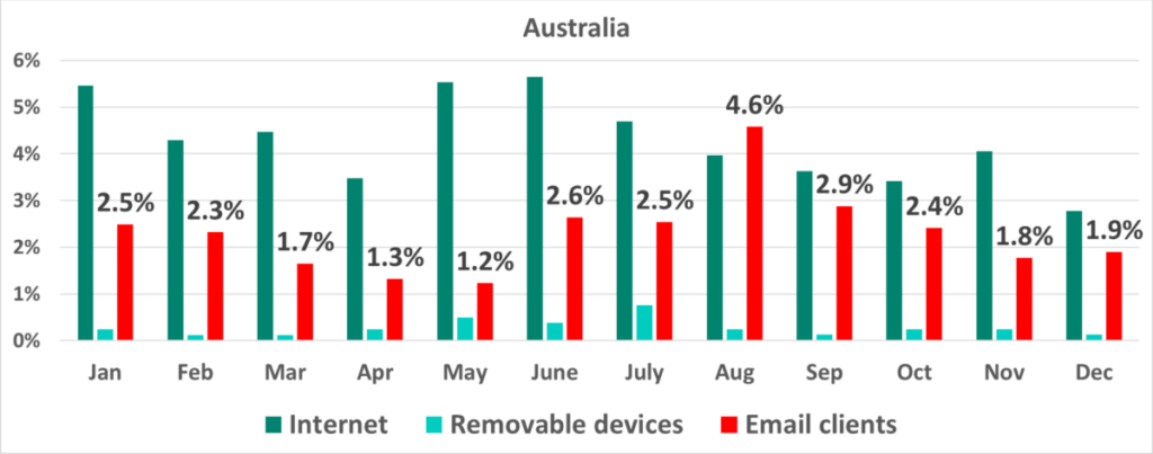


Fig. 3.8. Percentage of sources of threats [75].

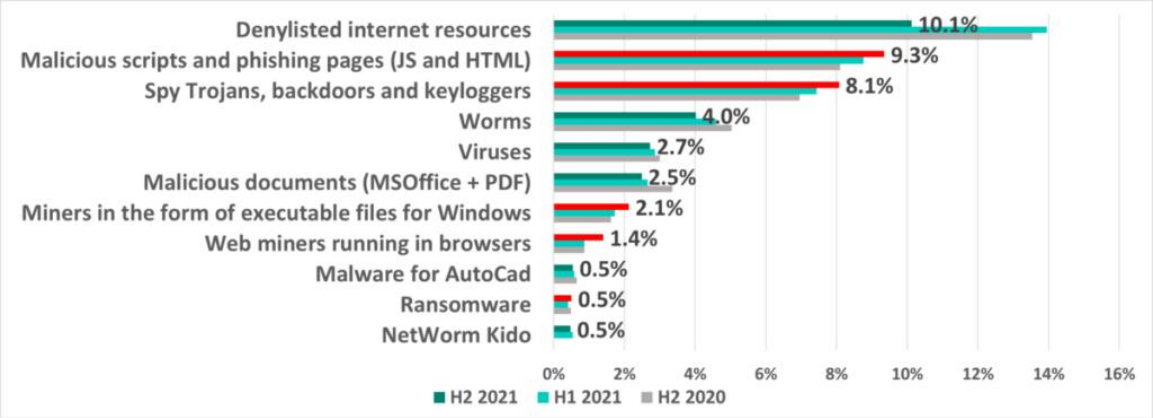


Fig. 3.9. Percentage of threat types [75].

Fig. 3.10. Breakdown of attacks against ICS industries [76].

According to the Trend Micro Zero Day Initiative report published in 2019 [77], vulnerabilities of SCADA systems were high during 2015-2019, as shown in Fig. 3.11. The vulnerabilities were found in several industrial devices, software, and types across different vendors. For instance, In 2015, vulnerabilities were found in Schneider Electric's ProClima software and in 2018 more vulnerabilities were found in Omron HMI software packages CX-Supervisor, see Fig. 3.12.

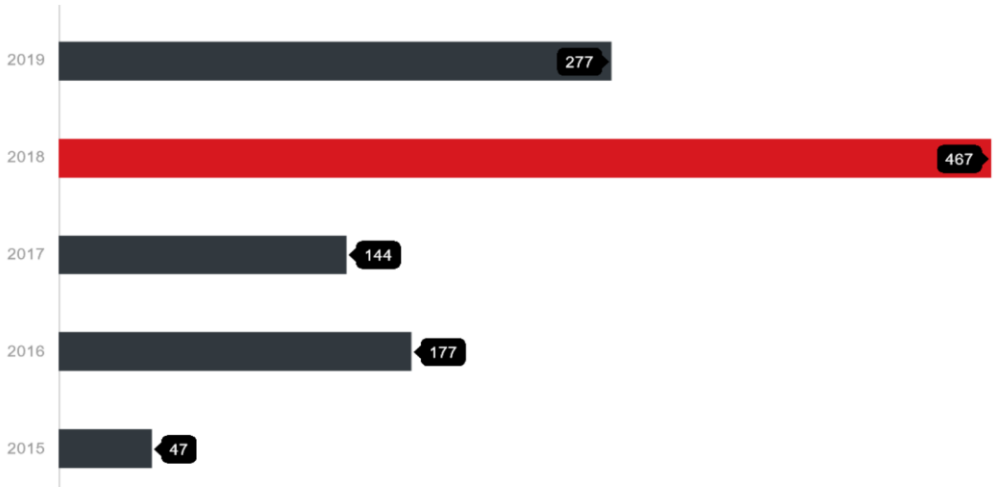


Fig. 3.11. Numbers of vulnerabilities found in SCADA systems [77].

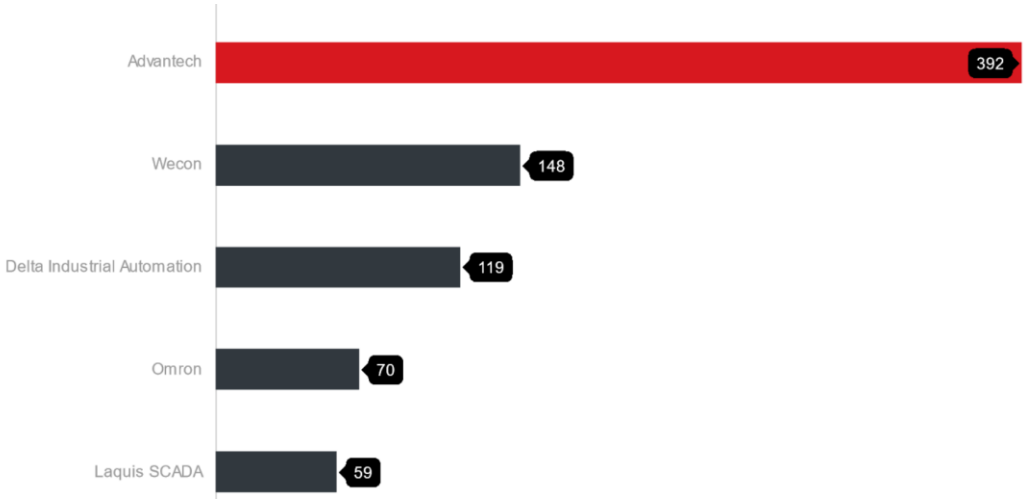


Fig. 3.12. Vulnerabilities found in PLCs and HMIs software packages [77].

3.5 A Review of PLC-BS Vulnerabilites and Threats

Based on the previous statistics, vulnerabilities of PLC-BS are the main reason of successful attacks. PLCs are not capable of handling extensive computation required to detect malware or malicious attacks. By gaining access to PLCs, adversaries can create serious damage to PLC-BS. Limited efforts have been made to detect threats within PLCs, such as PLCs' OS and PLCs code.

[78] [79] show that bad code practice and improper or poor ladder logic design would increase the risk of vulnerabilities within PLC code. Inexperienced programmers are more concerned about the functionality of automated devices than vulnerabilities or poor code practices.

[79] shows several bad code practices that could be exploited and risk the safety of PLC-BS. [81] provided ladder logic bombs (LLBs) where a ladder logic code was injected to execute malicious PLC code that manipulated receive status of field devices. The code could also be utilized to initialize DoS attacks. [80] presented a PLC-Blaster worm that targeted Siemens PLCs, SIMATIC S7-1200. The worm was hosted on Siemens PLCs, modified PLC code, manipulated outputs, and scanned associated networks to target other PLCs. [81] presented attack model that was able to access a Siemens PLC by using a SNMP Scanner and a SOCKS Proxy. The attack was able to modify the associated PLC program. [82] presented an external non-PLC program to verify PLC code integrity before loading it to the associated PLC. The research work introduced examples of bad code practice that could be exploited by adversaries. Racing conditions, duplicate or missing outputs, unused tags were provided as examples of bad code practice. [83] presented CLIK, a PLC logic attack. The attack consisted of bypassing security measures of a PLC, accessing and getting a binary copy of its code, decompiling the stolen binary code, and injecting a malicious code into the stolen one. Once the malicious code was injected, it was transferred and downloaded, as binary code, to the exploited PLC. [84] presented network attacks that manipulated memory values of PLC's inputs. [85] presented malicious attacks on PLCs, "Denial of Engineering Operations". The attacks were developed to access PLC code, retrieve a copy of it, modify it, recompile it, and downloaded back to the designated PLC. The process was accomplished without being detected by associated code designer software. As a countermeasure solution to detect malicious manipulations, a ladder logic decompiler termed 'Laddis' was introduced. Attacks in [86] were able to manipulate inputs and outputs of Siemen S7 PLC and remotely stopping it. [87] [88] introduced attacks that targeted Siemens S7 PLC code by retrieving the running ladder log code, decompiling it, manipulating it and transferring it back to the PLC in bytecode format. [89] presented attacks

that targeted Siemens S7-300 PLCs and associated TIA data. The authors injected interrupt code to attack the designated PLC Organization Block. [90] presented 'Shade', a deep packet inspection (DPI) technique. Shade employs certain algorithm techniques to monitor PLC code and detect any evasion injected into designated ICS networks. [91] presented a 'SABOT' tool that instantiated malicious payloads on Siemens PLCs. The payloads could be arbitrary manipulations of the targeted PLC code. [92] presented "Harvey" rootkit that exploits PLCs' firmware vulnerabilities to modify associated logic instructions. The malware was capable of monitoring and modifying PLC inputs received from field devices. Another rootkit to manipulate PLC's I/O values was introduced in [93]. In [94] a NuSMV model to verify the integrity of PLCs' function blocks, FB, code was introduced. [95] presented systematized knowledge of PLCs' threats, vulnerabilities, and recommended countermeasures. PLCspecif was introduced in [96] to validate and improve PLC programs. [97] demonstrated a methodology based on 'Cone of Influence' and NuSMV models to verify PLC programs. In [98] solutions were provided to verify safety program. [99] addressed a method to verify the integrity of PLCs' structured language programs. [100] [101] [102] introduced methods and techniques to verify and validate running PLC programs. [103] introduced 'Arcade', a statistical tool, to evaluate different PLC programs. [104] introduced HyPLC, a compilation and verification tool to provide guarantee proper correctness of the PLC programs.

In addition to code vulnerabilities, PLCs are vulnerable to OS attacks. Like any other OS, PLCs' OS or RTOS (real-time operating system) have their own vulnerability that could be exploited by adversaries especially if they are connected to outside networks or integrated with IoT network. The following are a couple of examples of vulnerabilities found in two major RTOS in the PLC domain. VxWorks, a RTOS for so many PLCs – such as Rockwell PLCs- and continuously functioning devices, was found vulnerable according to Armis researchers [105]. More than 2 billion devices run on VxWorks including aerospace, robots, industrial devices, etc., and roughly about 200 million devices among them appear to be vulnerable according to Armis researchers [106]. Exploitation of the exposed vulnerabilities would cause sever malfunctioning. Though Wind River, the developer of VxWorks, created a patch, it would not be easy to be deployed by companies. Nucleus RTOS is a real time operating system developed by Mentor Graphics, acquired by Siemens in 2021. In 2021 several vulnerabilities were found in Nucleus RTOS that were reported by Forescout and Medigate Labs [107]. The vulnerabilities can be exploited to take over a PLC or crash it.

Unlocked or unprotected PLCs create another vulnerability concern. Having an access point to a PLC allows user to upload any malicious code to the PLC, manipulate the current running one, or even upload new firmware. PLCs typically do not check whether the uploaded code is from a verified trusted source or not. Even with password protection, PLCs do not have enough defense mechanisms to stop attackers from retrieving or stop passing passwords [85] [91] [92]. [9] [11] [108] presented Stuxnet malware that severely attacked PLC-BS. Stuxnet was able to maliciously spy, attack, compromise, and even exploit other field devices to initialize attacks on other systems. By exploiting Siemens software and associated programs (HMIs - WinCC, PLC codes -Siemens Simatic Step7, PCs - Windows, etc.) and faking values, Stuxnet severely affected crucial field devices - taking advantage of multiple Windows zero days vulnerabilities [108]. Even though it was a malware that was specifically customized to target certain Siemens SCADA systems and programs, Stuxnet is just a typical threat that PLC-BS might face. In addition to Stuxnet attac, so many other attacks were carried out by other threats such as Flame, Guass, Duqu, Wiper, and BlackEnergy malware [39], [1]. There are more than 50 new Stuxnet-like attacks on SCADA threats discovered [109] [9] [10] [110]. Since Stuxnet's appearance, PLC-BS have attracted the attention of the hacker crowd.

Other vulnerabilities are related to PLC-BS networks. With the advancement of technology and the increase demand for remote access, PLC-BS networks become less obscure to hackers [111]. A Modbus stager tool was used to deploy a payload program through Modbus vulnerability [112]. In [80] the developed worm can scan PLC networks and target other S7 PLCs. In [113] several attacks on industrial networks were introduced including Denial of Service (DoS) and command injection. Exploiting vulnerability in Siemens S7 PLCs was addressed in [114]. [30] presented Packet manipulation such as latency, spoofing, eavesdropping, and deletion. [115] presented attacks on the communication stack: network layer, transport layer, application layer and the implementation of protocols such as TCP/IP, OPC, and ICCP. [116] presented attacks on industrial networks and methods to detect associated networks intrusion.

[117] [118] presented ARP Spoofing, password attack, and DoS. [11] presented Backdoors and Holes in Network Perimeters. [119] presented several industrial network intrusions and provided a detection solution using a DPI method. In [120] two attacks were introduced against SCADA: reply packet attack and Man in The Middle (MITM) attack. [121] Introduced several attacks against Siemens PLC and PROFINET test beds such as MITM cyber-attacks. Packet manipulation of logic was addressed in [90]. Detecting intrusion of SCADA networks was addressed in [122] [123] [124]. Recommendations and guidelines were provided in [125].

[126] presented Communications jamming, blocking, or hijacking; MITM. [127] presented a Controller-Aware False Data Injection (CaFDI) attack against ICS networks. CaFDI targeted associated PLC's inputs to generate malicious attacks.

3.6 Conclusion

This chapter presented a summary of major ICS cyber incidents that occurred since the year 2000. The chapter provided recent statistical reports about ICS vulnerabilities and threats. A special section was introduced that discussed all direct attacks against PLCs and the literature review of related research works.

Chapter 4 PLC Code-Level Vulnerabilities

4.1 Introduction

Much attention is usually directed toward the hardware portion of ICS or SCADA systems such as: industrial components, peripheral devices, or networks. Research works are more into external attacks against PLC-BS like network intrusion, compromised SCADA devices and DoS, but less attention is given to the ladder logic code vulnerability within PLCs [128]. It has been assumed that ladder logic code is secure and safe as long as the network is healthy and protected from malware or intruders. But that is not sufficient since the ladder logic itself has its own overlooked or unnoticed vulnerabilities. This chapter provides an overview of some critical vulnerabilities within the PLC ladder logic code or program and recommends corresponding steps or methods to keep PLCs safer and more secure. The chapter focuses on specific ladder logic code vulnerabilities and weak points that might be exploited by malicious attacks. Those weak points could be a result of intentional malicious pieces of code embedded within the ladder logic code or inadvertent ones such as bad code practices or human errors.
Indeed, not many solutions exist to help secure PLCs such as certificateless cryptography [129] or intrusion detection through expected response times under normal operating conditions such as [130] and [131].

4.2 Ladder Logic Code Vulnerabilities

Any ladder logic code that is not well structured and designed increases the risks of vulnerabilities and security holes; even though the programmer is conforming to the company's standards and recommendations. And that could be more aggravated if the logic is not written by professionals, which is mostly the case. Standards are very subjective and are mainly company oriented. Such standards are mainly created and instituted to keep systems functioning, well optimized, and safe, but less attention is given to security threats and vulnerabilities. Such cases create a back door to hackers or could inherit the PLC programs insecure and dormant or unnoticed threats.
The following are some main examples of bad coding scenarios that any programmer should avoid to reduce or eliminate any code vulnerabilities:

- *Using duplicated instructions:* reusing certain operands – Such as: OTE, counters, timers, and JSR. – more than once in the ladder code leads to undesired result. Fig. 4.1

shows an example of a duplicated OTE operand - Y1. The duplication in this logic makes Y1 triggered during its unintended time. The reason is that Y1 is going to be turned ON in the first rung and right away turns OFF if X2 is enabled. So, it goes ON or OFF based on the scanning result of the rung it belongs to. An unintended fluctuating value of an operand would make it hard to debug or notice. Keep in mind that duplicating certain outputs would not be allowed in some PLCs, but some others would allow it.

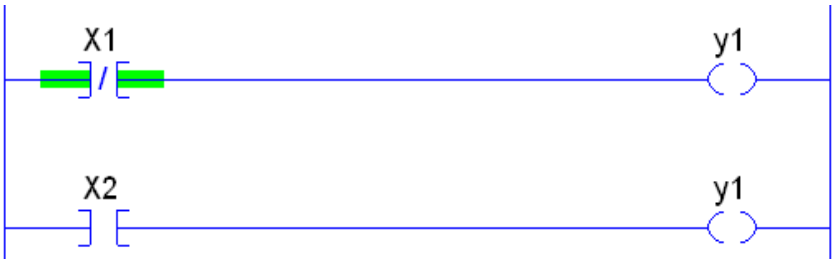


Fig. 4.1. Duplicating OTE operand.

- *Snooping:* a ladder logic code that can be written to log certain critical parameters and values to be leaked stealthily for spying purposes without affecting the logic flow and purpose. That can be done by utilizing array instructions like FIFO and some other arrays-based ones; e.g. “ADD ON” user defined instructions. Such instructions can be added unnoticed to the code and do not raise any suspicious or unusual behavior.

- *Missing certain coils or outputs:* occurs when a rung is missing a specific output coil (such as OTEs, latches or sets, unlatches, etc.) which other tag(s) depends on; see Fig. 4.2. Missing coils increases the risk of vulnerability threats. It is a warning sign that someone could be deliberately tampering with logic to deviate from certain critical values; risking the system to make wrong calls and decisions. Fig. 4.3 shows the proper way to handle OTE instructions where each rung has its proper pre-condition and output operands. Y11, an OTE instruction, depends on the value of the normally open instruction Y2. Not having Y2 instruction (deleted or replaced by non-useful false instruction) as a precondition to Y11, makes Y11 *rung*-condition-*out False (Y11 always OFF)* hard to notice.

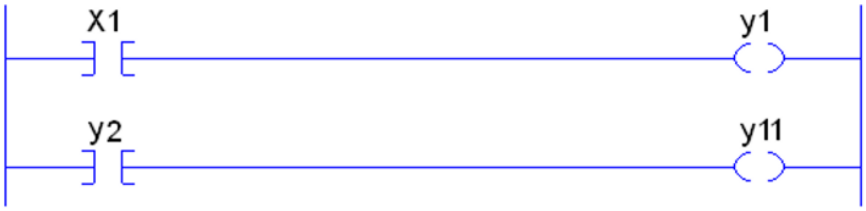


Fig. 4.2. Tag y2 is missing related input(s).

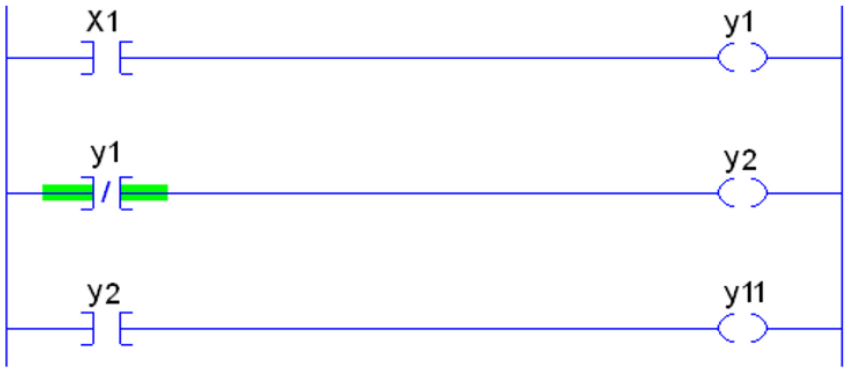


Fig. 4.3. Outputs instructions are properly energized.

- *Bypassing:* either by manually forcing the values of certain operands while the ladder logic is online or by using empty branches, jumpers, as shown in Fig. 4.4.

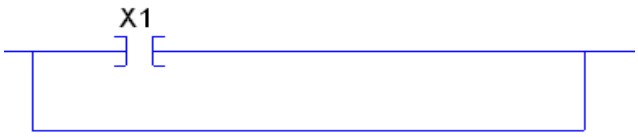


Fig. 4.4. Using an empty branch as a jumper.

- *DoS:* the user can write online or upload a malicious piece of ladder logic to the PLC that might be activated or triggered at a certain time. That could severely slow down the PLC, halt it, or cause major faults. The operator can't access the ladder logic, edit it, or monitor values in real-time. The attack can be done through:
    - Coding a repetitive SBR calls via JSR instructions.
    - Coding infinite loop via jumpers.
    - Nest timers and jumpers.
    - Improperly inserting MCR - Master Control Reset - instructions that de-energized non-retentive instructions like OTE coils.

- Coding certain ladder logic that might lead to fatal errors or major faults. Such faults might require restarting the PLC or re-uploading correct clean ladder logic which leads to data loss and temporarily shut down to the whole automated system associated with that PLC. The recovery could be time consuming and might cause damage to some meticulous industrial activities or devices; in addition to data loose of critical parameters or values. One of the solutions for such problems is to monitor jumpers and other looping routines using counters and timers. If the loops are going on more than expected, then warn the operator and halt certain suspicious routines.

- Using hard coded values: in certain situation using hard coded values or parameters endangers the process or its related program; see Fig. 4.5 . Numeric values are easier to modify than those driven by continuous feedback. Modification can be on purpose, inadvertent, or by malicious attacks. For instance, a programmer by mistake might enter a wrong value in the database table where the values of the instructions are easily displayed and accessible; that could also happen by toggling the values of the instruction displayed in the rung. Fig. 4.6 shows a solution that can keep source B numeric value updated even if it is modified inadvertently by a toggle or by updating the values in the PLC database table.

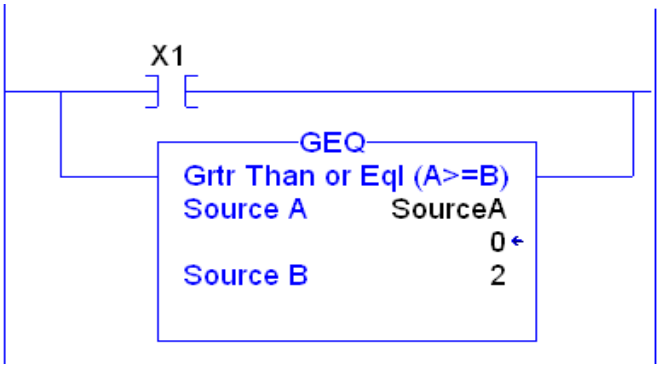


Fig. 4.5. Numeric values are vulnerable.

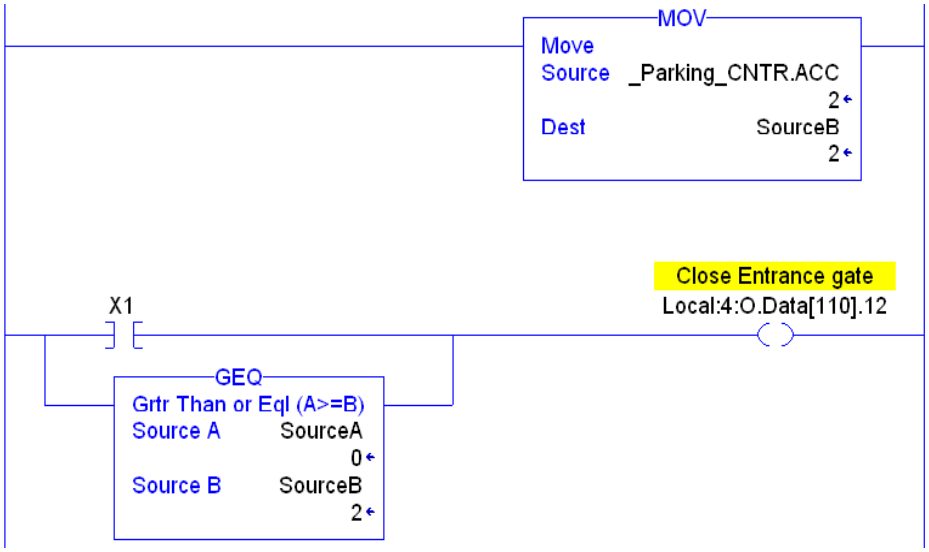


Fig. 4.6. Compare real-time numeric values not hard coded ones.

- *Racing:* occurs when two codes or operands of logic are racing against each other leading to inconsistent results and can be used to create a threat that could damage devices. Misplaced operands within the same code or even one rung - such as the racing scenario in timers – is a good example that often happens. Fig. 4.7 shows that having the done bit of the timer (tmr1), tmr1.DN, before the branch causes a racing problem if the timer's accumulator reaches the value of the Preset one's (assuming X1 is always ON). In other words, whenever the timer (tmr1) is done, it is reset again and the Valve01 is energized because the precondition, tmr1.DN, is false. There is always a chance the Valve01 will never get turned off or be de-energized. That would make it hard to locate the problem because the logic looks legitimate. The proper correction is shown in Fig. 4.8 .

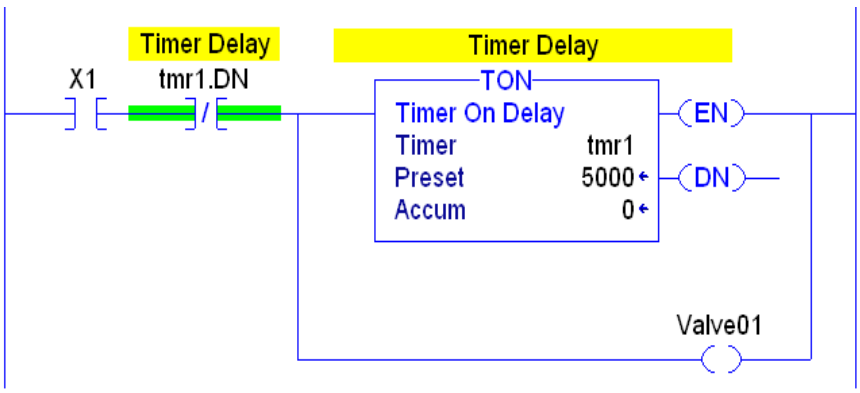


Fig. 4.7. Racing condition.

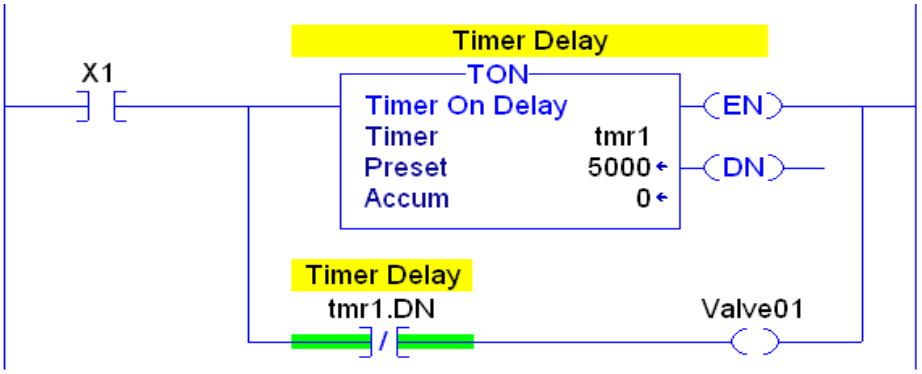


Fig. 4.8. Racing condition solved.

- *Lack of thorough diagnostics and alarm messages*: when there are no detailed and in-depth alarms, diagnostics, or preconditions the devices might be at great risk because the operator will only notice the damage after it occurs. Overall, the result is device damage or time delay in recovering, debugging, or maintaining. For instance, not setting an alarm message or warning for motor overload before enabling or while running it could damage the motor especially if the physical overload switch is compromised, missing, or malfunctioning.

  The problem will be more aggravated if the compromised device is critical – e.g., nuclear reactor – and yet lacking critical alarms or warnings. Another concern is when there are sufficient alarms and warning messages that can prompt the operator, but they got disabled either through another wrong or malicious piece of logic or by external user who manages to get through the code. A good practice is to add a ladder logic code that can simulate all faults' scenarios and check their status alive before running production. Another good practice is to create a heart-beat pulse bit – flashes every 50ms – that is synchronized with the alarms program section.

- *Compiler warning:* overlooking certain PLC compiler warnings would be critical since they might be a real threat; e.g. a compiler warns about duplicate outputs, see Fig. 4.9.

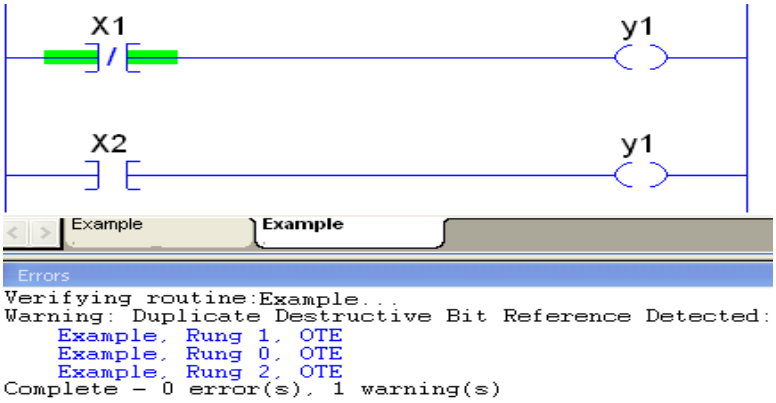


Fig. 4.9. Compiler warnings.

- *Unused tags or operands:* dormant malicious code or external attacks might take advantage of any unused tag because they are already predefined and using them will not trigger or not raise any flag. So many PLC programmers leave unused tags in the PLC database. Any malicious attack might exploit unused tags to trigger signals that might activate malicious output to interrupt or manipulate data. That can be done by utilizing instructions (timers, jumpers, etc.) that can overload the PLC OS, slow it down, truncate critical data, or generate certain datatype faults and errors.

- *Program Mode:* keeping the PLC in "Program Mode" or "Unlocked Mode" allows others to upload wrong or malicious ladder logic code; jeopardizing the whole automated system. The user can even wipe out the whole ladder logic or upload any suspicious one. Also, keeping the PLC in "Program Mode" makes the PLC vulnerable to any code manipulation with no need to delete or overwrite the whole program. It allows others to do online editing for ladder logic (add or delete pieces of code or data) while the ladder logic is running. That can be done by any user without being noticed since there will be no need to do a critical ladder logic code upload to the PLC; only critical uploads usually cause systems to stop and reset.

- *The lack of authentication*: before uploading new or modified ladder logic code to the PLC no authentication is conducted. Attackers can use this to upload malicious code, vulnerable code, or improper code or they can even compromise the PLC. To avoid that, comparison tools should be used to ensure the integrity of critical pieces of code. The comparison is to be made against a valid verified program.

4.3 Ladder Logic - Backdoors

Many companies and vendors assume that ladder logic is secure and not accessible by hackers and intruders because the PLC networks are usually air gapped. But that is not true for the following reasons:

- *Random threats:* the malicious attack could be done by malwares that are designed to affect certain PLC brands. A malware could be deployed remotely, by USB, or locally by infected PC on the PLC network. Such attacks could be initialized on purpose or inadvertent. Isolating PLC networks completely from others is not realistic because there is always a need to connect a PC or HMI – which could be infected - to a PLC where a programmer needs to monitor or edit ladder logic.
- *Internal threats*: could be because of upset employees, bad coding practice, infected logic written on infected PC, or intentional attacks; e.g. give remote access to hackers, open certain ports, or insert infected USB.
- *External threats*: external stealthy access to the PLC code to either keep a malicious code dormant and trigger it at a proper time or to be a "hit and run" scenario. Dormant pieces of logic could be used to steal sensitive information and parameters which could be used later on to sabotage or damage automated systems.

4.4 Conclusion

Vulnerabilities of PLCs are growing, leading to an increasing risk of threats and attacks. There are few works and scattered local efforts involved in improving PLC ladder logic code, but challenges remain. In this chapter we have solely focused on the code level vulnerabilities of ladder logic that resides and runs on PLCs. The chapter provides a summary and details of some major fundamental ladder logic code vulnerabilities and threats. Those vulnerabilities might be existing in any typical ladder logic: unnoticed or unknown — never thought of. Ladder logic code vulnerabilities could be dormant threats that can be triggered at any time risking the whole automated system that is associated with. Even though code vulnerabilities could occur because of bad coding practice, some might be unknown even to professional programmers.

In addition, we have provided solutions for the vulnerabilities mentioned above. Following the solutions and recommendations provided would highly mitigate, reduce, or eliminate malicious attacks or threat.

Chapter 5 Programmable Logic Controllers Based Systems (PLC-BS): Vulnerabilities and Threats

5.1 Introduction

Though PLC-BS are reliable, real-time devices that are widely used in most automated systems and industrial facilities, they are becoming a big concern. It has been proven that PLC are vulnerable like any other OS based systems, not mention that PLCs have limited resources and OS. PLCs are not designed to prevent and scan the code for malicious behaviors or stealthy attacks. Devices associated with PLCs such as HMI, peripherals and I/O devices, and networks are vulnerable and would risk PLC-BS. A study by Kaspersky Lab 2010 proved a new sophisticated level and era of attacks against PLC-BS software has started. Stuxnet was the first to include a PLC rootkit [108]. Stuxnet was able to maliciously spy, attack, compromise, or even exploit other machines to initialize attacks on other systems [9] [11]. It demonstrated a real sophisticated cyber security attack that catastrophically affected several areas of PLC-BS. By attacking the software (HMIs - WinCC, PLC codes -Siemens Simatic Step7, PCs - Windows, etc.) and faking values, Stuxnet severely affected crucial field devices taking advantage of multiple Windows zero days vulnerabilities [108]. Even though it is a very customized malware targeting Siemens SCADA systems, Stuxnet is just a typical threat that PLC-BS might face; especially if there are any cyberwarfare attacks. In addition to Stuxnet attack [108], so many other attacks were carried out by other threats such as Flame, Guass, Duqu, Wiper, and BlackEnergy malware [39], [1]. There are more than 50 new Stuxnet-like attacks on SCADA threats discovered [109]. Since Stuxnet's appearance, PLC-BS have attracted the attention of the hacker crowd.

In this chapter, we present critical PLC-BS security vulnerabilities and potential polices that can mitigate some of the threat.

5.2 PLC-BS Threats and Vulnerabilities

Nowadays, PLCs are more interconnected to many other systems and devices, see Fig. 5.1 to improve data monitoring. But with more connections, the risk of attacks becomes higher, and more vulnerabilities would be exposed and exploited.

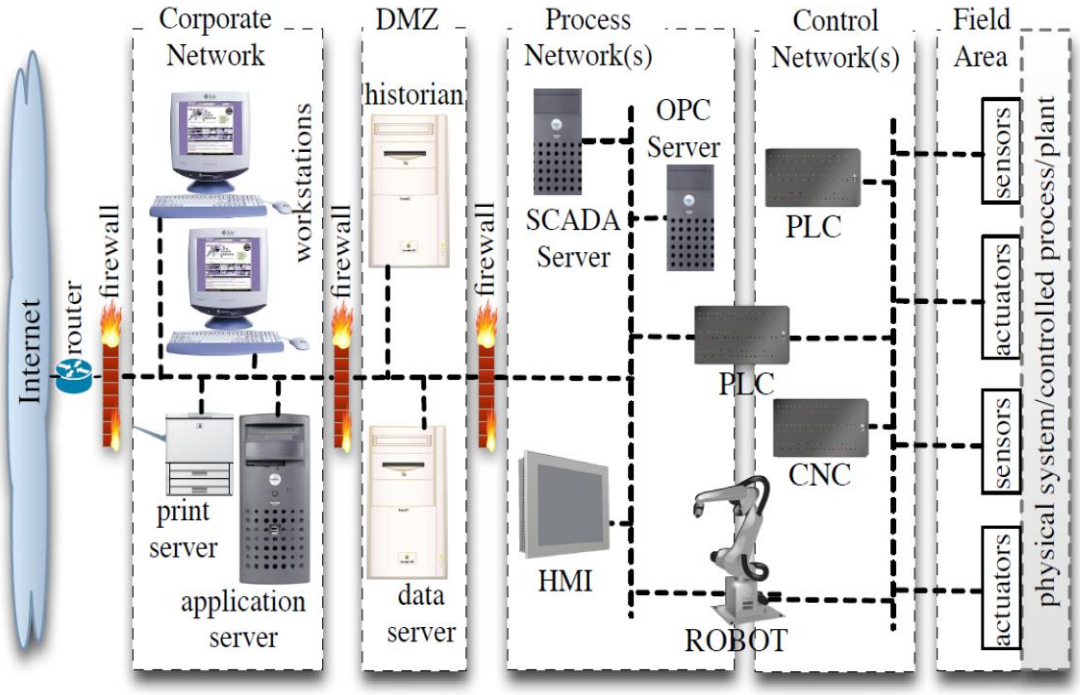


Fig. 5.1. Connections of PLC-BS to corporate networks and Internet [9].

The following is a categorization of the vulnerabilities that could be exposed and exploited in any PLC-BS:

- PLCs code vulnerabilities.
- PLCs vulnerabilities.
- HMIs and DTUs vulnerabilities.
- Field devices vulnerabilities.
- Network vulnerabilities.
- Network segmentation vulnerabilities.

5.2.1 PLC Code Vulnerabilities.

PLC code vulnerabilities has not been a great concern as that of network related ones. Companies, developers, and programmers have the tendency to believe that PLC code and associated programs are safe and secure since there are not connected to the internet or isolated from network intruders. But that is not correct. PLC code or a program can carry within nefarious destructive threats and vulnerabilities that hackers or regular disgruntled users might exploit. The vulnerabilities come from the way the code is written or designed. The following are some typical examples:

- ***Manipulated or missing faults messages*:** a malicious code can disable or suppress certain alarms. Basically, the manipulated logic code could stealthily deactivate critical faults, alarms, related logic code, or parameters. By that, operators would not notice any ongoing problems unless the system is down, i.e., recognizing threats after the damage occurs [78] .
- ***Disabling Outputs:*** that occurs when a condition-out rung or certain constructive tags, such as OTE instructions, get disabled or do not get scanned. For instance, hackers could add two Master Control Reset instructions (MCRs), as shown in Fig. 5.2, to disable several rungs and disable certain instructions. Once the MCR instruction is disabled, "Light" and "Light1" would be OFF regardless of the status of corresponding precondition instructions, "AB" and "CC".

Fig. 5.2. Disabling outputs when MCR goes OFF.

- ***Customized Function Blocks and ADD-ON instructions:*** user designed instructions, such as "ADD ON" instruction, can get very complex and hard to track. Hackers would exploit such instruction by tweaking them or add malicious code to them. Add ON instruction should be organized, certified, and continuously monitored by a company, protecting them with password would be a good choice if it is possible.
- ***Overflow:*** occurs when an instruction or an operand parameter length of input or output do not match what the PLC is expecting. It usually occurs because of bad code practices or malicious attacks trying to manipulate parameters [78] .

- ***Duplicated or misplacing certain instructions:*** duplicating or using certain instructions within the same routine or program, would result in unpredictable behavior, which in return risk associated devices and processes. Duplicating certain instructions would increase the debugging time [78]. Using the same timer within the same code would make that timer useless and unpredictable, see Fig. 5.3. Another example, if an MCR zone is nested within another MCR zone that would result in undesirable results.

