Investigating Security Issues in Programmable Logic Controllers and Related Protocols

Von der Fakultät 1 - MINT - Mathematik, Informatik, Physik,
Elektro- und Informationstechnik
der Brandenburgischen Technischen Universität Cottbus–Senftenberg
genehmigte Dissertation
zur Erlangung des akademischen Grades eines,

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

vorgelegt von

M.Sc. Wael Alsabbagh

geboren am 23.02.1989 in Hama, Syrien

Vorsitzende/r: Prof. Dr. Andriy Panchenko

Gutachter/in: Prof. Dr. Peter Langendörfer

Gutachter/in: Prof. Dr.-Ing. Ulrich Berger

Gutachter/in: Prof. Dr. Dimitrios Serpanos

Tag der mündlichen Prüfung: 20.12.2023

DOI: https://doi.org/10.26127/BTUOpen-6611

Abstract

Programmable Logic Controllers (PLCs) play a substantial role in Critical Infrastructures (CIs) and Industrial Control Systems (ICSs). They are programmed with a control logic program which determines how to control and operate physical processes such as nuclear power plants, petrochemical factories, water treatment systems, and many others. Unfortunately, those devices are not impervious to security threats and are susceptible to malicious attacks. In particular, vulnerabilities within the control logics of PLCs serve as entry points for such attacks. Such threats are known as control logic injection attacks, and primarily designed to sabotage physical processes controlled by exposed PLCs, resulting in disastrous damages to target systems as Stuxnet showed.

This thesis addresses various security issues and vulnerabilities in PLCs and their associated protocols that facilitate control logic injection attacks. Our primary focus is to analyze the security of both non-cryptographically and cryptographically protected PLCs, specifically examining the protective mechanisms implemented by vendors to secure the control logics executed by PLCs. To validate our findings and results, we have chosen PLCs from Siemens to conduct our experiments for two main reasons. Firstly, Siemens is the leader in the automation market, and its devices, especially the S7-300 PLCs, are currently employed in millions of diverse automation applications. Therefore, their security reflects the broader landscape of security for millions of operational ICSs. Secondly, Siemens has reportedly asserted that its latest PLCs production line, i.e., the S7-1500, incorporates a sophisticated integrity check mechanism, rendering such devices fully resistant to various cyberattacks, such as control logic injection attacks.

The first part of this work analyzes the authentication method implemented by non-cryptographically protected PLCs to prevent unauthorized access to control logics. Subsequently, a scenario involving a stealthy control logic injection attack is introduced, based on implementing a fake Programmable Logic Controller (PLC) approach. All experiments in this section are conducted in a real industrial setting using an S7-300 PLC and its corresponding S7 Communication (S7Comm) protocol.

The second part investigates the integrity check mechanism applied by modern cryptographically protected PLCs to secure control logics. Building on our findings, which include disclosed vulnerabilities, we execute a severe control logic injection attack that infects PLCs with a malicious interrupt block. All experiments in this section are performed in an actual industrial setting using an S7-1500 PLC and its S7 Communication Plus (S7CommPlus) protocol.

The last part focuses on the communication between PLCs and other industrial devices. Here, we investigate the security of the Profinet protocol and then perform an injection attack scenario. In this context, we demonstrate that adversaries lacking prior knowledge of both the target system and its network can manipulate critical data used as inputs/outputs in control logics, causing disastrous harm to the physical process. To create a real-world attack scenario, we execute our attack on a Profinet-based system using two S7-300 PLCs.

Finally, we conclude our investigations by suggesting mitigation solutions to enhance the security level of PLCs and their communication protocols. This contribution significantly improves the security posture of millions of operating devices worldwide. The presented results of this thesis contribute to advancing the state of research in the area of PLC security.

Zusammenfassung

Speicherprogrammierbare Steuerungen (SPSen) sind ein wesentlicher Bestandteil von kritischen Infrastrukturen (KIen) und Industriellen Kontrollsystemen (IKSen). Sie sind mit einer Steuerlogik programmiert, die festlegt, wie kritische Prozesse z.B. in Kernkraftwerken, petrochemischen Fabriken, Wasseraufbereitungsanlagen und vielen anderen Anwendungsgebieten gesteuert und betrieben werden. Leider sind diese Geräte nicht sicher und anfällig für bösartige Bedrohungen, insbesondere für solche, die Schwachstellen in der Steuerungslogik der SPSen ausnutzen. Solche Bedrohungen sind als Control Logic Injection-Angriffe bekannt. Sie zielen hauptsächlich darauf ab, physikalische Prozesse zu sabotieren, die von ungeschützten SPSen gesteuert werden, und können katastrophale Schäden an den Zielsystemen, wie Stuxnet gezeigt hat, verursachen.

Diese Arbeit befasst sich mit verschiedenen Sicherheitsproblemen und Schwachstellen in SPSen und den zugehörigen Protokollen, die Angriffe der Steuerlogik ermöglichen. Unser Hauptaugenmerk liegt auf der Analyse der Sicherheit sowohl von nicht kryptografisch als auch von kryptografisch geschützten SPSen, genauer gesagt auf den Schutzmechanismen, die von den Herstellern zum Schutz, der von den SPSen ausgeführten Steuerlogik eingesetzt werden. Um unsere Erkenntnisse und Ergebnisse zu validieren, verwenden wir aus den folgenden zwei Hauptgründen SPSen von Siemens für unsere Experimente. Erstens ist Siemens der Marktführer in der Automatisierungstechnik, und seine Geräte, insbesondere die S7-300 SPSen, werden derzeit in Millionen von verschiedenen Automatisierungsanwendungen eingesetzt. Daher spiegelt ihre Sicherheit das Gesamtbild der Sicherheit von Millionen von IKSen wider. Zweitens behauptet Siemens, dass die neueste SPS-Produktionslinie, d. h. die S7-1500, einen ausgeklügelten Integritätsprüfungsmechanismus enthält, der diese Geräte gegen verschiedene Cyberangriffe, z. B. Angriffe durch Injektion von Steuerungslogik, vollständig resistent macht.

Im ersten Teil dieser Arbeit wird die Authentifizierungsmethode analysiert, die nicht kryptografisch geschützte SPSen nutzen, um unbefugte Zugriffe auf die Kontrolllogik zu verhindern. Anschließend wird ein verdecktes (stealthy) Angriffsszenario für die Injektion von Steuerungslogik vorgestellt, das auf der Implementierung eines „Fake“ Speicherprogrammierbare Steuerung (SPS) Ansatzes basiert. In diesem Teil werden alle unsere Experimente in einer realen industriellen Umgebung mit einer S7-300 SPS und dem zugehörigen S7-Kommunikationsprotokoll durchgeführt.

Im zweiten Teil wird der Integritätsprüfungsmechanismus untersucht, den moderne kryptografisch geschützte SPSen anwenden um Kontrollsysteme zu sichern. Auf der Grundlage

unserer Erkenntnisse, d.h. der aufgedeckten Schwachstellen, führen wir einen schwerwiegenden Angriff auf die Steuerlogik durch, bei dem die SPS mit einem bösartigen Interrupt-Block infiziert wird. Alle Experimente in diesem Teil werden in einer realen industriellen Umgebung mit einer S7-1500 SPS und ihrem S7-Kommunikationsprotokoll Plus Protokoll durchgeführt.

Der letzte Teil konzentriert sich auf die Kommunikation zwischen SPSen und anderen industriellen Geräten. Dazu wird die Sicherheit des Profinet-Protokolls untersucht und anschließend ein Injektionsangriffsszenario durchgeführt. Hier zeigen wir, dass Angreifer sowohl ohne vorherige Kenntnis des Zielsystems als auch seines Netzwerks kritische Daten manipulieren können, die als Ein- und Ausgänge in einer Steuerlogik verwendet werden, was zu einer katastrophalen Schädigung des physischen Prozesses führt. Um ein realistisches Angriffsszenario zu erhalten, führen wir unseren Angriff auf ein echtes Profinet-basiertes System mit zwei S7-300 SPSen durch.

Zum Abschluss unserer Untersuchungen schlagen wir zukunftsweisende Lösungen vor, um die Sicherheit von SPSen und deren Kommunikationsprotokollen zu erhöhen und damit einen wichtigen Beitrag zur Sicherheit von Millionen von Industriekontrollgeräten auf der ganzen Welt zu leisten. Die in dieser Arbeit vorgestellten Ergebnisse erweitern den Stand der Forschung auf dem Gebiet der SPS-Sicherheit signifikant.

Acknowledgement

First and foremost, I would like to express my deepest gratitude to Prof. Dr. Peter Langendörfer for his exceptional guidance, unwavering support, and invaluable mentorship throughout my doctoral research. His insight and expertise have not only shaped the trajectory of my academic pursuits but have also been fundamental in fostering my growth as a scientist. I extend my sincere appreciation to Prof. Dr.-Ing. Ulrich Berger and Prof. Dr. Dimitrios Serpanos for their meticulous evaluation and constructive feedback, which have greatly enriched the quality of this dissertation.

I am quite grateful for Prof. Dr. Andriy Panchenko for his insightful comments and dedicated efforts during the defense. His contribution has added immense value to the refinement of my research.

I would like to acknowledge my colleagues at IHP and B-TU for their collaborative spirit and the enriching environment that has been a significant catalyst for my success. This dissertation would not have been possible without the collective support and encouragement from everyone of you.

A very special note of appreciation goes to my small family, whose encouragement and love have been my pillars of strength. I am also grateful to my parents for their sacrifices and belief in my abilities. To all my friends who have stood by me, I extend my heartfelt thanks.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Contents of this chapter are as follows:

1.1 Problem Statement

Industrial Control Systems (ICSs) re employed to automate physical processes, including production lines, electrical power grids, gas plants, telecommunications, transportation, and chemical and pharmaceutical facilities, among others. Each Industrial Control System (ICS) environment consists of a control center and field sites [4], (see Figure 1.1).

The physical processes are located at field sites and are controlled by Programmable Logic Controllers (PLCs). These PLCs read measurements from sensors and act accordingly by switching on/off appropriate actuators. PLCs are industrial computers (devices) that execute specific control logics, specifying how to operate physical processes. These devices are considered the central components of any ICS. Consequently, their security, robustness, and expected responses are major design concerns.

Unfortunately, ICSs, particularly their PLCs, have become attractive targets for adversaries seeking to disrupt control processes driven by compromised PLCs. Among the various types of attacks targeting PLCs, control logic injection attacks are considered the most severe. These threats exploit software or hardware flaws in PLCs, resulting in undesirable actions that could lead to fatal consequences for Critical Infrastructures (CIs), as demonstrated by incidents such as Stuxnet [17], TRITON [21], and Black Energy [46].

In typical control logic injection attacks, adversaries employ specific tools, libraries, and software to modify either the control logic program that the target Programmable Logic Controller (PLC) runs or to manipulate the program's inputs or outputs [119]. The malicious

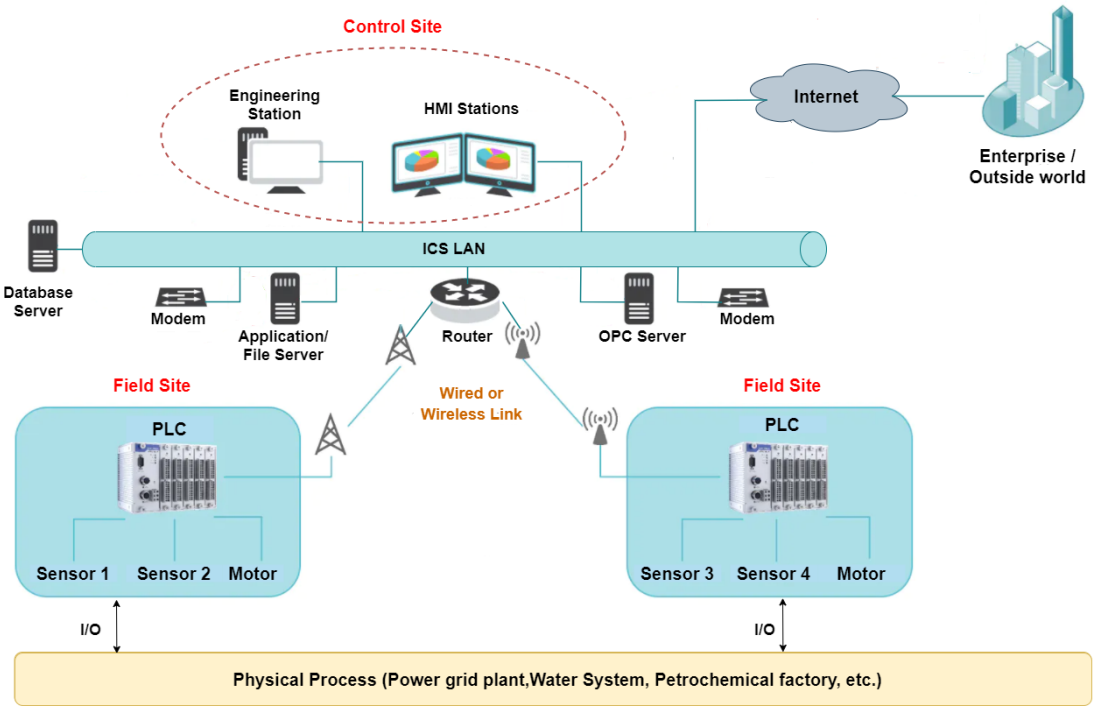


Figure 1.1: An Example of Industrial Control System Environment [6]

injection can be applied either to the high-level source code program (i.e., in one of the five programming languages defined in IEC-61131 [120]) or to the low-level bytecode program (i.e., to the machine code that PLCs read and execute). Another control logic attack scenario can be conducted by crafting certain data values involved in a PLC's program, which eventually exploit design flaws in the code.