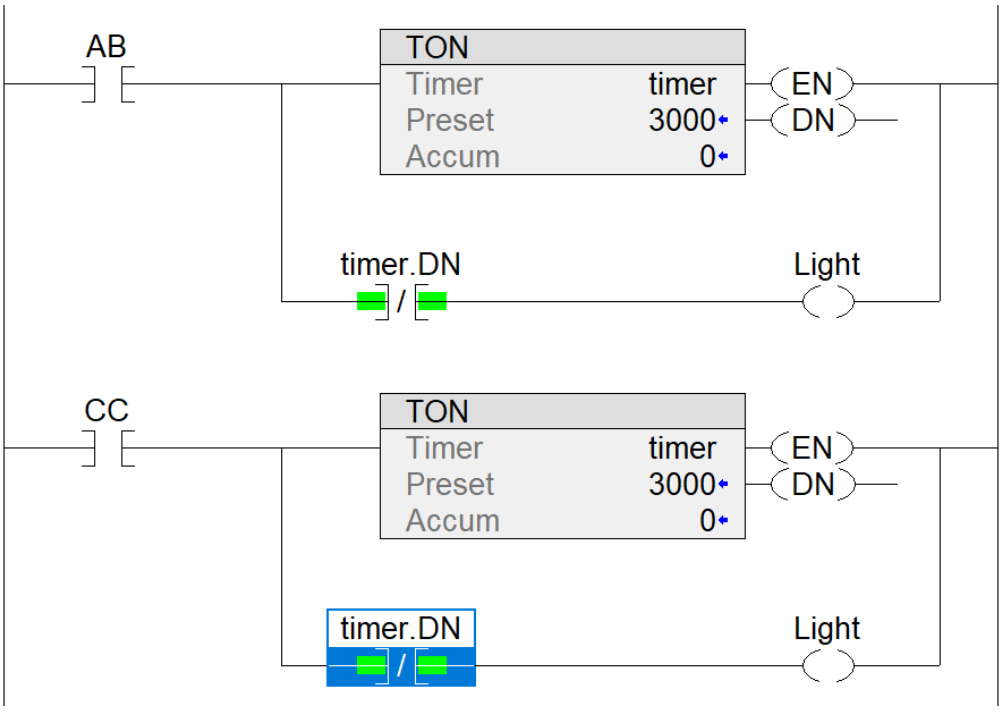


Fig. 5.3. Duplicating timer instructions.

- ***Unused tags:*** defining tags in the controller database that are not used in the logic could increase the level of threats; especially if the tags are not driven by a well-defined ladder logic or are for "ADD ON" or predefined instructions.
- ***Missing certain coils or instructions:*** can result in undesired behaviors. A user can exploit such situations to add an improper output that could severely affect the logic code and the associated controlled hardware. For example, missing an MCR instruction in a code that has several ones could lead to undesirable and dangerous code behaviors since MCR instructions must work in pairs.
- ***Bypassed instructions:*** That would be implemented by inserting an empty parallel branch, around certain instructions which would affect its rung condition-out, see Fig. 5.4. When hackers or regular developers use such techniques, it would go unnoticed which makes it difficult to debug and detect unless there is a clear compiler warning

which often are disregarded. In addition, by passing could be done by misplaced jumpers (JMP) or jump to subroutines (JSR), see Fig. 5.5.

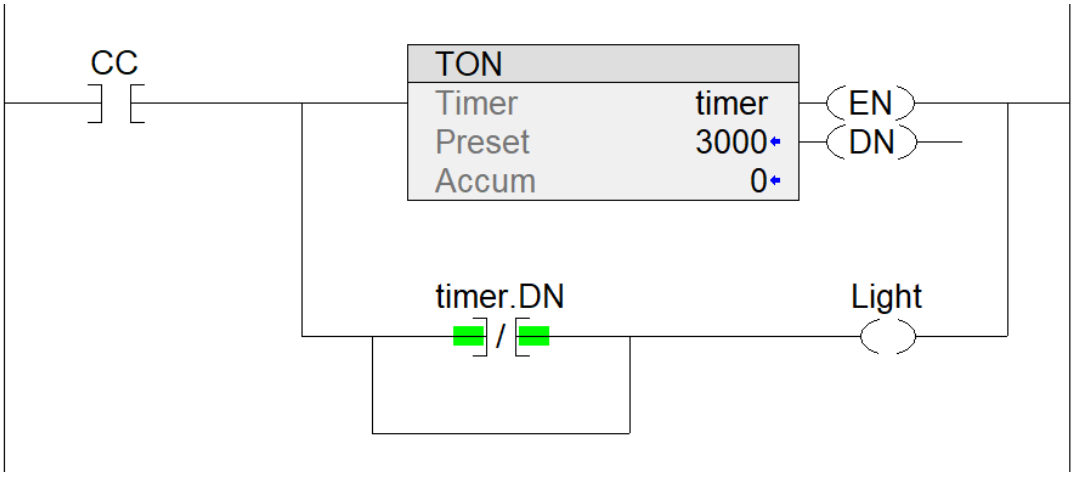


Fig. 5.4. Bypassing timer done bit.

Fig. 5.5. Bypassing all the rungs between JMP and LBL instructions.

- ***Hard coded values:*** in certain situations, using hard coded parameters in instructions like comparative ones could increase vulnerabilities. Parameters used in the comparison instruction can be manipulated by users, hackers, or malicious code without being noticed or constantly overwritten by the proper legitimate value, see Fig. 5.6. One of the solutions is to protect critical parameters by constantly moving a real-time value into it, using MOV instruction [76], see Fig. 5.7.

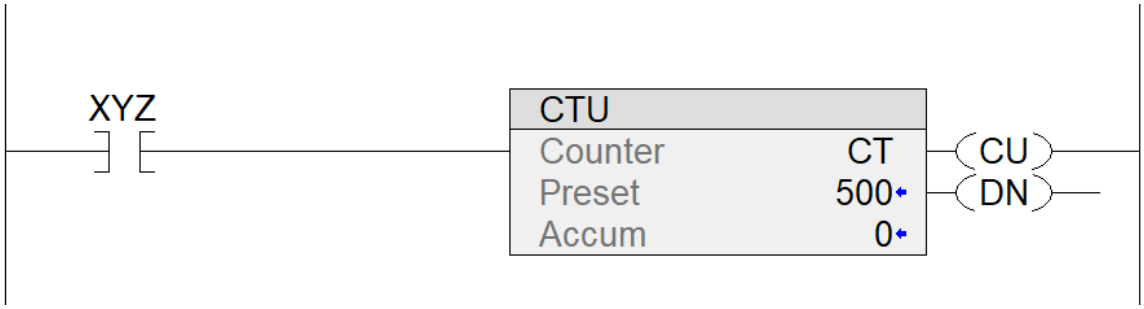


Fig. 5.6. Vulnerable "PRESET" value.

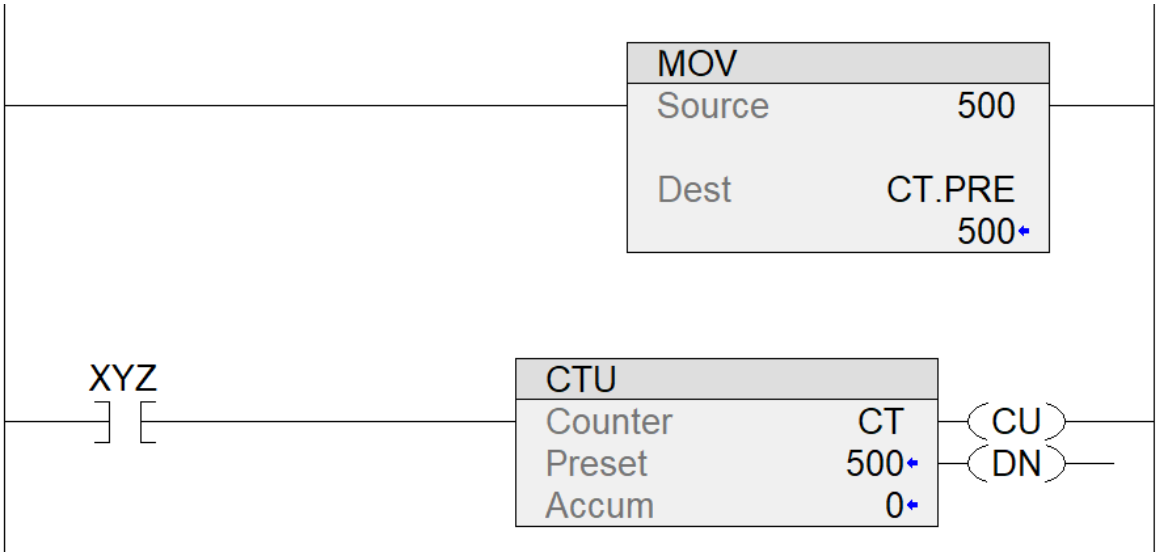


Fig. 5.7. Using "MOV" instruction to define a "PRESET" value.

- ***Racing:*** misplaced branches, calls, or instructions can lead to inconsistent results. The logic is going to behave in an undesired way based on the unpredicted result of the race. In Fig. 5.8, "tmr1.DN", which is a status of the timer "tmr1", is placed just before the parallel branches creating a racing scenario between the timer or energizing "Valve01". To avoid that, the instructions or the branches should be properly placed. "tmr1.DN", see Fig. 5.9, shows the proper location of the instruction to prevent any racing [76].

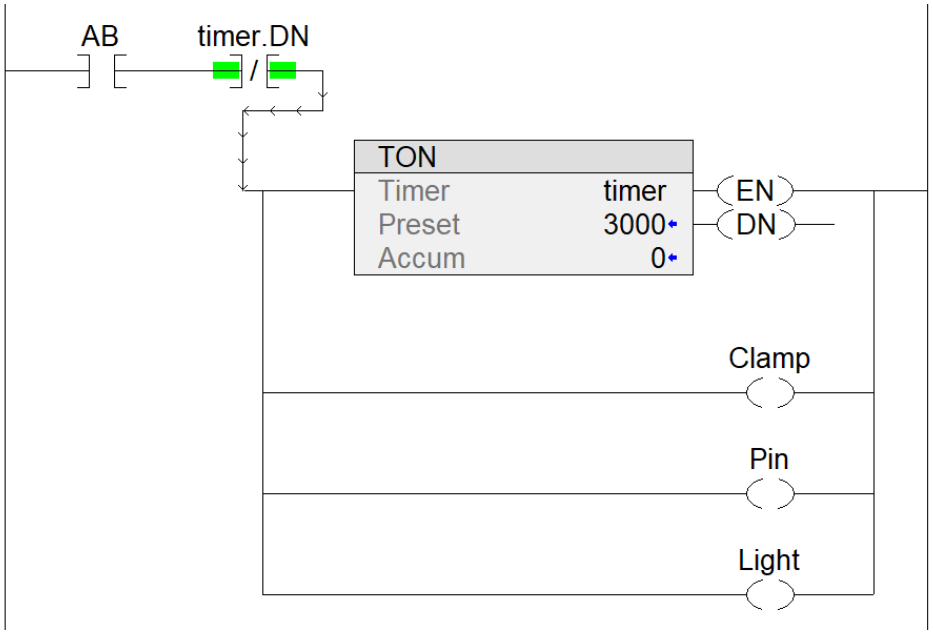


Fig. 5.8. Racing condition.

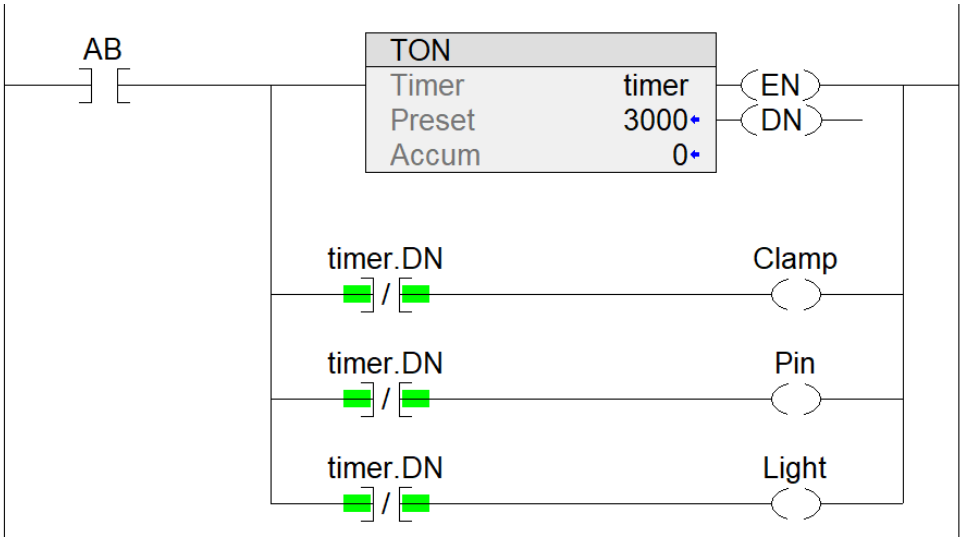


Fig. 5.9. Racing condition solved.

- ***Compiler warnings:*** Some compilers of logic design software might not be strict enough in compilingPLC code to give more flexibility to developers. PLC programmers should pay great attention to the compiler errors as well as warnings. Compiler warnings could have a great value that could shed the light on improper codes or instructions that could be exploited by malicious attacks.
- ***Infinite loops:*** using FOR instructions, JSR instructions, JMP instructions, nested timers, etc.
- ***Fatal Faults:*** trigger certain faults that could cause PLC program to crash.

5.2.2 PLC Vulnerabilities.

Since PLCs run commercial operating systems, they are vulnerable like any other known OS; Linux or Windows. PLCs are not designed to be cyber-resilient since they have limited resources and are insecure by design. They were never designed for resilience against threats and attacks. The following are some examples of PLCs' vulnerabilities:

- ***Obsolete PLC Firmware**:* PLC Firmware, i.e., OS, must always be up to date. A PLC OS such as VxWorks or OS-9, runs with the highest privileges and there is little memory protection between OS's tasks [132]. If the OS level vulnerabilities are exploited by an attacker, the system could be completely taken over allowing installation of malicious programs [133]. In general, PLCs' OS are fragile when it comes to security: not frequent update or patches and not built anti-malware application. They are not frequently patched or updated because their networks are isolated or limited and un-upgradable firmware. More reports are issued regarding PLC OS vulnerabilities [134] [135] [136]. Another issue is that some threats are not properly addressed or reported. That poor addressing is due to isolated PLC systems, undocumented problems, or untraceable threats. It is hard, for instance, to update a firmware vulnerability or report it to vendors if PLCs are not directly connected to the vendors' network or to the internet. Without good logging and traceability, vendors could not get enough details to report the problem and offer proper countermeasure solutions. That makes patching and even revalidating feedback difficult.
- **Unrestricted uploads:** having an access point to a PLC allows user to upload any malicious code to the PLC, manipulate the current running one, or even upload new firmware. PLCs typically do not check whether the uploaded code is from a verified trusted source or not. Also, PLCs have no capabilities to know whether the uploaded code is a malicious one or not. What matters is whether the code is compiled with no syntax or program errors. Any PLC compiled code can be uploaded to the PLC and overwrites or modifies the current running one.
- **Unlocked Mode:** PLCs are, most of the time, unlocked and not protected by any password. That would allow exploiters and hackers to access the running logic code, monitor tags, manipulate the code, or even download a totally wrong logic code. Some vendors offer physical security keys to lock PLCs, like turning the key to "Run". A locked PLC prevents any code modification, update, or download.

- **Non-Isolated Networks:** with more interconnect systems and the integration of IoT, PLCs or any associated device could be exploited by hackers to launch attacks against other PLCs or devices that are on the same network.

5.2.3 HMIs and DTUs Vulnerabilities.

HMIs, DTUs, and HTUs are becoming more remotely accessible and interconnected with other networks and devices. As such, they are becoming more vulnerable and are attracting more hackers and threats. Like any other computers, they are vulnerable to any threats within the network and inherit all the vulnerabilities of the OS that are built on. For instance, HMIs have become generic or off-shelf software products that are built on or share a common architecture computers' technology, languages, OS, and communication used in Information Technology (IT) systems like Windows OS, ActiveX, Java, etc. However, being generic software based, HMIs are becoming more susceptible and vulnerable. Attackers consider them regular PCs or vulnerable devices on an accessible network.

Attacking any HMI, including its related database, could lead to severe consequences on software (deleting or manipulating codes, alarms, or database records) as well as on hardware. Since HMIs are used by operators to control manual activities and to monitor real-time status and alarms, their roles are critical and extremely sensitive. Attacks on HMIs can either slow them down manipulate manual activities, fake status of alarms or data, or steal confidential logged information. An exploited HMI application that monitors and manually controls field devices, might consequently affect the functionality of related PLC-BS devices - encoders, VFDs, motors, etc. Software attacks are summarized as follows:

- ***External malware:*** that can be deployed either via internet, company's network, or locally by users – e.g., inserting an infected USB into a HMI, Server, or PC that is on the PLC-BS network. Malwares can spy and damage industrial systems, delay or block networks, or even include PLC rootkits [137] [111].
- ***Deception attacks:*** that includes a wrong unauthorized identity of a command sending device that can enable remote access and cause fatal damage to the software and hardware [115].
- ***SQL Injection:*** affects Web based HMIs and servers with database applications (some HMIs or servers). It is a way to take control of a system or to insert unexpected SQL statements into a query in order to manipulate a database [115].

5.2.4 Field Devices Vulnerabilities.

One of the main vulnerabilities that can be exploited to manipulate and risk PLC-BS integrity and reliability is faking hardware status. As real-time processors, PLCs monitor and control real-time status (inputs) or command (outputs). When any data sent to a PLC is not accurate or delayed, the PLC would execute wrong or improper commands. For example, when the status of a physical instrument or device is manipulated to send fake values, a PLC would act accordingly without detecting any abnormalities. Tampering with any value of an input or an output would deceive the PLC and lead to undesired ladder logic program results; endangering equipment, productivity, environment, and human life. That is accomplished by compromising the associated network which goes unnoticeably and without modifying any PLC logic code or firmware. In addition to faking their inputs/outputs, hardware devices can be vulnerable if the related PLC-BS programs are compromised; whether its HMI related or PLC ones – e.g. ladder logic code or database. The following is a summary of hardware vulnerabilities:

- Fake inputs: status, parameters, or values of the compromised sensors or input devices (e.g., proximity switches, safety emergency stops, safety switches, encoders, VFDs, etc.). Also, such vulnerabilities can lead to fake inputs carried out from the HMI to the PLC, e.g., operator's manual selection, operator's entered data, etc. [137].
- Fake outputs: status, parameters, or values of the compromised actuators or field devices (e.g., valves, VFDs, encoders, stepper motors, etc.) PLCs' or HMIs' outputs can also be faked; affecting other related devices [108], [137].
- Manipulated inputs and outputs values by tampering data integrity of the PLC, HMI, or other devices such as manipulating their database or tags to create severe hardware damages or threats to PLC-BS [138].
- Manipulated PLC ladder logic codes or HMI programs that damages hardware devices; Stuxnet exploited Siemens software to manipulate parameters of devices resulted in damaging critical hardware devices [108].
- Manipulated HMI functionality or PLC ladder logic codes by slowing them down to severely affect production or make them inaccessible.
- Deactivated alarms and critical messages or warnings; could delay response time and make it slower to detect the problem.

5.2.5 Network Vulnerabilities

Nowadays, most PLC-BS networks architecture distributes their functionalities across common or open standards protocols such as WAN and LAN; Ethernet/IP, DeviceNet, ControlNet, Profibus, PROFINET, and Modbus. However, the advancement in related technology have increased vulnerability and threats; they lack security-integrated mechanisms and they become less obscure to hackers [111]. Not providing security mechanisms to protocols makes the network vulnerable to:

- Packet manipulation (latency, spoofing, eavesdropping, deletion, etc.) [30].
- Attacks on the communication stack: network layer, transport layer, application layer and the implementation of protocols (TCP/IP, OPC, ICCP, etc.) [115].
- Remote or local field devices attacks; Intelligent Electronic Devices (IEDs).
- ARP Spoofing, password attack, and DoS [117] [118].
- Backdoors and Holes in Network Perimeters [11].
- Database attacks.
- Communications jamming, blocking, or hijacking; MITM attacks [126].

5.2.6 Network Segmentation Vulnerabilities

Many companies still assume that they are safe and secure if their industrial networks are off the internet or isolated [139]. There are still some who believe that segmenting a network as in Fig. 5.1 keeps PLCs networks secure and safe. They assume that air gapped industrial network, which is a way of network segmentation, secures all PLCs and associated field devices including HMIs. But segmenting the ICS network this way is not secure enough for the following reasons:

- ***USB threat:*** the malicious attack could be deployed by infected USB.
- ***Inherited:*** a malicious attack can be carried on by another infected computer or HMI that it is plugged to the same PLC-BS network. Also, some worms can go from one PLC to another PLC if they are on the same networks.
- ***Disgruntled employees*****:** an upset employee can create major damage and harms. He or she can sabotage the code, infect HMIs or PCs, write dormant malicious code within the ladder logic, or even open certain ports to hackers.
- ***Bad code practice*****:** a programmer might inadvertently write pieces of code that might damage certain machine or create DoS; e.g. infinite loops.

- ***Dormant access*:** some vulnerabilities might take years before noticing them. Their job is not to create direct damage. They are there just to eavesdrop, collect, and steal sensitive information and data.

5.3 Lack of Data Forensics

Whenever an attack occurs, a forensic investigation must take place in order to figure out the causes and responsibilities. It is typical to collect related data. By analyzing and reverse engineering all needed collected data, we can get better understanding of the attack's behavior, elements, techniques, etc. Also, further similar malicious attacks can be prevented and stopped. Since PLC-BS are critical and widely used in automated industries and critical infrastructure facilities, a thorough forensic investigation would have been a great help because of the following:

- To identify the root cause of the attack.
- To identify the potential elements and devices involved or exploited.
- To identify possible risks and weaknesses.
- To identify devices status and configuration just before the attack.
- To find the proper remedy of the attack and prevent future similar reoccurring ones.

Unfortunately, forensic methods or tools are difficult to apply or use in PLC-BS. Unlike traditional IT systems and related devices, PLC-BS are more complex and custom-made. The difficulties or impediments that make applying digital forensic very challenging - if not impossible most of the time - are as follows:

- Continuity: PLC-BS are continuously fed by field devices and I/O's. Mainly they are real-time devices that are continuously being updated with newer information; tracing previous ones would be hard if there are no continuous incremental backups.
- Volatility: critical information of running programs and hardware that can be used as evidence is located within volatile memory. PLCs, for instance, do not have proper hardware and software that log thorough code or firmware modifications or updates.
- Fast Response: since PLC-BS are real-time devices that are continuously fed by updated newer information, delaying forensic response would make it more difficult to analyze and trace the problem. The slower the response is, the less related data will be resided within the volatile memory; overwritten by newer ones.

- Validity and availability: Being real-time systems, PLC-BS care much about the validity, integrity, and availability of data more than security, encryptions, or backups. Slowing the scanning time of any running systems would create unfavorable problems. Therefore, using any tools or methods that could slow down PLC-BS would not be tolerated. That makes it difficult to embed any typical forensic tool.

5.4 PLC-BS Security Recommendations

Based on extensive experiments and studies that we conducted and PLC-BS audits, the following are some recommendations to keep PLC-BS protected against threats or at least mitigate the risks:

- Security First: as industries consider safety as a main factor while designing, updating, or functioning any PLC-BS, security should be paramount. That must consider the hardware, software, and networks. Companies have to come up with more detailed risk assessment, responses, and standardizations before implementing any PLC-BS projects.
- Cybersecurity is everyone's responsibility. All employees should be always aware and concerned about security. Employees should immediately report any insecure practice, insecure device, or skeptical behave.
- Cybersecurity must be an organization culture. For instance, an organization should offer periodic security training to its employees. Employees should be aware of the security threats and wrong practices that might affect their work areas and systems.
- Roles and Authentication: privileges to access information and devices should be properly restricted and well considered before assigning them to the employees. Privileges should be well validated, controlled, logged, and monitored (use unique IDs or access credentials). Unauthorized or non-monitored activity should be prevented or at least reduced to the minimum. Users should only have access to their daily related work and tasks. Automatic logging review and monitoring of users might also help.
- Air-gapped network: PLC-BS systems should have their own isolated private networks, as much as possible, or at least clearly distinguished from other networks.

- Redundant files backup and recovery tools: use several ways and tools including scripts to backup critical files and make it handy to recovery it or use it when needed.
- Daily checkup and comparison: as companies are so concerned about safety before and during running production lines, they should worry more about the integrity of the files they are running on the PLCs or the HMIs. That should be done by having a software tool that compares the ladder logic against the original trusted master file before starting the new production lines. There is always a chance that someone might sabotage the logic or create a dormant malware within the ladder logic code the goes unnoticed.
- Remote access and IoT: must be restricted either to certain device, areas, or sometimes disabled. If needed, it must only be enabled for limited duration and used by internal trained personnel from an approved monitored and controlled device; all communications should be filtered and checked. Systems or devices that do not need to be connected to other networks, including internet, must be properly segregated and isolated to avoid any threat [140].
- USB ports should be physically disabled on HMIs and on any other associated PCs. Only authenticated and approved USBs are to be allowed and must be used by administrators. Malware, like Stuxnet – spread via SCADA network through an infected USB storage device.
- Spare port of any device should be disabled.
- System logging: must be generated and kept for a reasonable amount of time in case are needed if any thing goes wrong.
- Periodic system auditing and periodic penetration testing.
- Continuous vulnerability and threats assessment and pre-emptive solutions.
- Periodic risk assessment and analysis. Risk assessment answers questions like: What can go wrong? What is the probability that it would go wrong? What is the impact or what are the consequences [141] [139].
- A cost–benefit analysis (CBA) is important, but without compromising security, safety, and data real-time validity, integrity, and availability.
- Dedicated and protected devices: limit connection to PLC-BS to certain dedicated devices. Make sure any end-user device or PC is safe and protected (e.g., antivirus and other security apps).

- Periodic updates: keep software including firmware up to date to reduce vulnerability and threats by deploying approved patches or updates from dedicated approved and safe PCs.
- Intrusion system detection: that should also include ‘traditional’ perimeter protection (e.g., antivirus, firewalls, etc.). They should always be kept up-to-date and ON.
- PLC-BS should be resilient and secure. Do not just focus on securing certain insecure field devices. Securing the whole systems is critical and a must.
- It is impossible to have 100% risk free system. But we should make it harder on any attacker to initialize any major or massive attack. If stopping a malicious attack is difficult, at least slow it down. Stopping or slowing down any attack can be done through fast detection, network segmentation, and recovery steps.

5.5 Next Generation: Secure by design

Currently, several research papers have explored how to enhance security or add some security features to PLC and PLC based devices. [142] is one of those research papers that can be summarized as follows:

- PLC signed firmware and secure boot.
- High percentage of ICS protocols will be able to integrate authentication into certain protocols. ICS Protocols authentication will prevent attackers from spoofing or sending illegitimate commands.
- IPsec communications protocol between Windows based computers and a few PLCs - Modicon M580. IPsec is implemented only among certain modules for authentication header protocol and uses pre-shared keys rather than certificates.
- Security Logging: get ICS security logs into a SIEM; partially implemented in Modicon M580 [142].
- Disabling unused Ethernet ports.
- Access Control List (ACL): restricts access by IP address based on the administrator criteria, used by Modicon M580 safety PLC [142].
- Syslog: a few PLCs, Modicon M580 safety PLC, have started supporting syslog. Security events or some PLC logs can be exported and managed [142].
- For old PLC-BS, add certain modules in front of the existing controllers to add or enhance security functions.

Those critical improvements are yet not widely adaptable or applicable. Mainly, most of PLC-BS can't adapt or integrate those features due to limited features and resources, backward compatibility problems, limited compatible modules, PLCs' scan time delay, and high costs which might require a major rip and replace. Some of the drawbacks of using IPsec:

- Adding IPSec to M580 PLC consumes extra 10ms for reading 10,000 variables [142].
- IPsec is exclusively used between M580 PLCs and Windows computers. It is not built to support communications among M580 PLCs [142]
- Generic or typical filed devices, like sensors or Armorblocks, may not work with encryptions and enhanced security due limited resources and the priority of real-time transmission. Even it was to be feasible to encrypt some devices, updating the algorithm of those field devices would be a difficult task.

Nevertheless, PLCs are still far from being capable to detect internal threats within their running code due to limited resources. While regular OS systems, such as Windows, have antivirus application and malicious detection tools, PLCs do not. Any code that is compiled without any error could be downloaded to a PLC and get executed whether its malicious code or not. A PLC can't detect and warn operators of any suspicious behavior nor it could intelligently eliminate any eminent or suspicious threat.

5.6 Conclusion

This chapter has provided an overview of the architectural components of PLCs - languages and hardware - in addition to an overview of associated industrial networks, field devices, HMIs, and DTUs. The chapter summarized the major vulnerabilities of PLC based devices and listed each group under its proper categorization. The vulnerabilities and threats against PLC_BS are categorized as: PLCs code vulnerabilities, PLCs vulnerabilities, HMIs and DTUs vulnerabilities, field devices vulnerabilities, network vulnerabilities, and network segmentation vulnerabilities. Every category shows its associated vulnerabilities and threats that could be exploited by attackers or malwares. A special attention has been given to illustrate, analyse, and evaluate certain ladder logic codes and bad programing practices. Not writing PLC programs professionally and up to recommendations provided would increase unnoticed vulnerabilities and threats. The absence of forensic data is yet another challenging weakness that we have discussed. Not having adequate and detailed forensics would encourage hackers to exploit such missing feature to cover their traces and further similar risks or attacks. It would

be hard if not impossible to reverse engineering any attack and collect enough evidence among PLCs. In addition, recommendations have been provided and suggestions to avoid future vulnerabilities and threats. Following those recommendations, would highly mitigate, reduce, or eliminate malicious attacks or threats. Finally, we have provided PLCs are not “secure by design”, they have their vulnerabilities like any other OS. Upcoming next generation PLCs and related security solutions have been presented; showing some features and challenges. We have explained the advantages of next gen PLCs – like Modicon M580 safety PLC – and their limitations; especially when it comes to resources, costs, and backward compatibility.

Chapter 6 Applied Methods to Detect and Prevent Vulnerabilities within PLC Alarms Code

6.1 Introduction

The demand for reliable, valid, and secure PLC alarms has been a great concern since the 2000 Maroochy Shire cyber event, Australia [28] [8] [143]. In the Maroochy malicious attack, an insider was able to suppress the PLC alarms to prevent them from notifying operators and central servers of any threats or abnormalities. Undetected suppressed or inactive alarms prevent real-time notification messages which in turn could mislead operators, delay maintenance, increase financial loss, risk the safety of ICS systems, or even jeopardize human life by preventing outgoing alarms from being raised and reported, operators were fooled into believing that the system was running normally. Though preventing alarms from being raised or being acted upon might not be considered as critical as other threats, it could risk critical devices or endanger human life [144]. Every PLC has a dedicated alarms code that is responsible for monitoring the status of the controlled devices and raising alarms to alert staff in case of any abnormalities or faults.

This chapter focuses on identifying and detecting vulnerabilities within the PLC alarms code and introduces countermeasure techniques to enhance the validity and the reliability of the code. A real-time test bed was implemented in ladder logic that was able to detect code abnormalities and prevent possible exploitations.

6.2 Related work

Very little work has been done on vulnerabilities of PLC alarms code. The concerns of researchers have been mostly focused on other kinds of vulnerabilities that are associated with general PLC codes [78] [79] [82] [81] [80] [137] [145], non-code related or general alarms analysis and management [144], [146] [147] [148] [149] [150] [151] [152] [153] [154], ICS networks related [155] [119] [156] [133] [157] [158], code intrusion and injection [137] [90] [159] [160] [121] [161], security credentials and accessibilities [83] [87] [89] [162], general recommendations and surveys [163] [164] , statistics [165] [166] [167], and assessment [168].

6.3 Roles of PLCs and their Dependencies in Alarms

ICS, typically, consist of PLCs, HMI devices, industrial networks, and field devices. Because of their important roles in monitoring alarms, the following is a brief explanation of each:

6.3.1 PLCs' Alarm Code

PLCs are the main controllers of ICS. PLCs handle several programs that control and monitor other dependencies, activities, and processes in a real-time manner. Among those programs, there are associated or dedicated codes to handle alarming messages. A PLC alarms code can be written in ladder logic, function block diagram (FBD), sequential function chart (SFC), or structured text (ST).

The role of a PLC alarms code is to raise alarms whenever faults or abnormalities are detected while monitoring field devices and operations. Once detected, the PLC alarms code sent raises alarms to HMIs to alert operators, see Fig. 6.1.

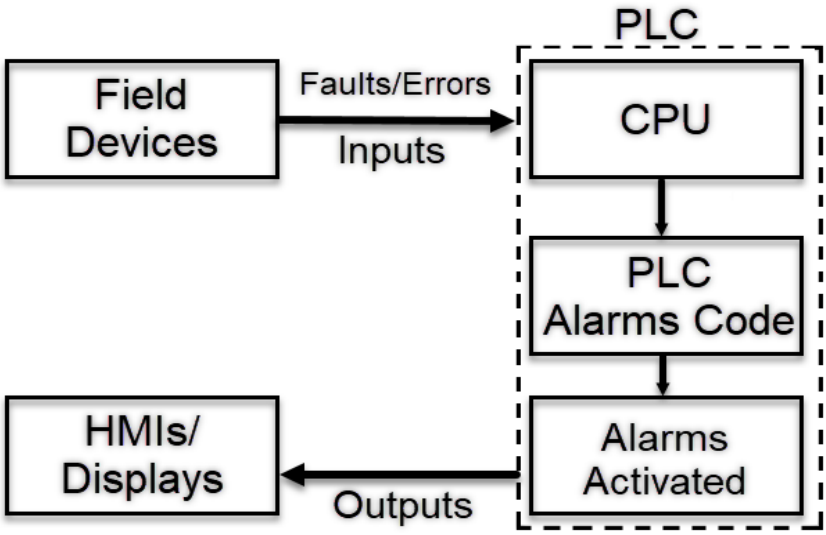


Fig. 6.1. PLC flow in capturing and triggering alarms.

6.3.2 HMIs' Role

HMIs act like visual aids to operators where any alarming messages raised by a PLC alarms code would be displayed and acknowledged by the operators. HMIs could be considered like a GUI interface of a PLC to operators to check messages and perform manual activities. HMIs could be as simple as touchscreen panels or could be more complex such as server-based ones where they can handle statistical tools, logging capabilities, ladder logic code monitoring in addition to alarming messages and manual activities.

When it comes to monitoring messages, HMIs are slaves to PLCs since alarming messages are controlled by PLC alarms code. If alarms within a PLC alarms code are suppressed, no messages would be displayed on HMIs.

6.3.3 Industrial Networks' Role

Industrial networks (such as EtherNet/IP) enable real-time communication among I/O modules, HMIs, PLCs, and associated field devices. If the network is down or slow, that would affect displaying proper alarming messages handled by PLCs.

6.3.4 Field Devices' Role

Field devices (such as limit switches, actuators, drives, etc.) are controlled and monitored by PLCs via industrial networks [6]. If any field devices or their associated are faulty, the designated PLC would detect that, and related alarms would be raised by the PLC alarms program. However, no alarms would be raised or triggered, if the PLC alarms program gets compromised, which could lead to critical consequences.
The following are examples of some field devices with a brief description of each of them:

- **Sensors' Role**: Sensors send the status of devices and instrumentations to PLCs. Even with highly reliable, thorough, and accurate sensors, any designated PLC alarms program would be not valid and reliable if its code gets compromised; where alarms could be suppressed, tampered with, or deleted.

- **Actuators and other Devices' Role** These are devices that translate received output signals from PLCs into practical operations. Good examples of such devices are actuators, inverters, and servo motors. Most of the time, output commands from PLCs to such devices must also be monitored, so PLCs could verify, stop, or adjust any of their behaviors accordingly. In case of any errors or issues, the PLC through its alarms program would raise alarming messages to alert operators.

- **I/O Modules' Role**: I/O modules, such as ArmorBlocks and I/O scanners [19], facilitate communications between PLCs and all other devices. They reduce labor costs, wiring, and maintenance costs. PLCs monitor and control all inputs and outputs PLCs through I/O modules; including feedback and status bits that are critical in any PLC alarms program.

6.4 PLC Alarms Overview

Alarms including faults, system errors, and warnings are essential indicators in protecting, maintaining, and enhancing automated systems. The alarms are simple to configure and develop in a PLC alarms code and HMIs. Despite their simplicity, they are critical, vital, and helpful in recovering from downtime or preventing damages [144]. Whenever there are any faults or abnormalities, alarms are raised or activated to protect devices and sent to HMIs to alert operators. To be useful and helpful, alarms should cover and monitor the status of all sensors, devices, I/O modules, and other dependencies that a PLC controls. Therefore, a PLC alarms code that handles alarms must be reliable, valid, and thorough. Thorough and reliable alarms would reduce maintenance costs, decrease troubleshooting time, and increase efficiency, and help in statics tools if logged [148].

The following are some useful information regarding the alarms raised by PLC alarms code:

6.4.1 Alarms Levels

PLC alarms can be broken into several levels or groups:

- System-stop alarms: these are raised whenever there are global, unsafe violations where more than one station is involved. For example: safety gates and perimeter E-Stops.
- Station-stop alarms: these only stop specific and certain devices or stations within a cell or a predefined zone.
- Cycle-stop alarms: they are non-immediate stops that take effect once a cycle is done.
- Warnings alarms: these are not designed to stop any system or devices but are used to initialize warnings in case of undesired conditions. Though these are not considered critical, they are helpful in predictive maintenance.

6.4.2 Alarms Prioritization

Prioritizing alarms helps in preventing flooding the system with messages and reduce operators' distractions. When an alarm is raised it indicates an abnormal state was detected by the PLC alarms code in a process, an operation, or a device. Since not all alarms carry the same weight, alarms prioritization must be taken into considerations to reduce distraction and accelerate maintenance response time, see Fig. 6.2.

The prioritization of alarms is categorized as follows:

- High Priority alarms: cause critical harm to humans and other living beings, environment, equipment, and huge economic losses. Such alarms must be given the highest attention and fastest response time by operators.
- Medium Priority alarms: Are not as serious and critical as the previous ones. They might cause serious financial loss or deteriorate the life service of devices or equipment in the long or short run.
- Low Priority alarms: minor financial impact, and less financial loss.

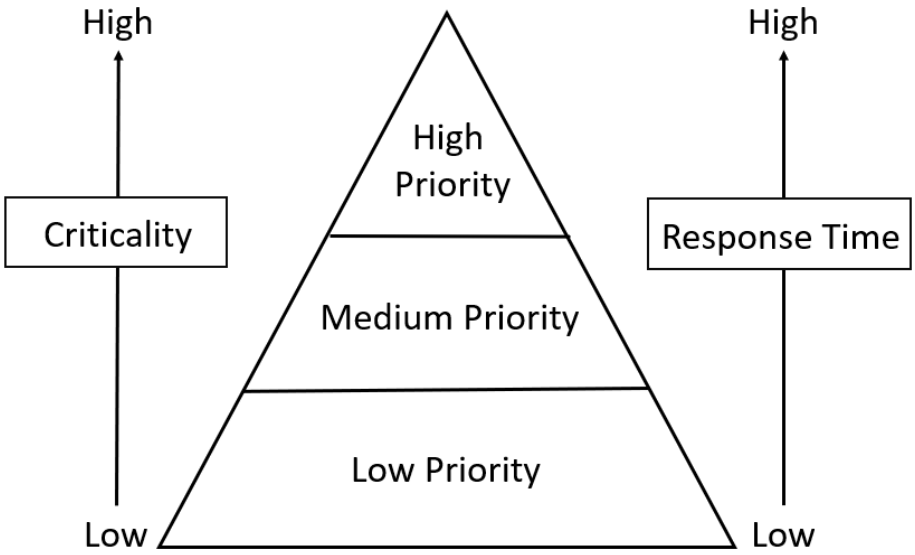


Fig. 6.2. Alarms priority based on criticality and time responses.

6.4.3 Supressed or Tampered Alarms Consequences:

If alarms within the PLC alarms code were to be suppressed or tampered with, they could cause the following:

- Risk the safety of workers or even other living beings.
- Impact the functionality and safety of devices.
- Impact public and private facilities and infrastructure.
- Impact on the environment.
- Increase maintenance costs and troubleshooting time.
- Prevent addressing critical alarming messages.

6.5 A Real-Time Testbed for PLC Alarms Code

In our testbed prototype, a ladder logic program was developed as a real-time PLC alarms code to perform several experiments.

6.5.1 Devices and Software Used

The experiments were conducted by using current and well-known industrial PLC-BS components in industrial systems. The following are the main devices and software used in implementing the testbed:

- PLC: Allen-Bradly/Rockwell GuardLogix5570 Safety Controller.
- Software: Rockwell Logix Designer version 33 to develop, monitor, and edit the ladder logic code.
- Allen-Bradley EtherNet/IP Network modules.
- Allen-Bradley ArmorStratix module.

6.5.2 Developing Alarms Ladder Logic Code

A typical ladder logic code was designed to monitor faulty devices and operations, as shown in Fig. 6.3, Fig. 6.4, and Fig. 6.5 . In the beginning a “Reset” setup code was implemented to clear non-retentive faults whenever it is needed, as shown in Fig. 6.3. For instance, by pressing the associated HMI push button (HMI_PB_Reset) of PV01 group, all faults related to that group would be cleared and acknowledged.

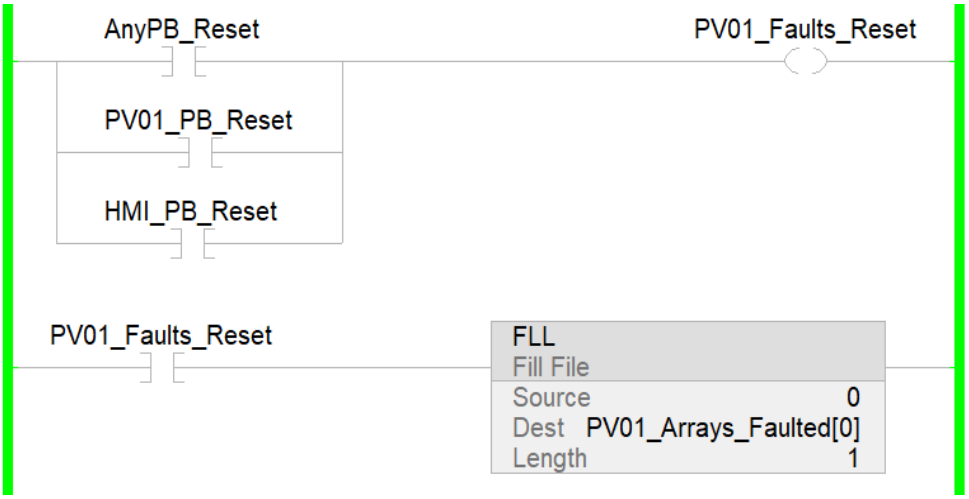


Fig. 6.3. Faults reset setup to clear non-retentive alarms.

If faults were to be acknowledged but not cleared, then that would be a sign of a real-time faulty device that must be resolved. Typically, whenever a field device is faulted, it would be reported to the respective PLC alarms code where a designated alarm message or tag, such as “_PV01_Arrays_Faulted[0].0”, would be enabled, as shown in Fig. 6.4. Overall, every detected fault would raise an alarm bit, i.e., triggering an HMI message to be displayed, as shown in Fig. 6.5.

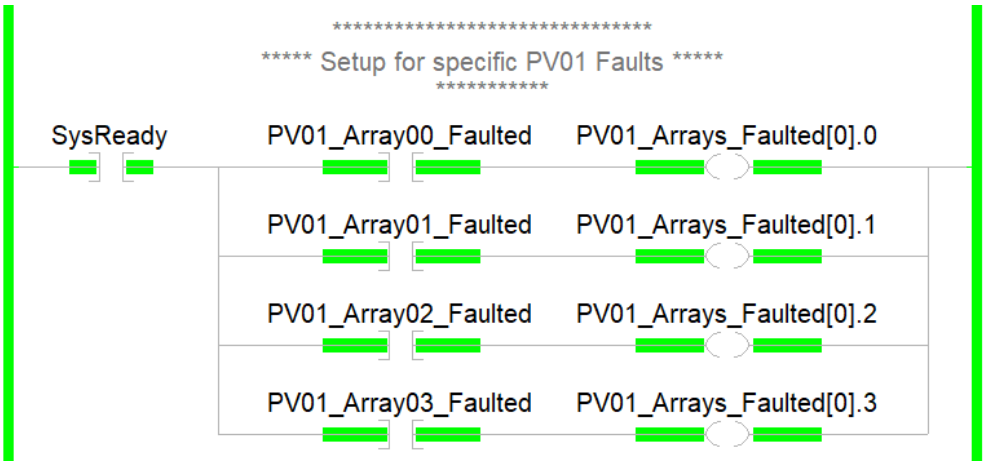


Fig. 6.4. Alarms setup for every fault related to PV01 group.

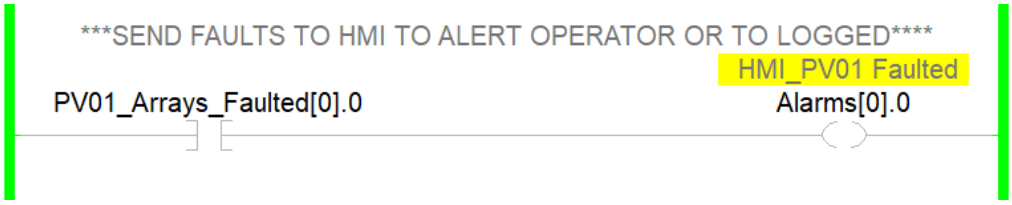


Fig. 6.5. Example of faults sent to HMI.

Another optional method that relies on File Bit Comparison instruction (FBC) was introduced, as shown in Fig. 6.6, based on the previous Alarms Ladder Logic Code. The implementation was done by using FBC instruction which makes detecting the occurrence of faults much easier.

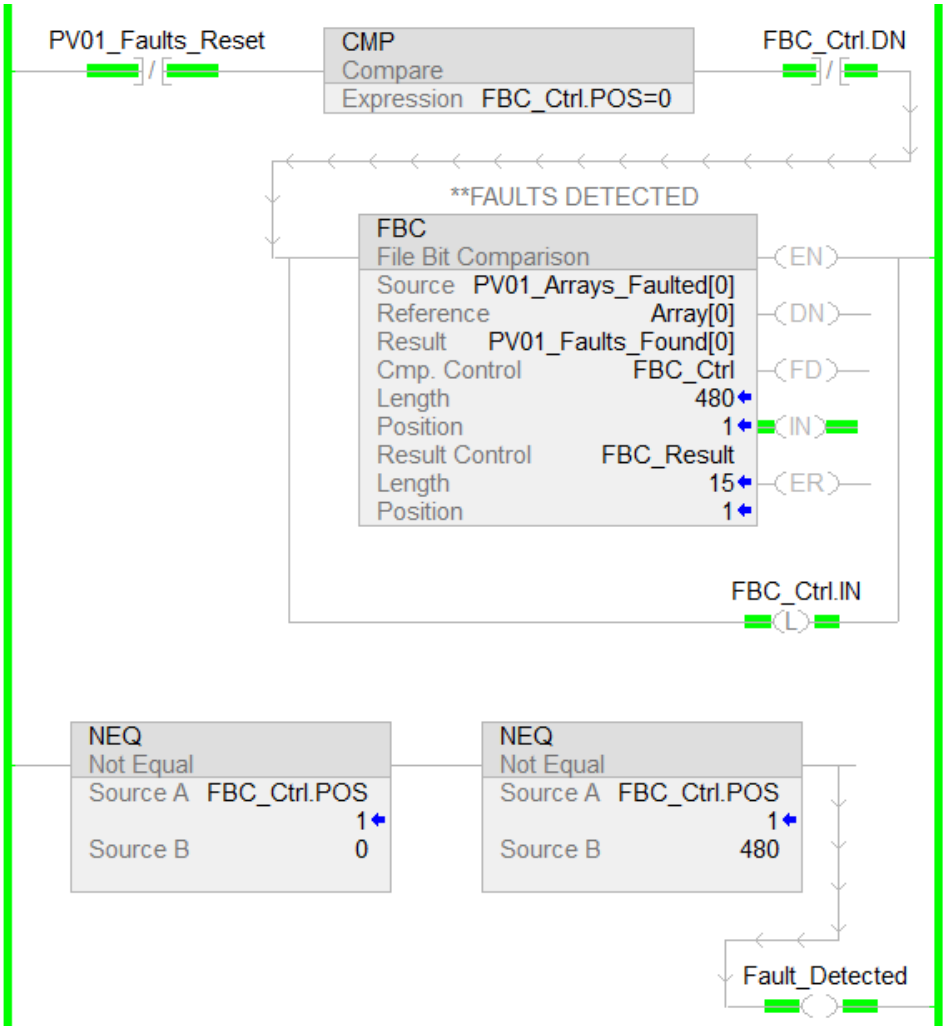


Fig. 6.6. Monitoring all PV01 related faults.

6.6 Case Study: Introducing Real-Time Attack Techniques to Exploit Vulnerabilities

A case study based on the previously described test bed was conducted to expose vulnerabilities within the PLC alarms code using a variety of exploits. The main target was to detect any or all suppressed or modified alarms within the PLC alarms code.

Adversaries might suppress, delay, or delete alarms raised by PLC code to evade detection to cause damage and disruption. Suppressed or inactive alarms prevent real-time notification messages which in turn could lead to undesired consequences. Bad code practices can also result in disruption to processes and cause damage.

The following is summary of the main vulnerabilities scenarios that might be exploited:

- All devices are functioning properly and reporting all abnormalities to the designated PLC but their alarms within the PLC code are deactivated, deleted, or tampered with. So, the PLC would obtain all status of devices and operations, but alarms would not be raised. That would be a sign of suppressed alarms within the PLC code.
- Alarms are delayed or skipped due to code modifications. That could reduce productivity, increase maintenance costs, or even risk the system.
- Alarms are faked to deceive operators, risking the safety of the devices.
- PLC alarms code is valid and legitimate, but the designated PLC is not receiving any faults from faulty devices. This could be due to physical problems, networks issues, or wrong configurations.

To mimic a malicious code attack where hackers would exploit vulnerabilities of a running PLC alarms code, real-time exploitation techniques were introduced into the test bed and applied without stopping any process or recompiling any routine. The attack models were stealthy and ran without being detected or noticed.
The following are four real-time attack models used to exploit PLC alarms code:

6.6.1 Skipping or Deleting Alarms Code

One of the techniques used in skipping some alarms or all alarms, was by embedding a “JMP” Jump instruction, named “LBL01” and “LBL” instructions, named “LBL01”. The CPU skipped scanning all alarms rungs (code lines) that are between “JMP” and “LBL” without energizing or raising any alarms bits within that range, see Fig. 6.7.
The logic would not distinguish whether skipping a code using “JMP” was legal or not. Therefore, operators would not be notified, and skipping the code would not be detected.

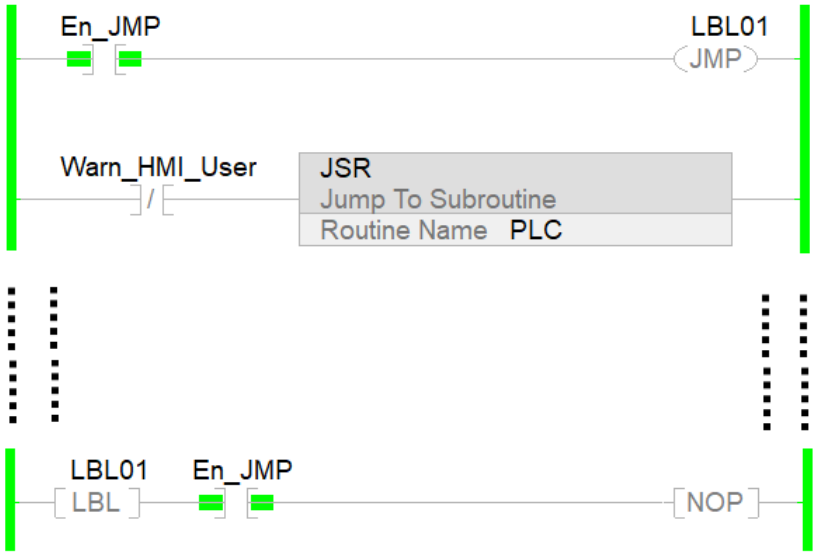


Fig. 6.7. The logic code between "JMP" and "LBL" is skipped.

6.6.2 Slowing Down the Scan Cycle of the PLC Alarms Code

Slowing down the scan time of the PLC alarms code means that the code was modified either by adding more code elements or suspicious loops. Some embedded finite loops would significantly slow down logic response for highly critical situations, or they might lead to a faulty PLC scenario. To slow down a scan time, a single "FOR" instruction was embedded to call another routine several times, looping for 500 times, see Fig. 6.8. Once the looping started, the scanning of the logic code was slowed down and then the PLC was faulted. When the number of loops was reduced to 300, the scanning of the logic was slower than normal but without faulting the PLC. The PLC was not able to detect or report to the operator any slower scanning of the PLC alarms code.

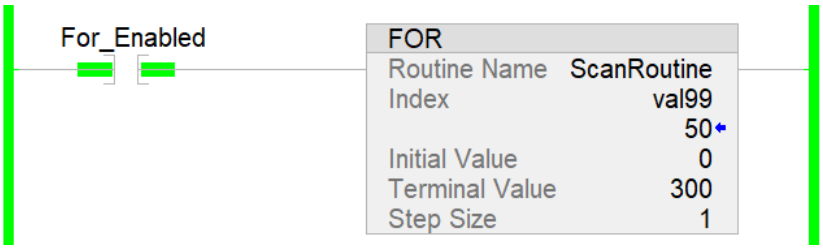


Fig. 6.8. Embedding a "FOR" to slow the scanning time of the code.

6.6.3 Deactivating or Suppressing Alarms

In this test, the alarms were suppressed when a single Boolean instruction, "Always_OFF", that is false (zero) was embedded before the energizing the alarms, as shown in Fig. 6.9.When

the “Always_OFF” bit was ANDed with any precondition instructions, like “SysReady”, the rung condition-out result was false i.e., all its output coils de-energized.

In other words, whenever the result of the rung-condition-in was false, the result of the rung-condition-out was false as well, i.e., suppressing all associated alarms, as shown in Fig. 6.10.

The PLC in this scenario would not detect any abnormal code behavior because all syntaxes and code elements are legal.

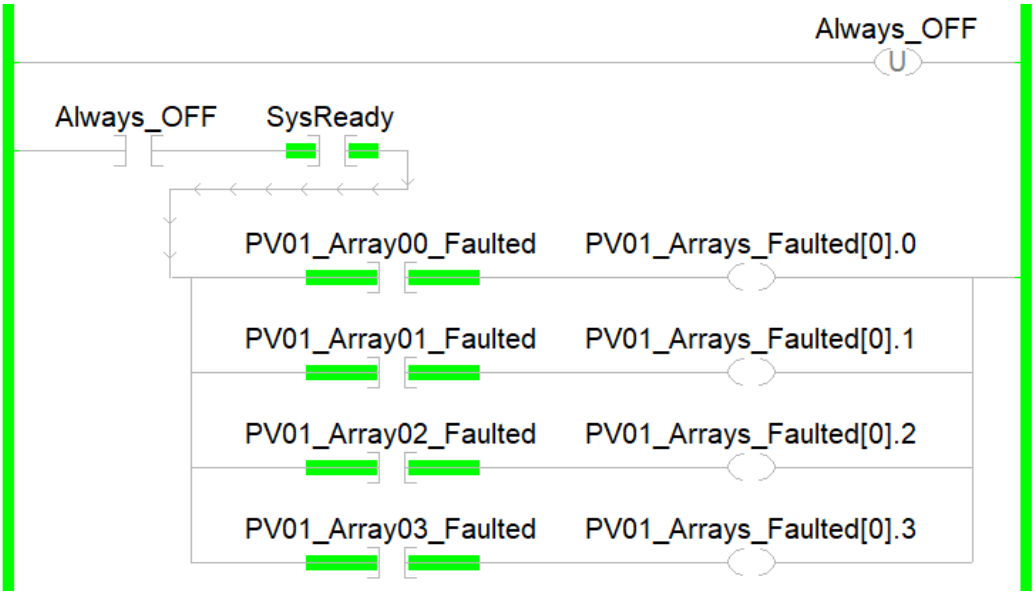


Fig. 6.9. Faults were deactivated when ANDed with a false bit.

Scope: MainProgram Show: All Tags

Name	Data Type	Force Mask	Value	Usage
PV01_Arrays_Faulted	DINT[32]	{...}	{...}	Local
PV01_Arrays_Faulted[0]	DINT		0	
PV01_Arrays_Faulted[1]	DINT		0	
PV01_Arrays_Faulted[2]	DINT		0	
PV01_Arrays_Faulted[3]	DINT		0	
PV01_Arrays_Faulted[4]	DINT		0	
PV01_Arrays_Faulted[5]	DINT		0	
PV01_Arrays_Faulted[6]	DINT		0	
PV01_Arrays_Faulted[7]	DINT		0	
PV01_Arrays_Faulted[8]	DINT		0	
PV01_Arrays_Faulted[9]	DINT		0	

Fig. 6.10. Suppressed or deactivated faults means values are set to zero.

6.6.4 Faking Alarms

The alarms in this scenario were faked rather than being skipped or deleted. Faking alarms would be difficult to detect since they would deceive the operators by showing faked updated alarms to hide long time suppressed critical messages. A running system without any faults for a long period of time would be unusual in ICS.

To counter such scenario, a flash bit called “Flash” was created based on code of two timers, as shown in Fig. 6.11. The “Flash” bit acted like a fluctuating false that occurred every predefined duration (toggled ON/OFF every few seconds). The “Flash” bit was placed before energizing alarms to fool the operators that alarms were active and occurring occasionally in a normal way, see Fig. 6.12.

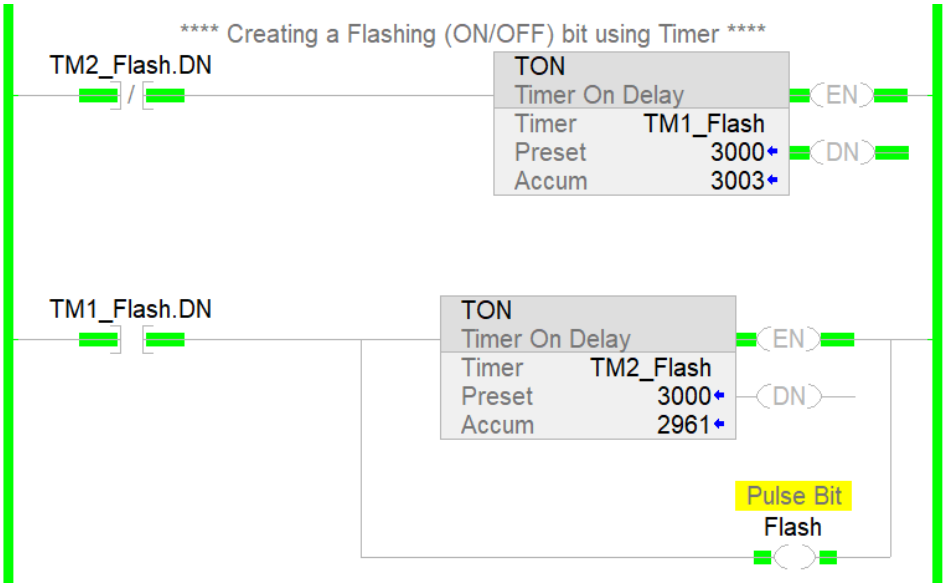


Fig. 6.11. Create a flashing bit using timer on delay instructions.

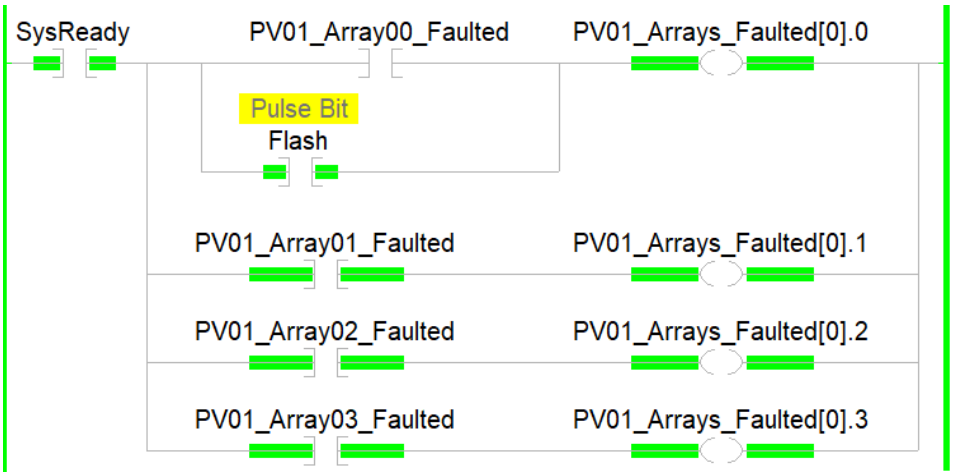


Fig. 6.12. Misleading diagnostics by adding a flashing bit.

6.7 Countermeasures

For the attacks deployed and implemented in the previous case study, countermeasure solutions were introduced and tested to prevent and detect each attack using real-time ladder logic code. The countermeasure solutions are listed as the following:

- A Scan Time Countermeasure Against Skipped or Deleted Code Attack.
- A Scan Time Countermeasure Against Slower Code Scan Cycle Attack.
- A Heartbeat Bit Countermeasure Against Inactive or Suppressed Alarms Attack.
- A Physical Plausibility Check to Detect Faked Alarms Attack.

The details of each counter-attack model are explained hereafter.

6.7.1 Method 1: A Scan Time Countermeasure Against Skipped or Deleted Code Attack

6.7.1.1 Scan Time Overview

The scan time of a code is the amount of time the CPU would take to scan the code within a scan cycle. Since the scan time value would be affected by any code modifications, storing a scan time value (or average of several scan time values) and using it as a reference point would be a reliable and effective approach. To do that a ladder logic code was developed and embedded within the PLC alarms code to capture the scan time value of the running PLC alarms code under normal conditions.

The average of several scan time samples was computed and stored as a reference point. It was proven that whenever portion of the code was skipped, the scan time was faster than the average, and vice versa. By using comparison instructions further ladder logic scanning would be stopped and staff would be warned whenever a scan time value is not within the range of the average one.

6.7.1.2 Developing Scan Time Code

Capturing the scan time of the PLC alarms code or any other specific routine is not a direct built-in feature in Rockwell PLCs. Built-in timer instructions were not applicable when capturing microseconds intervals.

To circumvent that, a “WallClockTime” object of a Get System Value (GSV) instruction was used, see Fig. 6.13.

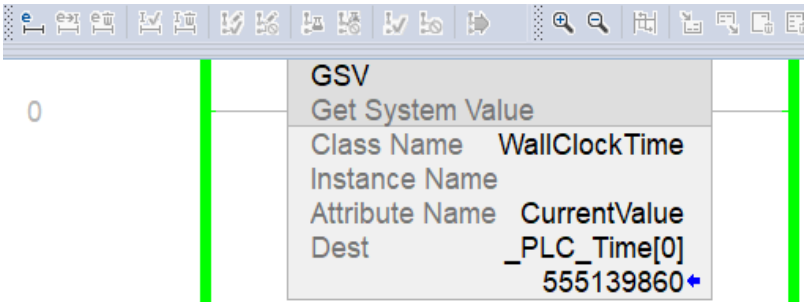


Fig. 6.13. Capturing the starting point of a scan time.

The "WallClockTime" is a continuous PLC built-in global timer that provides timestamps scheduled by the controller. A "WallClockTime" object was added to the beginning of PLC alarms code to timestamp the starting point of every initial scanning of the code. Then the captured timestamp value was stored into "_PLC_Time[0]" array, as shown in Fig. 6.13. Another "WallClockTime" object was added at the end of the code to capture its timestamp. The captured time value was stored into "_PLC_TimeNext[0]". By subtracting the values of the two arrays, "_PLC_Time[0]" and "_PLC_TimeNext[0]", we were able to get the time that the CPU took to scan the PLC alarms code within one cycle. The scan time captured was stored into "PLC_TimeResult[0]", as shown in Fig. 6.14. As the CPU keeps scanning the code, a new scan time value was captured with some marginal variance.

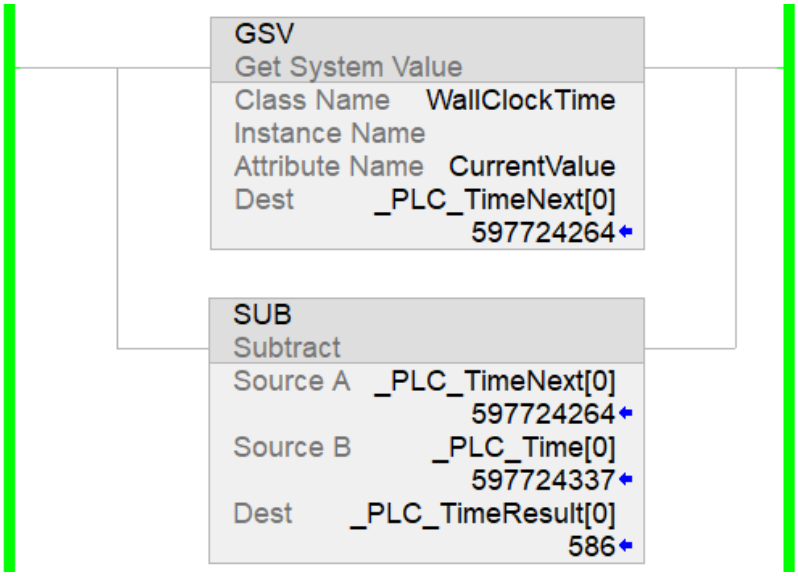


Fig. 6.14. Capturing the scan time by the end of the routine.

To get more accurate result with less discrepancies (due to clock latency), 10 scan time samples were captured and stored within FIFO arrays, from 0 to 9, as shown in Fig. 6.15 and Fig. 6.16. The average of those stored values of each scan time was calculated using “CPT” (Compute) instruction, and the result was stored in “ScanAverage” array, as shown in Fig. 6.17. The code is scalable and could get more enhanced if more reliable and accurate value of the scan time. For instance, If the code is set to capture more than 10 scan time, the average would be more accurate.

Scope: MainProgram Show: All Tags

Name	Data Type	Force Mask	Value	Usage
FIFO_Array[0]	DINT		599	
FIFO_Array[1]	DINT		580	
FIFO_Array[2]	DINT		572	
FIFO_Array[3]	DINT		595	
FIFO_Array[4]	DINT		598	
FIFO_Array[5]	DINT		597	
FIFO_Array[6]	DINT		590	
FIFO_Array[7]	DINT		601	
FIFO_Array[8]	DINT		595	
FIFO_Array[9]	DINT		598	

Fig. 6.15. Capturing several values of the scan time.

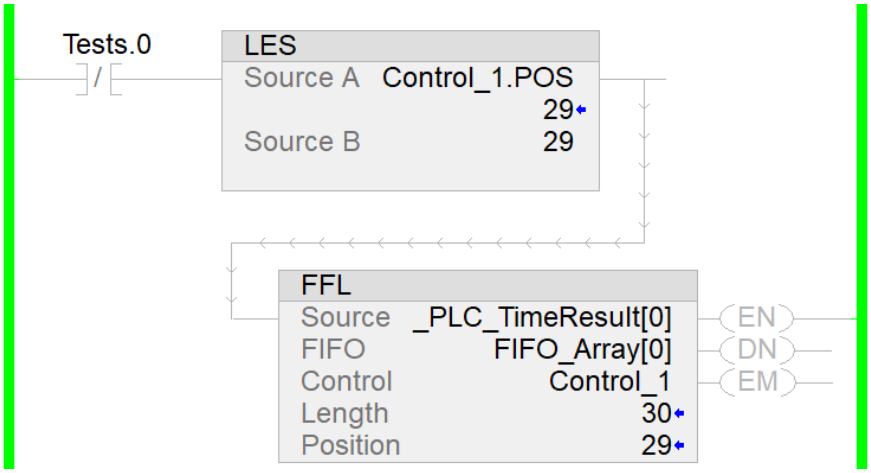


Fig. 6.16. Using a “FIFO" to capture several values of the scan time.

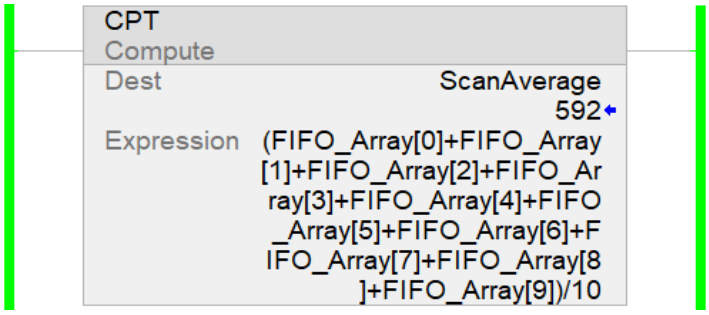


Fig. 6.17. Calculating the average time of ten scan time values.

6.7.1.3 Using Scan Time to Detect Skipped or Deleted Alarms Code

This method was introduced and tested to detect skipped or deleted alarms by checking and validating every scan time per PLC cycle. When a newly captured scan time was less than that of the reference scan average value, then the PLC stopped further code scanning (stop further code execution) and warned operators. After skipping all rungs related to alarms, a new scan time value was captured, "_PLC_TimeResult[0]". As expected, the scan time was tremendously less than that of the original average value, "ScanAverage[0]" (23 microseconds compared to about 592 microseconds). Several new scan values of the modified code were, as shown in Fig. 6.18. Then the average of those values was computed and stored in "ScanCurrentAvg[0]" to get a more accurate and stable value, see Fig. 6.19. By comparing the newly calculated average, "ScanCurrentAvg[0]", to that of the reference a discrepancy was detected and a warning message, "Warn_HMI_User" bit, was raised to alert operators, as shown in Fig. 6.20. The result of the comparison was also used as a precondition to stop further logic scanning, see Fig. 6.20.

Name	Data Type	Force Mask	Value	Usage
FIFO	DINT[29]	{...}	{...}	Local
FIFO[0]	DINT		23	
FIFO[1]	DINT		22	
FIFO[2]	DINT		21	
FIFO[3]	DINT		22	
FIFO[4]	DINT		22	
FIFO[5]	DINT		23	
FIFO[6]	DINT		22	
FIFO[7]	DINT		23	
FIFO[8]	DINT		22	
FIFO[9]	DINT		36	

Fig. 6.18. Capturing scan time values after logic was skipped.

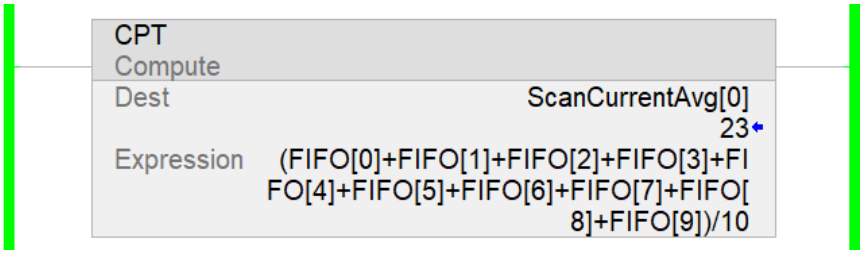


Fig. 6.19. The new scan time average of the modified code.

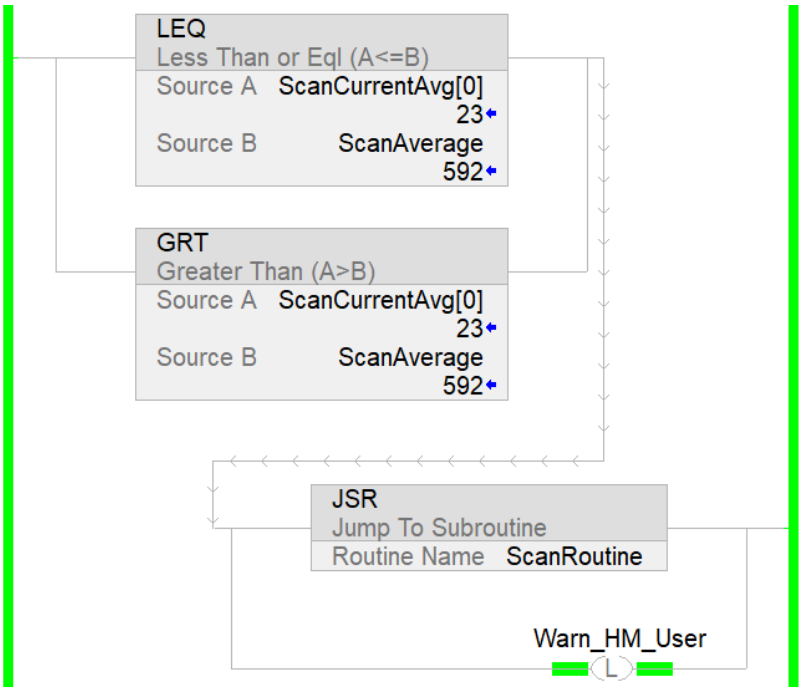


Fig. 6.20. Preventing further logic scanning.

6.7.2 Method 2: A Scan Time Countermeasure Against Slower Code Scan Cycle Attack

Since embedding more code or finite loops was not detected by the PLC, as demonstrated in the earlier attack model, a real-time countermeasure technique based on a scan time implementation was introduced. The PLC cycle would take more time to finish scanning the embedded code, i.e., increasing the scan time duration. When “For” instruction (with 300 steps) was embedded within the PLC alarms code, the scan time trap was able to detect that attack based on its scan time duration. Fig. 6.21 shows that the new scan value was above 230000 microseconds, significantly greater than that of the normal average value which was around 592 microseconds.

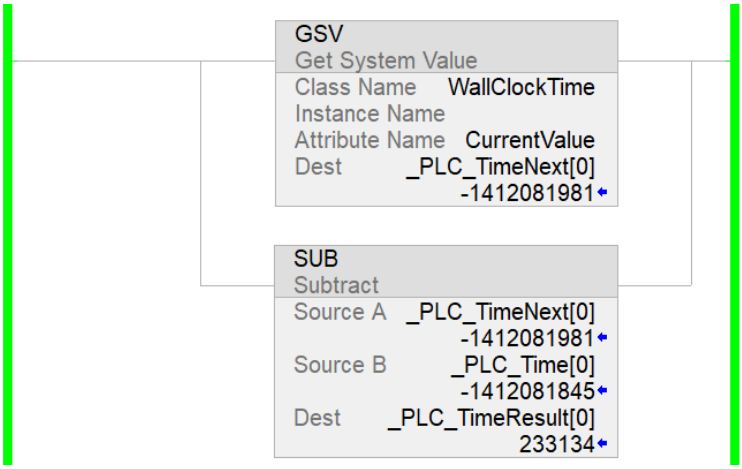


Fig. 6.21 The scan time was significantly increased after inserting "For" loop instruction.

For reliability, several scan time values were captured and stored, as shown in Fig. 6.22. Then the average of those was computed and stored in the array named "ScanCurrentAvg[0]", as shown in Fig. 6.23.

Scope: MainProgram Show: All Tags

Name	Data Type	Force Mask	Value	Usage
FIFO	DINT[29]	{...}	{...}	Local
FIFO[0]	DINT		233269	
FIFO[1]	DINT		234557	
FIFO[2]	DINT		233383	
FIFO[3]	DINT		236461	
FIFO[4]	DINT		232372	
FIFO[5]	DINT		235972	
FIFO[6]	DINT		233190	
FIFO[7]	DINT		236242	
FIFO[8]	DINT		232983	
FIFO[9]	DINT		236064	

Fig. 6.22. Several scans time values were captured and stored.

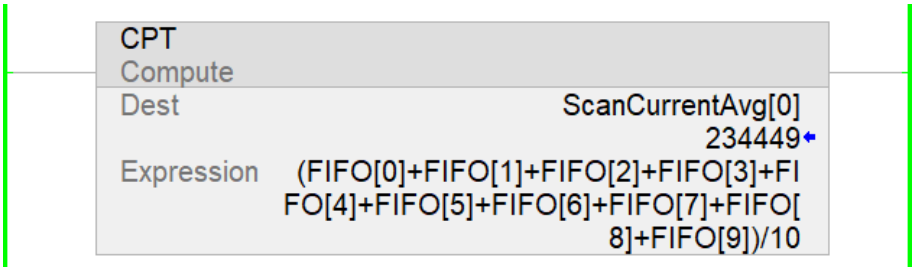


Fig. 6.23. The average scan time was computed.

Similar to the previous comparison code setup and techniques introduced in Section 6.7.1, once an abnormal scan time was detected, further logic scanning would not be allowed, and a warning message was sent to operators, as shown in Fig. 6.24 and Fig. 6.25.

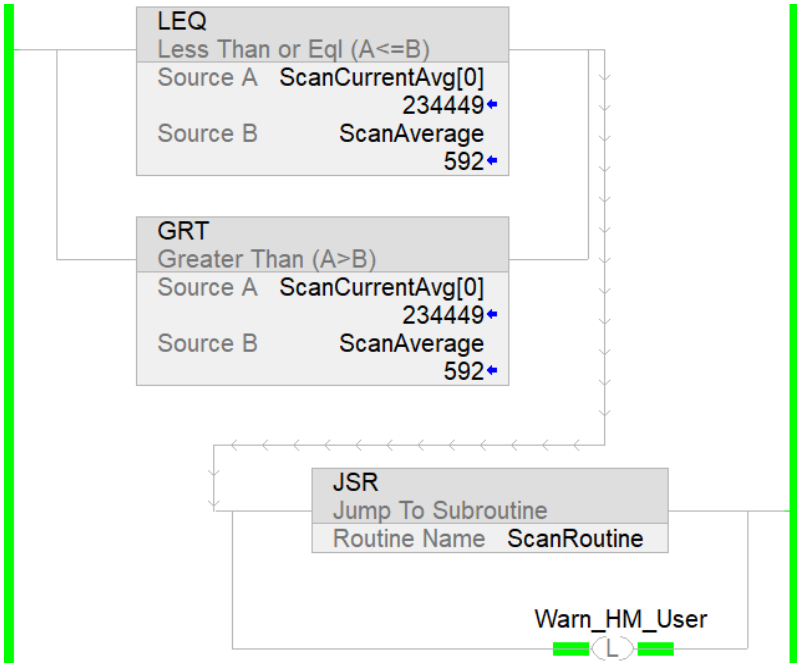


Fig. 6.24. Preventing further logic scanning and alerting staff.

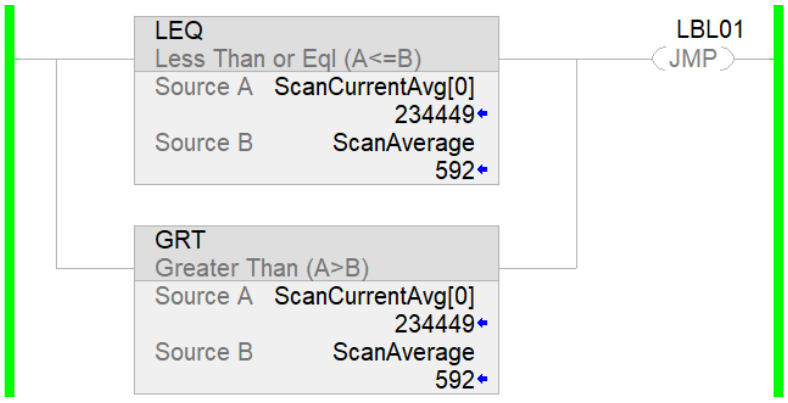


Fig. 6.25. Another way to prevent further logic scanning by using "JMP" instruction to skip certain code.

6.7.3 Method 3: A Heartbeat Bit Countermeasure Against Inactive or Suppressed Alarms Attack

A heartbeat code was developed to verify the validity and the real-time status of alarms. The alarms would be considered invalid if all faults were idle (all ON or all OFF) within a predefined duration of time. The resultant of the faults of each group was tied to a Boolean bit, such as "GroupAlarms05".

When "GroupAlarms05" had its status idle for 3600000 milliseconds, the heartbeat bit, "Heartbeat_OK", was disabled, as shown in Fig. 6.26 and Fig. 6.27. A dead or disabled heartbeat bit was then used to stop further logic scanning and to warn operators.