These threats have been a research hotspot and received increasing attention from security researchers since 2010, i.e., since Stuxnet [17], which is believed to be the first attack designed with the sole purpose of causing physical harm. Stuxnet targeted Iran's uranium facility at the Natanz enrichment plant and aimed at infecting the control logic program of Siemens S7-300 PLCs that control the variable frequency drives of centrifuges. As a consequence, the infected logic control confused the normal operation of the drives by modifying their motors' speed periodically from 1410 Hertz (Hz) to 2 Hz to 1064 Hz and then over again [70]. The shocking fact of the Stuxnet attack is that the Iranian plant was completely isolated from the outer world (air-gapped). Therefore, the infection occurred either via a Universal Serial Bus (USB) stick or a portable programming device.

In recent years, many industrial plants have become increasingly connected to the Internet. Consequently, various exposed devices, such as PLCs, can be publicly identified using specialized search engines for ICSs and Supervisory Control and Data Acquisition (SCADA) systems, such as *Censys* [25], *Shodan* [26, 27], *ZoomEye* [28], and *Ditecting* [29]. Table 1.1 presents the results of recent scans conducted with the assistance of these search engines, precisely on March 22, 2023. Appendix A provides all the queries used for each search engine listed in Table 1.1.

Table 1.1: The number of Internet-accessible PLCs on March 22, 2023

Protocol	**Port**	***Censys***	***Shodan***	***ZoomEye***	***Ditecting***
Modbus	502/TCP	35,018	61,902	31,706	14,173
Siemens S7	102/TCP	6,804	56,010	40,099	2464
DNP3	20000/TCP	597	962,297	14,750,352	367
BACnet	47808/TCP	15,514	31,157	56,767	11,750
Niagara Fox	1911/TCP	25,398	80,902	262,111	26,634
Ethernet/IP	44818/TCP	Not Available	65,402	27,637	1971
Phonix/PCWorx	1962/TCP	Not Available	48,924	316,365	168
Codesys	2455/TCP	Not Available	37,099	241,809	1298

It is noteworthy that the number of PLCs found varies significantly from one search engine to another. According to [30], the varying counts reported by *Censys* can be attributed, in part, to the fact that *Censys* removes a considerable number of entries compared to *Shodan*. Specifically, *Censys* updates its database by removing PLCs that were not found in the last week. This implies that, from our perspective, *Censys* may have access to fewer devices than *Shodan*. *ZoomEye*, not explicitly designed for searching only for ICS devices, lacks an understanding of certain protocols. Consequently, this results in a substantially larger number of findings for some open port protocols, such as Distributed Network Protocol 3 (DNP3). However, finding a comprehensive explanation is challenging, given that most search engines are proprietary, such as *Detecting*. Making accurate statements about the number of devices found by each search engine is difficult due to minor changes in dynamic Internet Protocol (IP) ranges leading to significant variations. Therefore, Table 1.1 merely highlights that many PLCs are exposed and accessible from the Internet.

Even though only one PLC may be reachable from the outside world, this exposed PLC is highly likely to be connected to internal networks housing many more PLCs. This concept is referred to as the deep industrial network [31]. Another significant observation was emphasized in [32]. The report revealed that many devices are exposed to the internet

without adequate security measures. Additionally, many ICS operators lack knowledge on how to protect their industrial components. Alternatively, they may possess the knowledge but fail to realize that a successful attack could incur greater costs than implementing security measures. Consequently, many manufacturers are reluctant to secure their industries unless explicitly instructed to do so, or they will apply security updates/patches only when absolutely necessary, such as in the event of a severe attack. Given that some of the exposed PLCs controlling ICSs and CIs have vulnerabilities and weak points, there is an urgent need for action. Hence, PLCs need to undergo more thorough testing to identify and address vulnerabilities effectively.

1.2 Research Objectives

The objective of this thesis is to initially investigate the security of PLCs and subsequently develop new attack approaches. These approaches aim to reveal new vulnerabilities or insecurity specifications within ICS environments, specifically targeting PLC devices and their related protocols. Our investigations encompass both the old generation of PLCs—non-cryptographically protected PLCs, and the modern generation—cryptographically protected PLCs. It is essential to note that the inclusion of old PLCs in our investigations is motivated by the extended life cycle of ICSs, which is typically much longer (20 to 30 years) compared to Information Technology (IT) systems. Consequently, numerous outdated PLCs remain operational and exposed to various cyber-attacks.

The primary focus of this work is to analyze attack vectors and demonstrate to the security research community, engineers, and industrial vendors the potential consequences of exploiting vulnerabilities. To this end, we address the following research questions, which reflect the outcomes of our work.

1. **Is the authentication method used in non-cryptographically protected PLCs sufficient to secure the control logics?**

   One of the initial queries when addressing the security of old PLCs (i.e., non-cryptographically protected PLCs) is how an attacker can access the control logic program running on a target device. A significant challenge here is to bypass the authentication method protecting PLCs (addressed in Chapter 3, Section 3.1).

2. **Can adversaries conduct a stealthy injection attack against non-cryptographically protected PLCs?**

   Another challenge that attackers typically face after gaining access to control logics

within PLCs is concealing their ongoing attacks from ICS operators (addressed in Chapter 3, Section 3.2).

3. **Are cryptographically protected PLCs truly resistant against control logic injection attacks?**

   Vendors have enhanced the security of their PLCs by integrating new devices with advanced security mechanisms (e.g., sophisticated integrity check methods) to thwart malicious adversaries from conducting cyber-attacks against PLCs. As part of this work, we investigate whether modern cryptographically protected PLCs are secure against control logic injection attacks (addressed in Chapter 4).

4. **Is it feasible to manipulate Input/Output (I/O) data values used in control logics even without any prior knowledge of the target systems?**

   Another research question examines how to manipulate the trusted communication between two connected devices/stations and manipulate inputs/outputs of control logic programs. To enhance the realism of our investigations, attackers should not be familiar with the target system, the physical process, the data exchanged, or the system parameters (addressed in Chapter 5).

1.3 Contributions

While working on this thesis, several vulnerabilities and weak points in PLCs and their related protocols were identified. These vulnerabilities allow attackers to conduct different control logic injection attack scenarios against both cryptographically and non-cryptographically protected PLCs. In the following, we summarize our main findings achieved in this work.

1. **Systematizing the control logic injection attacks targeting PLCs based on three attack scenarios.** Our study showed that most of the control logic injection threats were executed through one of the following three attack scenarios: 1) attackers have access to the Engineering Work Station (EWS), 2) attackers have access to the control network, 3) attackers have access to the PLC runtime environment [8, 13].

2. **Analyzing the authentication protocol used in non-cryptographically protected PLCs.** We tested the authentication protocol used in PLCs (e.g., S7-300 and S7-400) and revealed the encoding scheme that such a protocol utilizes to encode passwords. As part of our investigations, we disclosed a few vulnerabilities in the authentication protocol and revealed the 256 characters and their corresponding encoded bytes [1, 2].

3. **Performing a stealthy control logic injection attack against non-cryptographically protected PLCs.** As part of this attack, we introduced a new fake PLC approach that mimics the behavior of a real PLC [2]. This allows us to conceal an ongoing infection in the PLC without being noticed by the ICS operator. We implemented this attack scenario on an experimental setup using an S7-300 PLC.

4. **Introducing a *Decompiler -Compiler* approach to alter the control logic in its high-level format.** This approach allows attackers to extract critical semantics from the physical process and manipulate the control logic in its source code [2, 5]. To validate our approach, we tested both the *Decompiler* and *Compiler* on a real experimental setup based on S7-300 PLCs.

5. **Investigating the integrity check mechanism that modern (cryptographically protected) PLCs implement to secure their control logics.** As a case study, we analyzed the entire integrity check method used in the latest S7 PLC model, i.e., S7-1500 PLCs. We revealed several new vulnerabilities in these devices and their related S7 Communication Plus (S7CommPlus) protocol [6, 7, 11].

6. **Introducing a new injection attack scenario against cryptographically protected PLCs.** The threat allows attackers to manipulate the physical process without the need to be online at the point zero for the attack. To validate our approach, we performed the attack on an experimental setup using the most secured PLC model among Siemens SIMATIC families, i.e., S7-1500 [6, 7].

7. **Concealing the malicious injection in the PLC memory.** We found a security gap in the S7CommPlus protocol that allows malicious adversaries to hide their infection in the PLC memory until the very moment determined by the attackers [6, 7].

8. **Compromising the Profinet protocol, precisely the Application Relationship (AR) between connected PLCs.** To this end, we performed a fully-blind data injection attack scenario that alters critical I/O data exchanged over network packets [4, 10]. Our attack is based on integrating an I/O *Database* approach that assists attackers in replacing valid data with malicious ones without the need to be familiar with the control logics running in the connected PLCs.

1.4 Scientific Publications

During the time working at this thesis, several scientific papers/articles were published. This section provides a short description of our publications sorted with the oldest first as follows.

Journal Publications:

- **W. Alsabbagh** and P. Langendöerfer, "A New Injection Threat on S7-1500 PLCs - Disrupting the Physical Process Offline," in IEEE Open Journal of the Industrial Electronics Society, vol. 3, pp. 146-162, 2022, doi: 10.1109/OJIES.2022.3151528 [6].

  Summary: This paper presents a control logic injection attack against the latest and most secured SIMATIC S7 PLC. As a part of this work, we investigated and revealed two design vulnerabilities in the S7-1500 PLCs and performed an attack scenario that allows an adversary to disrupt the physical process without being connected to the target system at the time of the attack.

  Author Contributions: Conceptualization, W.A.; Attack approach, W.A.; Writing original draft, W.A.; Review and editing, P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh** and P. Langendörfer, "A Flashback on Control Logic Injection Attacks against Programmable Logic Controllers," Automation 2022, 3(4), 596-621, doi: 10.3390/automation3040030 [8].

  Summary: A flashback on the recent works related to control logic injection attacks against PLCs is presented. As a part of this paper, we introduced a new systematization based on the attacker techniques under three main attack scenarios. Finally, the action taken by the top vendors are presented and futuristic solutions to prevent such severe attacks are suggested.

  Author Contributions: Conceptualization, W.A. and P.L.; Systematization methodology, W.A. and P.L.; Writing original draft, W.A.; Review and editing, P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh** and P. Langendörfer, "Security of Programmable Logic Controllers and Related Systems: Today and Tomorrow," in IEEE Open Journal of the Industrial Electronics Society, vol. 4, pp. 659-693, 2023, doi: 10.1109/OJIES.2023.3335976 [13].

  Summary: In this article, we provide a detailed review of all aspects concerning the security of PLCs and related systems. This includes vulnerabilities, potential attacks, and security solutions including digital forensics. We aim to offer a precise analysis, addressing the shortcomings of previous studies.

Author Contributions: Conceptualization, W.A.; Systematization methodology, W.A.; Writing original draft, W.A.; Review and editing, P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

Conference Publications:

- **W. Alsabbagh** and P. Langendörfer, "A Remote Attack Tool Against Siemens S7-300 Controllers: A Practical Report," In: Jasperneite, J., Lohweg, V. (eds) Kommunikation und Bildverarbeitung in der Automation. Technologien für die intelligente Automation, vol 14. Springer Vieweg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-64283-2_1 [1].

  Summary: A remote IHP-Attack tool to generate a series of attacks against Siemens PLCs from S7-300 family is introduced. The tool consists of many functionalities e.g., scanners, authentication bypass attack, replay attack, and control logic injection attack.

  Author Contributions: Conceptualization, W.A.; Attack approach, W.A.; Writing original draft, W.A.; Review and editing, P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh** and P. Langendörfer, "A Stealth Program Injection Attack against S7-300 PLCs," 2021 22nd IEEE International Conference on Industrial Technology (ICIT), 2021, pp. 986-993, doi: 10.1109/ICIT46573.2021.9453483 [2].

  Summary: An advanced attack based on combining a replay and injection approaches on Siemens S7-300 PLCs is presented. This severe exploit enables adversaries to hide their injection by engaging a fake PLC, impersonating the real infected device. Furthermore, as a part of the attack, we introduced a decompiler to convert the low-level code of the user program to one of its high-level languages.

  Author Contributions: Conceptualization, W.A.; Attack approach, W.A.; Writing original draft, W.A.; Review and editing, P.L.; visualization, W.A; supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh** and P. Langendörfer, "Patch Now and Attack Later - Exploiting S7 PLCs by Time-Of-Day Block," 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS), 2021, pp. 144-151, doi: 10.1109/ICPS49255.2021.9468226 [3].

Summary: A novel control logic injection attack strategy is introduced. The exploit presented in this paper is based on patching an interrupt in the PLC's program which could be activated at a later time when attackers are neither connected to the target device nor to its network. This attack was conducted and tested on S7-300 PLCs, and allowed adversaries to disrupt the physical process offline.

Author Contributions: Conceptualization, W.A.; Attack approach, W.A.; Writing original draft, W.A.; Review and editing, P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh** and P. Langendörfer, "A Fully-Blind False Data Injection on PROFINET I/O Systems," 2021 IEEE 30th International Symposium on Industrial Electronics (ISIE), 2021, pp. 1-8, doi: 10.1109/ISIE45552.2021.9576496 [4].

  Summary: A new False Data Injection (FDI) attack against Profinet I/O based Systems is presented. The approach taken in this work is to manipulate specific bytes that represent either sensor readings or actuator values in the control logic depending on the transaction direction between the nodes. This attack is performed without any prior knowledge of the target system, the data exchanged, the physical process or even the system parameters. S7-300 PLCs and 343-1 Communication Processor (CP) were a part of the hardware industrial setting used in this work.

  Author Contributions: Conceptualization, W.A.; Attack approach, W.A.; Writing original draft, W.A.; Review and editing, P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh** and P. Langendörfer, "A Control Injection Attack against S7 PLCs - Manipulating the Decompiled Code," IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society, 2021, pp. 1-8, doi: 10.1109/IECON48115.2021.9589721 [5].

  Summary: A new control logic injection based on modifying the PLC's program in its high-level decompiled format is introduced. We employed an advanced decompiler and compiler which allows attackers to understand the physical process controlled by the victim PLC and alter the control logic program on the attacker will. This approach was the first work allows an attacker to modify the control logic program in its high-level format.

Author Contributions: Conceptualization, W.A.; Attack approach, W.A.; Writing original draft, W.A.; Review and editing, P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh** and P. Langendörfer, "No Need to be Online to Attack - Exploiting S7-1500 PLCs by Time-Of-Day Block," 2022 XXVIII International Conference on Information, Communication and Automation Technologies (ICAT), 2022, pp. 1-8, doi: 10.1109/ICAT54566.2022.9811147 [7].

  Summary: This paper takes the approach presented in [3] one more step in the direction of exploiting PLCs offline, and extends the experiments to cover the newly most secured Siemens PLCs line i.e., S7-1500 PLCs.

  Author Contributions: Conceptualization, W.A.; Attack approach, W.A.; Writing original draft, W.A.; Review and editing, P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh** and P. Langendörfer, "You Are What You Attack: Breaking the Cryptographically Protected S7 Protocol," 2023 IEEE 19th International Conference on Factory Communication Systems (WFCS), Pavia, Italy, 2023, pp. 1-8, doi: 10.1109/WFCS57264.2023.10144251. [11].

  Summary: This paper investigated the integrity check mechanism implemented in the latest and most secure S7CommPlus protocol. To this end, we analyzed and disclosed the newly encryption algorithms that Siemens applies to secure their PLCs. Based on our findings, we managed successfully to craft malicious packets and performed several attack scenarios.

  Author Contributions: Conceptualization, W.A.; Attacks approach, W.A.; Writing original draft, W.A.; Review and editing, P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by both authors. All authors have read and agreed to the published version of the manuscript.

And the following are the further contributions of the author not directly used in this thesis:

- P. Langendörfer, S. Kornemann, **W. Alsabbagh** and E. Hermann, "Information Security: The Cornerstonefor Surviving the Digital Wild," In: Madsen, O., Berger, U.,

Møller, C., Heidemann Lassen, A., Vejrum Waehrens, B., Schou, C. (eds) The Future of Smart Production for SMEs. Springer, Cham, doi 10.1007/978-3-031-15428-7_29 [9].

Summary: This paper discusses the very basics in the sense of how to prepare your company with respect to security. The essential issues are a proper information security governance framework that takes into account the managerial and organizational issues as well as proper technical means.

Author Contributions: Conceptualization, P.L.; Security approach, P.L., S.K., W.A., E.H.; Writing original draft, P.L.; Review and editing, S.K., W.A., E.H.; Visualization, P.L; Supervision, P.L. The paper was proofread and checked by all authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh**, S. Amogbonjaye, D. Urrego and P. Langendörfer, "A Stealthy False Command Injection Attack on Modbus based SCADA Systems," 2023 IEEE 20th Consumer Communications Networking Conference (CCNC), Las Vegas, NV, USA, 2023, pp. 1-9, doi: 10.1109/CCNC51644.2023.10059804. [10].

  Summary: This paper exploited the insecurity of the Modbus protocol and performed a stealthy false command injection scenario concealing the injection from the ICS operator. To this end, we implemented an I/O *Database* approach that allows an adversary to conceal his ongoing injection.

  Author Contributions: Conceptualization, W.A.; Attack approach, W.A., S.A., D.U.; Writing original draft, W.A.; Review and editing, S.A., D.U., P.L.; Visualization, W.A; Supervision, P.L. The paper was proofread and checked by all authors. All authors have read and agreed to the published version of the manuscript.

- **W. Alsabbagh**, C. Kim and P. Langendörfer, "Good Night, and Good Luck: A Control Logic Injection Attack on OpenPLC," IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society, Singapore, Singapore, 2023, pp. 1-8, doi: 10.1109/IECON51785.2023.10312570 [12].

  Summary: This paper conducted intensive investigations and discloses some vulnerabilities existing in the OpenPLC project, showing that an attacker without any prior knowledge neither to the user credentials nor to the physical process can access critical information and maliciously alter the user-program the OpenPLC executes.

  Author Contributions: Conceptualization, W.A.; Attack approach, W.A., Ch. K.; Writing original draft, W.A.; Review and editing, Ch. K., P.L.; Visualization, W.A;

Supervision, P.L. The paper was proofread and checked by all authors. All authors have read and agreed to the submitted version of the manuscript.

1.5 Overview of the Thesis

The Structure of this thesis is summarized in the outline below:

Chapter 1 - Introduction: the introductory chapter.

Chapter 2 - Literature Review: Section 2.1 introduces the structure of an ICS environment, while Section 2.2 introduces the architecture of PLCs, its run-time environment and control logic programs. In Section 2.3, we overview control logic vulnerabilities, and the injection attack scenarios targeting PLCs, while Section 2.5 introduces real-world attack scenarios against ICSs.

Chapter 3 - Investigating the Security of non-cryptographically protected PLCs: focuses on analyzing the authentication method of PLCs and introducing a stealthy control logic injection attack. Section 3.1 discusses the possibility of bypassing a password-protected PLC. Thereafter, a stealthy control logic injection approach is presented and conducted against S7-300 PLCs in Section 3.2.

Chapter 4 - Investigating the Security of cryptographically protected PLCs: focuses on the security of the modern PLCs generation. As a part of this chapter, analyzing the advanced protection mechanism used by the newest S7CommPlus protocol is given. As well as a severe control logic injection attack scenario aiming at disrupting the physical process controlled by an S7-1500 PLC is conducted.

Chapter 5 - Investigating the Security of the Profinet I/O Protocol: introduces a fully-blind false data injection attack against Profinet I/O based environments. As a part of this chapter, hijacking the trust AR between PLC stations is introduced and a severe FDI attack without any prior knowledge of the target system is performed.

Chapter 6 - Summary and Future Work: concludes this thesis and a discussion of open research questions and future directions are given.

CHAPTER 2

Literature Review

Contents of this chapter are as follows:

Parts of this chapter have already been published in the following papers:

- W. Alsabbagh and P. Langendörfer, "A Flashback on Control Logic Injection Attacks against Programmable Logic Controllers," Automation. 2022; 3(4):579-595. https://doi.org/10.3390/automation3040029 [8].
- W. Alsabbagh and P. Langendöerfer, "A New Injection Threat on S7-1500 PLCs - Disrupting the Physical Process Offline," in IEEE Open Journal of the Industrial Electronics Society, vol. 3, pp. 146-162, 2022, doi: 10.1109/OJIES.2022.3151528 [6].
- W. Alsabbagh and P. Langendörfer, "Security of Programmable Logic Controllers and Related Systems: Today and Tomorrow," in IEEE Open Journal of the Industrial Electronics Society, vol. 4, pp. 659-693, 2023, doi: 10.1109/OJIES.2023.3335976 [13].

2.1 Industrial Control Systems (ICSs)

ICSs are utilized to automate critical control processes in industrial infrastructures. They primarily consist of various devices, including sensors, actuators, and embedded computers interconnected via networks that cooperate to maintain the physical process at a desired state. Figure 2.1 illustrates the basic hierarchical scheme standardized by International Electrotechnical Commission (IEC) 62264 [33].

The ICS of each company consist of several layers, as depicted in the figure. Starting from the top, we find Enterprise Resource Planning (ERP), which facilitates entrepreneurial

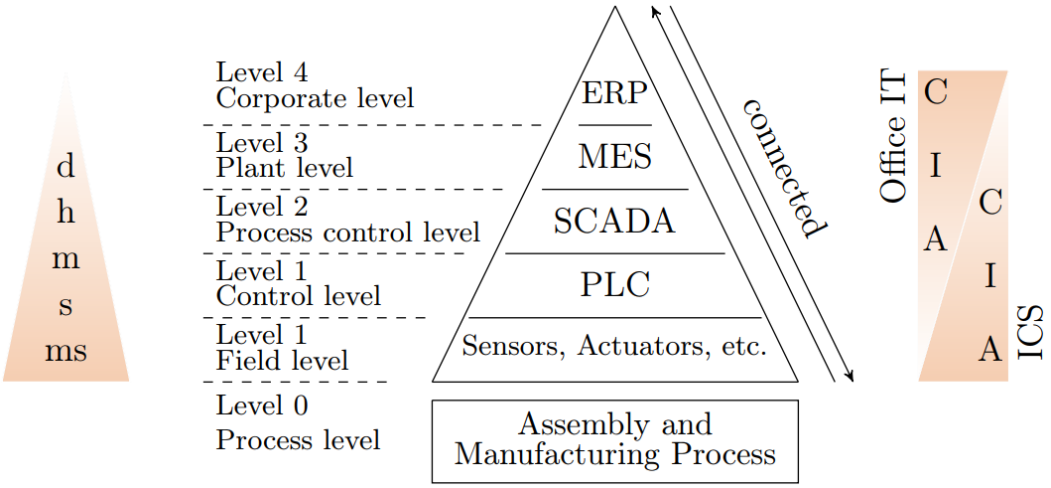


Figure 2.1: IEC 62264 Industrial Automation Pyramid, adopted from [34]

tasks in a timely manner. ERP systems mirror all entrepreneurial operations and should not involve any isolated solutions, enabling them to be utilized for the comprehensive control and monitoring of resources company-wide. Furthermore, ERP systems enhance cooperation through e-collaboration and facilitate communication flow. In level 3, we find Manufacturing Execution System (MES). It differentiates from similar systems (i.e., ERP systems) by connecting directly to distributed systems. The MES allows both production management and control. SCADA, located in level 2, is a system aimed at supervising and driving technical operations. The data of the supervised processes are displayed in a user-friendly way, enabling control to intervene in ongoing operations. Additionally, a historian server likely exists in the same SCADA layer to store all operational conditions, inputs, outputs, and other information for examination and evaluation. Consequently, SCADA systems can also generate statistical reports or identify potential faulty batches. In level 1, we find an Human Machine Interface (HMI) that permits an operator to interact with other industrial components in both the same and lower layers. Nowadays, the connectivity in SCADA systems relies more and more on Transmission Control Protocol (Transmission Control Protocol (TCP))-based techniques. At the control level (level 1), we also find PLCs, defined as embedded devices utilized to drive physical processes appropriately by running specific control logic programs. PLCs use field-bus protocols to read inputs from sensors and write outputs to actuators placed in the field level. ICS devices have a longer lifetime compared to IT system devices and operate strictly in hard real-time within certain time constraints [35].

The left side of Figure 2.1 illustrates the different timing requirements for each layer of an ICS. As shown, components located at the upper levels (e.g., level 4 and 3) have data transfer

times in the order of hours (h) or even days (d), while SCADA systems have data transfer times of seconds (s) to minutes (m). In contrast, in the lower layers (e.g., level 1) where hard real-time processes are operating, data exchange must occur in the order of milliseconds (ms). It is worth mentioning that the timing requirements for each ICS depend on the individual physical process it controls. For instance, the temperature in huge containers changes slowly, while the flow of water pipes can vary rapidly.

On the right side of Figure 2.1, the Confidentiality, Integrity and Availability (CIA) model is presented. The figure illustrates that the highest protection goals are reversed between office IT and ICS. In ICS systems, availability is the highest protection goal, whereas confidentiality is the most important protection goal in office IT.

2.2 Programmable Logic Controllers (PLCs)

In this section, we give an overview of the architecture of a standard PLC, its runtime environment and control logic program.

2.2.1 PLC Architecture

A PLC is an embedded small computer programmed on a digital basis that is utilized to drive plants, industries and facilities. Figure 2.2 introduces the architecture of a standard PLC. As can be seen, it is compromised of a power supply, I/O modules, an Operating System (OS), Random Access Memory (RAM), Electrically Erasable Programmable Read-only Memory (EEPROM) and interface to upload/download user-specific programs to/from the EWS. Both OS and user program are stored in the EEPROM. Input devices (e.g., sensors, switches, etc.) send measurements from the physical process environment to the PLC that processes them through its user program, and then acts by switching on/off certain output devices (e.g., motors, valves, etc.) accordingly.

2.2.2 Runtime Environment

PLCs operate with an OS that processes specific tasks in strict real-time through cyclic repetition-defined command sequences. Each execution cycle, also known as a scan cycle, comprises four steps, as illustrated in Figure 2.2. At the onset of each cycle, the Central Processing Unit (CPU) reads measurements from all sensors, storing the readings in an input image or a dedicated table. Subsequently, the CPU updates the current inputs of the user program with the new values. Following this, the program is processed, and the current output values are updated accordingly. The fourth step is designated for communications,

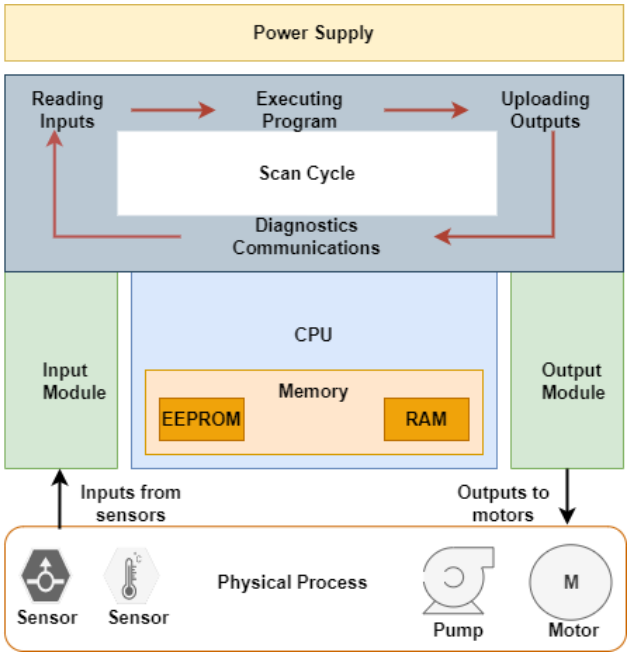


Figure 2.2: A typical PLC Architecture [6]

such as exchanging data with other devices connected to the PLC. At the conclusion of the communication time slice, the OS transitions the CPU to the maintenance phase, which is executed in the background, meaning the user remains uninformed. This phase encompasses various tasks, including updating internal clocks, managing internal registers, and executing memory management, among others.