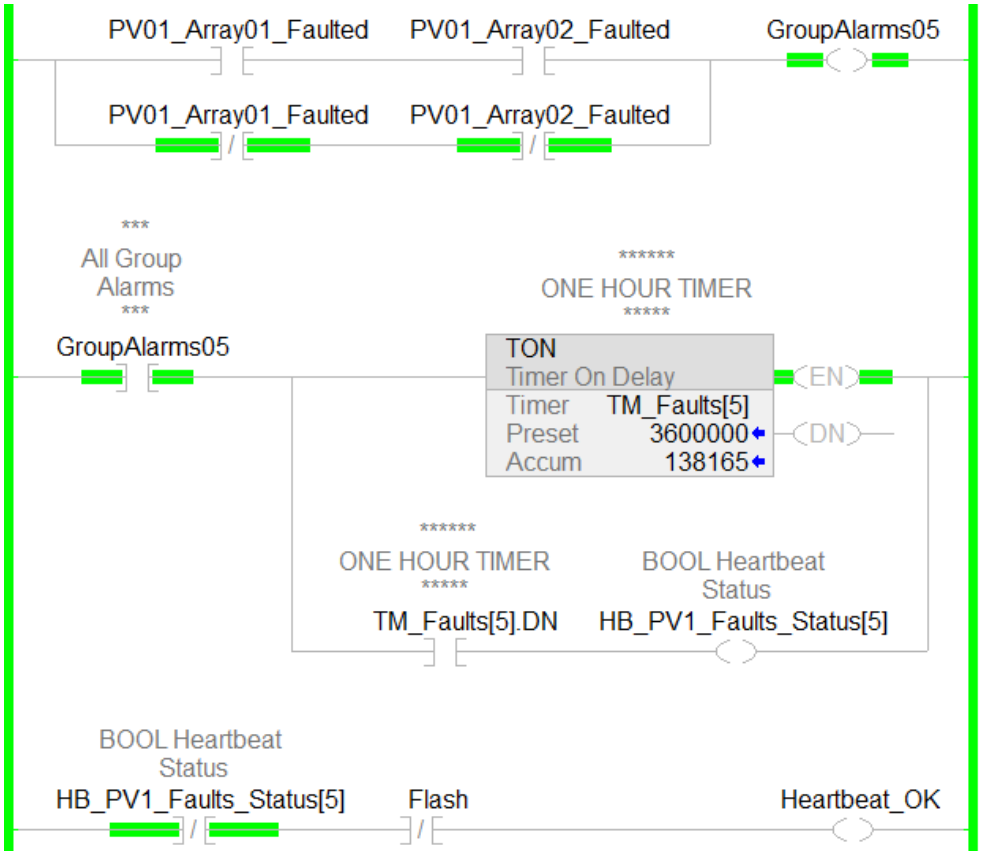


Fig. 6.26. Heartbeat setup based on alarms within a group.

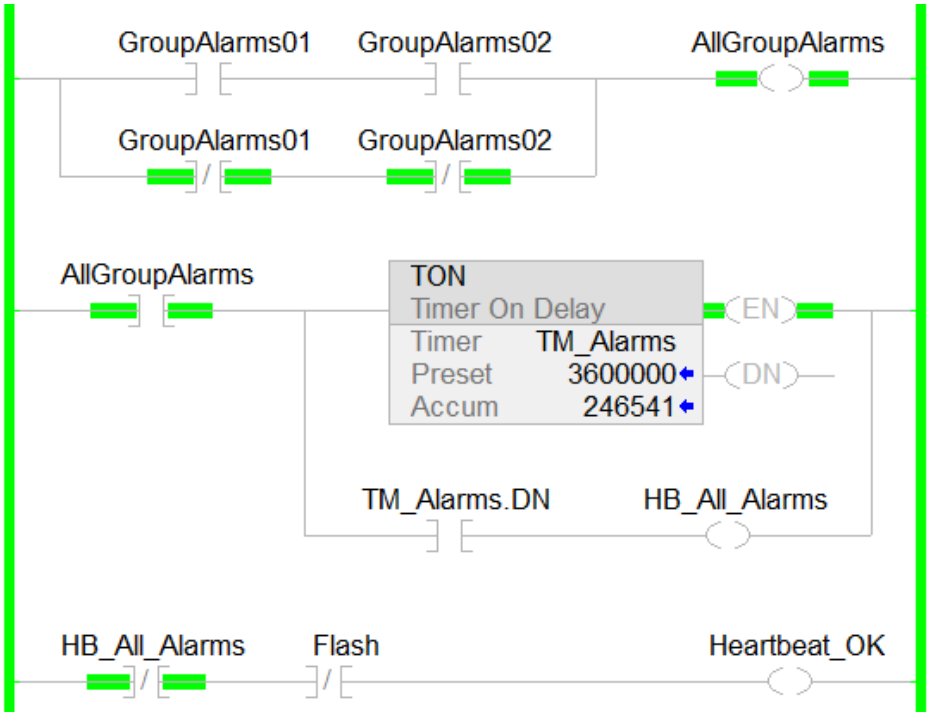


Fig. 6.27. Heartbeat setup based on groups of alarms.

6.7.4 Method 4: A Physical Plausibility Check to Detect Faked Alarms Attack

To detect faked alarms or unnoticed malfunctioning devices, a physical verification of all critical devices and instruments involved must be conducted. A physical plausibility check should be performed to validate whether the alarms are matching real-time status and behavior of field devices.

The following are some of the techniques that should be followed to properly validate the process of verification:

6.7.4.1 Validate a Device Operation Based on a Predefined Duration

Validation of the operation of devices and instruments must be based on predefined duration. The operation time of a device should be compared to a predefined or to a previously computed average while physically operating or checking the device. It is not enough to check, for instance, that a locking pin is extending without comparing its operation duration to a specific predefined time. If the behavior of an instrument or a device is taking more than its predefined duration, then the operators should be alerted, and any associated alarms should be raised.

6.7.4.2 Validate PLC Outputs

There should be a handshaking validation between a PLC and the controlled devices (such as actuators) through their statuses. A PLC should validate that an output is sent to the proper designated device by getting the proper expected feedback before running further code. In the absence of a device feedback or proper acknowledgment, associated alarms must be raised. For instance, when a PLC sends a program number to a robot, the robot must send back a matching program number. If both program numbers are not matching, then the PLC should alert the operators and stop executing the rest of related code.

6.7.4.3 Validate PLC Inputs

Validate the inputs of sensors and other monitored devices based on their proper physical operations. For instance, a clamp should indicate an exclusive status which is either “OPEN” or “CLOSED”, neither in between nor none. If the hardware indicates that both are active “TRUE” or both are “FALSE”, an alarm should be reported.

6.7.4.4 Validate Coils or Commands

Validate there is no conflict among PLC commands (such as energized coils or outputs) that work in pairs. A conveyor, for instance, should not get a request to move forward and backward at the same time. A pin, for instance, should not be commanded to extend and retract at the same time. Such invalid conflict should be monitored, logged, and prevented.

6.8 Results and Discussion

The main contribution of this chapter are the solutions that can be applied to mitigate real attacks on PLC alarms. It was shown how the ladder logic code of alarms could be compromised even though all other field devices, including HMIs, were properly functioning, leading to serious damage. Four attack models within the PLC alarms code were introduced and exploited designated PLC alarms code. Each one of the attacks relied on modifying the ladder logic of the PLC alarms code by embedding instructions that adversely affected the behavior of the alarms. Real-time countermeasures were deployed and successfully detected and prevented those introduced attacks.

To analyze and verify our previously calculated scan time values, real-time trends that captured and analyzed real-time scan time values were introduced and demonstrated using an experimental test bed. Each trend captured more than 50 real-time values of the PLC alarms code scan time, “_PLC_TimeNext[0]”. The values were captured and analyzed during a 2

millisecond interval. Under normal condition, attacks free scenarios, the values of the captured scan time in our test bed were between 469 and 606 microseconds, as shown in Fig. 6.28. All those captured values were close to the average that was calculated previously, about 592 microseconds, see Fig. 6.17.

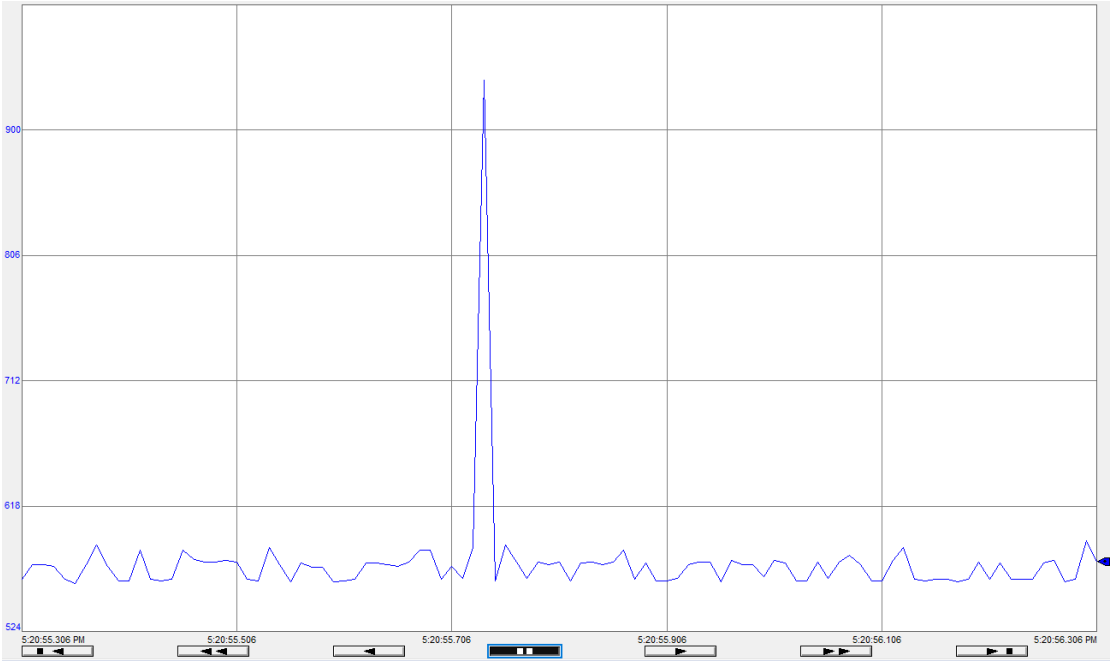


Fig. 6.28. Capturing several scan-time values under normal conditions during 1 second.

Similarly, when the skipping code attack was introduced the scan time value was significantly reduced. The trend diagram provided in Fig. 6.29 shows that the real-time values of the scan time were within the range of 20 and 23 microseconds. That was close to the average scan computed earlier, see Fig. 6.17.

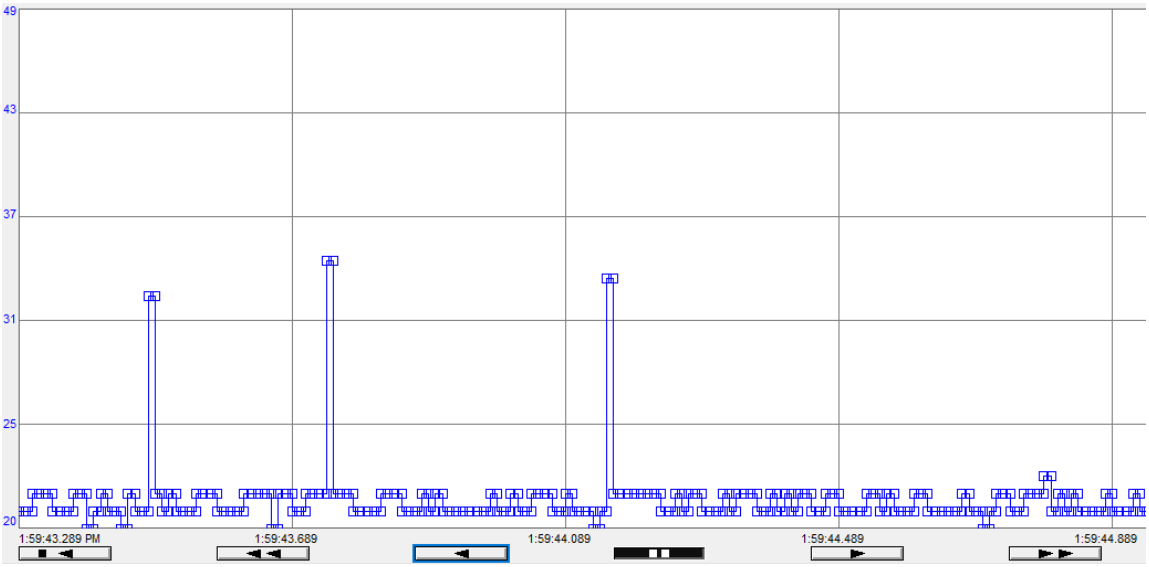


Fig. 6.29. Scan-time values of the compromised skipped code within 2 seconds.

Regarding the slow scan time attack, the captured real-time values of the scan time was around 2300 microseconds, as shown in Fig. 6.30. That was significantly more than the normal average scan and close to the abnormal average that was computed earlier, see Fig. 6.23. Similar measurements can be extracted from any PLC and analyzed to detect PLC alarms code abnormalities.

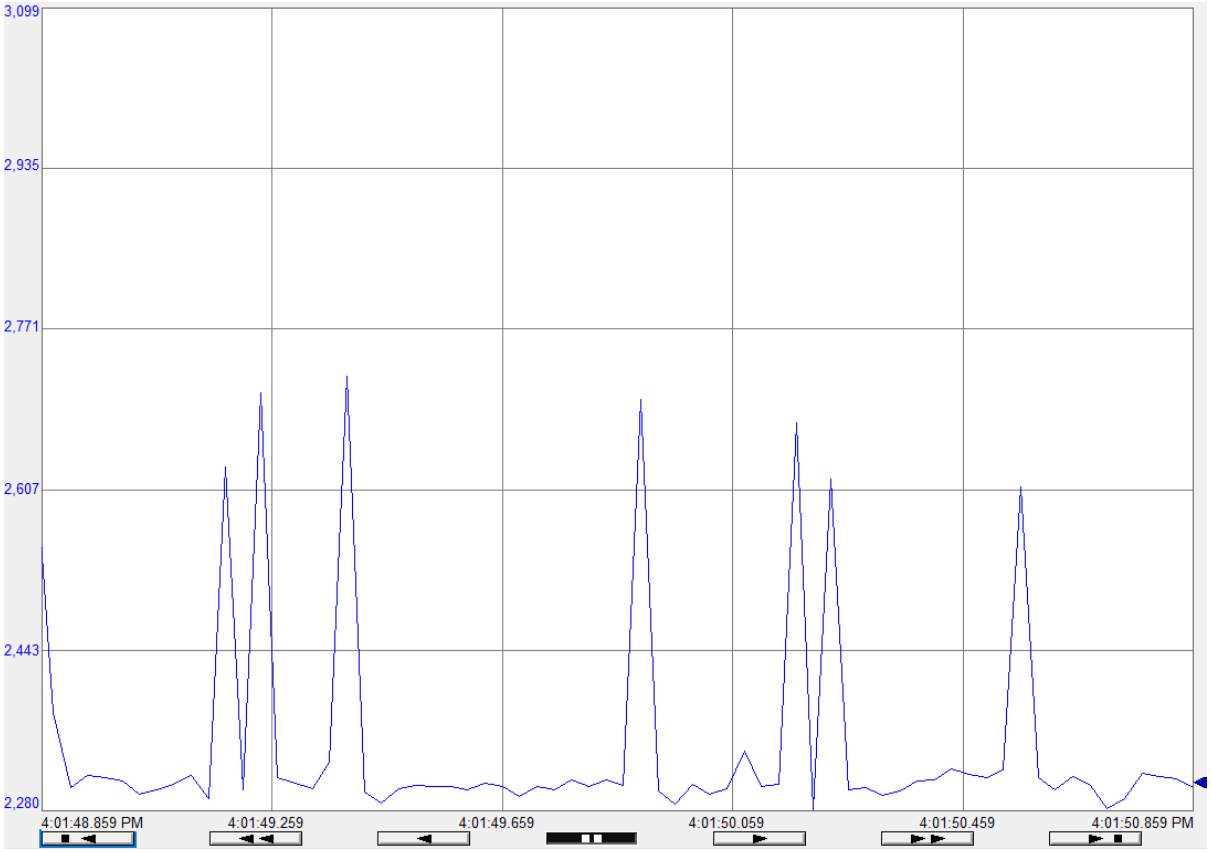


Fig. 6.30. Capturing several scan-time values when slow code attack was introduced.

6.9 General PLC Alarms Code Guidelines

6.9.1 PLC Alarms Code Best Practices

For safer systems and better troubleshooting and recovering, it is recommended that PLC alarms code follow these code practices:

- Alarms code must reside in dedicated and separated PLC routines to make it easier to authenticate, analyze, and troubleshoot.
- Raised alarms bits by PLC code must stay set (latched) once they are triggered and only cleared (reset) if faults are resolved and acknowledged by operators.
- Raised alarms must be logged into arrays within PLCs and sent instantly to designated HMIs where they would be clearly displayed and logged.
- Validate alarms raised within PLC alarms code with those displayed on HMIs. In some scenarios, a PLC could be triggering the proper messages on its side, but an HMI messaged could be misconfigured.
- Each alarm configured within the PLC alarms code must have a unique ID or identifier, so it can be properly initialized, set, and tracked. The identifier is a unique fault code that must be considered when implementing the alarms within the PLC code. Each alarm must be clearly displayed on any associated HMIs with its ID and proper descriptive message.
- Alarms within PLC alarms code must have standards that define naming conventions, tags, and data types.
- Alarms could be more useful if they are time stamped. That would help in knowing the initial root cause of the problem.
- PLC alarms code should be password protected.
- A comparison of the PLC alarms code must be performed with a previously validated file. The comparison should be performed every time the checksum of the PLC is changed.
- Raising the same alarm (output) in more than one place within the PLC code should be avoided, especially those are OTE (Output Energize) based.
- Empty branches must not be allowed.
- Forcing tag values should be avoided since forced bits could be hard to track.
- Avoid any racing scenarios within the PLC alarms code. Racing occurs when two routines or set of operands of logic are racing against each other.

6.9.2 Other Suggested Techniques

6.9.2.1 A Mirror PLC for Alarms

In some critical scenarios, adding a duplicate or a mirror PLC would be an effective use of redundancy. The redundant PLC can be used as an external backup for alarms, safety, and other critical routines. The access to a mirror PLC should be very limited and restrained. A mirror PLC can be used also as a real-time reference in checking if there are any code modifications within the primary PLC. If any discrepancies are found, then the operators must be alerted. Also, having a backup PLC would make compromising the code harder on hackers.

6.9.2.2 Effective Usage of HMIs and Logs

It would be more effective if PLC alarms, abnormal scenarios, scan time values, PLC cycles, etc. are sent and displayed on HMIs. Those charts would and logged information would be used in trends and statistical charts.

6.9.2.3 Alarms Test Mode

Enhance PLC alarms code to handle an automatic Alarms Test Mode (ATM) where operators can run it to verify all alarms. When ATM is enabled, the PLC would raise every alarm in a sequential manner and display all of them on the designated HMI. If an alarm is not raised on the HMI, then operators should be alerted.

6.10 Conclusion

The real-time test bed introduced in this chapter exposed some vulnerabilities of the ladder logic code responsible for raising alarm messages, i.e., PLC alarms code.

Four real-time attack techniques were introduced and applied within the PLC alarms code while the PLC was in “RUN” mode. None of the attacks stopped or interrupted the PLC, other ladder logic code, or got detected. All the attacks could be used to cause serious damage to a PLC based ICS. The attacks were based on different ladder logic scenarios and techniques.

It was proven that the PLC was defenseless in detecting or preventing any of those embedded attacks. Several effective countermeasure solutions against the described attacks were introduced: scan time code, heartbeat code, and verification guidelines of alarms and designated physical devices. One of the challenges faced in this experiment was in capturing the scan time of a fast cycle. The built-in timer instruction provided by Rockwell software were not capable of capturing time intervals within microseconds. To bypass that problem a “WallClockTime” object was used.

Chapter 7 Equipping PLC Code with Self-Detection Mechanisms to Detect and Prevent Vulnerabilities and Threats

7.1 Introduction

As adversaries are finding more ways to compromise the code of PLCs, relying on external perimeters and external counterattacks to protect PLCs is not any more sufficient. Equipping PLCs with self-defence and self-detection mechanisms is becoming more critical and essential. This chapter focuses on enabling PLCs to expose, detect, and prevent vulnerabilities within the code they are running. The chapter provides novel counterattack code techniques that enhances the security of ladder logic code to make it self-aware of code exploitations and abnormalities based on code behaviors. A real-time ladder logic code test bed that addresses six different real-time attack models were implemented and applied to a running PLC program to prove their typical defenseless mechanisms. The attacks were stealthy, and they effectively achieved their malicious goals in modifying the PLC code without being detected. With the deployment of our code countermeasures, the PLC detected and prevented those attack models including the detection of code abnormalities, modifications in the overhead time slice, and any manipulation or deterioration of certain field devices. The novel countermeasures against the introduced attack models were implemented, mainly, based on a novel Scan Time Code (STC) setup that monitors the runtime execution of a particular code per scan cycle. The introduced countermeasures can be implemented to any similar PLCs to enhance its defense mechanism and protect ICS.

7.2 Related work

Very little work, if not none, has been done on vulnerabilities of PLC code and how to protect within the PLC scanning time. Researchers are more focused on third party solutions that are related to securing networks or association PC based devices and software.

[78] [79] show that bad code practice and improper or poor ladder logic design would increase the risk of vulnerabilities within PLC code. Inexperienced programmers are more concerned about the functionality of automated devices than vulnerabilities or poor code practices.

[79] shows several bad code practices that could be exploited and risk the safety of PLC-BS.

[81] provided ladder logic bombs (LLBs) where a ladder logic code was injected to execute malicious PLC code that manipulated receive status of field devices. The code could also be

utilized to initialize DoS attacks. [80] presented a PLC-Blaster worm that targeted Siemens PLCs, SIMATIC S7-1200. The worm was hosted on Siemens PLCs, modified PLC code, manipulated outputs, and scanned associated networks to target other PLCs. [81] presented attack model that was able to access a Siemens PLC by using a SNMP Scanner and a SOCKS Proxy. The attack was able to modify the associated PLC program. [82] presented an external non-PLC program to verify PLC code integrity before loading it to the associated PLC. The research work introduced examples of bad code practice that could be exploited by adversaries. Racing conditions, duplicate or missing outputs, unused tags were provided as examples of bad code practice. [83] presented CLIK, a PLC logic attack. The attack consisted of bypassing security measures of a PLC, accessing and getting a binary copy of its code, decompiling the stolen binary code, and injecting a malicious code into the stolen one. Once the malicious code was injected, it was transferred and downloaded, as binary code, to the exploited PLC. [84] presented network attacks that manipulated memory values of PLC's inputs. [85] presented malicious attacks on PLCs, "Denial of Engineering Operations". The attacks were developed to access PLC code, retrieve a copy of it, modify it, recompile it, and downloaded back to the designated PLC. The process was accomplished without being detected by associated code designer software. As a countermeasure solution to detect malicious manipulations, a ladder logic decompiler termed 'Laddis' was introduced. Attacks in [86] were able to manipulate inputs and outputs of Siemen S7 PLC and remotely stopping it. [87] [88] introduced attacks that targeted Siemens S7 PLC code by retrieving the running ladder log code, decompiling it, manipulating it and transferring it back to the PLC in bytecode format. [89] presented attacks that targeted Siemens S7-300 PLCs and associated TIA data. The authors injected interrupt code to attack the designated PLC Organization Block. [90] presented 'Shade', a deep packet inspection (DPI) technique. Shade employs certain algorithm techniques to monitor PLC code and detect any evasion injected into designated ICS networks. [91] presented a 'SABOT' tool that instantiated malicious payloads on Siemens PLCs. The payloads could be arbitrary manipulations of the targeted PLC code. [92] presented "Harvey" rootkit that exploits PLCs' firmware vulnerabilities to modify associated logic instructions. The malware was capable of monitoring and modifying PLC inputs received from field devices. Another rootkit to manipulate PLC's I/O values was introduced in [93]. In [94] a NuSMV model to verify the integrity of PLCs' function blocks, FB, code was introduced. [95] presented systematized knowledge of PLCs' threats, vulnerabilities, and recommended countermeasures. PLCspecif was introduced in [96] to validate and improve PLC programs. [97] demonstrated a methodology based on 'Cone of Influence' and NuSMV models to verify PLC programs. In

[98] solutions were provided to verify safety program. [99] addressed a method to verify the integrity of PLCs' structured language programs. [100] [101] [102] introduced methods and techniques to verify and validate running PLC programs. [103] introduced 'Arcade', a statistical tool, to evaluate different PLC programs. [104] introduced HyPLC, a compilation and verification tool to provide guarantee proper correctness of the PLC programs.

Unlocked or unprotected PLCs create another vulnerability concern. Having an access point to a PLC allows user to upload any malicious code to the PLC, manipulate the current running one, or even upload new firmware. PLCs typically do not check whether the uploaded code is from a verified trusted source or not. Even with password protection, PLCs do not have enough defense mechanisms to stop attackers from retrieving or stop passing passwords [85] [91] [92]. [9] [11] [108] presented Stuxnet malware that severely attacked PLC-BS. Stuxnet was able to maliciously spy, attack, compromise, and even exploit other field devices to initialize attacks on other systems. By exploiting Siemens software and associated programs (HMIs - WinCC, PLC codes -Siemens Simatic Step7, PCs - Windows, etc.) and faking values, Stuxnet severely affected crucial field devices - taking advantage of multiple Windows zero days vulnerabilities [108]. Even though it was a malware that was specifically customized to target certain Siemens SCADA systems and programs, Stuxnet is just a typical threat that PLC-BS might face. In addition to Stuxnet attack, so many other attacks were carried out by other threats such as Flame, Guass, Duqu, Wiper, and BlackEnergy malware [39] [1]. There are more than 50 new Stuxnet-like attacks on SCADA threats discovered [109] [9] [10] [110]. Since Stuxnet's appearance, PLC-BS have attracted the attention of the hacker crowd.

General alarms analysis and management were discussed in [144] [146] [147] [148] [149] [150] [151] [152] [153] [154]. General recommendations and surveys were provided in [163] [164]. All the above research papers never mentioned or used the PLC scan time to detect coder abnormalities. As a matter of fact, none of the previously introduced papers developed ladder logic programs to detect and stop code abnormalities within the respective PLC.

7.3 PLC Scan

7.3.1.1 PLC Scan Overview

When a PLC is in "RUN" mode, the controller continuously scans all the listed routines of a program, unless it is not assigned by users to do so. It monitors all related physical and internal address inputs and energizes or deenergizes outputs according to the related instructions and

sequence of operation, see Fig. 7.1. For PLCs, such as Rockwell ones, a controller continuously scans each routine one at a time in the order they listed under “MainProgram”.

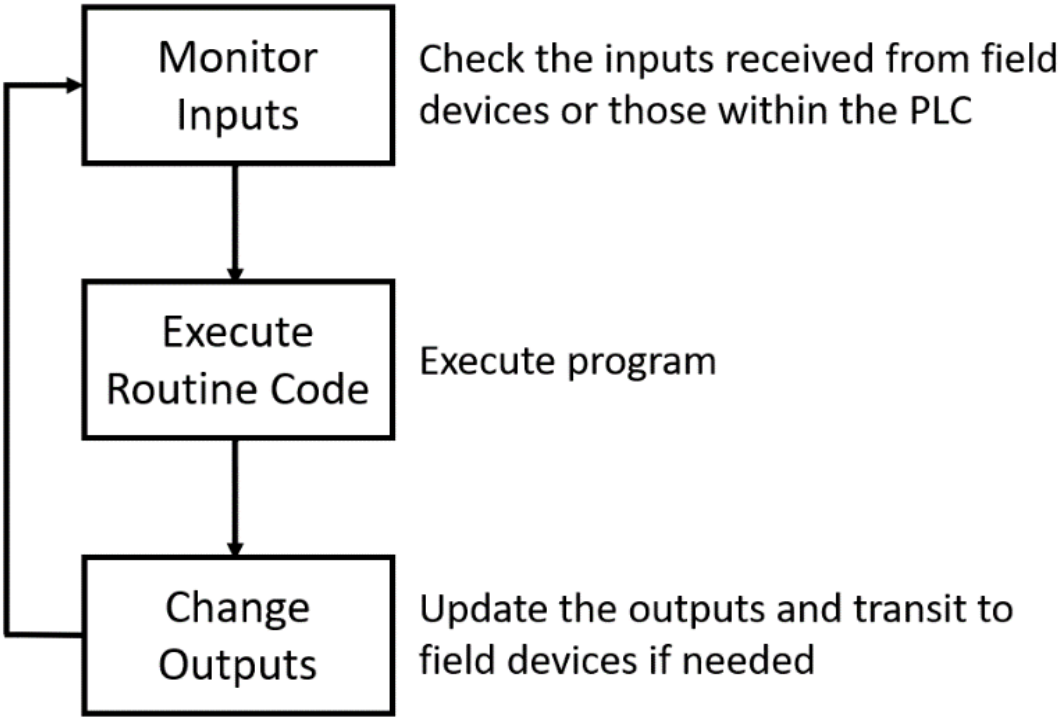


Fig. 7.1. Scan cycle of a code.

It starts scanning or executing each rung within a ladder logic routine starting from top to bottom left after scanning each rung from right to left, see Fig. 7.2.

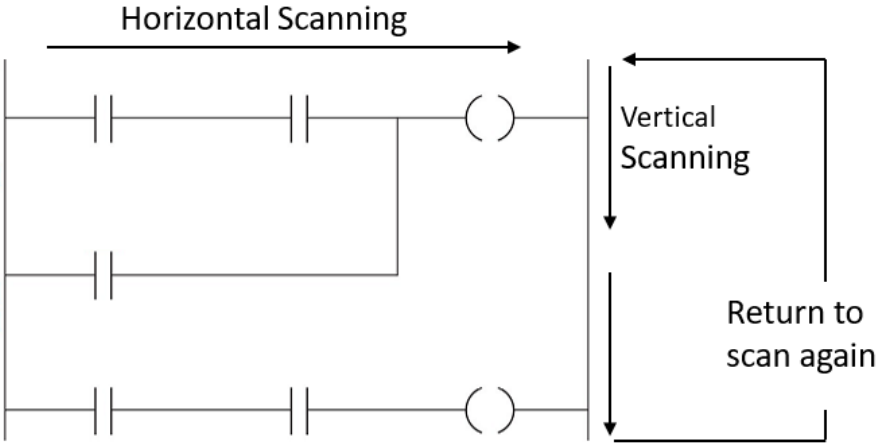


Fig. 7.2. Scan directions.

If the precondition instructions (such as inputs and comparative instructions) are “TRUE, the controller continuous to scan the reset of the rung (horizontally) to execute any available output instructions or other code elements (such as additions, counters, timers, etc.). If the precondition instructions of a rung are “FALSE”, the controller stops scanning the rest of the rung, i.e., doesn’t execute the rest of the instructions, and moves vertically down to scan the next rung. The status or value of any input instruction is either based on an internal memory value within the PLC or based on incoming signals of field devices that are sent to the PLC’s input modules, see Fig. 7.3. “Outputs” are tags that are set based on evaluating or triggering internal memory values.

The output values could be tied to physical addressed of a PLC’s output modules that control field devices such as motor or lights, see Fig. 7.3.

In addition to scanning all the routines within a program, the PLC controller’s performs other tasks such as checking PLC diagnostics and communication related ones. Once done, the PLC controller repeats the scanning process again in a continuous manner; unless it gets interrupted or scheduled to run periodically. When a PLC gathers and reads all related inputs of a program, executes its code, and updates all its related outputs, then that is called a scan cycle. If the scan cycle scans all routines within a program is called a program scan cycle. While if it scans only a specific routine, then it is called a routine scan cycle.

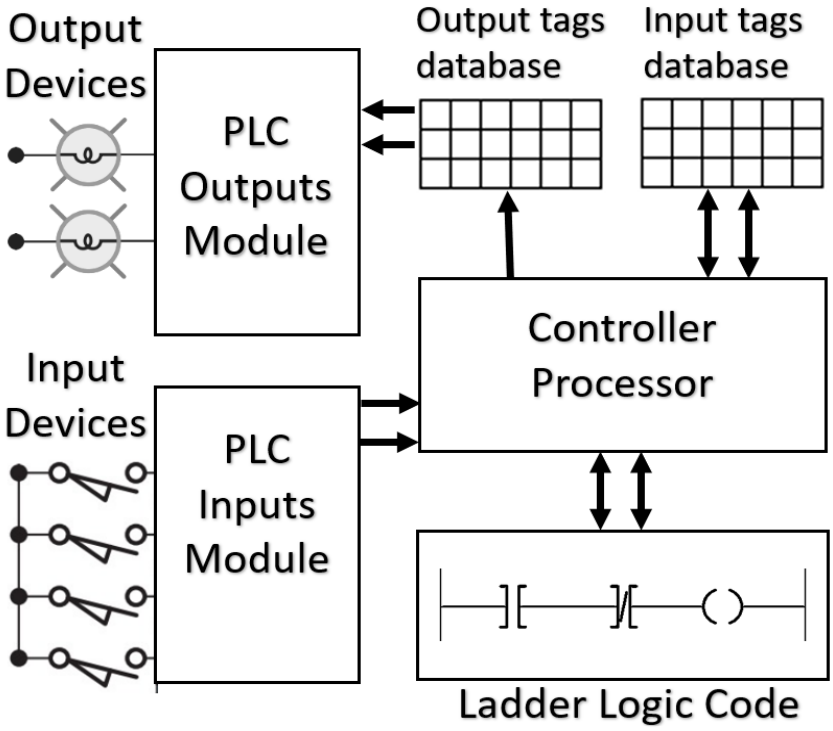


Fig. 7.3. Scan cycle.

7.3.1.2 PLC Scan Time

The scan time is the time a controller spends to scan and execute a certain designated program or routine including scanning and updating related physical tags, memory-based tags, networks, and other associated tasks within a scan cycle. So, the more code elements a CPU has to scan and execute per cycle, the slower the scan time would be. A slower scan time means that the program scan cycle is taking longer which adversely affects controlled operations, like slower response time. In certain scenarios, when a scan time of a code significantly exceeds the normal allocated time by the CPU per cycle, the watchdog would be triggered, and the PLC would be faulted. A faulted PLC would stop any processes or operations controlled by the PLC. A PLC scan time can be divided into two types: program and routine.

7.3.1.3 PLC System Overhead Time Slice

The PLC (GuardLogix 5570) used in this experiment, as well as many other PLCs produced by Rockwell Automation, allows users to change the execution time - allocated by the CPU - of background tasks, including the continuous task of a program. The modification of the scheduled allotted time of the CPU is done by selecting different percentage of the system overhead time slice, as shown in Fig. 7.4.

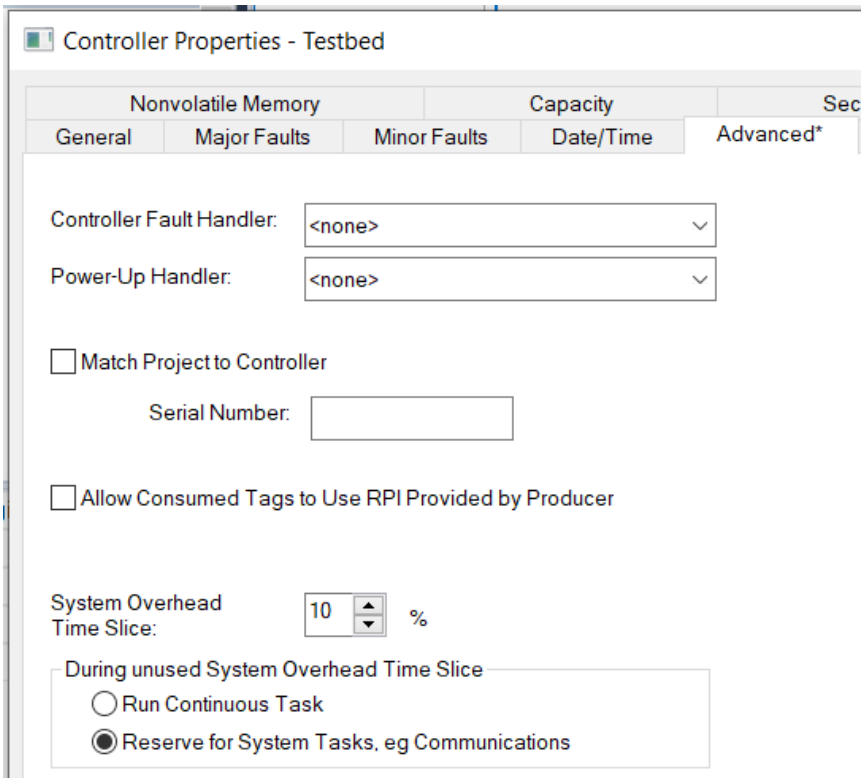


Fig. 7.4. Selecting a percentage of the time slice.

“Time slice” is the time that the CPU spends on unscheduled communication tasks, and its duration must not exceed one millisecond at a time. An unscheduled communication task is the task that is not configured through I/O configurations, such as communicating with logic designer software, communicating with HMIs, online edits, messages, etc. When time slice is modified, it changes the allocated time that a CPU spends in executing the continuous tasks (running the routines) and other background tasks.

Time slice only interrupts the continuous task in favor of other unscheduled communication, (assuming no periodic or event tasks). For example, whenever the allotted “time slice” is increased, the amount of time for the CPU to execute unscheduled communication tasks at a time is increased and less time is given to execute the continuous task. So, the overall scan time requires more time than usual to finish running all the routines (within the continuous task).

The relation between the time slice and the continuous task (such as scanning a routine) based on Rockwell documentations [169] [170] is as follows:

$$\boldsymbol{TS\%} = \left(\frac{\mathbf{1}}{\boldsymbol{CT}+\mathbf{1}}\right) \times \mathbf{100} \qquad (1)$$

Where TS% is the percentage of “time slice” that is set manually. CT represents the time the continuous task spent to execute its tasks (the time needed to run its listed routines). A task must be at least 1 millisecond [169] [170].

Based on Equation (1), CT can be defined as follows:

$$\boldsymbol{CT} = \left(\frac{\mathbf{100}}{\boldsymbol{TS\%}}\right) - \mathbf{1} \qquad (2)$$

For example, if “TS%” is set to 10%, then CT gets 9 milliseconds to execute its code and 1 millisecond would be dedicated to UCT (Unscheduled Communication Task), see Fig. 7.5 and Fig. 7.6.

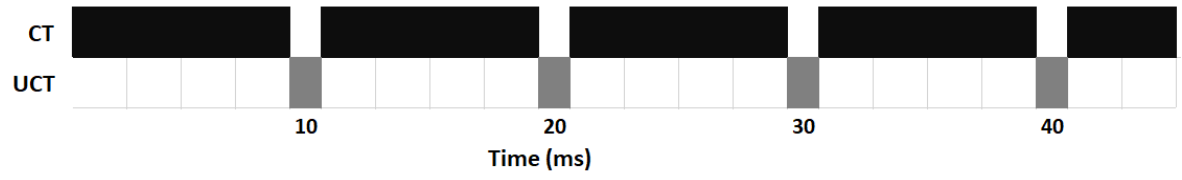


Fig. 7.5. CT and UCT during 10 milliseconds.

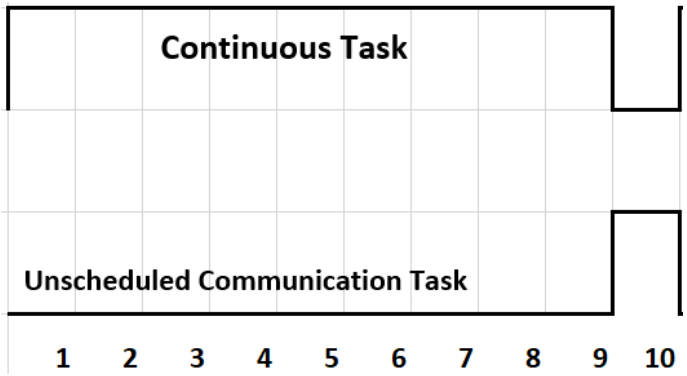


Fig. 7.6. A periodic duration of 10 milliseconds.

For a given 40 milliseconds of a code to execute, CT spends 9 milliseconds per cycle, and the unscheduled communication background task spends 1 millisecond. So, the overall scan time of the code is 44 milliseconds: 4 milliseconds for the unscheduled communication task and 40 milliseconds for the code execution. When "TS%" is set to 20%, CT gets 4 milliseconds per cycle, and the unscheduled communication background task gets 1 millisecond. So, the controller must cycle at least 5 times for the CT to complete the execution of 40 millisecond designated code. That increases the overall scan time of the code to at least 50 milliseconds, 5 milliseconds for the unscheduled communication task and 40 milliseconds for the code execution. Fig. 7.7 and Table 1 show more examples.

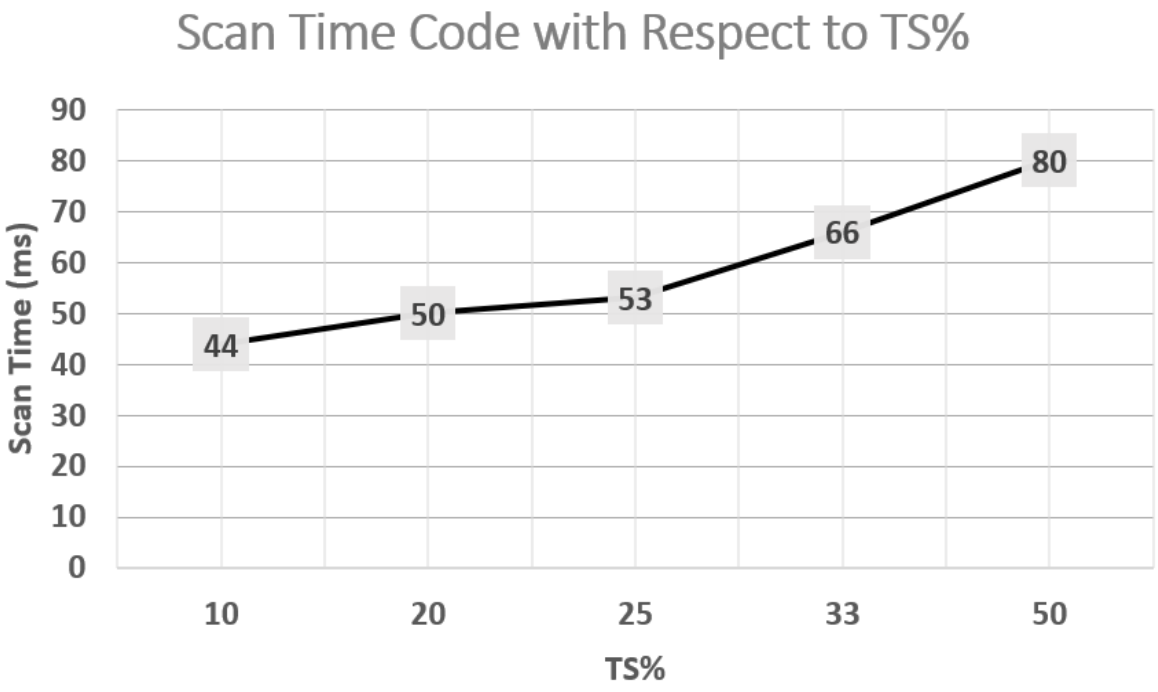


Fig. 7.7. Overall scan time was affected by the modification of TS%.

Table 1.
Scan times Value with Respect to CT and TS

Scan time (ms)	TS%	CT (ms)
44	10	9
50	20	4
53	25	3
66	33	2
80	50	1

7.4 A Case Study: STC Test Bed

7.4.1 Overview

Based on our analyses, we found that the time a PLC normally spends in scanning a routine or a program within one cycle could be employed beyond its typical usage. A scan time of a routine could be used as a real-time indicator of its code behavior to detect if there are any code modifications. By getting the average time of several scan times of a code under normal conditions (no attacks or malicious code) and within a specific time interval, that average could be used as a reference point. When a code is modified, its actual scan time value would be different than that of the reference scan time value. Generally, a faster or slower scan time might have critical consequences that adversely affect the automated process, devices, or staff, risking the whole system and environment. A change in a scan time could be because of one of the following:

- A slow scan time occurs because a suspicious code of finite or infinite loops are embedded.
- A slow scan time occurs if more code elements are embedded by adversaries to overload the PLC.
- A fast scan time occurs because a portion of the ladder logic code is deleted by hackers or adversaries.
- A slow overall scan time of the routines occurs because the overhead scan time slice is altered.
- A physical deterioration of filed devices. For example, a gate is taking longer than usual to be closed or open. That would make the program wait more than usual for the device’s feedback before updating any related outputs.

7.4.2 Test Bed and Experiment Setup

A real-time test bed experiment was introduced using a Rockwell GuardLogix 5570 controller, see Fig. 7.8. STC, a ladder logic-based code, was developed to capture the scan time of a routine, named "Main", per PLC cycle. The STC was designed to monitor, capture, and log the values of several scan time attempts of "Main" routine during a designated interval of time. Then, the average of the captured values was calculated. Initially, STC calculated the average scan time under normal and acceptable conditions to be used as a reference point. When the attack models were applied to "Main" routine, STC was able to log the new scan time values of the compromised routine and calculate the average scan time. By comparing the newly calculated average scan time of the compromised code to that of the reference average scan time, the attacks were detected. Once a discrepancy between the two compared values were found, STC setup warned operators and further scanning of "Main" routine or other associated routines were stopped.

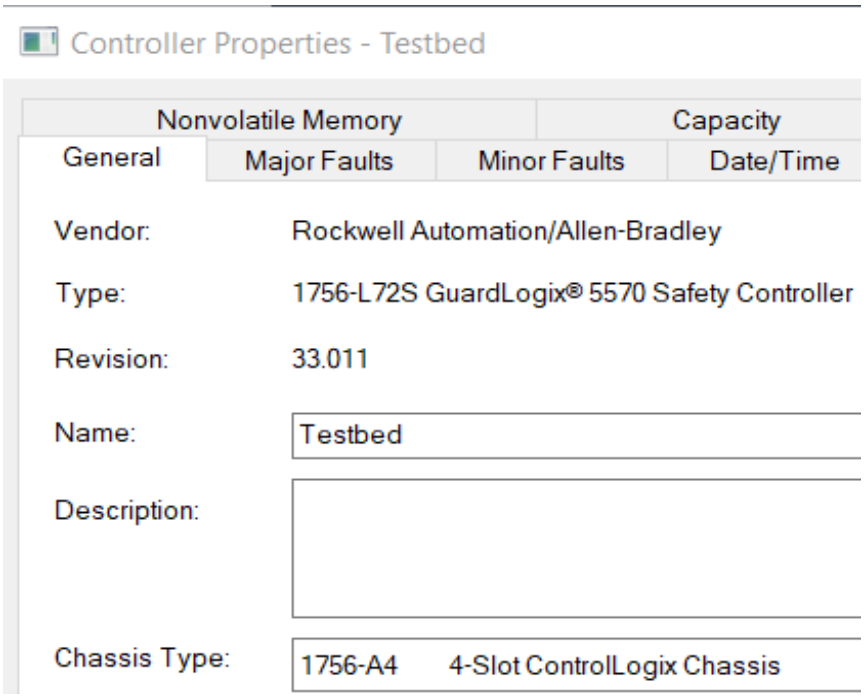
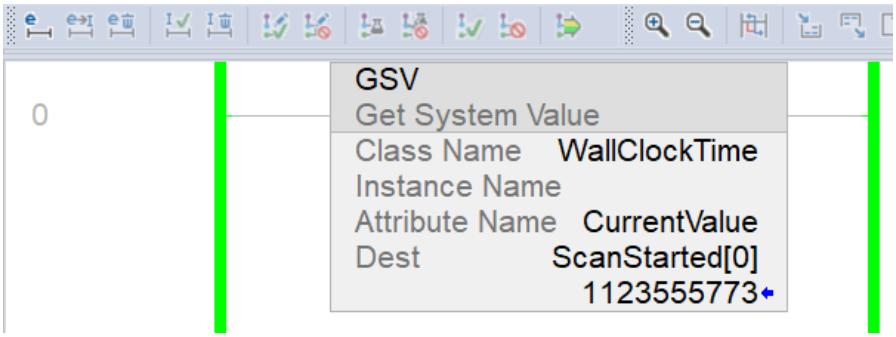


Fig. 7.8. Controller type.

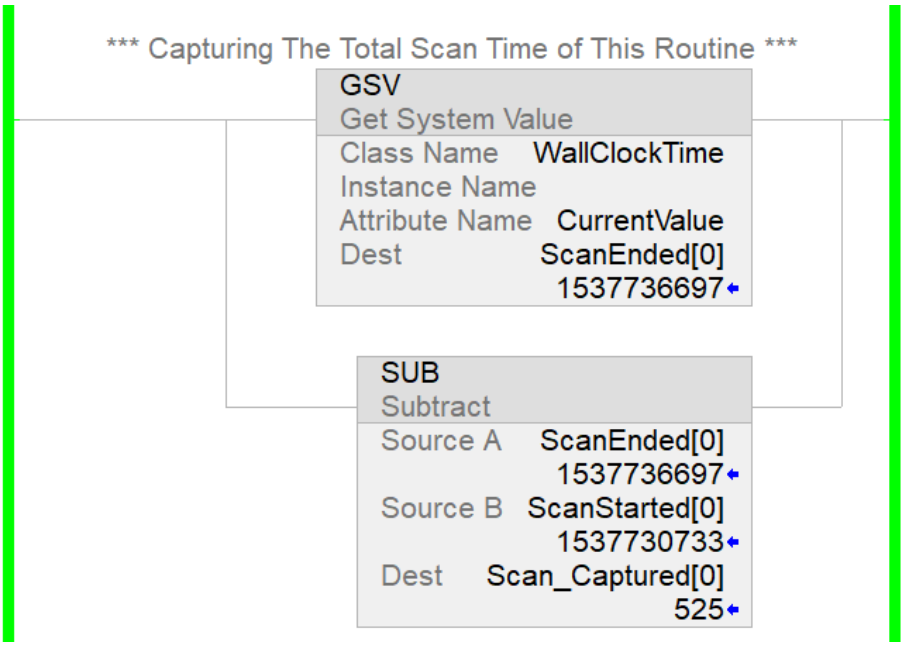
7.4.3 STC Setup

To capture timestamps of "Main" routine's scan time, STC used "WallClockTime", an attribute of "GSV" instruction. "WallClockTime" is an independent timestamp, not affected by timer instructions, code behaviors, or scanning scenarios.

The “WallClockTime” was added at the beginning and at the end of “Main” code to capture the elapsed time between two unique timestamps: the initial start of the scan and the end of it, see Fig. 7.9(a). Every time an initial timestamp was captured, by the beginning of the code execution, it got stored into the array “ScanStarted[0]”. And every time a timestamp value was captured by the end of the routine, it got stored into the array “ScanEnded[0]”, as shown in Fig. 7.9(b). Then those two unique stored timestamps got subtracted to calculate the net scan value of that routine and to be stored into “Scan_Captured[0]”.



(a)



(b)

Fig. 7.9 (a) Capturing the scan time of the routine in the beginning of the code. (b) Capturing the scan time by the end of the code and getting the time elapsed.

During each scan cycle, the new value of each scan time was monitored, captured, and stored in the array “ST_Values[30]”, as shown in Fig. 7.10. The values were not all the same. They varied but with marginal differences.

Name	Usage	Value
ST_Values	Local	{...}
ST_Values[0]		858
ST_Values[1]		502
ST_Values[2]		492
ST_Values[3]		491
ST_Values[4]		488
ST_Values[5]		919
ST_Values[6]		491
ST_Values[7]		490
ST_Values[8]		495
ST_Values[9]		477
ST_Values[10]		492
ST_Values[11]		490
ST_Values[12]		509
ST_Values[13]		476
ST_Values[14]		503
ST_Values[15]		489
ST_Values[16]		478
ST_Values[17]		504
ST_Values[18]		576
ST_Values[19]		495
ST_Values[20]		489
ST_Values[21]		502
ST_Values[22]		504
ST_Values[23]		488
ST_Values[24]		1253
ST_Values[25]		478
ST_Values[26]		478

Fig. 7.10. Storing values of scan time of “Main” routine

A “FIFO” instruction was used to log 30 captured values in the PLC data registry, as shown in Fig. 7.11. The captured scan time values had some marginal variances, as shown in Fig. 7.11. Because of such marginal variance, another code setup was introduced to calculate the average of all the 30 captured values using “CPT”, Comput, instruction. The calculated average was stored in “ScanAvg” tag, see Fig. 7.12.

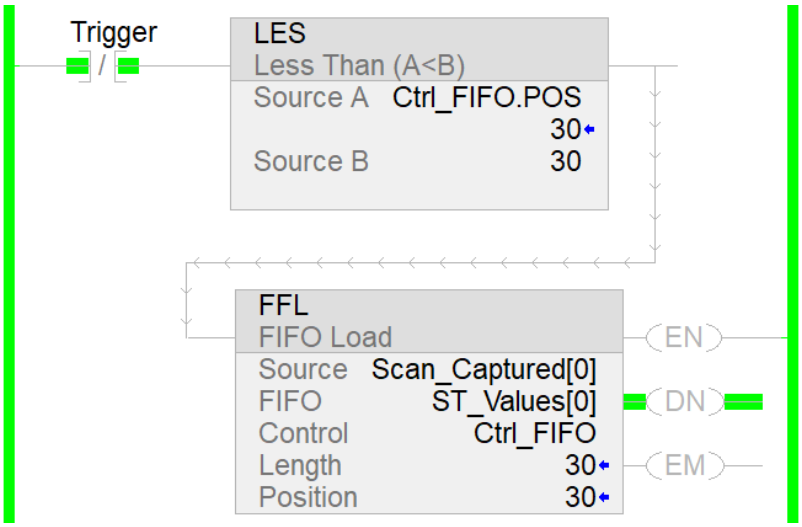


Fig. 7.11. Storing several scan time values using FIFO.

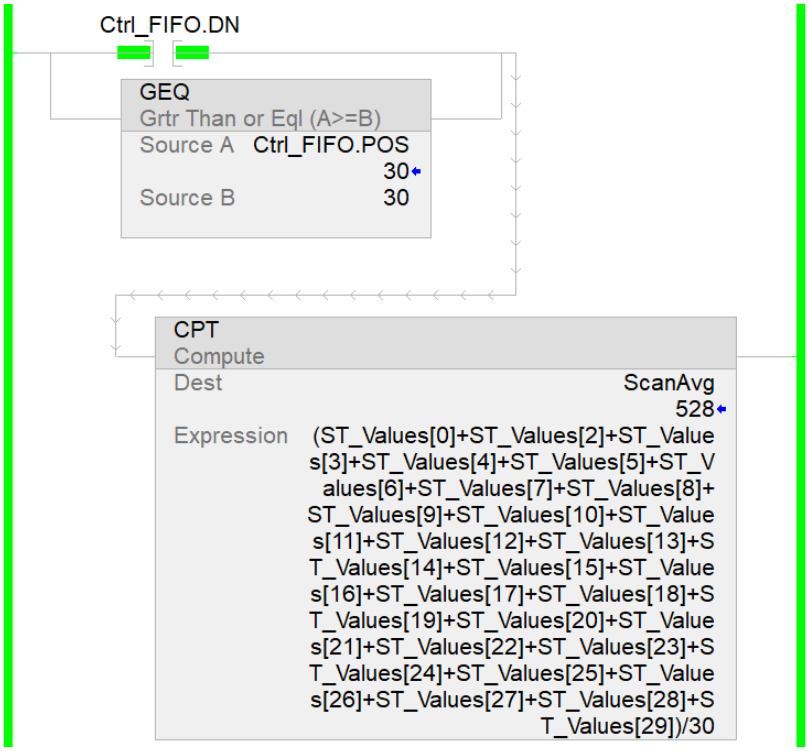


Fig. 7.12. Calculating the average scan time of 30 instances.

The scan average of the “Main” routine during that instance was 528 microseconds, as shown in Fig. 7.12. “ScanAvg” would be used as a baseline reference of “Main” routine in this test bed. It indicates the acceptable time that a PLC typically spends in completing the scan of “Main” code in one cycle. In some scenarios, if a more consistent and accurate average value is needed, then more data should be captured during a longer extended period.

Similarly, and after calculating “ScanAvg”, another setup was developed and embedded to capture any upcoming real-time scan time value of “Main” code, stored in “Scan_Captured[0]” tag, as shown in Fig. 7.13. Thirty captured values of “Scan_Captured[0]” during a designated time interval were stored in “ST_GVal[30]” array. A “FIFO” instruction was used to store those 30 scan values. The average of ST_GVal[30]” was stored in “STCapturedAvg” tag , as shown in Fig. 7.14.

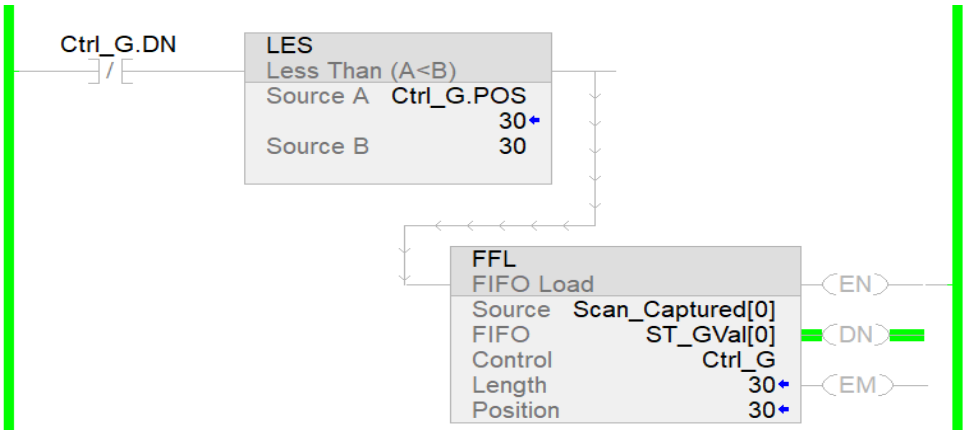


Fig. 7.13. “FIFO” to store 30 real-time captured scan time values.

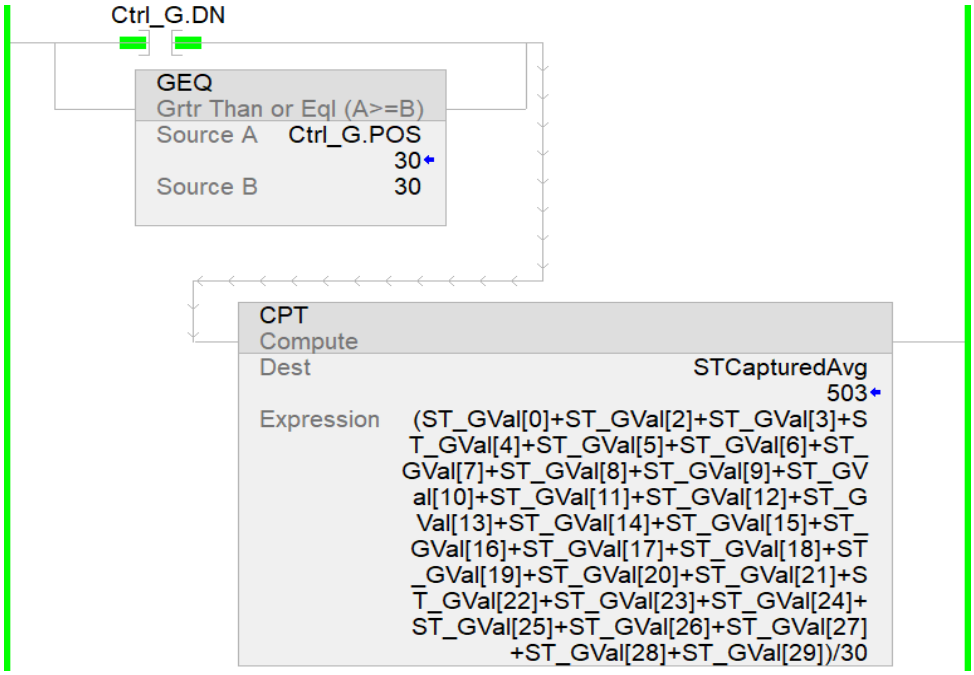


Fig. 7.14. The average scan time value of new real-time values.

By comparing the “STCapturedAvg” of the ongoing scan time of “Main” to the reference average, “ScanAvg”, the logic would be able to detect any code abnormality and raise an alerting message if the value of “STCapturedAvg” was not within the acceptable range.

7.5 Attacks Scenarios

Several attack models were conducted to exploit vulnerabilities in “Main” code and compromise it. The attacks were applied while the ladder logic code of “Main” was running. The attack models used different approaches to compromise the code and affect its behavior. They are summarized as follows:

- Embedding finite loops to slow down the scan time and overload the PLC.
- Embedding more code elements to overload the PLC scan time.
- Skipping portion of the code, i.e., skipping rungs that contained important instructions.
- Deleting code elements within the targeted code.
- Altering the overhead time slice system to significantly slow down the scan time of certain routines.
- Tampering with the behavior of field devices which could lead to systems damages.

The attacks were implemented while the PLC was continuously executing the code of “Main” routine. They achieved their intended purposes stealthily and without being detected.
By analyzing the attack models, we found that they did not just affect the code but also affected the scan time of the compromised code as well, except the attack that was deployed to tamper with the behavior of a field device. The change in the values of the scan time was captured manually as shown in Fig. 7.15.

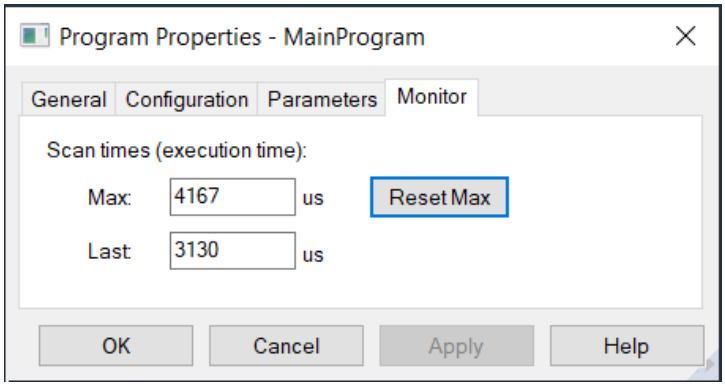


Fig. 7.15. Checking the scan time of the overall program using the properties of the “MainProgram” that includes all routines.

However, checking the scan time was limited since it only monitored the overall scan time of all the routine at once, and it was not accurate and reliable in certain cases. For instance, when

the percentage of the overhead “time slice” was increased, the overall scan time value did not show the changes in the scan time of certain routines. So, another method was introduced through our work which is discussed under “Countermeasures”, Section 7.6.

The following are the attacks that were implemented and applied with detailed explanations:

7.5.1 Attack 1: Embedding Finite Loops

One of the techniques that was implemented to overload and slow down the scan time of a routine (or a program) was by embedding a simple finite loop code using “FOR” instruction, as shown in Fig. 7.16. The overall scan time of the routines was significantly increased, i.e., the scan time became much slower, once “FOR” was embedded. The slow scan time was manually verified based on the program properties, as shown in Fig. 7.17.

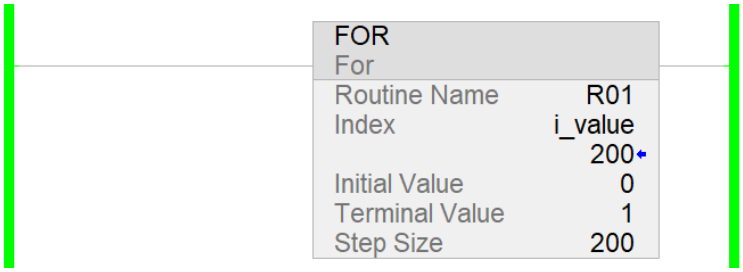


Fig. 7.16. Embedding “FOR” instruction

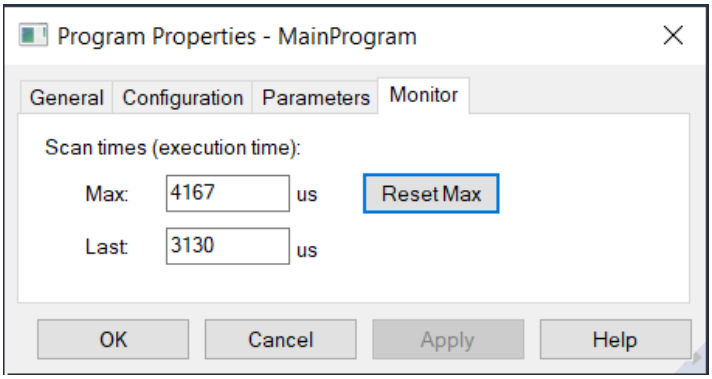


Fig. 7.17. Checking the scan time values after “FOR” was embedded.

The “FOR” loop was implemented in several ways to slow down the scanning time and overload the PLC:

- The “FOR” loop called another routine “R01” for 200 times before scanning the rest of “Main” code.
- The numbers of looping of “FOR” instruction were increased, “Step Size”, to 500 times not any further to guarantee not to trigger the PLC watchdog. For example, when the step size of a loop was set to 1000, the watchdog was triggered and faulted the PLC, as shown in Fig. 7.18 and Fig. 7.19.
- More loops were embedded to the code.

The logic neither detected the slower scanning nor reported the abnormal code element that caused a slower scan time.

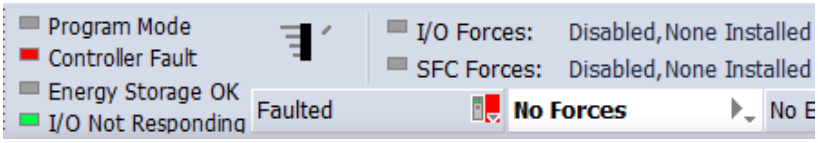


Fig. 7.18. PLC was faulted when “FOR” was set to a high value.

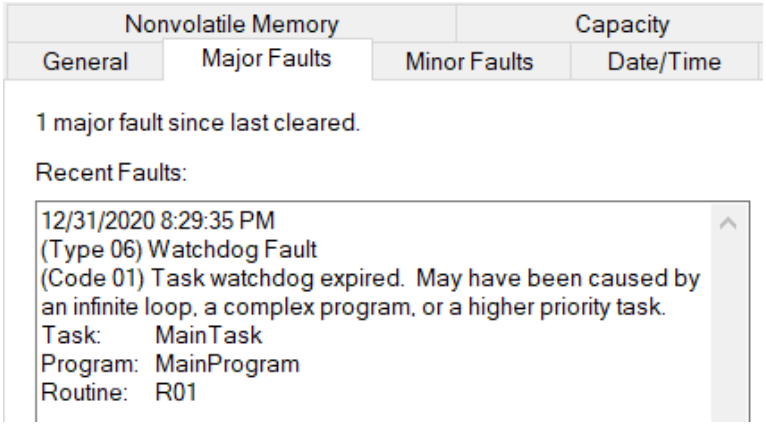


Fig. 7.19. Fault description after “FOR” was set to a high value.

7.5.2 Attack 2: Skipping Ladder Logic Rungs

Skipping portion of “Main” code was implemented by adding two instructions: “JMP” (Jump) and “LBL. The instructions were embedded while the PLC code was being executed without interrupting its continuous scanning. Once the PLC executed the rung that contained “JMP”, the controller skipped all the code after JMP instruction to the rung that started with “LBL” instruction and executed the rest of the code, see Fig. 7.20. Skipping rungs and associated code elements decreased the scanning time of “Main” routine since the overall program became 122 microseconds, see Fig. 7.21. That value was obtained by manually checking the overall scan time of all routines. The PLC was incapable of determining whether such skipping is

acceptable or suspicious. Also, the PLC never reported that a change of code or a scan time happened. Skipping code elements would have serious implications on the system and safety workplace.

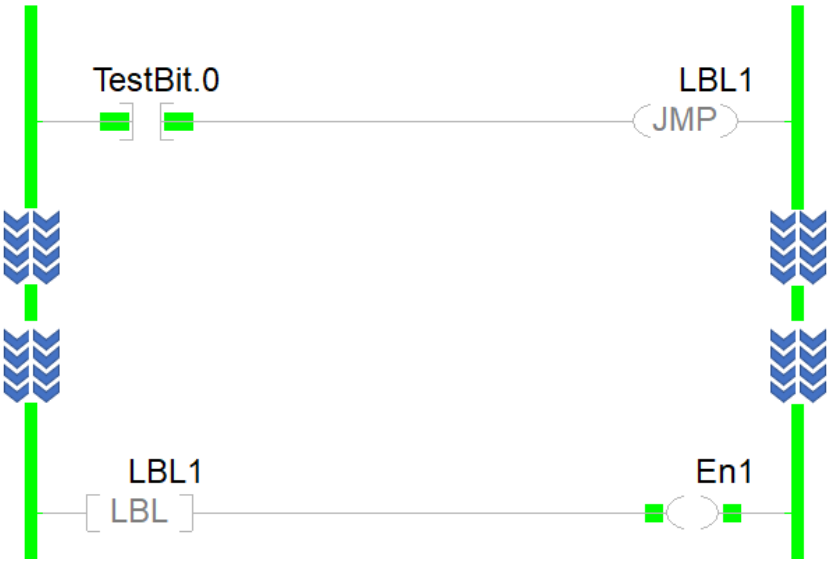


Fig. 7.20. Using JMP to skip ladder logic rungs.

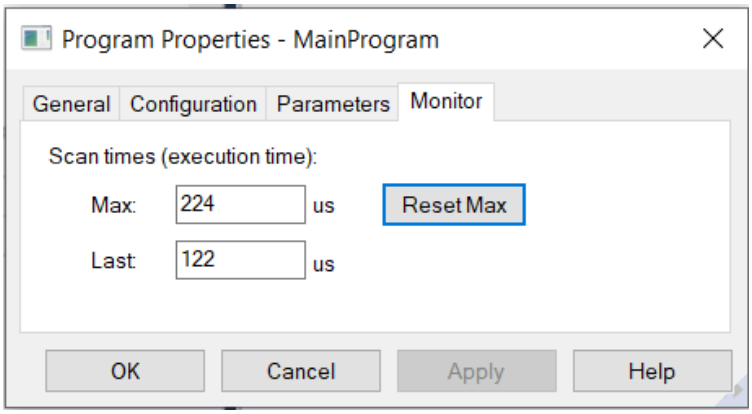


Fig. 7.21. The overall scan time of all routines after embedding JMP.

7.5.3 Attack 3: Deleting Code Elements.

In this critical scenario, several rungs of the running ladder logic code were deleted while the PLC was in “RUN” mode. The PLC compiler accepted the deleted code without reporting any

warning or interrupting the execution of the running code. By manually checking the overall scan time of the PLC program, the scan time was much faster, see Fig. 7.22. Based on this portion of the code was stealthily deleted while the PLC could not detect any of its threat. It had no way to distinguish whether the deletion was acceptable or suspicious. A tampered logic would risk critical safety operations, functionalities, or processes.

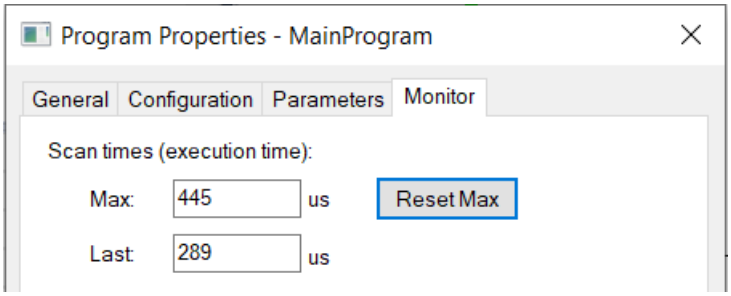


Fig. 7.22. The overall scan time of all routines after deleting code elements.

7.5.4 Attack 4: Embedding more Code Elements Mimicking Payload Code

In this attack, a couple rungs of code that had mathematical instructions, “XPY” (X to the Power of Y) were embedded, as shown in Fig. 7.23, which could be also replaced by any malicious payload code. Though the new code was embedded stealthily and not detected by the PLC, it turned out that it affected the overall scan time of the PLC program when the scan time was manually checked, see Fig. 7.24.

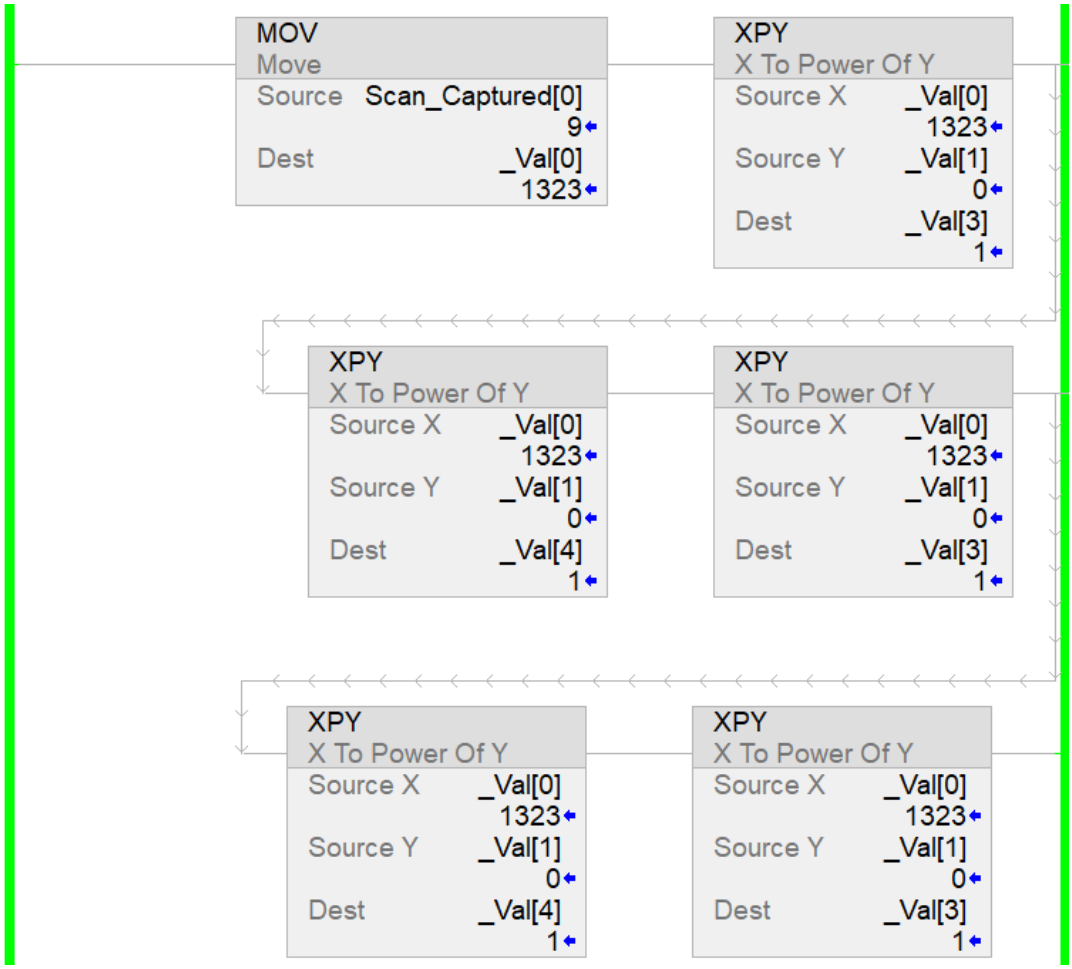


Fig. 7.23. Embedding code elements using "XPY" instruction.

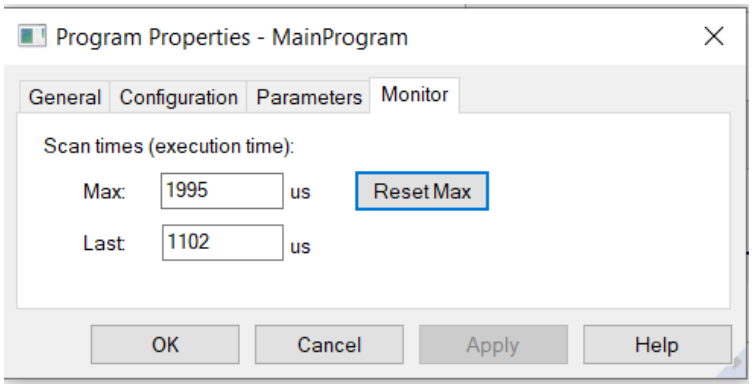


Fig. 7.24. Checking the scan time values after more code was embedded.

7.5.5 Attack 5: Modifying PLC Time Slice

While the PLC was running, the System Overhead Time Slice (SOTS), or for simplicity called "time slice," was changed from 10% to 90%. The modification of the SOTS was done by selecting a new percentage value from the drop-down menu, as shown in Fig. 7.25. The overall

scan time of all routines did not show any changes. But the scan time of all the tasks, including background ones, was tremendously increased (from 5.95 milliseconds to 81.04 milliseconds), as shown in Fig. 7.26. The modification of the new time slice was neither detected by the PLC nor was it reported. That would have an enormous impact on the PLC responses when it comes to communicating to other devices or monitoring fast devices such as servo drives or encoders.

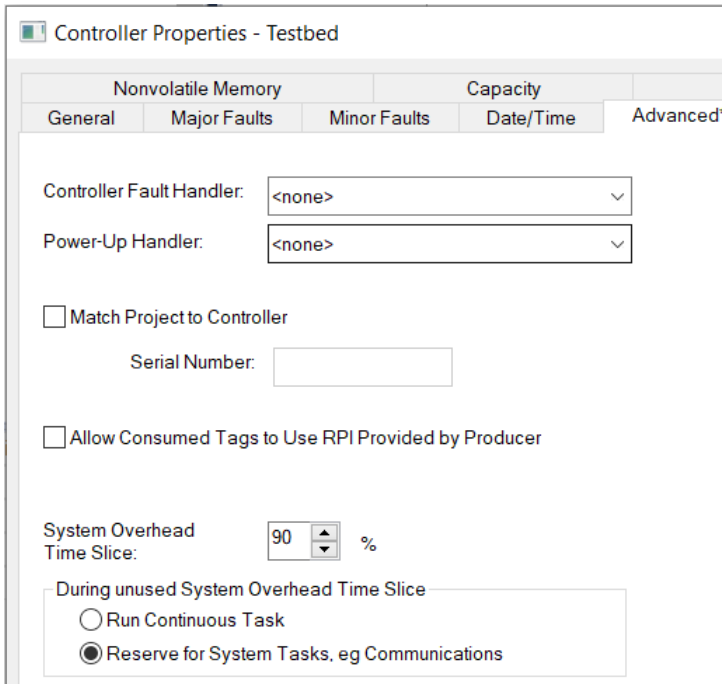


Fig. 7.25. Setting the "Time Slice" to 90%

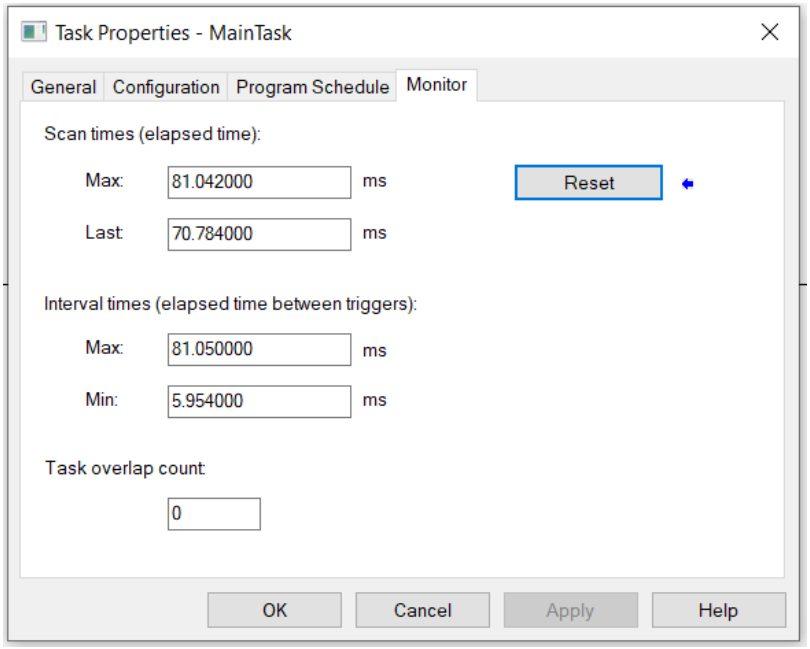


Fig. 7.26. The overall scan time of all tasks was significantly increased.

7.5.6 Attack 6: Tampering with the Behavior of Field Devices

The previously explained STC techniques can be used to detect any modifications of field devices, especially of the attacks targeted deleting any timer- based logic. But to enhance the security if the field devices code more techniques are introduced in this section.

Initially, an attack was deployed to compromise the functionalities of field devices by tampering with their ladder logic code setup to exploit their behaviors. The test bed developed in this experiment was a typical ladder logic setup of a pneumatic pin, targeting its extending and retracting operations. The attack was set to compromise either the outputs or the inputs of the pin through its logic code. A pin, typically, has two inputs (extended and retracted) and two outputs commanded by the PLC code (to extend and to retract), as shown in Fig. 7.27. Normally, whenever the pin ladder logic code commands a pin to extend (engage), the pin should be extended (engaged). And when the pin is commanded to retract (disengaged) it should be retracted (disengaged). The pin status (whether extended or retracted) should be reported back to the PLC, as shown in Fig. 7.28. The relation between a PLC and the pneumatic pin (as well as some other actuators) is not a direct closed loop, it must be properly coded. For instance, a typical code that cares more about the functionalities of the pin, it would not check and confirm the proper status of a pin without taking into consideration vulnerabilities that could be coming from bad code practice, attacks, or physical device deterioration.