2.2.3 Control Logic Program

A PLC control logic, also known as a user-program, is code authored by a user to dictate the operation of a PLC. It comprises different units (blocks), each associated with a specific task. All units collaborate to drive the PLC, maintaining the physical process at a desired state. An ICS operator writes control logic in a high-level programming language, i.e., in a source code version, using vendor-specific engineering software. The software compiles the source code into its low-level version (e.g., binary or bytecode) that the PLC reads and executes. Both control logic formats are illustrated below.

Source Code:

Various vendors manufacture different PLC models, including Siemens, Allen-Bradley, Mitsubishi, Schneider, and others. All PLCs in the automation market are programmed with

one of the five programming languages defined in IEC-61131 [54]: Ladder Diagram (LD), Structured Text (ST), Sequential Function Chart (SFC), Instruction List (IL), and Function Block Diagram (FBD). ICS operators write user-programs using vendor-specific software that includes built-in Integrated Development Environments (IDEs) and compilers. Some PLC models allow users to program in additional languages such as computer-compatible languages (e.g., BASIC, C, and assembly), proprietary-developed languages (e.g., Siemens GRAPH5[1]), and specific languages (e.g., boolean logic[2]).

Bytecode/Binary Code:

PLCs, like other small embedded computers, execute only low-level codes. Therefore, the engineering software needs to compile the readable version of any user-program into its bytecode or binary code before downloading it to the PLC. For instance, Siemens S7 PLCs read Machine Code 7 (MC7) bytecodes, while CODESYS devices execute binaries. However, the structure of PLC bytecodes or binaries is proprietary to the PLC vendors and not publicly documented. Consequently, any further exploration or investigation requires applying reverse engineering mechanisms.

2.2.4 PLC Example Application

Figure 2.3 shows a simple example application in which a PLC controls the movement of objects on a conveyor. For the given example, three sensors are positioned along the conveyor to determine the precise locations of the cartons and the current quantity of cartons on the conveyor. Under normal operating conditions, when an object reaches the conveyor, the first sensor (S1) reports to the PLC by transmitting an input value to the input module. In the subsequent execution scan cycle, the PLC reads this value and processes the control logic stored in the PLC's memory. Subsequently, the PLC issues a relevant control command to the output modules, activating the electrical switch (FWD) linked to the motor. The motor's rotation induces synchronized movement of the belt, facilitating the forward transport of the object through the looped movement of the belt.

In the event of an abnormal scenario, such as the arrival of a new object when the conveyor is already full (i.e., sensors S1, S2, and S3 are activated), the PLC deactivates the motor, bringing the belt to a halt. This process requires precise timing to prevent any disruption to the sequential movement of the objects.

[1] Simatic S5 PLC. https://en.wikipedia.org/wiki/Simatic S5 PLC.
[2] https://www.plcmanual.com/plc-programming

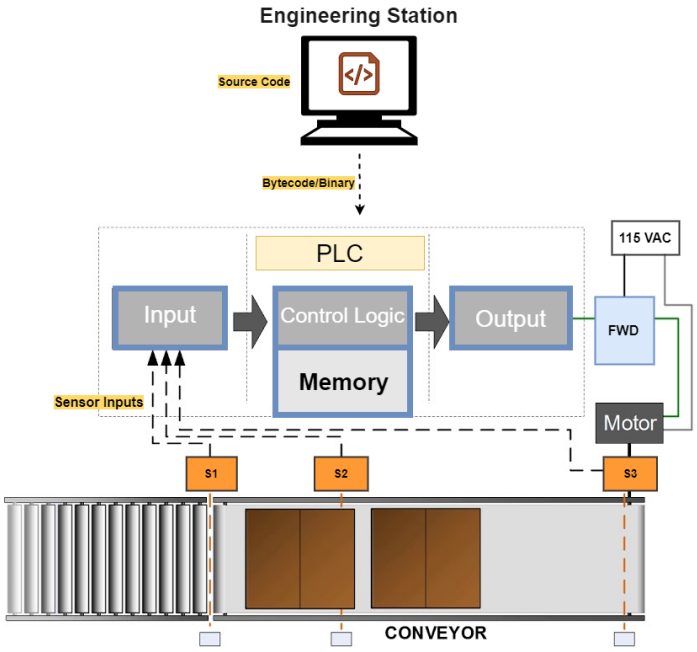


Figure 2.3: Example Application [8]

2.3 Control Logic Injection Attacks against PLCs

In this section, we provide an overview of the most common vulnerabilities that attackers exploit to access the control logics running in target PLCs and the security goals attackers aim to compromise through control logic injection attacks. Subsequently, we systematize and elaborate on the control logic attacks conducted and discussed in previous academic works, based on three scenarios.

2.3.1 Control Logic Vulnerabilities

Attackers typically seek entry points in the target system, leveraging existing vulnerabilities and weaknesses in the PLCs to achieve successful attacks. Most information concerning control logic exploits and vulnerabilities is derived from four primary sources: the Common Vulnerabilities and Exposures (CVE)[3], Industrial Control System Computer Emergency Response Team (ICS-CERT)[4], National Vulnerability Database (NVD)[5], and the Exploit Database[6]. All these projects are designed to provide lists of publicly disclosed information

[3]https://cve.mitre.org/cve/
[4]https://www.cisa.gov/uscert/ics
[5]https://www.nist.gov/programs-projects/national-vulnerability-database-nvd
[6]https://www.exploit-db.com/

security vulnerabilities and exposures related to ICSs and IT systems.

Using the search engine for each, we manually filtered vulnerabilities as control logic-related by matching specific keywords such as PLC, control logic, program injection, program modification, remote code execution. Subsequently, we scrutinized the descriptions of the resulting vulnerabilities and checked the online documentation of each affected PLC. Afterward, we extracted the number of vulnerabilities reported per year and generated a statistical report from the year 2010 (i.e., the first control logic-related vulnerability was reported) to the end of the year 2021, as presented in Figure 2.4.

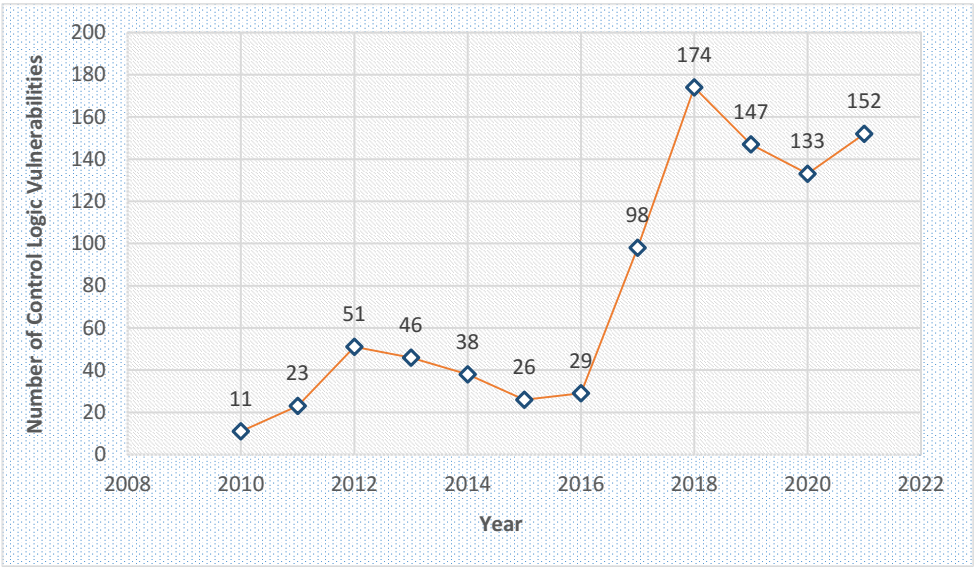


Figure 2.4: The number of related control logic vulnerabilities reported per year [8]

The rapid increase over the years, as demonstrated in our report, is primarily reflected in the steadily growing number of publications and the increasing popularity of PLC security within academic research. Specifically, after 2011 when Beresford [16], whose work is the most cited in this field, presented a roadmap for threats against PLCs, there has been a notable surge in academic studies investigating control logic injection attacks. In the following section, we illustrate the most common control logic vulnerabilities related to the four aforementioned projects.

V1 - Race Condition

A race condition scenario occurs when two or more threads attempt to access a shared resource simultaneously. This situation commonly arises when an output variable relies on multiple sensor readings. In the example provided in Figure 2.3, the exploitation of a race condition vulnerability takes place when a sensor reports to the PLC that objects arrive in their zones either faster or slower than expected at a configured moment. As a result, the PLC may transmit a false control command to the motor, leading to an undesired state.

V2 - Variables without Use

This vulnerability is exploited when a variable's value is assigned but never used in the PLC's program. This implies that if an input variable is uninitialized, attackers can provide an illegal value for it in a running program. Similarly, attackers can leverage unused output variables to induce a specific behavior. In the case of our example application, this vulnerability occurs when a user assigns new input variables for the sensors, i.e., S1, S2, and S3, without removing the previous variables. Consequently, an attacker can initialize one input variable to a malicious value, causing the PLC to read and process it during the running program.

V3 - Hidden Jumpers

A hidden jumper could involve either bypassing a portion of the program, forcing it to jump to an empty branch, or jumping to another branch based on a condition configured by the attacker. Attackers can exploit this vulnerability by embedding malware in the bypassed portion of the program. In the case of our example application, an attacker can create a force by activating a mechanism within the PLC, causing the device to override certain elements, such as the portion where a PLC updates sensor readings.

V4 - Improper Input Validation

In this vulnerability, attackers can craft illegal input values based on the types of the input variables, causing unsafe behavior. For example, attackers can send a specially crafted packet to the PLC, providing an input index that is out-of-bounds for an input array. Exploiting this vulnerability may render the target PLC inaccessible from the network, resulting in a Denial of Service (DoS) condition.

V5 - Predefined Hierarchical Memory Layout

PLCs typically adhere to a specific programming format, which includes designations such as I for input, Q for output, and specific data sizes (e.g., X for BOOL and W for WORD). Additionally, a hierarchical address is employed to specify the location of the corresponding port. Adversaries may exploit this format to predict variables during the program's execution.

V6 - Real-time Constraints

Each execution cycle should complete its task within a pre-configured maximum cycle time to ensure real-time execution. However, for programs in which tasks are executed without interruption, i.e., non-preemptive multitask programs, one task must wait for the completion of another task before initiating the next scan cycle. To launch synchronization attacks, adversaries can generate loops or flood the target PLC with a large number of I/O operations to extend the execution time slice.

2.3.2 Security Goals

The security goals that adversaries aim to undermine through control logic injection attacks are associated with the security properties of the CIA model [60]. In the following section, we provide a more detailed description of each security property.

Confidentiality

It is a security property that ensures critical information is only disclosed to authorized entities. There is a vast amount of sensitive information in PLCs, including the PLC vendor, Media Access Control (MAC) address, and I/O data from sensors and actuators, among other details. The attacks violate the confidentiality of PLCs by stealthily monitoring the execution of PLC programs, leveraging existing vulnerabilities (e.g., V2 and V3).

Integrity

It is a security property that prevents unauthorized modification or introduction of information by unauthorized users or systems. The attacks violate integrity by compelling PLCs to induce unsafe behaviors in the physical process, primarily exploiting vulnerabilities V4 and V5.

Availability

It is a security property that ensures that data, systems, or devices are accessible when required. The loss of availability could undermine the industrial process, as the timeliness of

real-time information is often crucial for the control process. The attacks violate availability by exhausting PLC resources (e.g., memory or processing power) and causing a DoS condition.

2.3.3 Control Logic Injection Attack Scenarios

Most control logic injection threats are executed through one of three attack scenarios. In the following, we describe these scenarios with the assistance of our provided application in Figure 2.3 and provide an overview of previous studies discussing control logic injection attacks.

Scenario 1 - Attackers have access to the EWS

For this type of attack, adversaries can access the EWS (see Figure 2.5). Attackers may be internal, meaning those who already have access to the engineering station, or external, indicating those who can exploit specific vulnerabilities [55, 56] in the EWS to gain access.

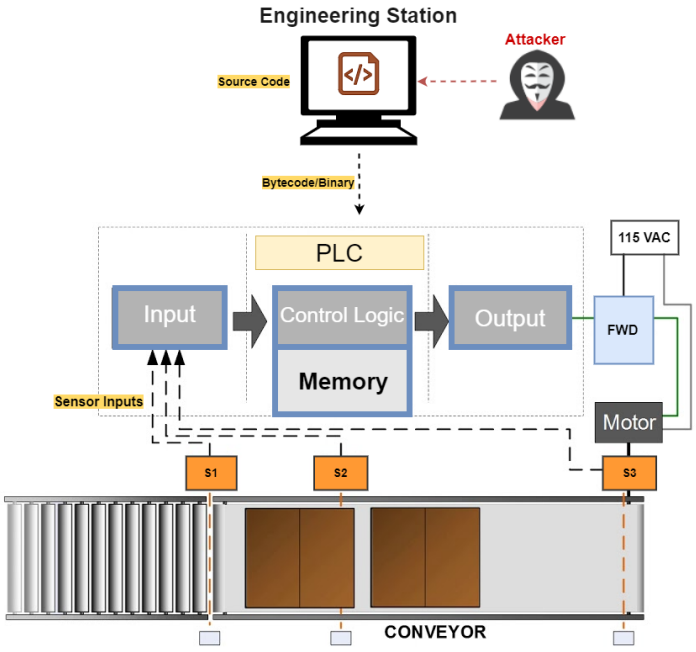


Figure 2.5: Attack Scenario 1 [8]

At the source code level, attackers aim to perform a stealthy injection in a manner that does not introduce noticeable changes to the fundamental functionality of a PLC program. This injection may also be disguised as mistakes commonly made by novice programmers. This implies that attacks can be concealed by appearing as unintentional instances of poor coding

practices, as demonstrated in [61, 62]. The authors of these works concentrated on attacks against graphical languages, such as Ladder Diagrams (LD), because slight modifications to such languages may not be readily observable.