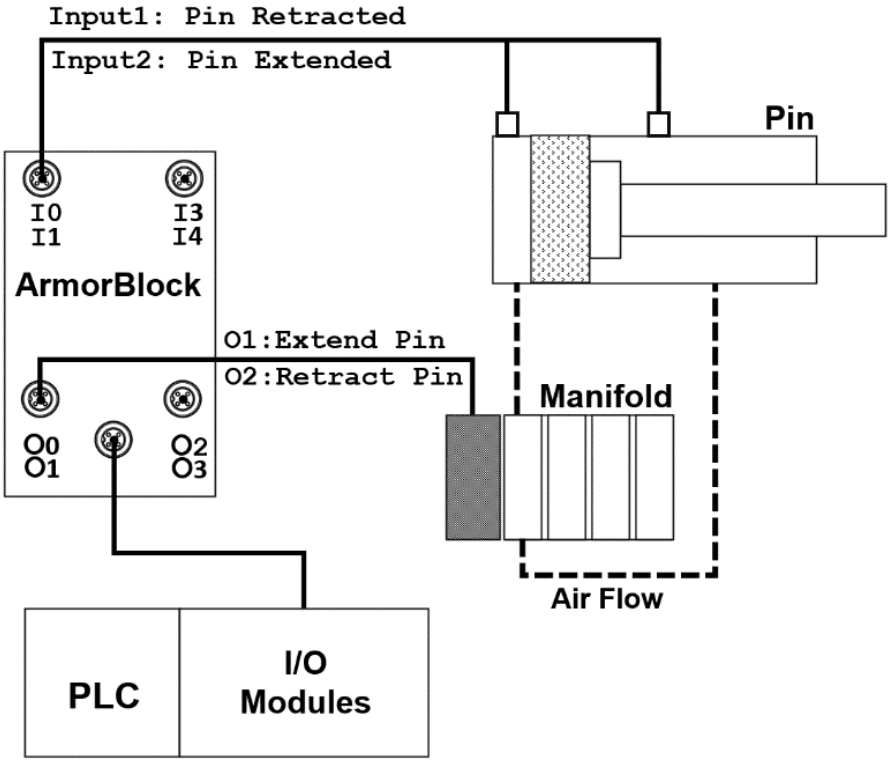


Fig. 7.27. Pneumatic Pin with hardwire diagram.

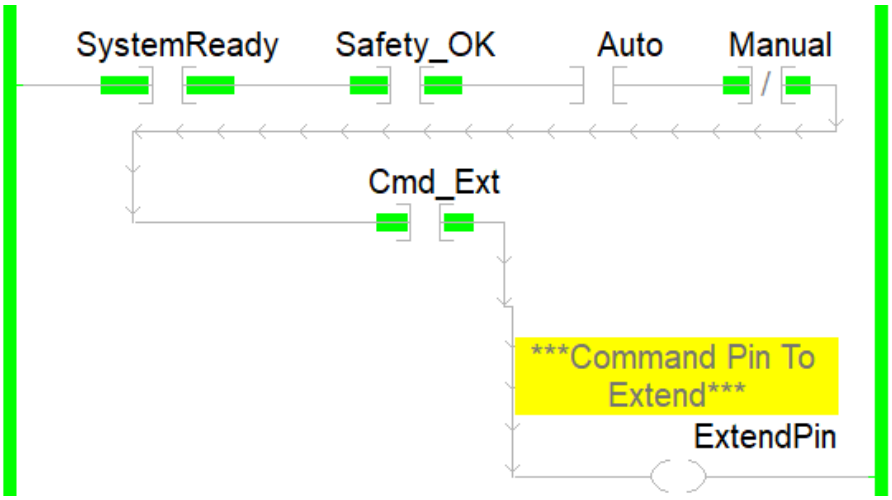


Fig. 7.28. Ladder logic Setup of a Pin.

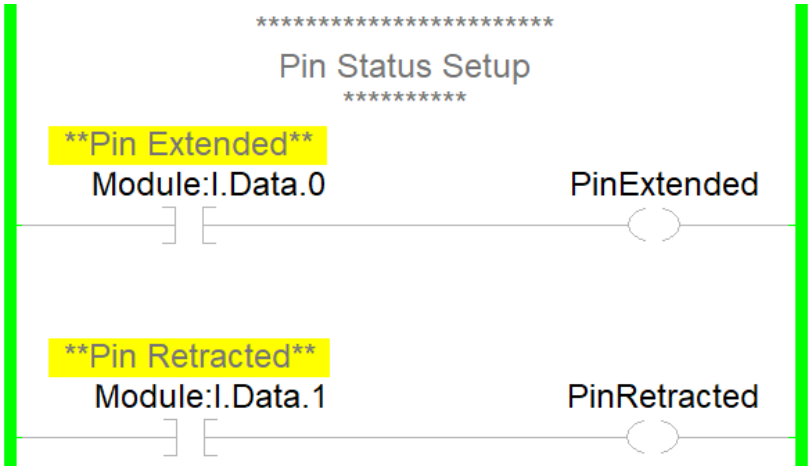
Based on the deployed attack, the ladder logic code of the pin was vulnerable and compromised without interrupting any code execution. The following is a list of vulnerabilities that might be exploited:

- Swapping Statuses
- Swapping Outputs
- Embedding Always ON status
- Embedding Always OFF status
- Deleting Outputs
- Delaying status
- Delaying Outputs

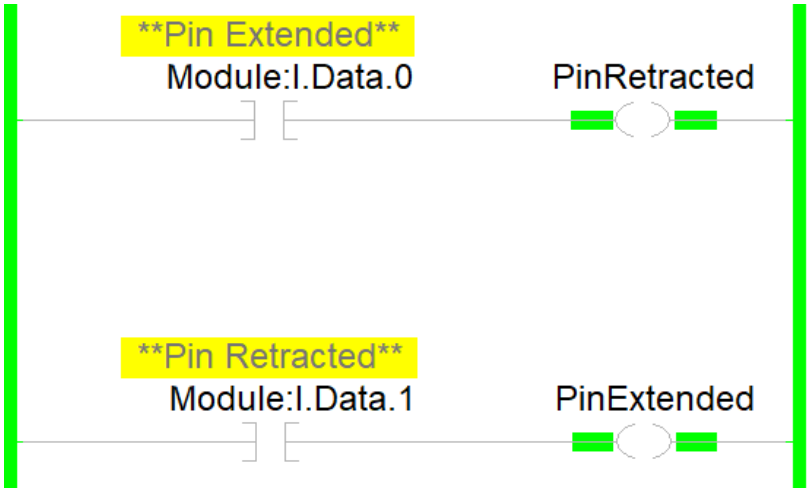
a) Swapping Statuses Attack

This attack compromised the ladder logic code of the pin by tampering with its statuses. The two statuses of the pin (extended and retracted) were edited and swapped while the logic code was running. The extended status bit was swapped with the retracted status bit, as shown in Fig. 7.29. So, the logic of the pin was reading extended while physically it was retracted, and vice versa. This attack would severely affect any process or functionalities associated with that

device. Relying on wrong device status would have devastating consequences in certain scenarios depending on the critical role of the compromised field devices.



(a)



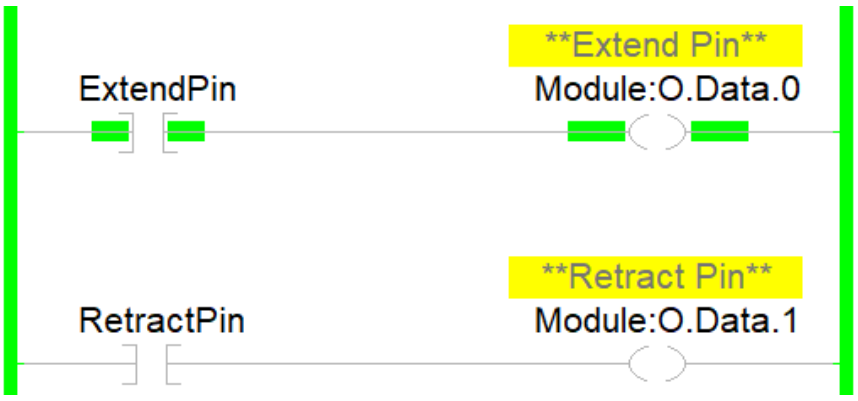
(b)

Fig. 7.29. (a) Logic setup to monitor whether the pin was extended or retracted. (b) The inputs were swapped while the logic was running.

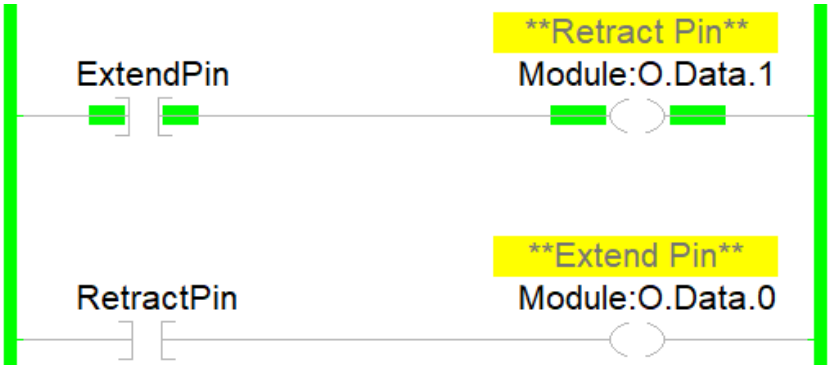
b) Swapping Outputs Attack

This attack targeted the outputs. The outputs in the pin logic code were swapped to fool the process and create damage. The command that was responsible for extending the pin was modified to retract it instead. And rather than commanding the pin to retract, the pin code was modified to extend, as shown in Fig. 7.30.

The PLC did not detect such modification, and it was implemented without any need to recompile the overall running PLC program.



(a)



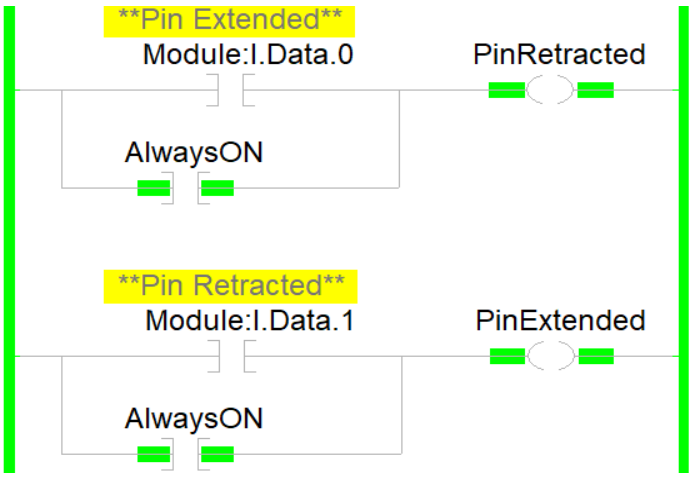
(b)

Fig. 7.30. (a) Logic setup to extend or retract a pin. (b) The outputs were swapped while the logic was running

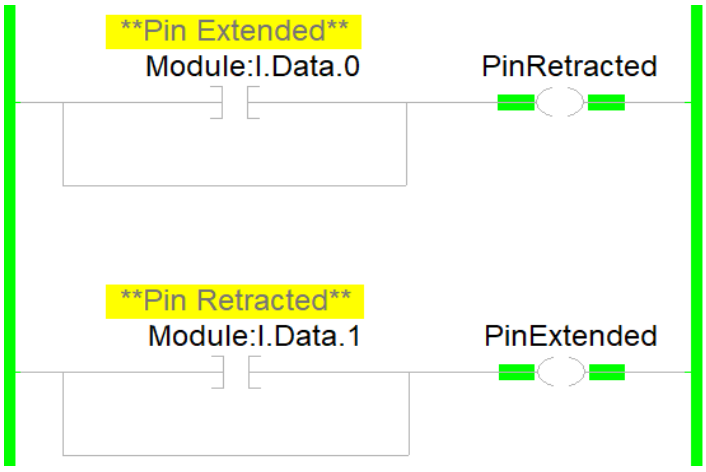
c) Embedding "Always ON" Status Attack

The attack embedded two bits to make both statuses (inputs) of the pins ON, i.e., always extended and retracted at the same time, as shown in Fig. 7.31(a).

Another method to make both inputs ON was by embedding an empty branch in parallel of all precondition instructions, see Fig. 7.31(b). The controller was not able to detect those modifications.



(a)



(b)

Fig. 7.31. (a) Forcing the status bit to be always ON. (b) An empty branch was used to bypass any preconditions.

d) Embedding “Always OFF” Status Attack

The target of this attack was to alter one of the statuses (inputs) in the pin code by disabling it to create damage in the system. For instance, one of the inputs was set to be deenergized (OFF) all the time. That means the PLC was reading that the pin was always extended but never retracted, as shown in Fig. 7.32. The logic code was not able to detect such abnormal behavior.

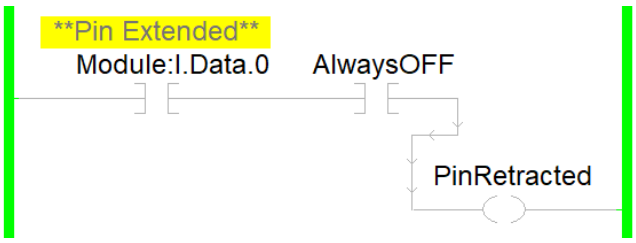


Fig. 7.32. Forcing the status bit of the pin to show not retracted regardless of the actual device feedback.

e) Deleting Outputs Attack

The deployed attack was implemented to delete one of the outputs of the pins without interrupting the running code. It replaced one of the outputs with “NOP” instruction. “NOP” is an instruction that does not perform any operation, as shown in Fig. 7.33.

The deletion of the output and replacing it with “NOP” was implemented without faulting the controller or getting detected.

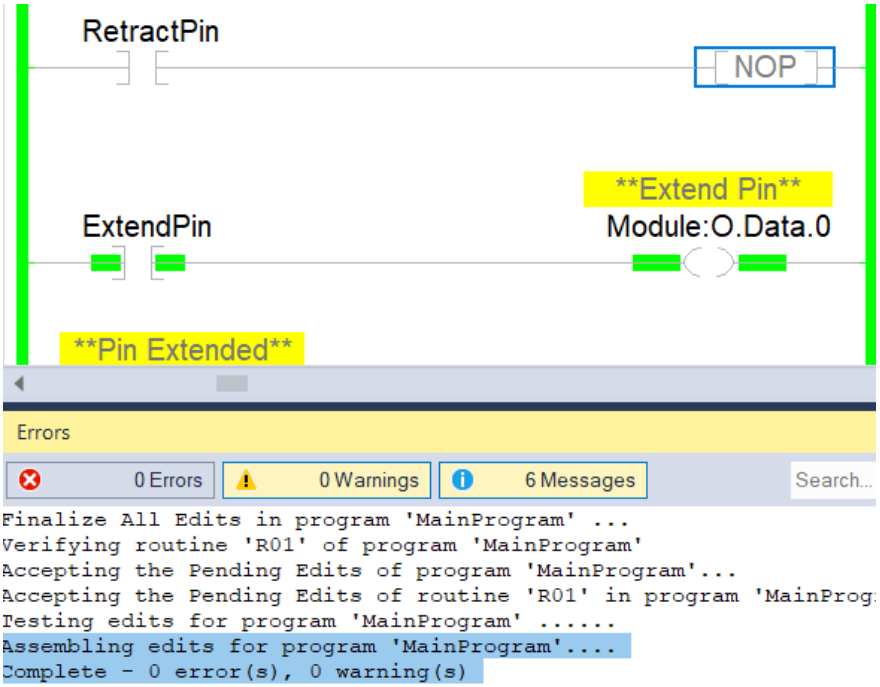


Fig. 7.33. Deleting one of the outputs to prevent commanding the pin to retract.

f) Delaying Status Attack

A timer delay instruction, TON -Timer ON Delay, was embedded in this attack to delay reading the statuses of the pin, as shown in Fig. 7.34. The attack was able to delay the real-time monitoring by 2 seconds. Implementing this threat was done while the PLC was executing the associated code and was not detected by the controller. Such delay would risk the system safety and reliability since delaying the real-time response would impact critical processes.

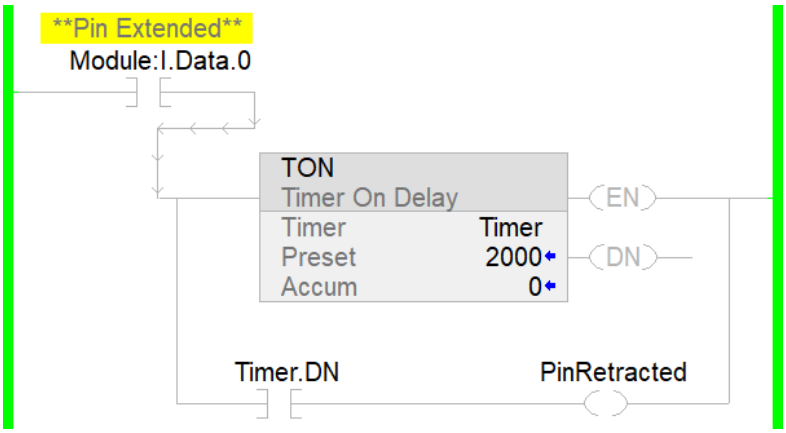


Fig. 7.34. Delaying a status of the pin

g) Delaying Outputs Attack

A TON (Timer ON Delay) instruction was embedded to delay energizing the outputs of the pin, as shown in Fig. 7.35. The attack made the PLC controller wait longer before executing next associated operations. Implementing this threat did not require any overall program download or recompiling. Delaying the statuses would risk the system if a quick real-time response is critical.

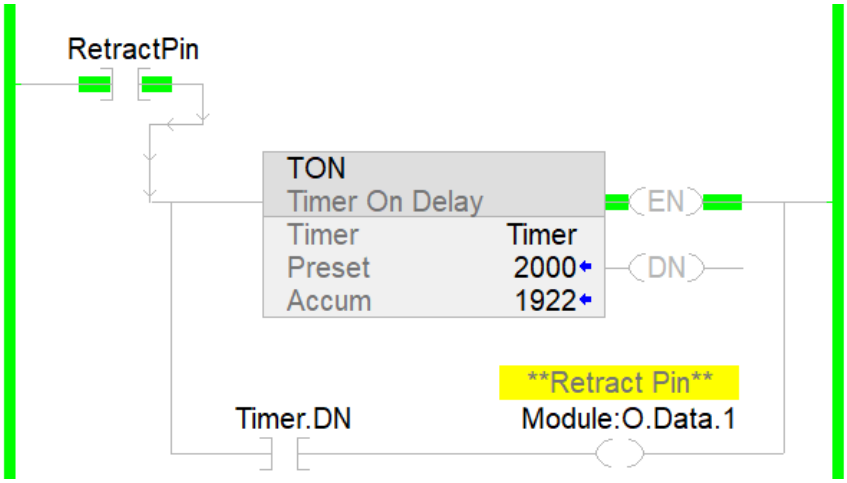


Fig. 7.35. Delaying the output for 2 seconds before energizing it.

7.6 Countermeasures

7.6.1 Countermeasures Against Infinite Loops

This method was implemented based on the scan time code discussed in Section STC Setup. A ladder logic was developed to detect the suspicious finite loops based on the scan time code. The embedded attack of finite loops made the PLC spend more time in finishing the execution of that exploited routine.

By capturing the new scan time of the compromised routine and comparing it to the reference average scan time “ScanAvg”, previously calculated, the PLC detected the code abnormalities. To enhance this countermeasure, several scan-time values, 30 values, within a designated duration were captured, as shown in Fig. 7.36.

Name	Usage	Value	Force Mask
ScanTime1	Local	0	
ST_GVal	Local	{...}	{...}
ST_GVal[0]		1360	
ST_GVal[1]		1350	
ST_GVal[2]		1444	
ST_GVal[3]		1357	
ST_GVal[4]		1347	
ST_GVal[5]		1345	
ST_GVal[6]		1347	
ST_GVal[7]		1357	
ST_GVal[8]		1343	
ST_GVal[9]		1352	
ST_GVal[10]		1347	
ST_GVal[11]		1360	
ST_GVal[12]		1607	
ST_GVal[13]		1342	
ST_GVal[14]		1597	
ST_GVal[15]		1349	
ST_GVal[16]		1349	
ST_GVal[17]		1437	
ST_GVal[18]		1428	
ST_GVal[19]		1344	
ST_GVal[20]		1361	
ST_GVal[21]		1440	
ST_GVal[22]		1348	

Fig. 7.36. Capturing several scan-time values after embedding “FOR”.

Then, the average scan time of the exploited code was calculated and stored in "STCapturedAvg" array element, as shown in Fig. 7.37. The average scan time, "STCapturedAvg", of the compromised code was significantly higher than that of the reference average "ScanAvg", 1336 microseconds compared to 528 microseconds, as it was previously shown in Fig. 7.12.

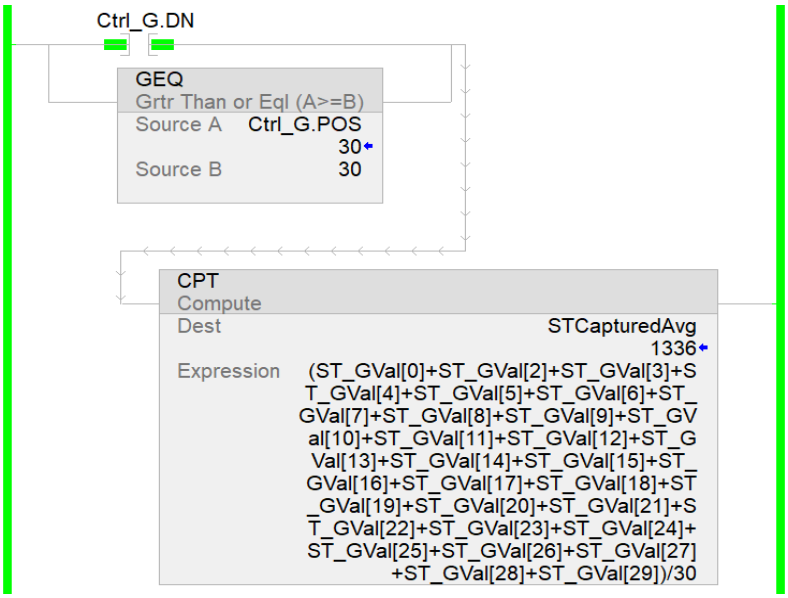


Fig. 7.37. Capturing the average of the scan time after embedding "FOR" instruction.

This countermeasure solution was able to detect other related finite loops attacks based on the scan time values as well, see shown in Fig. 7.38.

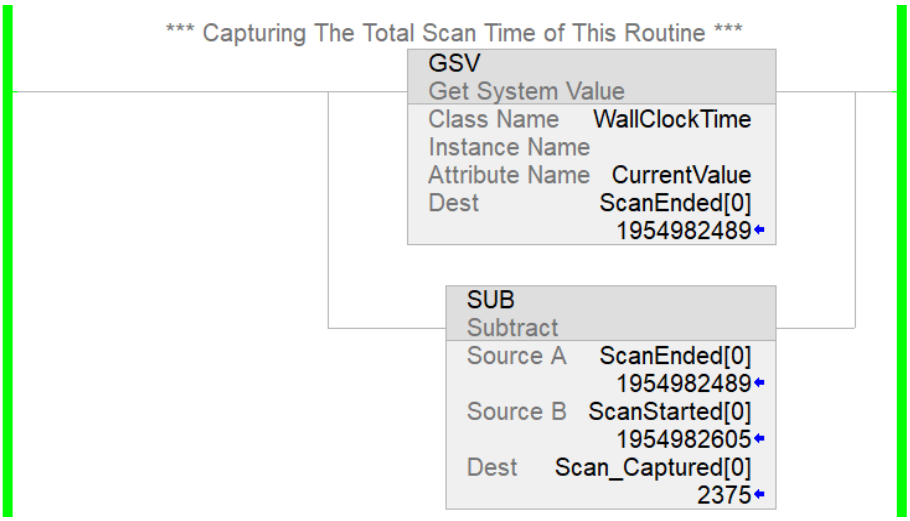


Fig. 7.38. Capturing a higher scan time value after increasing number of loops.

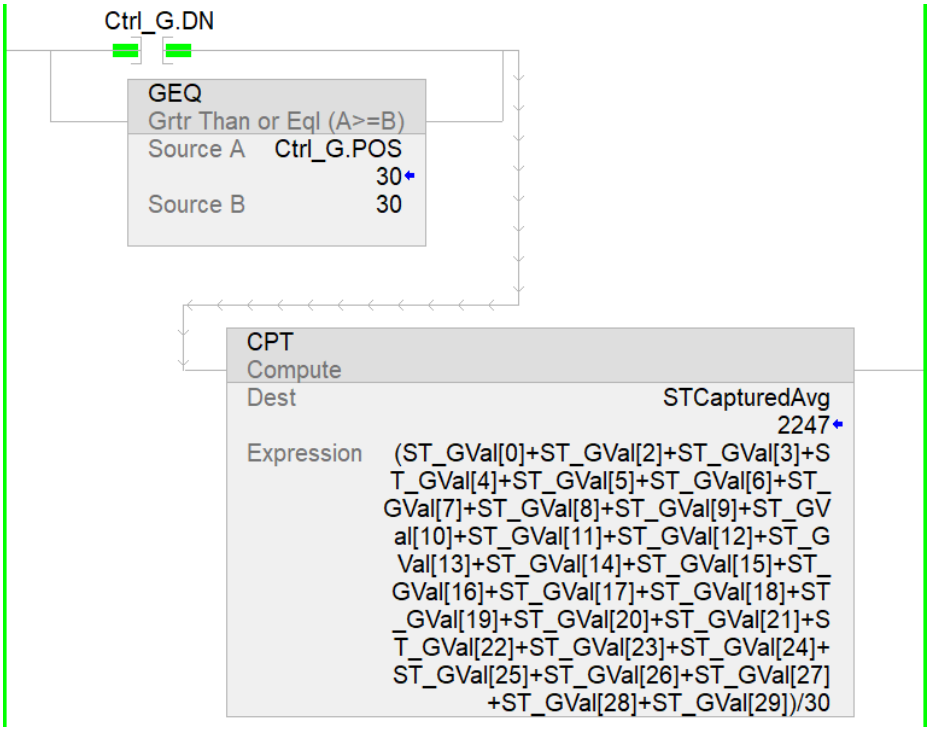


Fig. 7.39. Average scan time after embedding another "FOR" loop.

The average of other finite loops attacks was calculated as well, see Fig. 7.39 and Fig. 7.40.

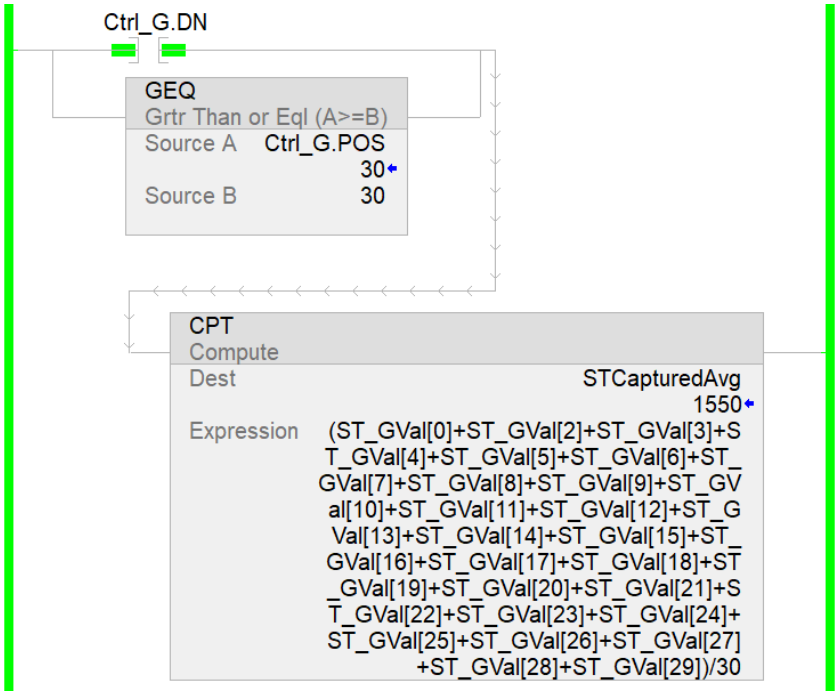


Fig. 7.40. The average scan time after increasing the loop size.

Every time the countermeasure code detected a discrepancy between “ScanAvg” and “STCapturedAvg”, it stopped further scanning of the code and warned operators, as shown in Fig. 7.41.

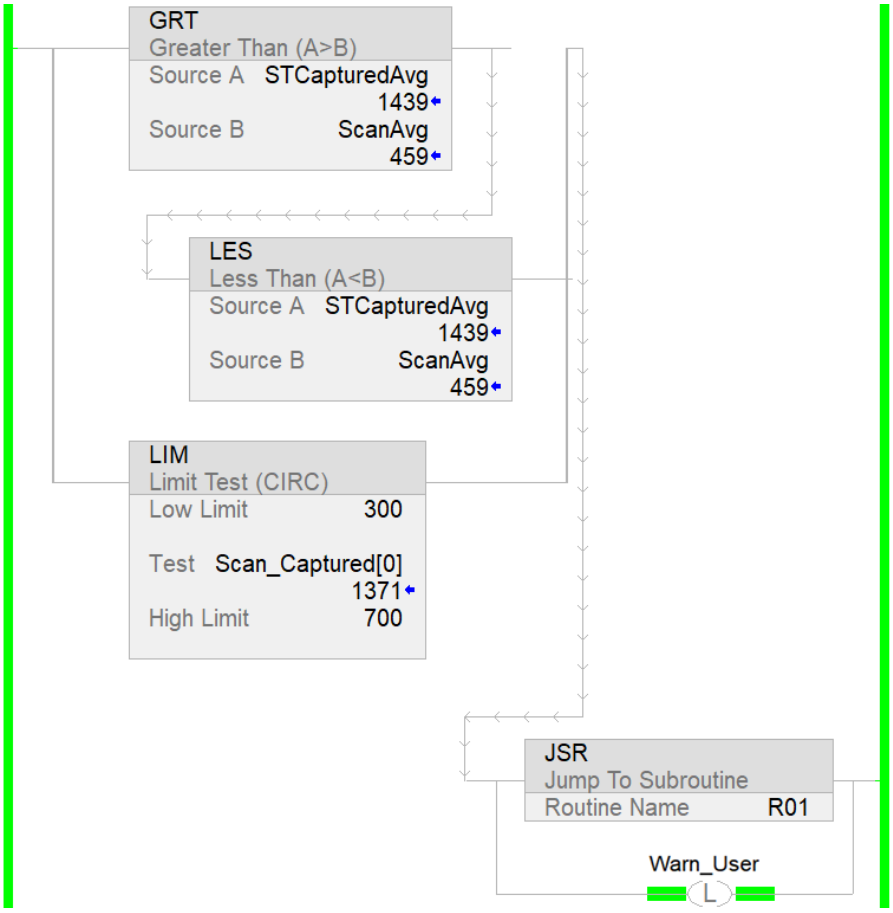


Fig. 7.41. Stopping further code scanning.

7.6.2 Countermeasures Against Skipped Code Attack

This countermeasure solution used a ladder logic trap based on scan time setup to detect the suspicious skipping of code elements within the “Main” routine. The countermeasure code monitored and captured the new scan time average of the exploited code and calculated its average based on 30 scans, as shown in Fig. 7.42. The average scan time of the exploited code, “STCapturedAvg”, was 9 microseconds, much lesser than that of the reference average value, “ScanAvg”. By comparing both averages, the code abnormality was detected, and further logic was not allowed.

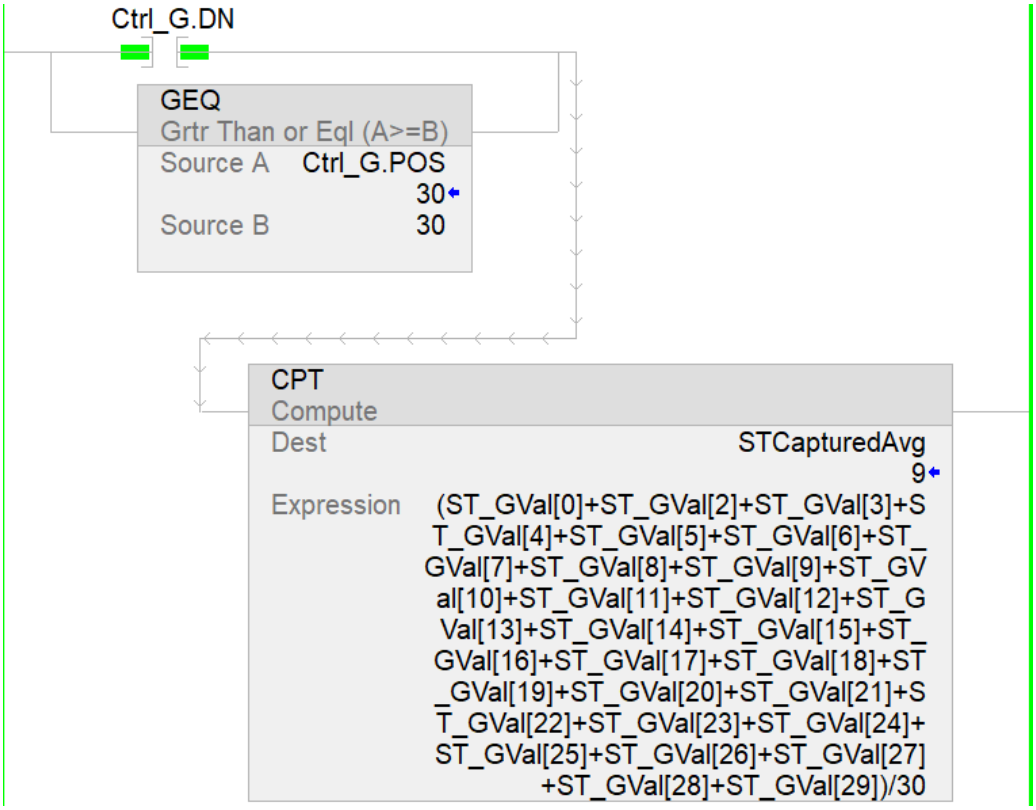


Fig. 7.42. Calculating the average of the scan time after a JMP instruction was embedded.

7.6.3 Countermeasures Against Deleted Code Attack

After monitoring 30 scan time values of the compromised code and calculating the average, “STCapturedAvg”, this method detected the code exploitation, see Fig. 7.43. The “STCapturedAvg” was 20 microseconds which was less than that of the stored reference average, “ScanAvg”. Once detected, the controller stopped further code scanning and alerted operators.

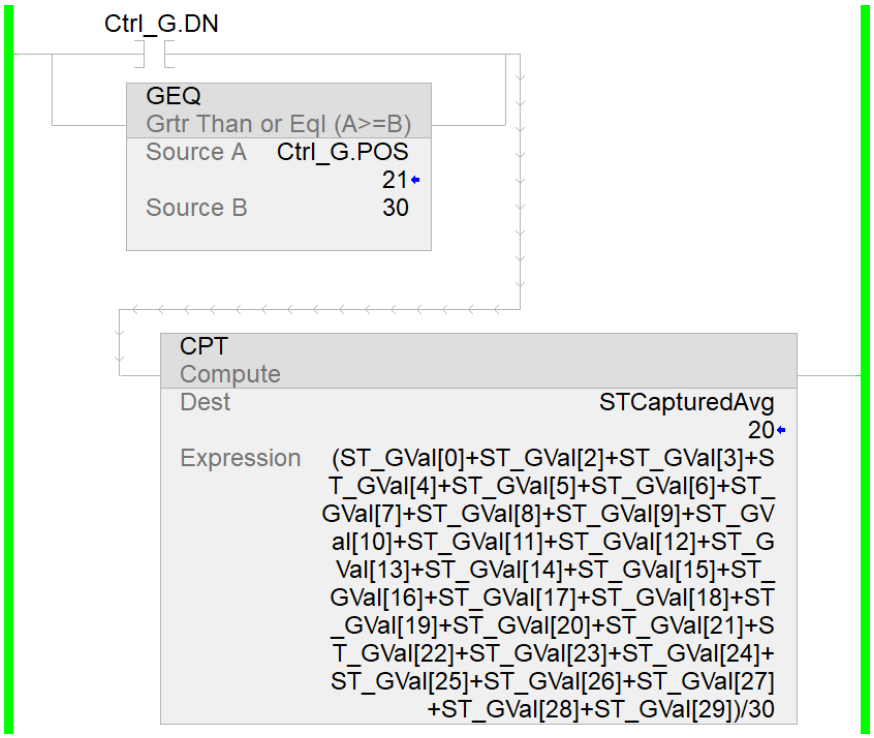


Fig. 7.43. Calculating the average of the scan time after code elements were deleted.

7.6.4 Countermeasure Against Embedding more Code Elements

Similar to the previous countermeasure solution, several scan time values of the exploited code were captured, and then the average was calculated and stored in “STCapturedAvg” array, as shown in Fig. 7.44 and Fig. 7.45.

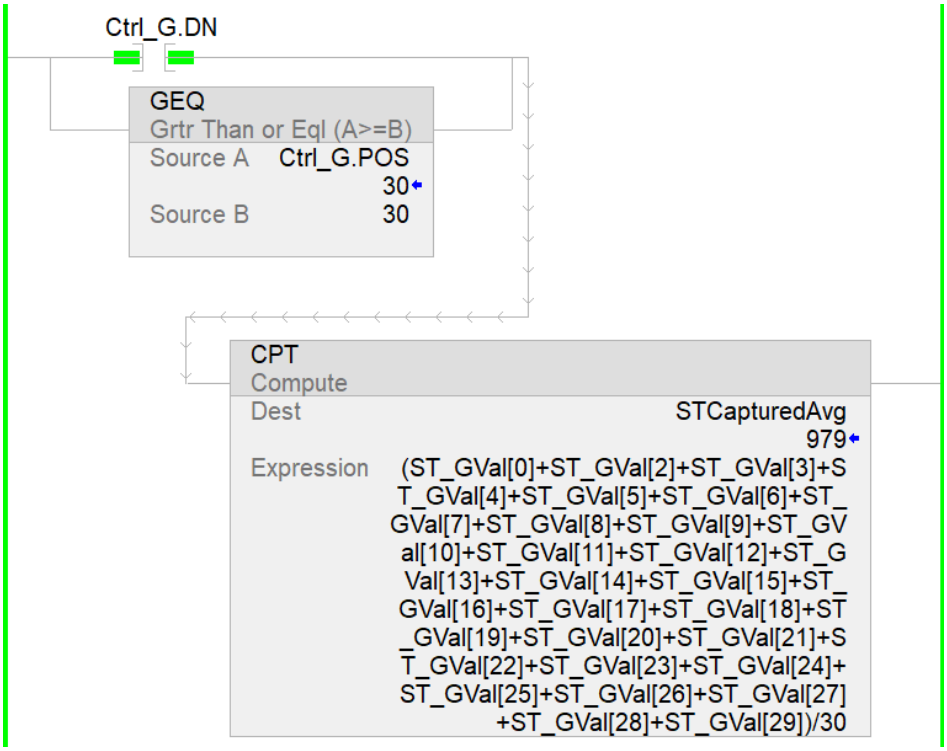


Fig. 7.44. The average scan time value after embedding more code elements.

The average scan time of the exploited code, “STCapturedAvg”, was 979 microseconds, and the was compared to the reference average, “ScanAvg”. By comparing the two averages, the countermeasure solution detected the suspicious code and stopped further logic scanning of the routine.

Name	Usage	Value
ST_GVal		927
ST_GVal[0]		913
ST_GVal[1]		914
ST_GVal[2]		902
ST_GVal[3]		1008
ST_GVal[4]		921
ST_GVal[5]		936
ST_GVal[6]		909
ST_GVal[7]		907
ST_GVal[8]		922
ST_GVal[9]		923
ST_GVal[10]		1001
ST_GVal[11]		922
ST_GVal[12]		957
ST_GVal[13]		925
ST_GVal[14]		1223
ST_GVal[15]		922
ST_GVal[16]		909
ST_GVal[17]		924
ST_GVal[18]		921
ST_GVal[19]		950
ST_GVal[20]		911
ST_GVal[21]		912
ST_GVal[22]		910
ST_GVal[23]		919
ST_GVal[24]		927
ST_GVal[25]		909
ST_GVal[26]		918

Fig. 7.45. Several scan time values were captured after embedding more code elements.

7.6.5 Countermeasure Against Altered Scan Time Slice

When the system overhead time slice was modified to 80%, the introduced STC was able to detect it. The countermeasure solution monitored and captured 30 scan values, calculated the average. The average scan time, “STCapturedAvg”, of the compromised program was compared to the reference average to stop further scanning and warn staff. The modification of the time slice drastically affected the overall scan time, as shown in Fig. 7.46. The exploitation of the time slice raised the scan time to an average of 10734 microseconds.

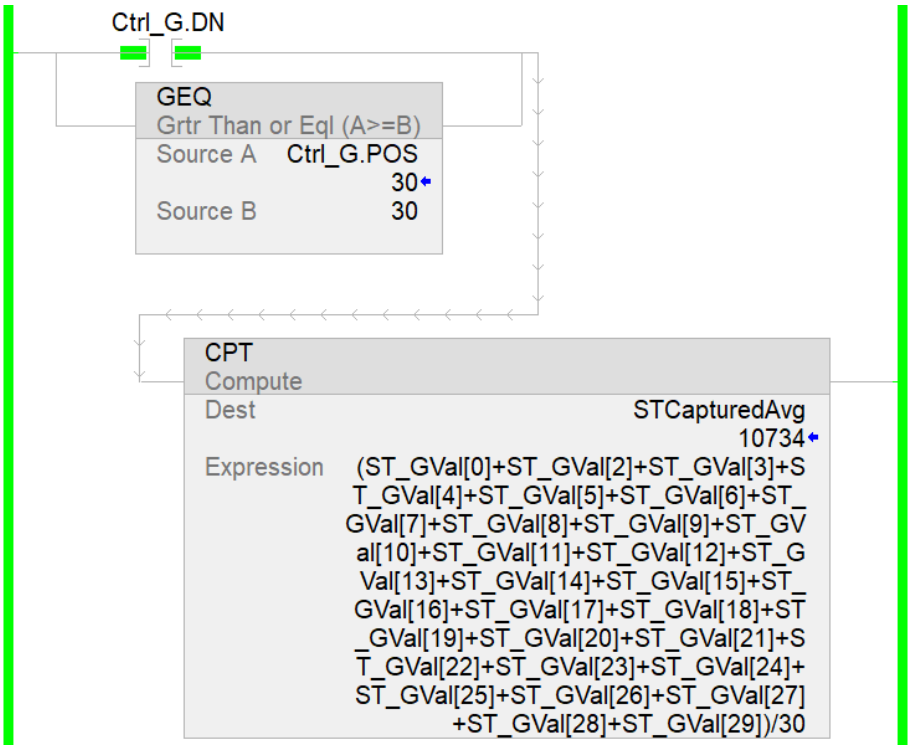


Fig. 7.46. The average of the scan time after TS% was increased.

7.6.6 Countermeasure Against Devices Behavior Attack

The stealthy attack scenarios that were applied to the pin code example were detected by this countermeasure solution based on using TON (timer) instruction and monitoring non-exclusive pin behavior state. The TON instruction was used to detect whether the pin was extending or retracting during an acceptable window of time, as shown in Fig. 7.47.

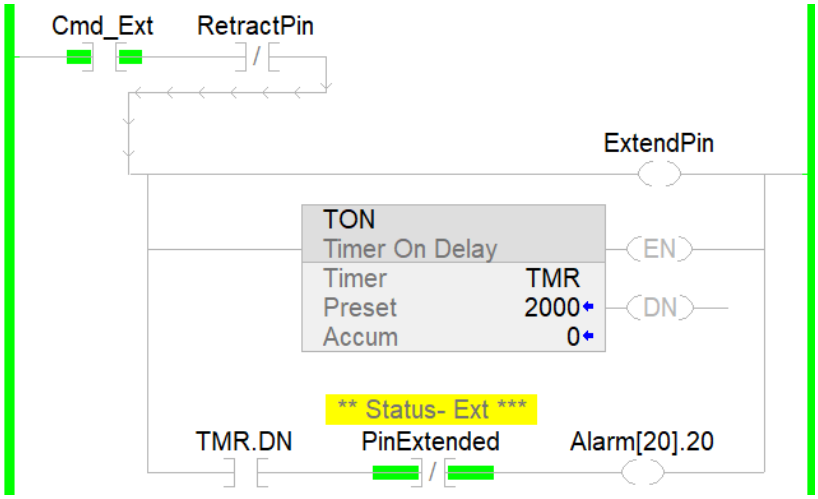


Fig. 7.47. Check if the pin was extended within 2000 milliseconds.

The code, in Fig. 7.47, also prevented commanding the pin to retract or extend at the same time. Another approach was used to detect a non-exclusive status indication. It checked that the pin status was either extended or retracted. When the pin indicated two statuses, extended and retracted, at the same time, as shown in Fig. 7.48 or Fig. 7.49, then operators were warned.

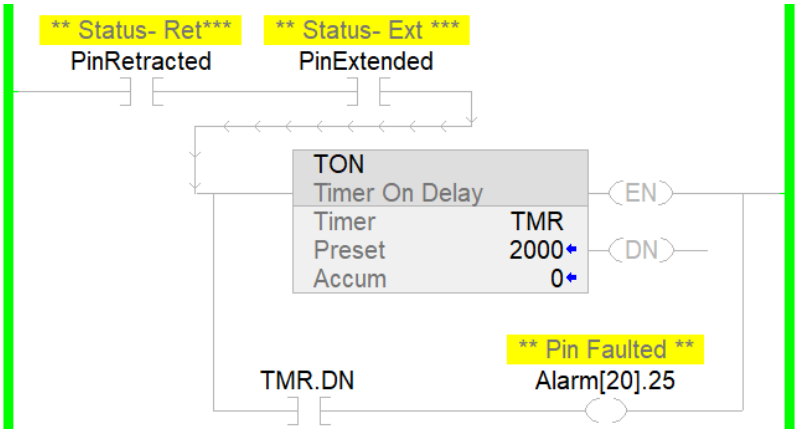


Fig. 7.48. Check if the pin was extended and retracted at the same time after 2000 milliseconds were elapsed.

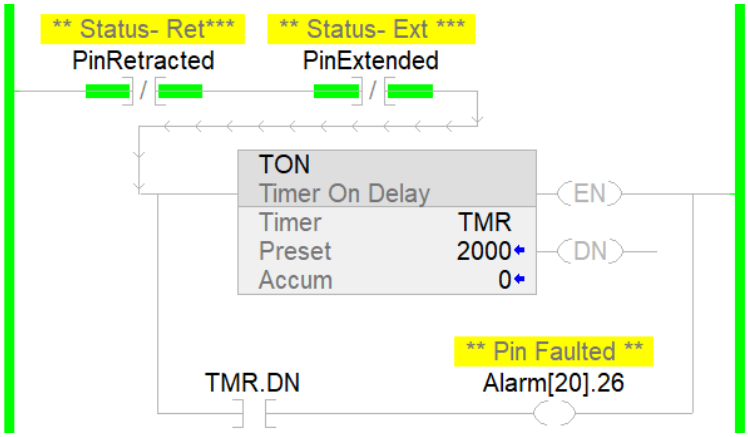


Fig. 7.49. Check if the pin was not extended and not retracted at the same time after 2000 milliseconds were elapsed.

7.7 Results and Discussion

A test bed with several stealthy attack scenarios was introduced and implemented to expose the vulnerabilities of the PLC code. Then countermeasure solutions were introduced and successfully detected and prevented those applied threats.

7.7.1 Scan Time Analysis

The countermeasure solutions were mainly based on several STC setups. They were implemented and successfully detected and prevented real attacks that could be embedded by adversaries within the PLC code. Six attack models were applied within the PLC code.
The attacks were able to exploit and compromise the PLC code without being detected by the controller. The PLC was defenseless in detecting or exposing the attacks or other similar vulnerabilities. It did not have any defense mechanisms to detect or even prevent such attacks or other vulnerabilities. The role of the controller was only to verify the validity of the code elements and syntaxes, as shown in Fig. 7.50.

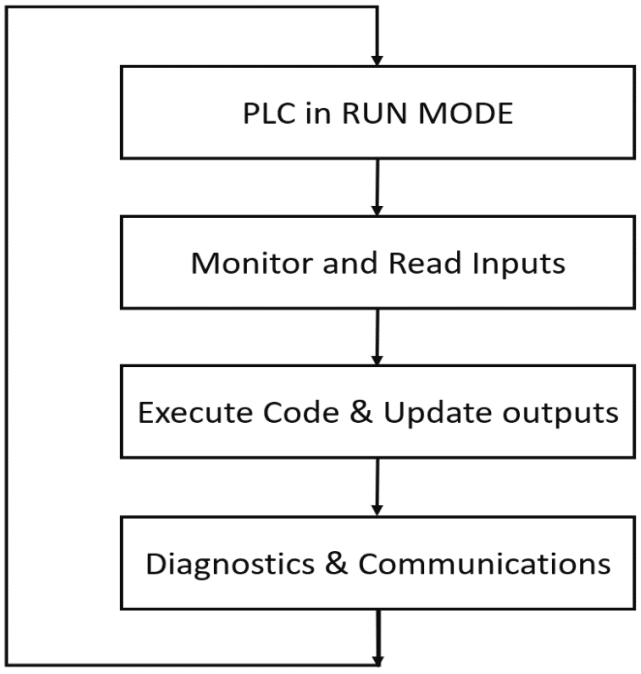


Fig. 7.50. A typical PLC scan cycle without any defense mechanisms.

The objective was to create a defensive code within the PLC code to detect and prevent attacks. The defensive mechanism was based on scan time of the targeted routine, as shown in Fig. 7.51.

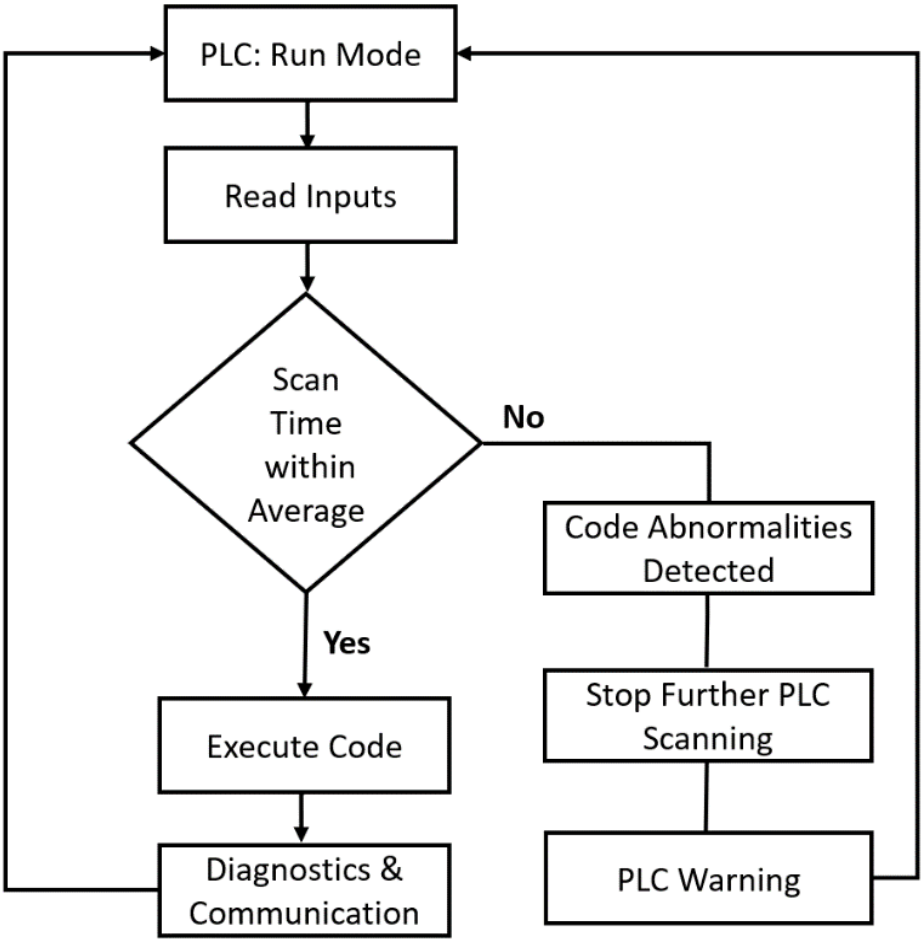


Fig. 7.51. Adding a defense mechanism to detect code abnormalities.

By analyzing the attacks that were deployed, we found that five out of them significantly affected the scan time. Therefore, the introduced countermeasures were designed to compare a stored, reliable average scan of a routine to any newly monitored ones. If discrepancies were to be found, the controller stopped further scanning and warned operators.

Real-time trends that captured and analyzed real-time scan time values of "Main" routine were conducted as well. They were used to validate and analyze the changes in the values of the scan time of "Main" Routine during a designated period.

The trends of the scan times were compared to the captured average values conducted in our experiments and they were within expectations.

Each trend captured more than 60 real-time values of the "Main" routine scan time, "Scan_Captured[0]". The values were captured and analyzed during a 400-millisecond interval.

Under normal conditions, the values of the captured scan time values were around 530 microseconds, as shown in Fig. 7.52. That was close to the previously calculated reference average, see Fig. 7.12.

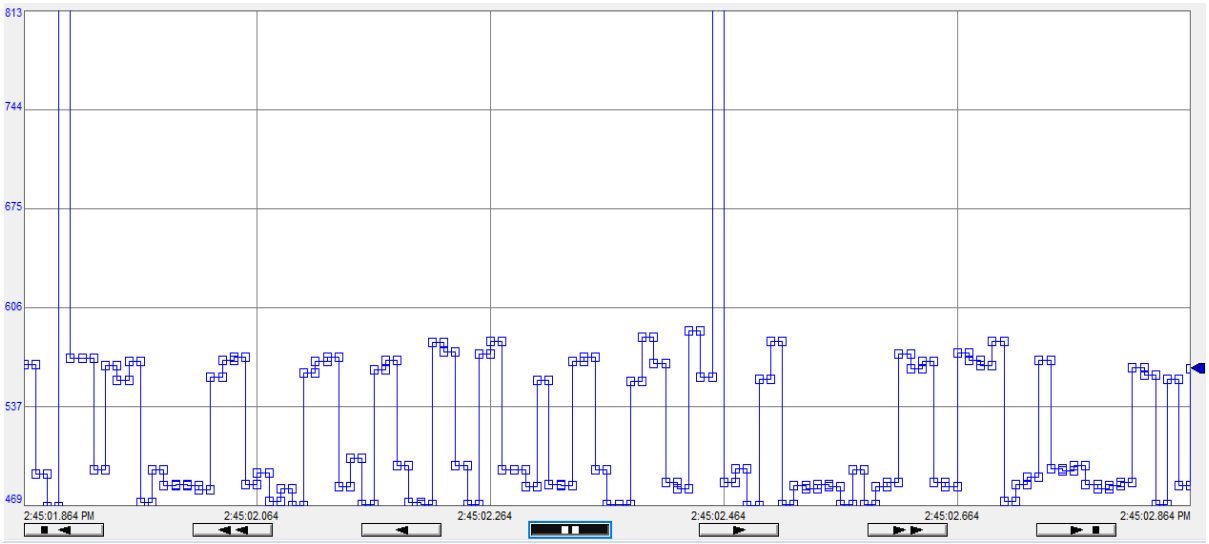


Fig. 7.52. The captured scan time values were around 530 microseconds during 1 millisecond duration.

For the compromised “Main” routine, affected by the “FOR” loops, the scan time values where around 1360 microseconds, as shown in Fig. 7.53. That was close to the reference average, previously calculated in our test bed experiment, see Fig. 7.37.

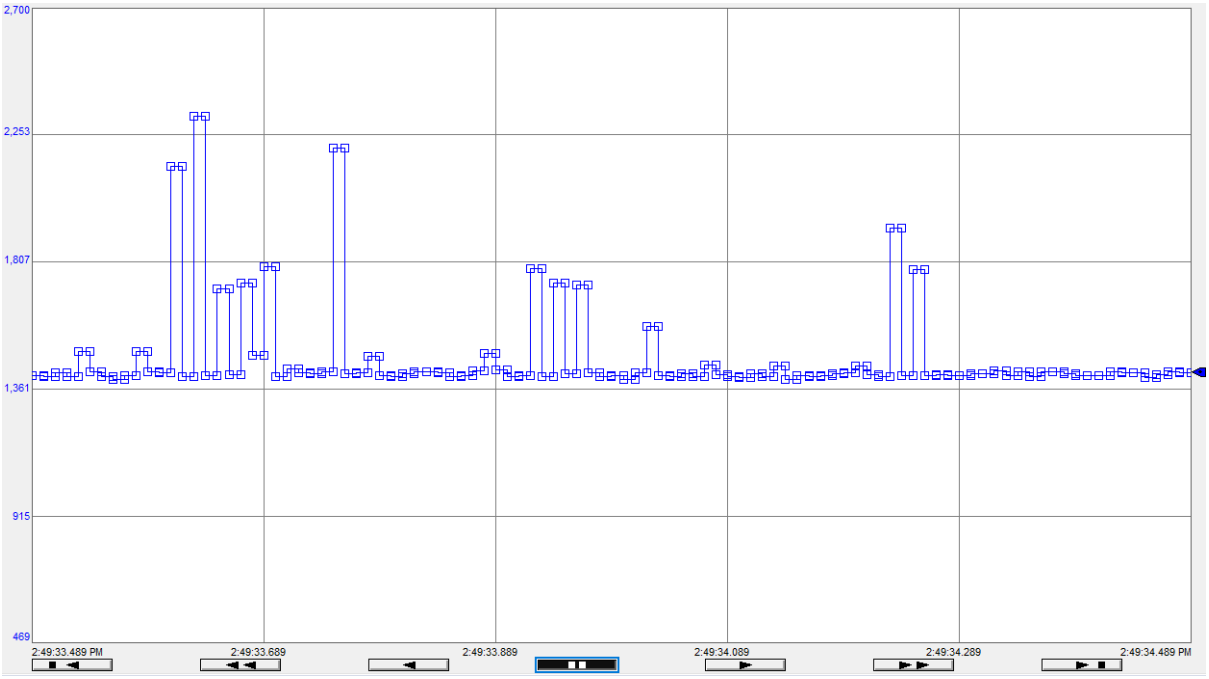


Fig. 7.53. The captured scan time values were around 1360 microseconds during 1 millisecond duration.

The scan time values of the compromised “Main” routine, attacked by skipping rungs, were around 8 microseconds, as shown in Fig. 7.54. The values are within the range of the reference average, see Fig. 7.42.

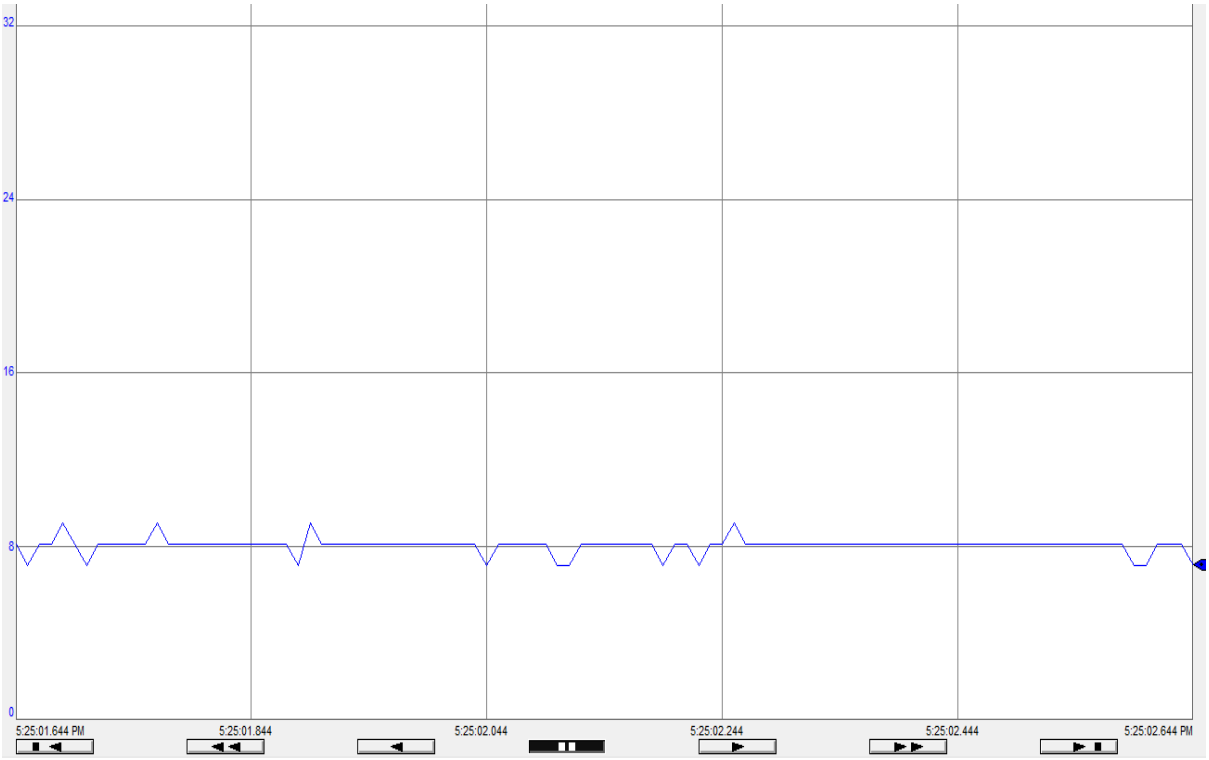


Fig. 7.54. The captured scan time values were around 8 microseconds during a 1 millisecond duration.

The Scan time values of the compromised “Main” routine, which had more code elements embedded to it, were around 930 microseconds, as shown in Fig. 7.55. That was close to the calculated reference value, as shown in Fig. 7.44.

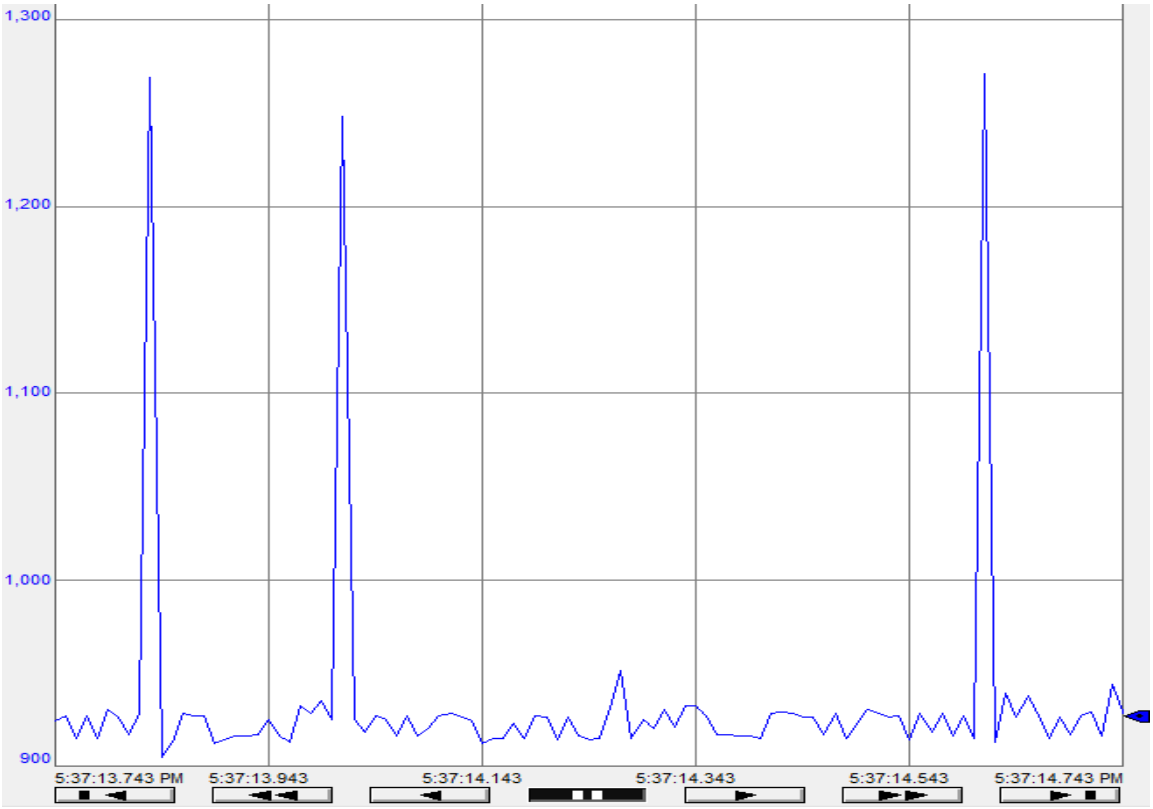


Fig. 7.55. The captured scan time values were around 930 microseconds.

The Scan time values of the compromised "Main" routine, that had portion of the code deleted, were around 18 microseconds, as shown in Fig. 7.56. The captured results shown in this trend during a 1 millisecond interval was close to the calculated reference value, as it was previously shown in Fig. 7.43.

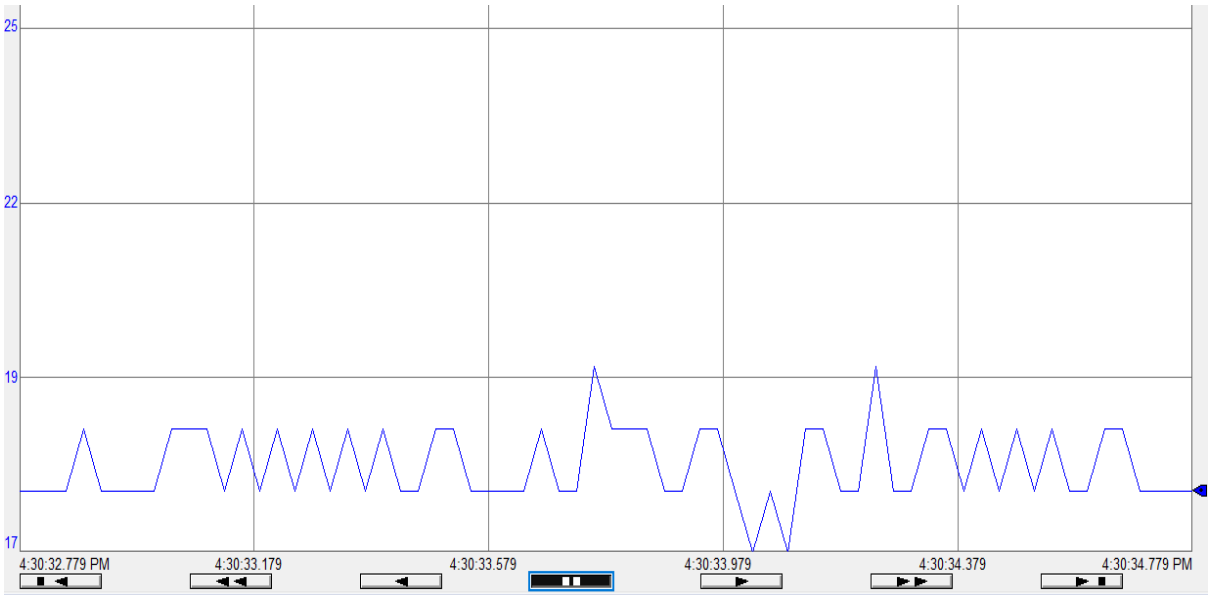


Fig. 7.56. The captured scan time values were around 18 microseconds during a 400-millisecond duration.

The Scan time values of the compromised code, “Main” routine which was affected by modifying the time slice to 80%, were around 10,000 microseconds, as shown in Fig. 7.57. That captured values showed in the real-time trend was close to the previously calculated reference, shown in Fig. 7.46. In this test the more percentage, 80%, of the time slice was given to tasks other than continuous tasks.

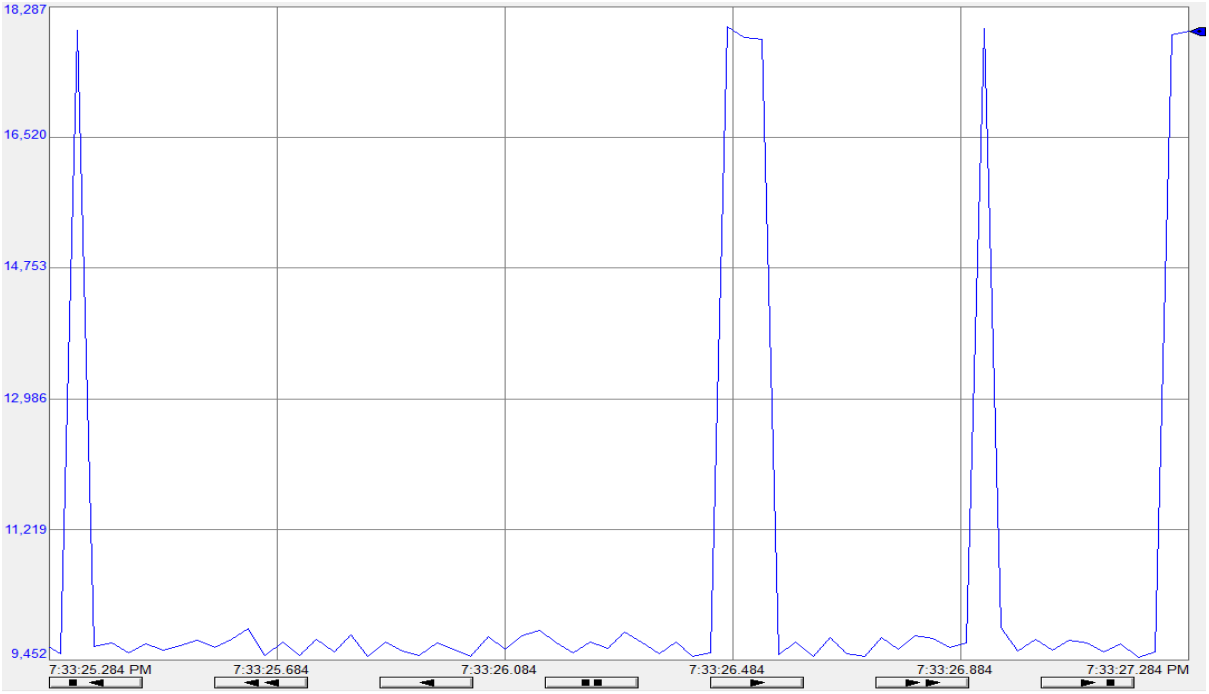


Fig. 7.57. The captured scan time values were around 10000 when the TS% was increased to 80%.

7.7.2 Analyzing Devices Deterioration and Tampering

For the sixth attack model that had to do with tampering with the field devices behavior, a suggested pin code setup was introduced and applied mitigate and prevent abnormalities. In addition to its role in detecting code abnormalities, the provided solution would be also applicable in detecting any physical deterioration of the pin or other similar devices The countermeasure solution was mainly based on two major factors. The first factor was based on a timer, “TON”. The timer was used to check the duration between sending a signal to the pin (commanding the pin to extend or retract) and the feedback coming from the pin (whether it was extended or retracted). Whenever the duration took more than the designated time (2000 milliseconds), an alarm was activated. The second important factor was to detect that there was always an exclusive status indicator of the pin reported to the PLC, either retracted or extended.

7.7.3 Limitations and Considerations

During the conducted experiment, monitoring and capturing the scan time of a PLC routine required novel techniques that had to be developed and implemented.

The following is a summary of the PLC limitations:

- The captured scan time values of a routine were inconsistent and not always the same per scan cycle. That was because they were captured in microseconds and under on-going different factors and conditions. Because of such discrepancies, our code did not depend on one captured value of a scan time but took the average of 30 values within a designated duration of time. To get more accurate average readings for real industrial environment, more data to be collected over more extended period.
- All the kinds of timers whether they are retentive (RTO) or non-retentive (TON and TOF-Timer OFF Delay) could not be utilized for timing a process that was faster than 1 millisecond. So, the timer instructions were useless when it comes to measuring the time of any code executions that were, typically, faster than 1 millisecond. So, provided timers were not useful in detecting scan time of code per scan cycle.
- The GuardLogix 5570 controller had no built-in instructions to capture the scan time of a specific routine. Because of such limitations the STC was developed and used.
- The only available feature that we were able to utilize in this test bed was the GSV instruction that is limited to capture the scan time of the “MainProgram”, i.e., the overall scan time of all the routines at once - as shown in Fig. 7.58 and Fig. 7.59. Relying on the overall scanning time was not applicable to certain scenarios especially if more accurate analysis and specific detection were needed for a particular routine or a task. Also, relying on the overall scan time of all routines was not always reliable and valid. For instance, when the TS% was increased to 80%, the overall scan time showed no abnormal scan time value, but the STC was able to capture the increase in the scan time whenever it affected a particular routine since it was monitoring the scan time at the routine level.

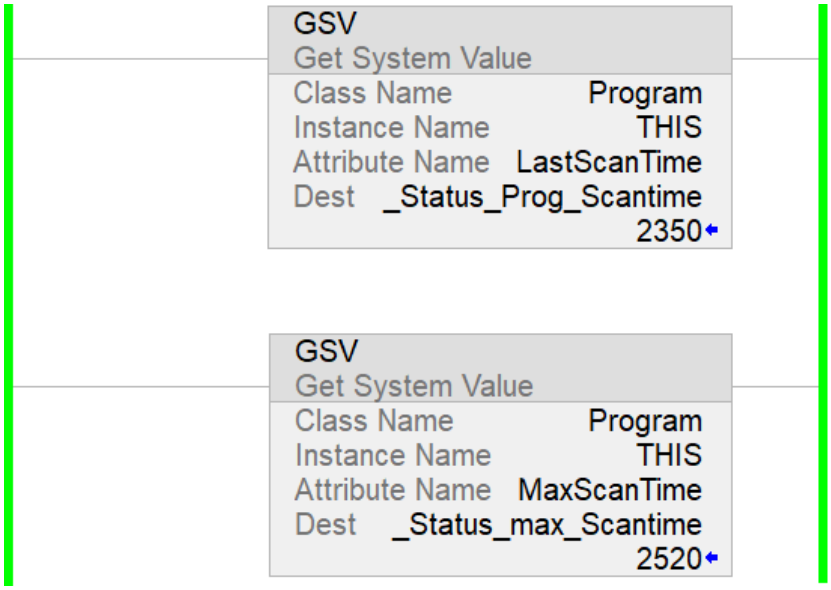


Fig. 7.58. Checking the scan time of the overall program from the logic code using GSV instruction.

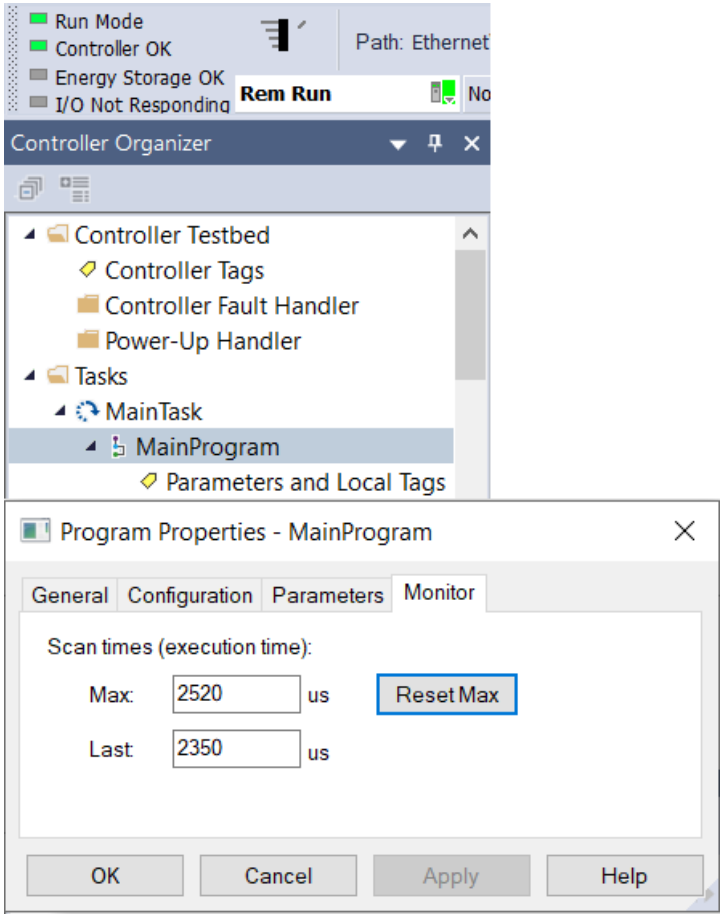


Fig. 7.59. Another way to check an overall scan time of the program during the experiment.

7.8 Conclusion

After setting up a real-time test bed based on ladder logic code, several vulnerabilities in the PLC code were exposed and exploited. The designated PLC was defenseless in detecting or protecting against any of the code exploitations. The exploitation of the code was achieved by developing and implementing six attack models that were deployed while the PLC code was running. The attacks were stealthy in compromising the targeted ladder logic code without being detected or prevented by the PLC though they modified the running code. All of the proposed attacks could be carried on and deployed to any other PLC based systems, causing critical consequences to associated ICS. The attacks used different techniques for different scenarios.

On the other hand, several countermeasures against those six attack models were introduced. The countermeasures were mainly based on two factors. The first one, STC, was based on capturing and analyzing the scan time of the running code of a particular routine. Several scan time values were captured under good, normal conditions and their average was calculated and used as a reference average. The reference average calculated by STC was used in ongoing comparison with every scan time value of a particular routine per PLC cycle. Whenever a discrepancy was found, the controller stopped further code execution and warned staff.

The second factor was about validating field devices behaviors, such as pins or clamps. Several vulnerabilities in the ladder logic code were exposed and a countermeasure, based on ladder logic code, against them was introduced. The countermeasure was able to detect and prevent devices abnormalities, whether it was because of a regular physical deterioration scenario or a cyber-attack.

The main challenge we faced in this experiment was to capture the time a PLC would spend in executing a routine per cycle. Capturing the scan time of a routine per cycle was critical to analyze any code modifications of abnormalities. A typical Rockwell PLC, like the one used in our test bed, could only provide the scan time of “MainProgram”, including the scan time of all routines. That was not useful in analyzing the scan time of a particular routine or code. Also, the experiment proved that the overall scan time of “MainProgram”, provided by GSV, was not accurate when the overhead time slice was drastically modified. To resolve the limitation of the GSV, a STC was introduced to capture the scan time value of any particular routine or code.

Chapter 8 Conclusions and Future Work

It is proven through this work that PLC codes are vulnerable, and our contributed work was capable to mitigate certain vulnerabilities within PLCs codes without the need of third-party solutions or external tools. PLCs, by design, do not have built in features that could enable them to detect malicious code attacks, manipulations, or undesired behaviors. PLCs typically rely on external security perimeters to reduce vulnerabilities and threats. So, adversaries who might get access to a PLC, could manipulate the code without being detected. Due to the limitations of the PLCs' OS and the absence of antivirus, unique code techniques were provided in this work. Those techniques could be embedded to any PLC code to enhance its security level.

They are focused on enhancing the security of the ladder logic code within a PLC to make it self-aware of any code vulnerabilities or suspicious code behavior. It provided code solutions that equipped PLCs with self-defense mechanisms to detect and prevent certain code threats, cyber-attacks, and inadvertent code practice.

This research work provided real-time novel countermeasures against exploitation of those vulnerabilities. The novel contribution against PLC code exploitation was addressed by developing several real-time techniques that could be embedded to PLC code to detect and prevent suspicious code modifications and behaviors within PLC routines. Those techniques were, mainly, based on developing a code, STC, that could capture, monitor, and analyze the scan time value during the execution of a particular code, routine, or overall program. The STC, a ladder logic-based code, was designed and implemented while the PLC was running without interrupting any running code.

While [13] claimed that monitoring the runtime of a payload program is impossible, the STC setup described in this thesis was able to monitor the time a PLC spent in executing a particular code and utilized it to detect code abnormal behaviors. The introduced STC setup was able to detect and prevent several attacks that targeted a PLC running code by monitoring the execution time of the targeted code. When the scan time of the compromised code took more or less than the preferred average, the PLC stopped scanning further code and warned staff.

The attacks used in our test bed were developed and deployed to a real-time ladder logic code, based on Rockwell Automation PLC. They used different scenarios and malicious techniques to expose and exploit PLC code vulnerabilities. Those cyber-attacks were successful and

stealth. They achieved their purpose of modifying and compromising the targeted PLC code without being detected and without interrupting any running code execution.

Some attacks were able to manipulate and fake the status of field devices through PLC code. Other attacks were able to overload, delete, or skip certain PLC code. Other stealthier attacks were able to suppress alarms.

This research also exposed some vulnerabilities in PLC alarms code, i.e., the ladder logic code responsible for raising or triggering alarm messages to alert staff and operators. Four real-time and stealth attacks were introduced and deployed targeting a running PLC alarms code. The attacks stealthily manipulated the PLC alarms code and suppressed them fooling staff that system has no faults or issues. The attacks were embedded to a running PLC code without being detected, that could cause significant damage if it were to be deployed to infrastructure or public facilities. Several effective countermeasure solutions against the described attacks were introduced: scan time code, heartbeat code, and verification guidelines of alarms and designated physical devices.

In addition to providing details of some major fundamental ladder logic code vulnerabilities threats, and actual documented incidents, general recommendations to enhance the security of PLCs' code were presented.

The main challenge faced in this work was in capturing the runtime execution of a particular code or the time a PLC would spend in executing a routine per cycle because typical PLCs are not capable of capturing time intervals in microseconds.

One of Our future goals is to expand our work to develop PLCs-aware code that detect code abnormalities and vulnerabilities. Currently, PLCs are only used as controller executers without detecting malicious code or attacks. Our provided STC solutions could be redesigned to detect more code abnormalities and vulnerabilities.

Moreover, the provided STC model would be a good starting point in forensic data analysis. The absence of forensic data is another challenging weakness of PLCs. By using our provided STC logic model, the logic would be able to have adequate and detailed forensics because of the logged captured scan times of PLC routines. Forensics collected directly from PLCs would help in uncovering the traces of certain malicious code that affect the scan time of a routine or a program as discussed in the work.