Serhane et al. [61] explored LD code vulnerabilities and poor coding practices that may serve as the root cause of bugs, subsequently exploited by attackers. The authors demonstrated that attackers could generate uncertain output variables; for instance, forcing two timers to control a specific output value could lead to a race condition (V1). Such a scenario might result in significant damage to the infected devices, akin to the Stuxnet incident [17]. Another scenario highlighted by the authors is that adversaries can compromise certain functions, set specific operands to desired values, or introduce empty branches/jumps (V3). To achieve stealthy modification, attackers could use instruction sets or determined commands to insert false parameters and values (V2). The authors also discussed the possibility of applying infinite loops via jumps and using timers to activate the branch containing malicious instructions only at a time determined by an attacker. This scenario could severely slow down or even halt the operation of the attacked device.

Valentine [62] introduced an attack scenario where an adversary installs a jump to a subroutine function and changes the intercommunication between two or more ladders in an LD code (V3). The author showed that an attacker with access to an engineering station could use a Jump (JMP) command to insert their own malicious code at the mislabeled location and cause multiple errors before the Return (RTN) command returns the code to the intended location. This leads the PLC to run the original control logic program up to the point of the JMP, read the inserted code, and then continue along the original path, with the possibility of introducing a new set of parameters.

In addition to these works, McLaughlin et al. [63] introduced a control logic attack to confuse the behavior of a PLC. The authors analyzed an I/O trace of the exposed PLC to produce a set of inputs to achieve the desired PLC outputs. Their attack comprises two steps. First, they speculated on what the control logic looks like by constructing a model of the PLC's internal logic from the captured I/O traces. Then, they searched for a set of inputs that cause the model to calculate the desired malicious behavior. Their attack was evaluated against a set of representative control systems, proving that it is a feasible threat against insecure sensor configurations. Please note that [63] can also be classified among those attacks grouped in scenario 3. This is because the manipulation happens at the PLC's input level. However, since the authors assumed in their study that an attacker has access to the EWS and can read the source code, we elaborated on this work in scenario 1.

Scenario 2 - Attackers have access to the Control Network

In this scenario, the EWS is inaccessible to attackers, but they can reach the control network, as illustrated in Figure 2.6.

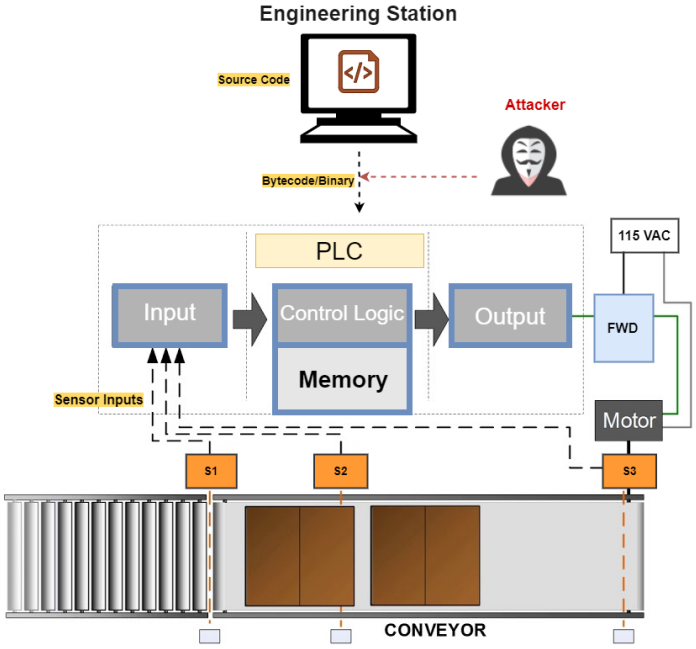


Figure 2.6: Attack Scenario 2 [8]

In such a situation, adversaries aim to intercept, modify transmissions, and exploit vulnerabilities in network communication [57–59]. This involves using packet-sniffing tools like Wireshark[7] to capture the network traffic between the EWS and the victim PLC(s). Subsequently, they extract program bytecode/binary from specific packets and maliciously alter the control logic using one of the following techniques:

- Modifying the control logic code in its decompiled format:

To achieve this objective, adversaries initiate control logic attacks by initially reverse engineering the program in its bytecode/binary format. Subsequently, they modify the decompiled code, recompile the altered code, and ultimately transmit the malicious code to the Programmable Logic Controller (PLC) over the network. Such attacks occur during payload transmission between engineering stations and PLCs, or vice versa. Studies within this research group have investigated program reverse engineering, a primary challenge in such attacks. This challenge arises from the fact that several features of PLCs are not supported

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

in normal instruction sets [64, 65]. However, existing studies such as [66–71] have discussed modifying decompiled code using various reverse engineering methods.

McLaughlin [67] executed an injection attack on a train interlocking program, reverse engineering a malicious program utilizing decompiled configuration code. This code facilitated the extraction of the field-bus Identifier (ID) dedicated to the PLC model and vendor. Subsequently, hints about the physical process structure were retrieved. Based on these clues, McLaughlin designed a program generating unsafe behaviors for the train, resulting in conflict states for signals. Notably, he targeted timing-sensitive signals and switches as part of a real attack scenario, successfully overcoming the security solution presented in [72] without alarming the detector. In a subsequent work, McLaughlin et al. [68] introduced the *SABOT* attacking tool, requiring a high-level illustration of control logic. The tool utilized public channels to obtain descriptions, creating specific behavior for control processes. McLaughlin leveraged vulnerability V4 of the control logic, focusing on the security goal I. However, these attacks were limited to Siemens PLCs, and the author did not provide sufficient illustration of the reverse engineering method employed.

In [66], the authors introduced a manual reverse engineering approach, developing a decompiler called *Laddis*, capable of decompiling low-level binary ladder logic to a higher-level presentation. *Laddis* was utilized for three control logic injection attacks, interfering with engineering operations of downloading and uploading PLC control logic. These scenarios involved injecting malicious control logic, replacing it with normal logic, or uploading malformed logic to crash the engineering software. *Laddis* could decompile ladder logic program network traffic in both upload and download directions but was limited to the Programmable Controller Communication Commands (PCCC) protocol and Allen-Bradley RSLogix 500 engineering software. The authors targeted security goal I, aiming to stay stealthy, and leveraged vulnerability V4 of the control logic for a successful attack.

Keliris et al. [69] presented the open-source decompiler *ICSREF*, which reverse engineers CodeSys-based control logic and creates malicious payloads. The authors demonstrated that skilled attackers can enable dynamic process-aware payload generation, allowing sophisticated attacks even against air-gapped systems without prior knowledge. *ICSREF* leveraged vulnerability V4, manipulating Proportional-Integral-Derivative (PID) controller functions and parameters. The tool deduced the physical features of the process, enabling the creation of substituted binaries for severe attacks.

Kalle et al. [70] demonstrated the remote attack *CLIK* on the control logic of Schneider PLCs, specifically Modicon M221 and its vendor-supplied engineering software (SoMachine-Basic). *CLIK* aimed to introduce malicious logic automatically into a target PLC. The authors developed a decompiler called *Eupheus* as part of this work, transforming low-level

control logic into a high-level IL program. *CLIK* leveraged vulnerability V4 of the control logic, targeting security goal I, but was limited to Modicon M221 PLCs.

A group of researchers introduced the control logic forensics framework *Reditus* [71] for control logic injection attacks. It automatically extracts and decompiles control logic from network traffic dumps without manual reverse engineering or prior knowledge of ICS protocols. *Reditus* leveraged vulnerability V4 to perform control logic injection attacks on Modicon M221 PLC and its SoMachine-Basic engineering software.

- Modifying the control logic code in its compiled format:

Here, attackers intercept packets containing program bytecode and replace the original bytes representing the code with malicious ones. The modified bytes are either identified using reverse engineering methods beforehand or pre-recorded from another session prior to the attack. After the modification, attackers push the crafted packet to the PLC over the network.

Several studies discussed hijacking specific network packets between the EWS and PLCs. Beresford [16] exploited packets (e.g., ISO-TSAP[8]) between PLCs (Siemens S7-300 and S7-1200) and their engineering software, i.e., Totally Integrated Automation (TIA) Portal. These packets contain critical data, such as variable names, data block names, and also PLC model and vendor information. The author mapped the values extracted from memory dumps to their corresponding variables in the PLC and successfully manipulated these variables to cause undesired behavior. Beresford tested his attacks only on S7 PLCs and its S7 Communication (S7Comm) protocol, revealing new vulnerabilities V4 and V5 in S7 PLCs and targeting security goal I. However, Beresford made the entry point through network traffic, ignoring the fact that security measures could have enabled Deep Packet Inspection (DPI) between the PLC and the EWS.

In 2015, Klick et al. [31] presented an injection of malware into the control logic of a Siemens S7 PLC, precisely S7 314C-2 PN/DP, without disrupting the service. The authors showed that a knowledgeable adversary with access to a PLC can download and upload code to it, as long as the code consists of an MC7 bytecode. They introduced a so-called *PLCinject* tool, which crafts a payload with an Simple Network Management Protocol (SNMP) scanner and Proxy. Their investigations disclosed a vulnerability existing in the predefined memory layout of SIMATIC PLCs (V5); that allows attackers to place a malicious payload at the beginning of the main Organization Block (OB). This security gap changes the execution sequence of the control logic, turning the PLC into a gateway mode, i.e., security goal I was broken. Since the authors assumed that attackers are already familiar with the PLC's

[8] https://www.rfc-editor.org/rfc/rfc1006

memory layout, this study can also be grouped among the attacks in scenario 3.

Using *PLCinject*, Spenneberg [74] performed a worm that can spread from one PLC to another by copying itself and adapting the next target PLC to execute the worm along with the running control logic. The worm was designed using a state machine where the current state is stored in a global variable. At the start of each cycle, the appropriate code in the worm is called. Thus, the maximum cycle time is never violated. *PLC-Blaster* utilized a variety of anti-detection mechanisms, e.g., the worm evaded the anti-replay byte method, it was stored in the less used block of the running program on the target PLC, and could meet the maximum scan cycle limit. This severe attack implemented an endless loop triggering an error condition within the PLC with the impact of a denial of service condition. Please note that this work can also be presented with the attacks in scenario 3, as the author assumed that an attacker has access to the PLC's run-time environment, leveraging the vulnerability V5 and targeting the security goals C and A.

Lei et al. [75] demonstrated a worm that can break the security wall of the S7CommPlus protocol that Siemens SIMATIC S7-1200 PLCs utilize. The authors first used Wireshark software to analyze the communications between the TIA Portal software and S7 PLCs. Then, they applied reverse debugging software (WinDbg [9]) to break the encryption mechanism of the S7CommPlus protocol. Afterwards, they demonstrated two attacks. First, a replay attack was performed to start and stop the PLC remotely. In the second attack scenario, the authors manipulated the input and output values of the victim, causing serious damage to the physical process controlled by the target PLC. The presented worm in this work was designed to attack only specific PLCs, i.e., S7-1200, and its S7CommPlus protocol, exploiting the vulnerability V5 and targeting security goal I.

The researchers behind *Rogue7* [76] created a rogue engineering station that can impersonate the TIA Portal to the latest Siemens S7 PLCs line, running various firmware versions and using the latest S7CommPlus protocol. Their so-called *Rogue7* tool can perform several attacks against S7-1500 PLCs. They performed first a typical start/stop attack and then downloaded a replayed control logic program to the target PLC. In their most advanced attack - the stealth program injection attack - they managed successfully to maintain the source program as the engineer expects to see, while programming the victim PLC to run a malicious bytecode that the engineer will never see. They achieved their attacks through detailed reverse-engineering analysis, as well as key generation and cryptographic primitives. As part of their research, they analyzed versions of the S7 protocol running on different generations of PLCs, e.g., 1200 and 1500 family. The authors targeted security goal I exploiting the

[9] http://windbg.org/

vulnerability V5.

Hui et al. [77] investigated different potential exploits against Siemens S7-1211C PLCs and their TIA Portal software. The authors used Windbg and Scapy[10] tools in their investigations and showed that the anti-replay mechanism of the S7CommPlus protocol is vulnerable. Based on their findings, they performed several attacks against the tested PLC, e.g., session stealing, phantom PLC, cross-connecting controllers, and a control logic injection attack. They proved that if adversaries could obtain information about the anti-replay mechanism and 20-byte integrity check that S7 PLCs use, the possibility of executing a malicious payload is very high. In a follow-up work, Hui et al. [78] analyzed and identified specific necessary bytes to craft valid network packets and demonstrated a successful replay attack on S7-1200 PLCs. The presented attacks in this work opened the door for attackers to manipulate the control logic that the victim PLC runs. However, both works were limited to S7 PLCs and did not take any security means, that might be implemented, into account. The authors exploited the vulnerability V5 in S7 PLCs and targeted security goals C and I.

Scenario 3 - Attackers have access to the PLC's runtime environment

Figure 2.7 depicts this scenario. Attackers, here, have neither access to the EWS nor to the payload transmitted between the stations. But, they can still speculate about the logic of the control program by reaching the PLC's runtime environment including the PLC firmware, memory, and I/O traces.

At this level, existing works have explored two types of threats. First, there is a focus on firmware modification, also known as control logic corruption. In this scenario, adversaries maliciously target the firmware or memory of a PLC. Secondly, there is control logic manipulation, where adversaries alter the input values provided to the control logic. It is important to note that input manipulation can originate from two main sources: communication between the PLC and its engineering software, or from sensor readings scanned during the execution of the control logic (see Figure 2.3). In the following, we present the existing studies on each of these threats.

- Control Logic Corruption Attacks

In recent years, a number of attacks aiming at manipulating the firmware of PLCs have been published. Basnight [79] explored vulnerabilities in a well-known PLC (Allen-Bradley ControlLogix) to perform an intentional firmware modification attack. The author analyzed the firmware update validation to discover weaknesses that facilitate firmware counterfeiting.

[10] https://scapy.net/

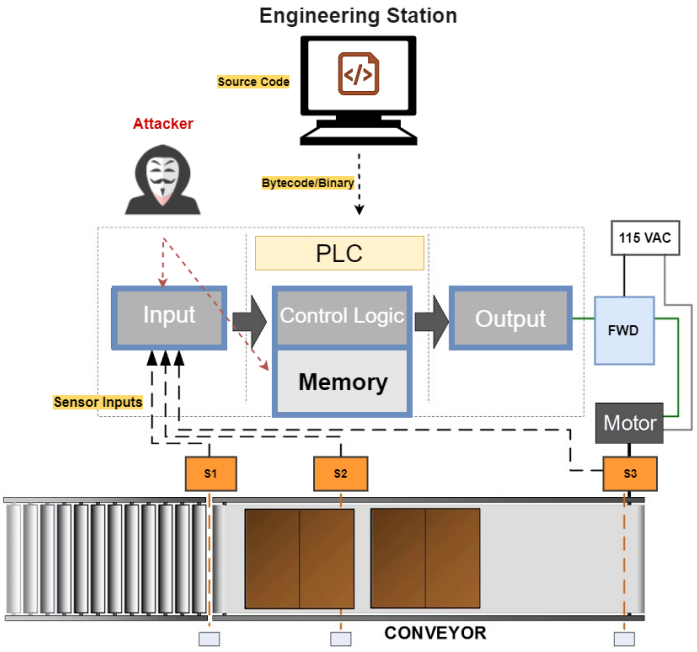


Figure 2.7: Attack Scenario 3 [8]

After that, he created a counterfeit firmware sample that was uploaded and executed on a ControlLogix L61 PLC. The firmware sample used to compromise the PLC was obtained directly from the vendor website as a firmware update package. In a follow-up work, Basnight [80] presented two methods of control logic corruption attacks on Allen-Bradley PLCs. The first method used immediate values in instructions to infer a reasonable image base, while the second method used a hardware debugger to halt a PLC to obtain a memory dump. The image base could be found by manually analyzing common instruction patterns in the memory dump. This vulnerability allowed the authors to execute arbitrary code in a PLC by exploiting the firmware update feature. Both studies were tested on Allen-Bradley PLCs, targeting security goal I, and exploited vulnerability V4 existing in ControlLogix PLCs.

Peck et al. [81] demonstrated how, using commonly available tools, an adversary can learn how firmware is loaded into different field device Ethernet cards and write his own malicious firmware before loading that malicious firmware into the field device Ethernet cards. In their experiments, the authors found a lack of source and data authentication on firmware uploads in both Koyo and Rockwell Automation PLCs. As a proof of concept, they uploaded modified web pages that are available from a similar PLC module. This study compromised vulnerability V4 in the control logic of PLCs and targeted security goal I.

Schuett [82] performed control logic corruption attacks on PLCs using the Joint Test Action

Group (JTAG) interface. The author first extracted the firmware image and performed static and dynamic analysis to identify execution paths and generate memory dumps. The firmware is then re-packaged with a malicious attack that triggers a DoS attack with a combination of control commands by writing a sentinel value to an unused flash memory area. Rais et al. [83] proposed a JTAG-based framework, namely *Kyros*, for reliable PLC memory acquisition. *Kyros* systematically creates a JTAG profile of a PLC through hardware assessment, JTAG pins identification, memory map creation, and optimizing acquisition parameters. As a case study, the authors implemented *Kyros* on Allen-Bradley PLCs and revealed the tested PLC's memory dumps that are basically used in typical firmware modification attacks. Both studies targeted security goal I and exploited vulnerability V4.

Garcia [84] presented *HARVEY*, a rootkit that, once installed in the device's firmware, has the capability to inspect the control logic and then modify its instructions. The rootkit can evade operators viewing the HMI by faking sensor input to the control logic to generate adversarial commands that an operator would expect to see. *HARVEY* could enlarge the harm to the control process and result in extremely huge failures without operators being alarmed about the ongoing attack. However, the authors conducted their investigation on the assumption that attackers already have access to the PLC firmware, which was less observed than the user program. *HARVEY* was also aware of the control process that the PLC handles and could exploit vulnerabilities V4 and V5 to intercept the measurement inputs that are used by this process, targeting security goals C and I.

- Control Logic Manipulation Attacks

Many academic endeavors have explored the manipulation of I/O data exchanged between engineering stations and PLCs over specific network packets. Among the various attacks discussed by Beresford in [16], it was demonstrated that an adversary could unveil the mapping technique between the names and input variables utilized by the control logic running in the victim PLC. Such knowledge would empower the adversary to modify the program based on specific needs, potentially causing disastrous damages to the physical processes. However, the likelihood of successfully mapping the variables through memory probing is minimal.

In the work of Lim et al. [85], a hijacking attack was executed to interrupt and alter the command-37 packets transmitted between a PLC (Schneider Tricon) and its EWS. Crafted packets containing inputs to industrial PLCs, commonly employed in nuclear power plant settings, were manipulated. The authors employed reverse engineering techniques to identify the general structure of the Tricon communication protocol. Subsequently, a successful control logic injection was conducted, causing common-mode failures for all modules and

necessitating a reset of the Tricon PLC. This work exploited vulnerabilities V4 and V5, targeting security goals I and A. However, the authors did not consider the potential detection of the attacker payload or input data by a DPI security method, had it been enabled.

To circumvent DPI, Yoo et al. [86, 87] introduced a stealthiness injection by dividing the malware transmission into small fragments and transferring one byte per packet with substantial padding of noises. This approach is due to the fact that the DPI mechanism relies on merging packets to detect any abnormal traffic, and thus it is unable to disclose a very small payload. The authors exploited vulnerability V5 to control the victim PLC by injecting malicious code. The studies also discussed the possibility of conducting a stealthy program modification and input manipulation at the network level. These attacks were applied to two industry-scale PLCs, specifically Modicon M221 and Allen-Bradley MicroLogix 1400. The attack scenarios presented in these works successfully evaded well-known intrusion detection methods such as signature-based intrusion detection and payload-based anomaly detection [88].

A research group in [89] demonstrated a control logic injection attack aimed at manipulating the data stored in the PLC's intermediate register. To achieve this, the authors read data values representing inputs, inverted these values, and wrote them back to the victim. This operation was repeated in a loop until the PLC ceased running. The authors exploited vulnerabilities V4 and V5, targeting security goals C and I. However, their attack was limited to Siemens PLCs, and did not consider implemented security measures.

In [90], the authors introduced Ladder Logic Bombs (LLB), a combination of a typical control logic injection attack and a control logic manipulation attack. The adversary's malware was covertly inserted into an already running LD program as a regular subroutine that is only activated under specific conditions. Once triggered, the malware impersonates a legitimate sensor reading with false values. LLB was customized to bypass any inspection method by providing legitimate instructions/names similar to those previously used in the same industrial setup. It is noteworthy that this study targeted security goal I and successfully exploited vulnerabilities V4 and V6, causing unsafe behavior of the attacked PLCs.

McLaughlin et al. [63] conducted a Controller-aware False Data Injection (CaFDI) attack, allowing adversaries to obtain partial information about the target subsystem and produce predictable malicious results. CaFDI operates in two main steps: first, constructing a model representing the PLC control logic based on the I/O traces, and second, seeking a set of inputs leading the model to calculate the desired malicious behavior. In other words, CaFDI searches for an I/O path that could be used as the sensor readings for the control logic. Evaluation on a set of representative control systems demonstrated the feasibility of exploiting insecure sensor configurations.

Additionally, Xiao [91] demonstrated that an attacker could collect fault-free I/O traces and formalize a representative model using a Non-Deterministic Autonomous Automation with Output (NDAAO). He built a word set of NDAAO sequences, seeking unobserved false sequences from the word set to infect the exposed sensors. His attack exploited vulnerability V4, resulting in the improper operation of the physical process controlled by the infected PLC.

In a similar setting, Abbasi et al. [92] combined the control flow of the program to exploit certain pin control operations, leveraging the absence of hardware interrupts associated with the PLC's pins. Their attack allows an attacker to reliably take control of the control process while remaining stealthy to both the PLC runtime and the ICS supervisor who observes the process through an HMI. This attack did not require modifications of the PLC logic, traditional kernel tampering, or hooking techniques normally observed by anti-rootkit tools. The authors introduced two variations of their attack implementations: the first allows extremely reliable manipulation of the process at the cost of requiring root access, while the second implementation allows manipulation without root access.

2.4 Online-Offline Control Logic Injection Attacks

Control logic injection attacks can also be classified based on their impact on the physical process into two groups: online attacks and offline attacks. In the following, we elaborate on each group in more detail.

2.4.1 Online Attacks

They are designed to modify the original control logic program by utilizing its engineering software. The physical process controlled by the infected device is affected immediately after the successful injection of malicious code. Figure 2.8 illustrates this attack scenario. These types of attacks are constrained and necessitate that attackers must be connected to the target at point zero for the attack, thereby increasing the likelihood of being discovered by ICS operators beforehand or being detected by security measures.

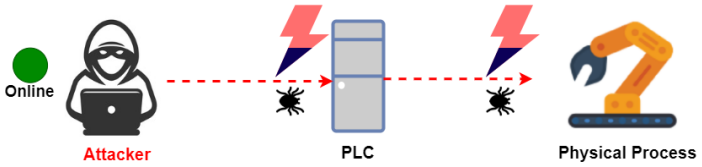


Figure 2.8: Disrupting the physical process online [6].

All the aforementioned attacks are considered online attacks, as adversaries are required to be connected (online) to the target device or the system's network at the moment of the attack.

2.4.2 Offline Attacks

The attacks in this class are quite similar to the ones mentioned in the prior class but differ in that an adversary does not aim to attack the physical process immediately after gaining access to the target device. This means that they patch their malicious code once they access an exposed PLC, then close any live connection with the target, keeping their patch inside the PLC's memory in idle mode. Subsequently, the malicious code/patch triggers the attack and compromises the physical process at a later time determined by the attacker, even without being connected to the system network (see Figure 2.9). Such attacks are more severe than the online ones, as the patched PLC continues executing the original control logic correctly without being disrupted for hours, days, weeks, months, and even years until the very moment determined by the attacker.

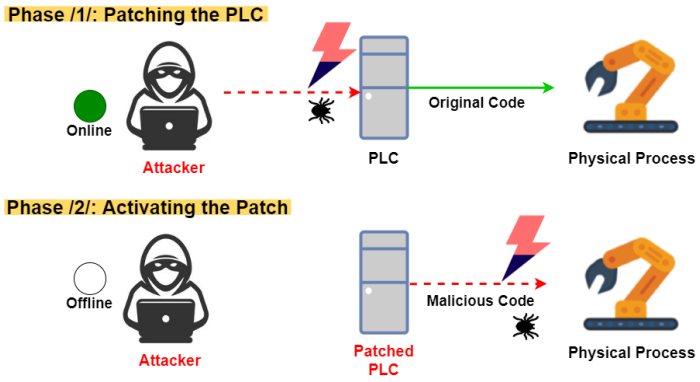


Figure 2.9: Disrupting the physical process online [6].

A prime example of an offline attack is presented in [3]. We introduced a novel approach involving the injection of a Time of Day (ToD) interrupt code into the target PLC, which disrupts the execution sequence of the control logic at the specified time set by the attacker. Our evaluation results demonstrated that an attacker could confuse the physical process even when disconnected from the target system. Although our research work was tested solely on an older S7-300 PLC and was specifically aimed at forcing the PLC into *STOP* mode, the attack proved successful and managed to interrupt the execution of the original control logic code running in the patched PLC.

The only practical means of detecting our attack occurs when the Industrial Control System

(ICS) operator requests the program from the PLC and compares the online code running in the infected device with the offline code stored on the engineering station. However, in this chapter, we overcome this challenge, as illustrated later in Section 4.2.

2.5 Real-world Control Logic Injection Attacks against ICSs

In the past, specifically before 2010, ICS environments were not the focus of attackers, and only a few successful attacks were reported. This can be attributed to several factors. Firstly, attacking industrial plants is more challenging than targeting IT systems, requiring specialized engineering knowledge about the industrial system adversaries aim to compromise. Another contributing factor is that sharing incidents publicly is not a common practice in the industrial domain. It is believed that companies tend to avoid reporting successful attacks to mitigate the risk of damaging their reputations.

However, a survey aimed at investigating different cyber-attacks against ICSs [36] revealed a significant increase in the number of reported incidents in recent years. In Table 2.1, we present a selection of attacks, specifically control logic injections that affected real-world industrial plants, sorted by the date of the attack, with the oldest incidents listed first. In this context, we provide details such as the year the incident occurred, the name, and the location.

Table 2.1: Chosen reported incidents targeted real-world industrial plants, listed in a timeline

Year	Type	Name	Place
2000	Attack	Maroochy Water [38]	Australia
2010	Malware	Stuxnet [17]	Iran
2010	Malware	Night Dragon [39]	Kazakhstan, Taiwan, Greece and U.S
2011	Malware	Duqu [40]	Europe, Asia and North Africa
2012	Malware	Shamoon [41]	Saudi Arabia and Qatar
2013	Malware	Havex [42]	U.S and U.K
2014	Malware	German Steel Mill [20]	Germany
2014	Malware	Trisis [45]	Middle East
2015	Malware	BlackEnergy [46]	Ukraine
2016	Malware	CrashOverride [47]	Ukraine
2017	Malware	NotPetva [48]	Ukraine
2017	Malware	TRITON [21]	Saudi Arabia
2018	Malware	VPNFilter [51]	Worldwide
2020	Malware	EKANS [52]	U.S and Europa
2021	Attack	No Name [53]	Australia

In 2000, a malicious attack occurred in Queensland (Australia), specifically targeting Maroochy Water services. According to the report [38], a former employee exploited the system that utilizes radio communication to connect nearly 150 pumping stations in the field to the control center. The attacker successfully leveraged a vulnerability in the radio channel between the stations, masquerading as a legitimate controller and sending control commands to the pumping stations. This resulted in the adversary halting certain pumps and blocking them from sending warnings to the legitimate control center, leading to the release of millions of liters of polluted water into the public water supply for almost three months.