Another future work is to customize our provided STC solutions along with PLC suppressed alarms detection code to prevent similar Maroochy Shire cyber events from occurring against industrial companies [28] [8] [143]. Companies lack the proper means to verify scan time of individual routines or to detect code abnormalities. The provided solutions in this work could

be a simple code methodology that can be deployed to any PLC program to keep PLC code secure and less vulnerable.

Nevertheless, this work could be enhanced to get integrated with emerging technologies such as Artificial Intelligence (AI) or predictive maintenance. Monitoring and logging the behavior of PLC code through the provided STC solutions could be utilized to create valuable datasets. The datasets could be used by another third-party programs to suggest and predict typical possible vulnerabilities, threats, and proper countermeasure solutions of designated PLC-BS devices or PLC code that a company is operating or wants to add on.

Finally, it would be beneficial to make a detailed document that includes systematization and taxonomy of all attacks and vulnerabilities of PLC-BS. The document could be used in generating recommendations and guidelines to enhance ICS recommended standards. The document would be based on our provided work in addition to other emerging threats, vulnerabilities, and countermeasure solutions.

Chapter 9 References

[1] R. M. Lee, M. J. Assante and T. Conway, "Analysis of the cyber attack on the Ukrainian power grid. Defense use case," *Protocol TLP: White,* 2016.

[2] "Overload: Critical Lessons from 15 Years of ICS Vulnerabilities.," Fireeye.com, [Online]. Available: https://www2.fireeye.com/rs/848-DID-242/images/ics-vulnerability-trend-report-final.pdf. [Accessed 25 July 2019].

[3] Kaspersky Lab, [Online]. Available: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2016/07/07190426/KL_REPORT_ICS_Statistic_vulnerabilities.pdf..

[4] "Internet Security Threat Report," Symatec, April 2016. [Online]. Available: https://docs.broadcom.com/doc/istr-16-april-volume-21-en. [Accessed 02 02 2022].

[5] K. Sheridan, "Take (Industrial) Control: A Look at the 2018 ICS Threat Landscape," Informa PLC - Black Reading, 06 September 2018. [Online]. Available: https://www.darkreading.com/risk/take-industrial-control-a-look-at-the-2018-ics-threat-landscape. [Accessed 02 02 2022].

[6] K. a. A. A. Group, "State of Industrial Cybersecurity: survey by Kaspersky and ARC Advisory Group," Kaspersky Lab, 29 August 2019. [Online]. Available: https://ics-cert.kaspersky.com/publications/news/2019/08/29/survey-2019/. [Accessed 02 02 2022].

[7] E. Kovacs, "Some ICS Security Incidents Resulted in Injury, Loss of Life: Survey," Wired Business Media, 24 October 2019. [Online]. Available: https://www.securityweek.com/some-ics-security-incidents-resulted-injury-loss-life-survey. [Accessed 02 02 2022].

[8] S. M. Nabil Sayfayn, "Cybersafety Analysis of the Maroochy Shire Sewage Spill," Massachusetts Institute of Technology, May 2017. [Online]. Available: https://web.mit.edu/smadnick/www/wp/2017-09.pdf. [Accessed 02 02 2022].

[9] R. Langner, "Stuxnet: Dissecting a Cyberwarfare Weapon," *IEEE Secur. Priv.,* vol. 9, p. 49–51, 2011.

[10] J. Weiss, *Stuxnet: Cybersecurity Trojan Horse,* 2010.

[11] T. Nash, *Backdoors and Holes in Network Perimeters - A Case Study for Improving Your Control System Security,* Vols. vol 1.1,, 2005.

[12] Mcafee, "Mcafee.com," Mcafee, [Online]. Available: https://www.mcafee.com/enterprise/en-ca/security-awareness/ransomware/what-is-stuxnet.html. [Accessed 02 02 2022].

[13] H. Yang, L. Cheng and M. C. Chuah, "Detecting Payload Attacks on Programmable Logic Controllers (PLCs)," in *IEEE Conference on Communications and Network Security (CNS)*, 2018.

[14] E. BYRES, "PLC security risk: controller operating systems," Tofino industrial security solution, 05 10 2011. [Online]. Available: https://www.tofinosecurity.com/blog/plc-security-risk-controller-operating-systems. [Accessed 02 02 2022].

[15] M. CHALFANT, "'Crash Override' malware heightens fears for US electric grid," The Hill, 15 6 2017. [Online]. Available: https://thehill.com/policy/cybersecurity/337877-crash-override-malware-heightens-fears-for-us-electric-grid. [Accessed 02 02 2022].

[16] ISA.org, [Online]. Available: https://www.isa.org/standards-publications/isa-publications/intech-magazine/2012/october/system-integration-iec-61131-3-industrial-control-programming-standard-advancements/. [Accessed 25 July 2019].

[17] Rockwell, "Rockwell Automation," Rockwell Automation, May 2018. [Online]. Available: https://literature.rockwellautomation.com/idc/groups/literature/documents/rm/1756-rm099_-en-p.pdf. [Accessed 02 02 2022].

[18] L. H. Eccles, "A smart sensor bus for data acquisition," 1998.

[19] A. Serhane, M. Shraif, H. Chehadi, A. Harb and A. Mohsen, "Optimizing solar systems using DeviceNET," in *2017 29th International Conference on Microelectronics (ICM)*, 2017.

[20] I. C. -. K. Lab, "WannaCry on industrial networks: error correction," Kaspersky Lab, 22 June 2017. [Online]. Available: https://ics-cert.kaspersky.com/publications/reports/2017/06/22/wannacry-on-industrial-networks/. [Accessed 02 02 2022].

[21] I.-C. Kaspersky, "Threat landscape for industrial automation systems. Statistics for H1 2021," Kaspersky Lab, 09 September 2021. [Online]. Available: https://ics-cert.kaspersky.com/publications/reports/2021/09/09/threat-landscape-for-industrial-automation-systems-statistics-for-h1-2021/. [Accessed 02 02 2022].

[22] H. M. V. D. S. a. C. F. JOSHUA RAY, "Cyber Threat Intelligence Report Volume 1," Accenture, 15 JULY 2021. [Online]. Available: https://www.accenture.com/us-en/insights/security/cyber-threat-intelligence-report-2021. [Accessed 02 02 2022].

[23] E. Kovacs, "Ransomware Hit SCADA Systems at 3 Water Facilities in U.S.," Wired Business Media, 15 October 2021. [Online]. Available: https://www.securityweek.com/ransomware-hit-scada-systems-3-water-facilities-us. [Accessed 02 02 2022].

[24] A. Press, "Small Kansas Water Utility System Hacking Highlights Risks," Wired Business Media, 13 April 2021. [Online]. Available: https://www.securityweek.com/small-kansas-water-utility-system-hacking-highlights-risks. [Accessed 02 02 2022].

[25] K. I. CERT, "Percentage of computers on which malicious objects were blocked," Kaspersky ICS CERT, 24 April 2020. [Online]. Available: https://ics-cert.kaspersky.com/publications/reports/2020/04/24/threat-landscape-for-industrial-automation-systems-overall-global-statistics-h2-2019/. [Accessed 02 02 2022].

[26] K. I. CERT, "Threat landscape for industrial automation systems. 2019 Report at a glance," Kaspersky Lab, 24 April 2020. [Online]. Available: https://ics-cert.kaspersky.com/publications/reports/2020/04/24/threat-landscape-for-industrial-automation-systems-2019-report-at-a-glance/. [Accessed 02 02 2022].

[27] I. C. -. K. Lab, "Threat landscape for industrial automation systems. Vulnerabilities identified in 2019," Kaspersky Lab, 2020 April 24. [Online]. Available: https://ics-cert.kaspersky.com/publications/reports/2020/04/24/threat-landscape-for-industrial-automation-systems-vulnerabilities-identified-in-2019/. [Accessed 02 02 2022].

[28] J. S. a. M. Miller, "Lessons learned from the Maroochy water," *Critical Infrastructure Protection, Springer,* p. 73–82, 2007.

[29] J. Riley, "Mysterious 08 Turkey Pipeline blast opened new cyberwar," Bloomberg, 10 December 2014. [Online]. Available: https://www.bloomberg.com/news/articles/2014-12-10/mysterious-08-turkey-pipeline-blast-opened-new-cyberwar. [Accessed 02 02 2022].

[30] B. Gourley, "Most violent cyber attack noted to date: 2008," CTOVision, 13 December 2014. [Online]. Available: www.ctovision.com/violent-cyber-attack-noted-date-2008-pipeline-explosion-caused-remote-hacking/. [Accessed 02 02 2022].

[31] Newswire, " 2008 Turkish Oil Pipeline explosion," Newswire, 10 December 2014. [Online]. Available: https://www.homelandsecuritynewswire.com/dr20141217-2008-turkish-oil-pipeline-explosion-may-have-been-stuxnet-precursor. [Accessed 02 02 2022].

[32] K. Zetter, "An unprecedented look at Stuxnet: The world's first," Wired, 03 November 2014. [Online]. Available: https://www.wired.com/2014/11/countdown-to-zero-day-stuxnet/. [Accessed 02 02 2022].

[33] ICS-CERT, "Advisory (ICSA-10-238-01B): Stuxnet malware mitigation," ICS-CERT, 15 September 2010. [Online]. Available: https://www.cisa.gov/uscert/ics/advisories/ICSA-10-238-01B. [Accessed 02 02 2022].

[34] Langner, "To Kill a Centrifuge," Langner, November 2013. [Online]. Available: https://www.langner.com/wp-content/uploads/2017/03/to-kill-a-centrifuge.pdf. [Accessed 02 02 2022].

[35] ICS-CERT, "ICS Advisory (ICSA-11-041-01A) McAfee Night Dragon Report (Update A)," ICS-CERT, 11 February 2011. [Online]. Available: https://www.cisa.gov/uscert/ics/advisories/ICSA-11-041-01A. [Accessed 02 02 2022].

[36] ICS-CERT, "ICS-CERT," Advisory (ICSA-11-041-01A): McAfee Night Dragon report, 11 February 2011. [Online]. Available: https://www.cisa.gov/uscert/ics/advisories/ICSA-11-041-01A. [Accessed 02 02 2022].

[37] K. Zetter, "Attackers stole certificate from Foxconn to hack Kaspersky with Duqu 2.0," Wired, 15 June 2015. [Online]. Available: https://www.wired.com/2015/06/foxconn-hack-kaspersky-duqu-2/. [Accessed 02 02 2022].

[38] ICS-CERT, "ICS Joint Security Awareness Report (JSAR-12-151-01A) sKyWIper/Flame Information-Stealing Malware (Update A)," ICS-CERT, 05 June 2012. [Online]. Available: https://www.cisa.gov/uscert/ics/jsar/JSAR-12-151-01A. [Accessed 02 02 2022].

[39] [. A. Gostev, *The Flame: Questions and Answers*, 2012.

[40] ICS-CERT, "ICS Joint Security Awareness Report (JSAR-12-222-01) Gauss Information-Stealing Malware," ICS-CERT, 09 August 2012. [Online]. Available: https://www.cisa.gov/uscert/ics/jsar/JSAR-12-222-01. [Accessed 02 02 2022].

[41] ICS-CERT, "Alert (AA21-201A) Chinese Gas Pipeline Intrusion Campaign, 2011 to 2013," ICS-CERT, 20 July 2021. [Online]. Available: https://www.cisa.gov/uscert/ncas/alerts/aa21-201a. [Accessed 02 02 2022].

[42] ICS-CERT, "ICS Joint Security Awareness Report (JSAR-12-241-01B), Shamoon/DistTrack Malware (Update B)," ICS-CERT, 16 October 16 2012. [Online]. Available: https://www.cisa.gov/uscert/ics/jsar/JSAR-12-241-01B. [Accessed 02 02 2022].

[43] T. K. a. S. M. Shimon Prokupecz, "Former official: Iranians hacked into New York dam," CNN, 22 December 2015. [Online]. Available: https://www.cnn.com/2015/12/21/politics/iranian-hackers-new-york-dam/index.html. [Accessed 02 02 2022].

[44] ICS-CERT, "ICS Advisory (ICSA-14-178-01), ICS Focused Malware," ICS-CERT, 30 June 2014. [Online]. Available: https://www.cisa.gov/uscert/ics/advisories/ICSA-14-178-01. [Accessed 02 02 2022].

[45] BBC, "Hack attack causes 'massive damage' at steel works," BBC, 22 December 2014. [Online]. Available: https://www.bbc.com/news/technology-30575104. [Accessed 02 02 2022].

[46] K. ZETTER, "A Cyberattack Has Caused Confirmed Physical Damage for the Second Time Ever," Wired, 08 January 2015. [Online]. Available: https://www.wired.com/2015/01/german-steel-mill-hack-destruction/. [Accessed 02 02 2022].

[47] ICS-CERT, "ICS Alert (ICS-ALERT-14-281-01E, Ongoing Sophisticated Malware Campaign Compromising ICS (Update E)," ICS-CERT, 10 December 2014. [Online]. Available: https://www.cisa.gov/uscert/ics/alerts/ICS-ALERT-14-281-01B. [Accessed 02 02 2022].

[48] M. Mimoso, *BlackEnergy malware used in attacks against industrial control systems*, 2014.

[49] D. Reading, "'Energetic' Bear Under The Microscope," Dark Reading, 01 August 2014. [Online]. Available: https://www.darkreading.com/attacks-breaches/-energetic-bear-under-the-microscope. [Accessed 02 02 2022].

[50] K. ZETTER, "Everything We Know About Ukraine's Power Plant Hack," Wired, 26 January 2016. [Online]. Available: https://www.wired.com/2016/01/everything-we-know-about-ukraines-power-plant-hack/. [Accessed 02 02 2022].

[51] N. Zinets, "Ukraine hit by 6,500 hack attacks, sees Russian 'cyberwar'," Reuters, 29 December 2016. [Online]. Available: https://www.reuters.com/article/us-ukraine-crisis-cyber-idUSKBN14I1QC. [Accessed 02 02 2022].

[52] E. Kovacs, "Attackers Alter Water Treatment Systems in Utility Hack: Report," SecurityWeek, 22 March 2016. [Online]. Available: https://www.securityweek.com/attackers-alter-water-treatment-systems-utility-hack-report. [Accessed 02 02 2022].

[53] S. Chan, "Cyberattacks Strike Saudi Arabia, Harming Aviation Agency," The New York Times, 1 December 2016. [Online]. Available: https://www.nytimes.com/2016/12/01/world/middleeast/saudi-arabia-shamoon-attack.html. [Accessed 02 02 2022].

[54] CISA, "Alert (TA17-163A) CrashOverride Malware," CISA, 12 June 2017. [Online]. Available: https://www.cisa.gov/uscert/ncas/alerts/TA17-163A. [Accessed 02 02 2022].

[55] D. V. a. S. Young, "White House blames Russia for 'reckless' NotPetya cyber attack," Reuters, 15 FEBRUARY 2014. [Online]. Available: https://www.reuters.com/article/us-britain-russia-cyber-usa/white-house-blames-russiafor-reckless-notpetya-cyber-attack-idUSKCN1FZ2UJ. [Accessed 02 02 2022].

[56] "Software: NotPetya," The MITRE Corporation, [Online]. Available: https://collaborate.mitre.org/attackics/index.php/Software/S0006. [Accessed 02 02 2022].

[57] K. C. a. R. G. David Voreacos, "Merck Cyberattack's $1.3 Billion Question: Was It an Act of War?," Bloomberg, 3 December 2019. [Online]. Available: https://www.bloomberg.com/news/features/2019-12-03/merck-cyberattack-s-1-3-billion-question-was-it-an-act-of-war. [Accessed 02 02 2022].

[58] T. H. Team, "Dragonfly: Western energy sector targeted by sophisticated attack group," Broadcom, 20 October 2017. [Online]. Available: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/dragonfly-energy-sector-cyber-attacks. [Accessed 02 02 2022].

[59] R. Hackett, "Hackers Have Penetrated Energy Grid, Symantec Warns," Fortune, 06 September 2017. [Online]. Available: https://fortune.com/2017/09/06/hack-energy-grid-symantec/. [Accessed 02 02 2022].

[60] ICS-CERT, "ICS Advisory (ICSA-18-107-02) Schneider Electric Triconex Tricon (Update B)," CISA, 17 April 2018. [Online]. Available: https://www.cisa.gov/uscert/ics/advisories/ICSA-18-107-02. [Accessed 02 02 2022].

[61] D. Inc, "TRISIS Malware: Analysis of Safety System Targeted Malware," Dragos, 13 12 2017. [Online]. Available: https://www.dragos.com/wp-content/uploads/TRISIS-01.pdf. [Accessed 02 02 2022].

[62] M. ATT&CK, "Software: Triton, TRISIS, HatMan," The MITRE Corporation, 11 October 2021. [Online]. Available: https://collaborate.mitre.org/attackics/index.php/Software/S0013. [Accessed 02 02 2022].

[63] I. C. -. K. Lab, "Industrial solutions may be affected by Spectre and Meltdown vulnerabilities," Kaspersky Lab, 12 January 2018. [Online]. Available: https://ics-cert.kaspersky.com/publications/news/2018/01/12/spectre-meltdown/. [Accessed 02 02 2022].

[64] J. Horn, "Reading privileged memory with a side-channel," Project Zero team at Google, 03 January 2018. [Online]. Available: https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html. [Accessed 02 02 2022].

[65] CISA, "Alert (AA18-337A) SamSam Ransomware," CISA, 03 December 2018. [Online]. Available: https://www.cisa.gov/uscert/ncas/alerts/AA18-337A. [Accessed 02 02 2022].

[66] P. Paganini, "Xenotime threat actor now is targeting Electric Utilities in US and APAC," Security Affairs, 15 June 2019. [Online]. Available: https://securityaffairs.co/wordpress/87125/breaking-news/xenotime-targets-us-apac.html. [Accessed 02 02 2022].

[67] "Threat Proliferation in ICS Cybersecurity: XENOTIME Now Targeting Electric Sector, in Addition to Oil and Gas," Dragos Inc, 14 06 2019. [Online]. Available: https://www.dragos.com/blog/industry-news/threat-proliferation-in-ics-cybersecurity-xenotime-now-targeting-electric-sector-in-addition-to-oil-and-gas/. [Accessed 02 02 2022].

[68] P. R. a. V. V. Warren Mercer, "PoetRAT: Python RAT uses COVID-19 lures to target Azerbaijan public and private sectors," Talos, 16 April 2020. [Online]. Available: https://blog.talosintelligence.com/2020/04/poetrat-covid-19-lures.html. [Accessed 10 March 2022].

[69] CISA, "FBI Releases Indicators of Compromise for RagnarLocker Ransomware," CIS, 08 March 2022. [Online]. Available: https://www.cisa.gov/uscert/ncas/current-activity/2022/03/08/fbi-releases-indicators-compromise-ragnarlocker-ransomware. [Accessed 10 03 2022].

[70] I. C. C. C. -. IC3, "RagnarLocker Ransomware Indicators of Compromise," Internet Crime Complaint Center - IC3, 07 March 2022. [Online]. Available: https://www.ic3.gov/Media/News/2022/220307.pdf. [Accessed 10 03 2022].

[71] K. Sheridan, "Colonial Pipeline Cyberattack: What Security Pros Need to Know," DarkReading, 10 May 2021. [Online]. Available: https://www.darkreading.com/operations/colonial-pipeline-cyberattack-what-security-pros-need-to-know. [Accessed 02 02 2022].

[72] FBI, "FBI Statement on Network Disruption at Colonial Pipeline," FBI National Press Office, 09 May 2021. [Online]. Available: https://www.fbi.gov/news/pressrel/press-releases/fbi-statement-on-network-disruption-at-colonial-pipeline. [Accessed 02 02 2022].

[73] J. Vijayan, "Attackers Exploit Log4j Flaws in Hands-on-Keyboard Attacks to Drop Reverse Shells," DarkReading, 04 January 2022. [Online]. Available: https://www.darkreading.com/application-security/attackers-using-log4j-flaws-in-hands-on-keyboard-attacks-to-drop-reverse-shells. [Accessed 02 02 2022].

[74] M. T. I. C. (MSTIC), "Guidance for preventing, detecting, and hunting for exploitation of the Log4j 2 vulnerability," Microsoft, 11 December 2021. [Online]. Available: https://www.microsoft.com/security/blog/2021/12/11/guidance-for-preventing-detecting-and-hunting-for-cve-2021-44228-log4j-2-exploitation/. [Accessed 02 02 2022].

[75] I. C.-. Kaspersky, "Threat landscape for industrial automation systems. Statistics for H2 2021," Kaspersky Lab, 03 March 2022. [Online]. Available: https://ics-cert.kaspersky.com/publications/reports/2022/03/03/threat-landscape-for-industrial-automation-systems-statistics-for-h2-2021/. [Accessed 10 March 2022].

[76] I. Security, "X-Force Threat Intelligence Index 2022," IBM Security, February 2022. [Online]. Available: https://www.ibm.com/downloads/cas/ADLMYLAZ. [Accessed 10 March 2022].

[77] T. Micro, "One Flaw too Many: Vulnerabilities in SCADA Systems," Trend Micro, 16 December 2019. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/one-flaw-too-many-vulnerabilities-in-scada-systems. [Accessed 02 02 2022].

[78] A. Serhane, M. Raad, R. Raad and W. Susilo, "PLC Code-Level Vulnerabilities," in *2018 International Conference on Computer and Applications (ICCA)*, 2018.

[79] A. Serhane, M. Raad, R. Raad and W. Susilo, "Programmable logic controllers based systems (PLC-BS): vulnerabilities and threats," *SN Appl. Sci.*, vol. 1, 2019.

[80] R. Spenneberg, M. Brüggemann and H. Schwartke, *PLC-blaster: A worm living solely in the PLC.*

[81] J. Klick, S. Lau, D. Marzin, J.-O. Malchow and V. Roth, *Internet-facing PLCs -A New Back Orifice.*

[82] S. E. Valentine, "PLC Code Vulnerabilities Through SCADA Systems," USA, 2013.

[83] S. Kalle, N. Ameen, H. Yoo and I. Ahmed, "CLIK on PLCs! Attacking Control Logic with Decompilation and Virtual PLC," in *Proceedings 2019 Workshop on Binary Analysis Research*, Reston, 2019.

[84] N. M. J. M. G. R. I. M.-M. Andres Robles-Durazno, "PLC memory attack detection and response in a clean water supply system," *International Journal of Critical Infrastructure Protection*, vol. 26, p. 100300, 2019.

[85] S. &. D. S. &. Y. H. &. A. I. &. R. V. Senthivel, "Denial of engineering operations attacks in industrial control systems," in *18th ACM Conf. Data Appl. Secur. Privacy*, 2018.

[86] L. D. a. M. L. C. Lei, "The spear to break the security wall of S7CommPlus," Black Hat, 2017. [Online]. Available: https://www.blackhat.com/docs/eu-17/materials/eu-17-Lei-The-Spear-To-Break%20-The-Security-Wall-Of-S7CommPlus-wp.pdf. [Accessed 2022 02 02].

[87] W. Alsabbagh and P. Langendorfer, "A stealth program injection attack against S7-300 PLCs," in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, 2021.

[88] W. Alsabbagh and P. Langendorfer, "A control injection attack against S7 PLCs -manipulating the decompiled code," in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, 2021.

[89] W. Alsabbagh and P. Langendorfer, "Patch now and attack later - exploiting S7 PLCs by time-of-day block," in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2021.

[90] Y. Hyunguk and A. Irfan, "Control logic injection attacks on industrial control systems," in *ICT Systems Security and Privacy Protection*, Cham, Springer International Publishing, 2019, p. 33–48.

[91] S. McLaughlin and P. McDaniel, "SABOT: Specification-based payload generation for programmable logic controllers," in *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, New York, New York, USA, 2012.

[92] F. B. M. H. C. A.-R. S. O. M. S. A. Z. Luis A. Garcia, "Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit," in *Annual Network and Distributed System Security Symposium (NDSS 2017)*, San Diego, California, 2017.

[93] A. A. a. M. Hashemi, "Ghost in the PLC: Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack," Black Hat, 2016. [Online]. Available: https://www.blackhat.com/docs/eu-16/materials/eu-16-Abbasi-Ghost-In-The-PLC-Designing-An-Undetectable-Programmable-Logic-Controller-Rootkit-wp.pdf. [Accessed 2022 02 02].

[94] O. Pavlovic and H.-D. Ehrich, "Model checking PLC software written in function block diagram," in *2010 Third International Conference on Software Testing, Verification and Validation*, 2010.

[95] R. Sun, A. Mera, L. Lu and D. Choffnes, "SoK: Attacks on industrial control logic and formal verification-based defenses," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021.

[96] D. Darvas, E. Blanco Vinuela and I. Majzik, "A formal specification method for PLC-based applications," 2015.

[97] D. Darvas, B. Fernández Adiego, A. Vörös, T. Bartha, E. Blanco Viñuela and V. M. González Suárez, "Formal verification of complex properties on PLC programs," in *Formal Techniques for Distributed Objects, Components, and Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, p. 284–299.

[98] D. Darvas, I. Majzik and E. Blanco Viñuela, "Formal verification of safety PLC based control software," in *Lecture Notes in Computer Science*, Cham, Springer International Publishing, 2016, p. 508–522.

[99] V. Gourcuff, O. De Smet and J.-M. Faure, "Efficient representation for formal verification of PLC programs," in *2006 8th International Workshop on Discrete Event Systems*, 2006.

[100] B. Fernandez Adiego, D. Darvas, E. B. Vinuela, J.-C. Tournier, S. Bliudze, J. O. Blech and V. M. Gonzalez Suarez, "Applying model checking to industrial-sized PLC programs," *IEEE Trans. Industr. Inform.*, vol. 11, p. 1400–1410, 2015.

[101] D. F. Bender, B. Combemale, X. Crégut, J. M. Farines, B. Berthomieu and F. Vernadat, "Ladder metamodeling and PLC program validation through time Petri nets," in *Model Driven Architecture – Foundations and Applications*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, p. 121–136.

[102] E. Brinksma and A. Mader, "Verification and optimization of a PLC control schedule," in *SPIN Model Checking and Software Verification*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, p. 73–92.

[103] S. Biallas, J. Brauer and S. Kowalewski, "Arcade.PLC: a verification platform for programmable logic controllers," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE 2012*, New York, New York, USA, 2012.

[104] L. Garcia, S. Mitsch and A. Platzer, "HyPLC: Hybrid programmable logic controller program translation for verification," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, New York, NY, USA, 2019.

[105] L. H. Newman, "An Operating System Bug Exposes 200 Million Critical Devices," Wired, 29 July 2019. [Online]. Available: https://www.wired.com/story/vxworks-vulnerabilities-urgent11/. [Accessed 02 02 2022].

[106] Amris, "11 zero day vulnerabilities impacting billions of mission-critical devices.," Amris, 14 December 2020. [Online]. Available: https://www.armis.com/research/urgent11/. [Accessed 02 02 2022].

[107] C. Cimpanu, "NUCLEUS:13 vulnerabilities impact Siemens medical & industrial equipment," Recorded Future, 09 November 2021. [Online]. Available: https://therecord.media/nucleus13-vulnerabilities-impact-siemens-medical-industrial-equipment/. [Accessed 02 02 2022].

[108] M. L. C. E. Falliere N, "W32.stuxnet dossier," Symantec, 2011. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf. [Accessed 25 July 2019].

[109] M. P, "Stuxnet-like attacks beckon as 50 new Scada threats discovered," April 21 2011. [Online]. Available: http://www.v3.co.uk/v3-uk/news/2045556/stuxnet-attacks-beckon-scada-threatsdiscovered.

[110] J. Leonard, D. Kundaliya, D. Berman, C. Staff, J. Oetjen, I. Hall, A. Filev, P. Cochrane and D. Trott, *Computing.*

[111] K. Lab, "Kaspersky Industrial CyberSecurity: Solution Overview 2018," Kaspersky Lab ICS CERT, 2018. [Online]. Available: https://media.kaspersky.com/en/business-security/enterprise/KICS-Solution-Overview-EN.pdf. [Accessed 02 02 2022].

[112] B. Merino, "Modbus Stager: Using PLCs as a payload/shellcode distribution system," 13 December 2016. [Online]. Available: https://www.shelliscoming.com/2016/12/modbus-stager-using-plcs-as.html. [Accessed 2022 02 02].

[113] T. M. B. R. D. R. W. Gao, "SCADA Control System Command and Response Injection and Intrusion Detection," pp. 1-9, 2010.

[114] K. M. S. S. H. Hui, "Vulnerability analysis of S7 PLCs: Manipulating the security mechanism," *Int. J. Crit. Infrastructure Protection*, vol. 35, 2021.

[115] S. Amin, X. Litrico, S. Sastry and A. M. Bayen, "Cyber security of water SCADA systems—part I: Analysis and experimentation of stealthy deception attacks," *IEEE Trans. Control Syst. Technol.*, vol. 21, p. 1963–1970, 2013.

[116] H. G. a. N. Tippenhauer, "HAMIDS: hierarchical monitoring intrusion detection system for industrial control systems," *Syst. Secur. Privacy*, pp. 103-111, 2016.

[117] C.-W. Ten, J. Hong and C.-C. Liu, "Anomaly detection for cybersecurity of the substations," *IEEE Trans. Smart Grid*, vol. 2, p. 865–873, 2011.

[118] U. K. Premaratne, J. Samarabandu, T. S. Sidhu, R. Beresh and J.-C. Tan, "An intrusion detection system for IEC61850 automated substations," *IEEE trans. power deliv.*, vol. 25, p. 2376–2383, 2010.

[119] Y. Yang, K. McLaughlin, T. Littler, S. Sezer, B. Pranggono and H. F. Wang, "Intrusion Detection System for IEC 60870-5-104 based SCADA networks," in *2013 IEEE Power & Energy Society General Meeting*, 2013.

[120] H. J. a. K. Jones, "DOI: http://dx.doi.org/10.14236/ewic/ics-csr2014.5," in *ICS-CSR 2014*, 2014.

[121] M. Noorizadeh, M. Shakerpour, N. Meskin, D. Unal and K. Khorasani, "A cyber-security methodology for a cyber-physical industrial control system testbed," *IEEE Access*, vol. 9, p. 16239–16253, 2021.

[122] W. S. a. P. Z. M. Wan, "Double behavior characteristics for one-class classification anomaly detection in networked control systems," *IEEE Trans. Inf. Forensics Security*, vol. 12, pp. 3011-3023, 2017.

[123] H. A. D. E. K. J. W. H. a. J. C. F. Zhang, "Multilayer data-driven cyber-attack detection system for industrial control systems based on network system and process data," *IEEE Trans. Ind. Informat.*, vol. 15, pp. 4362-4369,, 2019.

[124] A. S. Z. K. P. W. S. a. R. K. I. H. Lin, "Semantic security analysis of SCADA networks to detect malicious control commands in power grids," *SEGS*, pp. 29-34, 2013.

[125] E. U. A. F. N. A. I. Security, "Communication network dependencies for ICS/SCADA Systems," December 2016. [Online]. Available: https://www.enisa.europa.eu/publications/ics-scada-dependencies/.

[126] M. Mallouhi, Y. Al-Nashif, D. Cox, T. Chadaga and S. Hariri, "A testbed for analyzing security of SCADA control systems (TASSCS)," in *ISGT 2011*, 2011.

[127] S. McLaughlin and S. Zonouz, "Controller-aware false data injection against programmable logic controllers," in *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2014.

[128] S. Valentine and C. Farkas, "Software security: Application-level vulnerabilities in SCADA systems," in *2011 IEEE International Conference on Information Reuse & Integration*, 2011.

[129] Z. Zhang, W. Susilo and R. Raad, "Mobile ad-hoc network key management with certificateless cryptography," in *2008 2nd International Conference on Signal Processing and Communication Systems*, 2008.

[130] S. Qazi, R. Raad, Y. Mu and W. Susilo, "Securing DSR against wormhole attacks in multirate ad hoc networks," *J. Netw. Comput. Appl.*, vol. 36, p. 582–592, 2013.

[131] S. Qazi, R. Raad, Y. Mu and W. Susilo, "Multirate DelPHI to secure multirate ad hoc networks against wormhole attacks," *J. Inf. Secur. Appl.*, vol. 39, p. 31–40, 2018.

[132] B. Zhu, A. Joseph and S. Sastry, "A taxonomy of cyber attacks on SCADA systems," in *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, 2011.

[133] Z. Zhang, Z. Lv, J. Mo and S. Niu, "Vulnerabilities Analysis and Solution of VxWorks," in *Proceedings of the 2nd International Conference on Teaching and Computational Science*, Paris, 2014.

[134] "Vulnerability Note VU#362332 -Wind River Systems VxWorks debug service enabled by default," CERT Coordination Center - Carnegie Mellon University, 02 08 2010. [Online]. Available: https://www.kb.cert.org/vuls/id/362332/. [Accessed 02 02 2022].

[135] "Schneider Electric Modicon Quantum Vulnerabilities (Update B) - ICS Alert (ICS-ALERT-12-020-03B)," CISA.org - ICS-CERT, 09 April 2012. [Online]. Available: https://www.cisa.gov/uscert/ics/alerts/ICS-ALERT-12-020-03B. [Accessed 02 02 2022].

[136] "Vulnerability Note VU#840249, Wind River Systems VxWorks weak default hashing algorithm in standard authentication API (loginLib)," CERT Coordination Center - Carnegie Mellon University, 02 Aug 2010. [Online]. Available: https://www.kb.cert.org/vuls/id/840249. [Accessed 02 02 2022].

[137] N. Govil, A. Agrawal and N. O. Tippenhauer, "On ladder logic bombs in industrial control systems," in *Computer Security*, Cham, Springer International Publishing, 2018, p. 110–126.

[138] S. Sridhar and G. Manimaran, "Data integrity attacks and their impacts on SCADA control system," in *IEEE PES General Meeting*, 2010.

[139] F. S. Gazijahani and J. Salehi, "Robust design of microgrids with reconfigurable topology under severe uncertainty," *IEEE Trans. Sustain. Energy*, vol. 9, p. 559–569, 2018.

[140] D. Minoli, K. Sohraby and B. Occhiogrosso, "IoT considerations, requirements, and architectures for smart buildings—energy optimization and next-generation building management systems," *IEEE Internet Things J.*, vol. 4, p. 269–283, 2017.

[141] Y. Cherdantseva, P. Burnap, A. Blyth, P. Eden, K. Jones, H. Soulsby and K. Stoddart, "A review of cyber security risk assessment methods for SCADA systems," *Comput. Secur.*, vol. 56, p. 1–27, 2016.

[142] "Modicon M580 Safety eBrochure," Schneider , 2018. [Online]. Available: https://www.se.com/us/en/download/document/998-20233655_GMA/. [Accessed 02 02 2022].

[143] S. Ismail, E. Sitnikova and J. Slay, "Towards developing SCADA systems security measures for critical infrastructures against cyber-terrorist attacks," in *ICT Systems Security and Privacy Protection*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, p. 242–249.

[144] P. Goel, A. Datta and M. S. Mannan, "Industrial alarm systems: Challenges and opportunities," *J. Loss Prev. Process Ind.*, vol. 50, p. 23–36, 2017.

[145] E. Blanco Viñuela, D. Darvas and V. Molnár, "PLCverif re-engineered: An open platform for the formal analysis of PLC programs," 2020.

[146] S. Charbonnier, N. Bouchair and P. Gayet, "Analysis of fault diagnosability from SCADA alarms signatures using relevance indices," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014.

[147] P. M. Nasr and A. Y. Varjani, "Alarm based anomaly detection of insider attacks in SCADA system," in *2014 Smart Grid Conference (SGC)*, 2014.

[148] D. H. Rothenberg, Alarm management for process control, second edition: A best-practice guide for design, implementation, and use of industrial alarm systems, Momentum Press, 2018.

[149] P. Urban and L. Landryova, "Process knowledge building an optimized alarm system," in *Proceedings of the 2015 16th International Carpathian Control Conference (ICCC)*, 2015.

[150] P. Urban and L. Landryova, "Identification and evaluation of alarm logs from the alarm management system," in *2016 17th International Carpathian Control Conference (ICCC)*, 2016.

[151] J. Zhu, C. Wang, C. Li, X. Gao and J. Zhao, "Dynamic alarm prediction for critical alarms using a probabilistic model," *Chin. J. Chem. Eng.*, vol. 24, p. 881–885, 2016.

[152] V. B. Soares, J. C. Pinto and M. B. de Souza, "Alarm management practices in natural gas processing plants," *Control Eng. Pract.*, vol. 55, p. 185–196, 2016.

[153] N. A. Adnan, Y. Cheng, I. Izadi and T. Chen, "Study of generalized delay-timers in alarm configuration," *J. Process Control*, vol. 23, p. 382–395, 2013.

[154] P. Mahmoudi-Nasr, "Toward modeling alarm handling in SCADA system: A colored Petri nets approach," *IEEE Trans. Power Syst.*, vol. 34, p. 4525–4532, 2019.

[155] H. Hui and K. McLaughlin, "Investigating current PLC security issues regarding Siemens S7 communications and TIA portal," 2018.

[156] C. Bellettini and J. L. Rrushi, "Vulnerability analysis of SCADA protocol binaries through detection of memory access taintedness," in *2007 IEEE SMC Information Assurance and Security Workshop*, 2007.

[157] A. Al-Abassi, H. Karimipour, A. Dehghantanha and R. M. Parizi, "An ensemble deep learning-based cyber-attack detection in industrial control system," *IEEE Access*, vol. 8, p. 83965–83973, 2020.

[158] Priyanga, K. Krithivasan, Pravinraj and S. Sriram V.S., "Detection of cyberattacks in industrial control systems using enhanced principal component analysis and hypergraph-based convolution neural network (EPCA-HG-CNN)," *IEEE Trans. Ind. Appl.*, vol. 56, p. 4394–4404, 2020.

[159] S. D'Antonio, F. Oliviero and R. Setola, "High-speed intrusion detection in support of critical infrastructure protection," in *Critical Information Infrastructures Security*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, p. 222–234.

[160] J. Qian, X. Du, B. Chen, B. Qu, K. Zeng and J. Liu, "Cyber-physical integrated intrusion detection scheme in SCADA system of process manufacturing industry," *IEEE Access*, vol. 8, p. 147471–147481, 2020.

[161] H. Janicke, A. Nicholson, S. Webber and A. Cau, "Runtime-monitoring for industrial control systems," *Electronics (Basel)*, vol. 4, p. 995–1017, 2015.

[162] W. A. a. P. Langendörfer, "A Remote Attack Tool against Siemens S7–300 Controllers: A Practical Report," *KommA 2020*, vol. 11, 29 Oct 2020.

[163] R. Ma, Q. Wei and Q. Wang, "A survey of offensive security research on PLCs," *J. Phys. Conf. Ser.*, vol. 1976, p. 012025, 2021.

[164] M. Humayun, M. Niazi, N. Z. Jhanjhi, M. Alshayeb and S. Mahmood, "Cyber security threats and vulnerabilities: A systematic mapping study," *Arab. J. Sci. Eng.*, vol. 45, p. 3171–3189, 2020.

[165] O. Andreeva, S. Gordeychik, G. Gritsai and O. Kochetova, *Industrial control systems vulnerabilities statistics.*

[166] "Overload: Critical Lessons from 15 Years of ICS Vulnerabilities," [Online]. Available: https://www2.fireeye.com/rs/848-DID-242/images/ics-vulnerability-trend-report-final.pdf. [Accessed 25 July 2019].

[167] *Programmable logic controller (PLC) market 2020-2025 - leading players are Schneider electric, Rockwell automation, Siemens, Mitsubishi electric and Omron - ResearchAndMarkets.Com*, 2020.

[168] C.-W. Ten, C.-C. Liu and G. Manimaran, "Vulnerability assessment of cybersecurity for SCADA systems," *IEEE Trans. Power Syst.*, vol. 23, p. 1836–1846, 2008.

[169] "Logix 5000 Controllers Design Considerations," Rockwell Automation Publication, 2020. [Online]. Available: https://literature.rockwellautomation.com/idc/groups/literature/documents/rm/1756-rm094_-en-p.pdf. [Accessed 02 02 2022].

[170] "Programming Manual Original Instructions Logix 5000 Controllers Tasks, Programs, and Routines," Rockwell Automation Publication, 2020. [Online]. Available: https://literature.rockwellautomation.com/idc/groups/literature/documents/pm/1756-pm005_-en-p.pdf. [Accessed 02 02 2022].

[171] M. Zhang, C.-Y. Chen, B.-C. Kao, Y. Qamsane, Y. Shao, Y. Lin, E. Shi, S. Mohan, K. Barton, J. Moyne and Z. M. Mao, "Towards automated safety vetting of PLC code in real-world plants," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[172] I.-C. Kaspersky, Kaspersky Lab, 09 September 2021. [Online]. Available: https://ics-cert.kaspersky.com/publications/reports/2021/09/09/threat-landscape-for-industrial-automation-systems-statistics-for-h1-2021/. [Accessed 02 02 2022].