Stuxnet [17], reported in 2010, is considered the first worm designed to maliciously cause physical damage in ICSs. It stands out from previous worms/malwares due to its complexity, advanced skills required for execution, and deep knowledge involved. Stuxnet targeted Siemens PLCs, specifically S7-325 and S7-417, programmed and monitored by Siemens STEP 7 software. The adversaries exploited zero-day vulnerabilities, spreading the worm via removable media such as USB, later inserted into the control system. They also used sophisticated methods to disguise the worm as legitimate code, evading detection by the ICS operator. Notably, the worm used valid certificates with private keys hijacked from two individual companies to sign specific Windows drivers [22]. The altered control logic disrupted the regular operation of the drives by periodically changing the motor speeds from 1,410 Hz to 2 Hz, 1,064 Hz, and repeating. The Stuxnet attack played a significant role in raising awareness of ICSs security, challenging the belief that isolated and air-gapped systems were immune to attacks.

Night Dragon [39] employed a combination of attack techniques targeting critical information and sensitive data in gas, oil, and chemical industrial plants. The attackers aimed to access financing files related to gas and oil fields, potentially causing millions of dollars in losses and impacting the global energy sector. Successful collection of information from PLCs preceded the launch of the Night Dragon attack.

Another incident in 2011, known as Duqu [40], involved sophisticated injection attacks against various control systems in different countries, including Switzerland, Netherlands, France, Ukraine, Iran, India, Vietnam, and Sudan. Duqu modules established communication with the Command and Control (Command and Control (CC)) server, patching further malicious codes for network enumerations and gathering information about the target system. Due to the similarity in the attack scenario, Duqu is believed to share source code with Stuxnet [17], indicating potential access to Stuxnet's source code. Unlike Stuxnet, Duqu was not designed for physical damage but focused on collecting information for future attacks.

Shamoon [41], in 2012, targeted Saudi Aramco in Saudi Arabia and RasGas in Qatar, aiming to sabotage the plants. Attackers utilized central computers (approximately 30,000

computers) with the Shamoon malware to infect other systems connected to the network. The malware spread through the network, deleting sensitive files that defined system operations. The actual damage caused by Shamoon remains unknown, but it is believed to involve the removal or destruction of data files related to production.

In 2013, Havex (also known as Dragonfly) exploited Remote Access Trojan (Remote Access Trojan (RAT)) to target ICSs, particularly SCADA systems and PLCs in the global energy sector [42]. Havex aimed to enable attackers to control victim systems and manipulate parameters, data, and control commands remotely. The malware employed tactics such as phishing attacks, watering hole attacks, and compromised vendors to deliver malicious code to employees and plant malicious RAT on their computers.

In December 2014, adversaries gained access to a steel industry in Germany, controlling the blast furnace [20]. The attackers utilized spear-phishing and other social engineering attack approaches to access the control network from the corporate network, causing severe physical harm to the infected system by compromising industrial components such as PLCs.

Trisis, reported in 2017, is believed to be the first publicly reported ICS malware targeting a petrochemical plant [45]. The attackers injected the corporate IT network of a petrochemical company and then infiltrated the control network by stealing credentials and accessing the engineering workstation. Despite a programming flaw limiting its execution, Trisis shut down the entire plant twice, demonstrating the potential for physical damage [49].

BlackEnergy [46] was introduced in December 2015, causing a significant blackout in Ukraine. Although there is no official report detailing the attack scenario, security experts believe that attackers injected the malware into a regional energy provider through a phishing attack, leading to a coordinated blackout.

The malware VPNFilter [51], reported in 2018, aimed at scanning routers and Network Attached Storage (NAS) in more than 100 countries [46]. It shared similarities with the RC4 implementation bug in the BlackEnergy malware, indicating potential cooperation between the founders of the two attacks. VPNFilter injected critical network components, including routers and NAS servers, utilized by various industrial plants.

In June 2020, a malware named EKANS targeted Honda offices in the United State (U.S.), Europe, and Japan [52]. Hackers created a customized ransomware to gain access to a Honda internal server, using a decryption approach to exploit the secret key encrypting messages exchanged with other stations. The attackers impacted the production line and industrial components in the lower level of the target system.

In November 27, 2021, a group of Chinese hackers launched a sustained ransomware attack on CS Energy's two thermal coal plants in Queensland [53]. The attackers attempted to bypass authentications of internal corporate systems to gain access to generators that circulate

3,500 Megawatt (MW) of electricity into the grid. The attack was thwarted by separating the company's corporate and operational computer systems, preventing the adversaries from accessing the generators. The attackers were reportedly less than 30 minutes away from shutting down the power.

CHAPTER 3

Investigating the Security of Non-Cryptographically Protected PLCs

Contents of this chapter are as follows:

Parts of this chapter have already been published in the following papers:

- W. Alsabbagh and P. Langendörfer, "A Remote Attack Tool Against Siemens S7-300 Controllers: A Practical Report," In: Jasperneite, J., Lohweg, V. (eds) Kommunikation und Bildverarbeitung in der Automation. Technologien für die intelligente Automation, vol 14. Springer Vieweg, Berlin, Heidelberg. [online]. Available: https://doi.org/10.1007/978-3-662-64283-2_1 [1].

- W. Alsabbagh and P. Langendörfer, "A Stealth Program Injection Attack against S7-300 PLCs," 2021 22nd IEEE International Conference on Industrial Technology (ICIT), 2021, pp. 986-993, doi: 10.1109/ICIT46573.2021.9453483 [2].

- W. Alsabbagh and P. Langendörfer, "Patch Now and Attack Later - Exploiting S7 PLCs by Time-Of-Day Block," 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS), 2021, pp. 144-151, doi: 10.1109/ICPS49255.2021.9468226 [3].

- W. Alsabbagh and P. Langendörfer, "A Control Injection Attack against S7 PLCs - Manipulating the Decompiled Code," IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society, 2021, pp. 1-8, doi: 10.1109/IECON48115.2021.9589721 [5].

This chapter examines the feasibility of compromising the security of non-cryptographically protected PLCs, and conducting a stealthy control logic injection attack afterwards. To this end, we developed a full attack chain to maliciously alter the control logic, and tested our approach on an S7-300 PLC. For all the experiments conducted in this chapter, we use the attacker model presented in figure 3.4, assuming that the adversary gains access to the level-3

network of the Purdue Model [1] (i.e., control network). This assumption is based on real ICS attacks e.g., TRITON [49] and Ukrainian power grid attack [46] that gained access to the control center via a typical IT attack vector such as infected USB sticks and social attacks e.g., phishing attacks. We also assume that the adversary gains access to the target's network e.g., he can run Wireshark or any other packet-sniffing tool to capture exchanged packets between the connected stations. After the level-3 network access, an attacker can make use of software and libraries to communicate with the target PLC over the network. As our attacker model and assumptions have already been reported to hold true in reports on real world attacks, we are convinced that our attack is a realistic one.

The rest of the chapter is structured as follows. In Section 3.1, we investigate the possibility of bypassing a password-protected PLC. Afterwards, we present and implement our stealthy control logic injection approach in Section 3.2.

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

3.1 Authentication Issues in PLCs

Contents of this section are as follows:

Most automation vendors provide their PLCs with a user-authentication approach based on entering a password each time the user wishes to make changes in the PLC's control logic. This means that if a legitimate user wants to read, write, or update the program running in a PLC using its corresponding engineering software, the latter requests a password and then launches a specific protocol, normally proprietary to the vendor, to authenticate the user. This procedure aims to protect PLCs, specifically their control logics, from any unauthorized access.

In this section, we take a non-cryptographically protected PLC from Siemens (e.g., S7-300) as a test device and conduct an intensive study based solely on examining the network traffic exchanged between the PLC and its engineering software (TIA Portal software). Our investigations reveal serious vulnerabilities and exploitable flaws in the design of the authentication protocol, such as a small-sized encryption key, missing nonces, and a very weak encryption algorithm. Our findings in this section were confirmed by testing their proof-of-concept using the attacks derived from the *MIRTE ATTCK* tactics and techniques [126]. We perform different types of authentication attacks, including unauthorized sniffing passwords, retrieving the plain-text password, and setting/resetting/updating password attacks. Finally, we discuss the fundamental design issues in the authentication mechanism.

3.1.1 Password Policy

PLC vendors secure their devices with password-based user authentication to prevent unauthorized access attempts aimed at tampering with the control logics currently executed by the PLCs. According to the S7-300 documentation [95], the S7-300 PLCs offer users three levels of access control: no protection, write protection, and read/write protection. By default, the

PLCs are set to the no-protection level, requiring no authentication. In this level, users can access the PLC's program without any restrictions.

In the write protection level, any attempt to modify the control logic—such as downloading a new program, updating the configuration of a currently running program, or clearing data requires the user to provide the password with which the PLC is protected. The read/write protection level is the most restrictive among the three, where any read from or write to the PLC is password authenticated.

Engineering software (e.g., TIA Portal) enables a legitimate user to set an 8-character *ASCII* password. If the configured password is less than 8 characters, the engineering software pads the remaining characters with white spaces. Setting a new password involves changing the protection level in the engineering software to either write or read/write protection. Subsequently, the user sets the desired password before downloading the changes to the PLC's memory. Additionally, a copy of the password is locally stored in the project files on the EWS. After successful password configuration, the PLC authorizes each command delivered, based on the chosen protection level, by validating the provided password.

In subsequent interactions, the engineering software will automatically pad the password within the request packets sent to the PLC, initiating a specific authentication protocol to check the password for each request, as explained in the next subsection.

3.1.2 Authentication Protocol

As mentioned earlier, PLCs authenticate users by initiating proprietary protocols with vendors, known as authentication protocols. These protocols are determined by a specific set of steps that PLCs follow after receiving new requests from users. For example, the authentication protocol used in S7-300/S7-400 PLCs is described in Figure 3.1. In subsequent interactions, the engineering software will automatically append the password to the request packets sent to the PLC, initiating a specific authentication protocol to verify the password for each request, as explained in the next subsection.

Technically, a client (TIA Portal) encrypts the password by using a shared key (K), before sending it to the PLC with an authentication packet. After that, the PLC replies to the TIA Portal by sending an error code that has a value '0' if the authentication is successful and other values for a failed authentication.

Algorithm 3.1 depicts the encryption algorithm utilized in the authentication protocol. As can be seen, it takes an 8-bytes password ($ASCII$) as input ($P0$ .. $P7$), and a single byte secret key (K). Then it substitutes each character of the password with the help of a proprietary substitution table. Afterwards an XOR process is performed for the first two

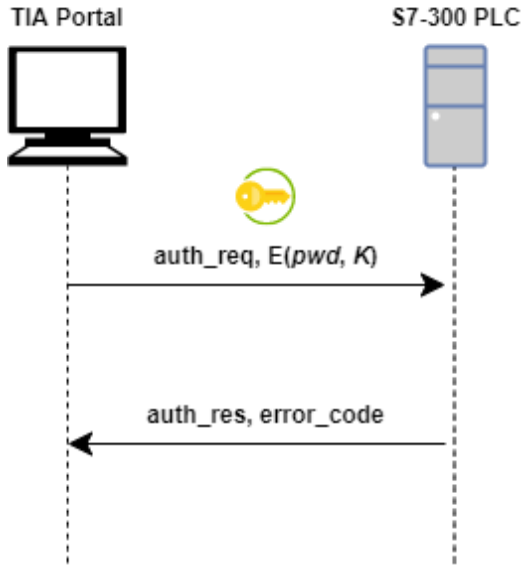


Figure 3.1: S7 authentication protocol used in S7-300 PLCs

Algorithm 3.1: Pseudocode of S7-300 Encryption Algorithm

Input: Password (*P0* .. *P7*), K (one-byte secret key)
Output: Encrypted_Password (*E0* .. *E7*)
for $i = 0$ *to* 7 **do**
  $S_i = Sub(P_i)$
  if $i < 2$ **then**
    $E_i = k \oplus S_i$
  else
    $E_i = k \oplus E_{i-2} \oplus S_i$
  end
end

characters with the secret key K, while the remaining characters are encrypted by performing *XOR* with K and the output of the encryption process for the *i-2* character i.e., *Ei-2*. Our investigation showed that such a protocol does not utilize any integrity checksum which exposes it for different attacks as illustrated in the next subsection.

3.1.3 Authentication Protocol Vulnerability

The authentication protocol, depicted in Algorithm 3.1, applies an uncomplicated encryption mechanism. This exposes the protocol to brute-force attacks. For instance, if an adversary managed successfully to brute-force one plain-text pair, he eventually would be able to recover the secret key (K) used to encrypt the 8 *ASCII* characters password. Furthermore, our

analysis showed that S7-300 PLCs use a key space as small as 8 bytes ($2^8 = 256$) characters which allows exhaustive key search attacks. Another vulnerability that we noticed in our experiments is that the PLC does not vary the secret key (K) from a session to another. This eases the attacker's task much more since he can record remarkable and repeated paradigms in the encrypted messages once he figures out the key.

3.1.4 Memory Structure

Configuring the PLC with a new password involves the following steps: the user initially opens the engineering software (e.g., TIA Portal) and adjusts the protection level from no protection to either write or read/write protection level. Subsequently, the user selects an 8-character password and uploads the new configuration into the PLC, precisely into its memory. The memory of the PLC is organized into labeled blocks, each containing specific types of information, as illustrated in Figure 3.2.

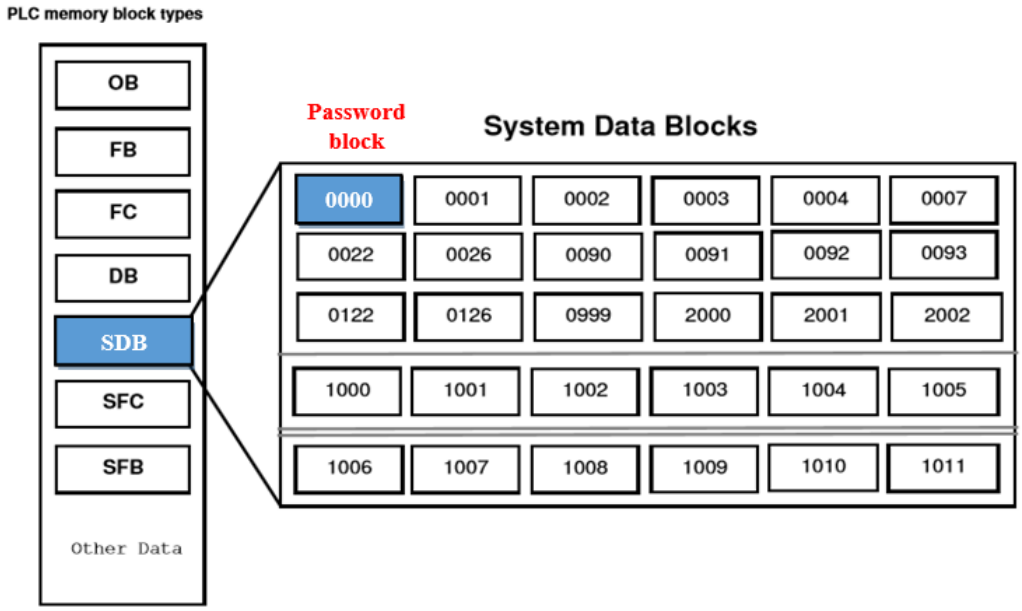


Figure 3.2: S7-300 PLC memory structure [2]

The PLC utilizes most of these blocks to store the user control logic program in sub-blocks based on individual tasks. Consequently, the user program downloaded to the PLC is also segmented into blocks. Through the analysis of these blocks and sub-blocks, we discovered that the user password is situated in the System Data Block (SDB). This block is further divided into multiple sub-blocks, each assigned a distinct role. For example, sub-blocks ranging from 0000 to 0999, as well as from 2000 to 2002, contain data that undergoes changes

in each new download process. The remaining sub-blocks are categorized into two groups: sub-blocks numbered from 1000 to 1005 contain data, while sub-blocks from 1006 to 1011 contain configuration data.

Our investigations revealed that whenever a user downloads a new control logic program into the PLC, the latter updates the data in all sub-blocks of the SDB, except for sub-block 0000, which stores the user password. Consequently, we can conclude that if an attacker aims to modify the user password, they must first clear the content of the System Data Block 0 (SDB0) block using a customized command before being able to write a new password into SDB0 with another command.

3.1.5 Revealing the Plain-text Password

In the following, we present our attack approach to retrieve a PLC's password in plain text. To accomplish this, we first need to identify the specific packet and byte shift where the password is located. Subsequently, we decode the encoded bytes to obtain the password in clear text.

- Sniffing the Password over the Network

The first step that an attacker needs to take is capturing a specific packet containing the user password from the transmitted packets between the TIA Portal and PLC. To achieve this, a network sniffing tool (Wireshark) is employed to record the entire network traffic during a password-setting process. As the user sends a new configuration command to the PLC for password setting, we create and download different passwords (20 passwords) one by one to the PLC to gather a sufficient number of samples. For each password, we record the network traffic and filter the packets, retaining only the TCP streams. Subsequently, we compare the TCP packets of all 20 passwords using a byte comparison tool, namely Burp Suite Comparator[2]. This tool aids in identifying differences and similarities between the packets.

This method enables us to determine the precise packets where the passwords are located, along with the specific position of the password within each packet. Furthermore, the 8-character password is not transmitted in plain text; it is encoded in each packet. This implies that the engineering software employs a specific encoding scheme to encode the password before downloading it to the device. It is noteworthy that when the user sets the PLC with no-protection level, we observed that the packets sniffed during the download of a new configuration do not differ in size compared to other protection levels, i.e., write protection

[2]https://github.com/nccgroup/pcap-burp

and read/write protection. Based on this observation, we can conclude that the PLC fills the password space with 8 random bytes when no-protection level is selected.

- Reverse Engineering for the Encoding Scheme (Decoding Scheme)

In the previous step, we determined the location of the 8 bytes within the configuration packet. The next step is to retrieve the plain-text version by decoding each byte of the password. Attempts using typical encoding schemes such as Uniform Resource Identifier (URL), Hexadecimal (HEX), *ASCII*, Base64, single-byte, and multiple-byte *XOR* failed to yield the password in its plain-text version.

Other encoding scheme algorithms, including full-fledged cryptographic methods (Data Encryption Standard (DES), Advanced Encryption Standard (AES), Rivest Cipher 4 (RC4), etc.), and hashing (Message Digest 5 (MD5), Secure Hash Algorithm (SHA)-512, etc.), were ruled out in our investigations for the following reasons. First, our investigations revealed that the TIA Portal and PLC do not establish handshake messages to generate a secret key (non-cryptographic) before any password configuration. Second, assuming that the TIA Portal uses hashing functions before sending the password to the PLC, the encrypted bytes should be entirely mixed compared to the plain text version. However, in our case, this is not true; a byte-by-byte encoding scheme is used in the ciphertext.

The *XOR* encoding scheme is widely used in the cybersecurity world and is well-suited for devices with limited resources. As discussed earlier, the scheme used to encode the password does not follow a regular *XOR* approach, such as single-byte or multiple-byte *XOR*. Considering the byte-by-byte encoding scheme used here, along with the lightweight encoding requirements, we focused on applying a customized *XOR* scheme. To achieve this, we sampled a representative list containing pairs of plain-text passwords and encoded-text passwords from the network. Subsequently, we ran a script to brute-force each byte. For a clearer understanding of the decoding process applied in this work, we illustrated a graphical diagram in Figure 3.3.

After the decoding process, we managed successfully to retrieve the plain-text password that the PLC is protected with. All our decoding scheme results are presented in Table 3.1.

- Compromising the Write Protection Level

In this scenario, the PLC is configured with the second protection level, which is the write protection. Under this level, the user is required to provide the correct password only when downloading or updating the control logic program in the PLC, while uploading the program from the PLC does not require authentication. Consequently, an attacker can create

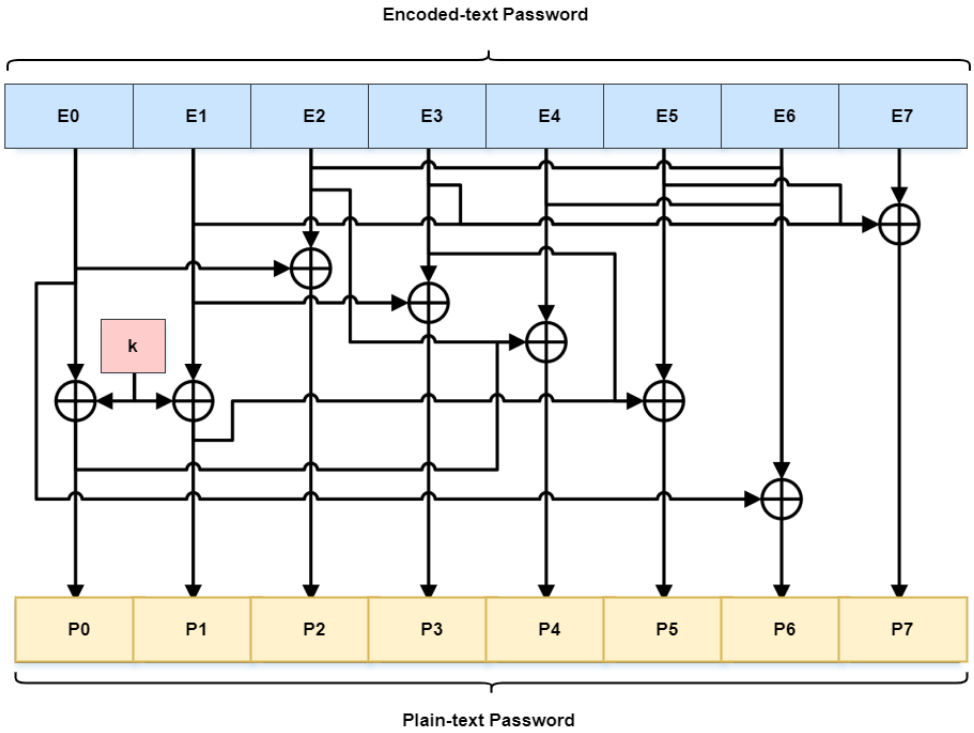


Figure 3.3: Decoding Scheme to retrieve the plain-text password: the input of this scheme is the 8-bytes encoded password (*E0 .. E7*), while the output is the 8 characters password (*P0 .. P7*).

an upload request packet and send it to the target PLC to obtain the user-program and, subsequently, the content of the memory blocks through network packets (e.g., the content of SDB0, representing the encoded password). After obtaining the content of the SDB0 block, the attacker can apply our decoding method (see Figure 3.3) to decrypt the password and retrieve the plaintext version.

- Compromising the Read/Write Protection Level

For this protection level, the user configures the PLC with read/write protection, meaning that any download/upload request sent from the user to the PLC should undergo authentication. Unlike the previous protection level (write protection), the attacker here cannot read the contents of the memory blocks without providing the correct password. Thus, they need to listen to the network and wait until an authentication request message is sent from the TIA Portal to the target PLC. Once they record an authentication packet, they can read the content of the SDB0 block, i.e., the encrypted password, and decrypt it using our decoding method, similar to compromising the write protection level scenario.

3.1.6 Replay Attacks to Subvert the Authentication

Our investigations revealed that even if the attacker cannot retrieve the plaintext password, they are still able to compromise the authentication of the PLC. This is possible by replaying a set of packets related to a specific valid command without authorization. Such a replay attack comprises three steps: 1) sending a forged command to the target device (e.g., start, stop, clear block, update configuration, etc.), 2) recording the network streams, and 3) pushing the crafted packets back to the PLC at a later time.

In a legitimate scenario, such as when a new password is being downloaded to the PLC, the content of the SDB0 block is updated. Therefore, the download process first reads the content of the SDB0 to check whether this block already has a value, indicating that the PLC is already configured with a password, or if it is clean, meaning the PLC is configured with no protection level. There are three scenario cases as follows:

a) The PLC is currently configured with no protection level, and the user wants to set a new password.

b) The PLC is already configured with a password, and the user wants to update the old password with a new one.

c) The PLC is already configured with a password, and the user wants to remove it, i.e., configuring the PLC with no protection level.

- Setting a New Password

For the first case, the attacker captures specific network packets responsible for configuring the PLC with a new password and subsequently sends these packets as replies to the target in a new communication session. The attacker's objective is to configure the victim PLC with a new password. Consequently, the attacker selectively replies only to the packets responsible for updating the content of SDB0, excluding all other packets. This approach ensures that the hardware and software settings of the PLC remain unchanged. Consequently, the authentication attack remains undetected by the engineering station. Following a successful configuration, the legitimate user is unable to access the infected PLC as authentication is no longer granted.

- Updating an Old Password

Here, the attacker's objective is to replace an existing password with a new one. This implies that the PLC is already configured with a password, and SDB0 already possesses a value. Employing a procedure similar to the first case (explained in Section 3.1.6) did not succeed in updating the old password. Our investigations revealed that we cannot overwrite/update

the content of sub-block SDB0 by simply replaying pre-recorded packets, as in the first case scenario. The PLC will broadcast Finish (FIN) packets to close the connection whenever an attempt is made to overwrite SDB0. To overcome this challenge, we first need to clear the content of block SDB0 and then reply with packets that write the SDB0 block with a new password.

However, the TIA Portal does not provide a legitimate command to clear the SDB0 block. For this reason, we searched for packets responsible for deleting different memory blocks and customized them to meet our requirements, i.e., to clear the SDB0 block. Using the aforementioned two-step approach, we successfully managed to change the password of the PLC via an unauthorized workstation without even knowing the old password.

- Removing an Existing Password

Here, an attacker aims to clear the block SDB0 from its configuration. To achieve this, we only employed the first stage of the aforementioned attack scenario, i.e., updating an old password. After successfully clearing SDB0, we managed to remove the password, and the PLC no longer requires authentication from the user.

3.1.7 Attacks Evaluation

All our experiments were conducted on a Siemens S7-300 CPU (*6ES7315-2EH14-0AB0*[3]) with firmware version V3.2.8 and TIA Portal version V16. Figure 3.4 illustrates the attacker model utilized throughout this thesis, where the attacker is positioned on the field site, i.e., the control network, and can establish communication with the PLCs operating in the same network. All the threats introduced in this thesis are executed using the *MITRE ATTCK* tactics and techniques [126]. These techniques are pertinent to ICSs and are elaborated as follows [12]:

a) T1040 - Network sniffing. Adversaries are capable of sniffing the exchanged packets at specific time e.g., authentication process, download program, upload program, etc.

b) T1555 - Credentials from password stores. Adversaries are capable of sending a forged command to read the content of memory blocks that store the password.

c) T1110.002 - Password Cracking. Adverseness leverage vulnerabilities to crack the password once they are capable of sniffing the network traffic.

d) T1098 - Unauthorized Password Reset. Adversaries send a crafted request to reset the password.

[3]https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7315-2EH14-0AB0

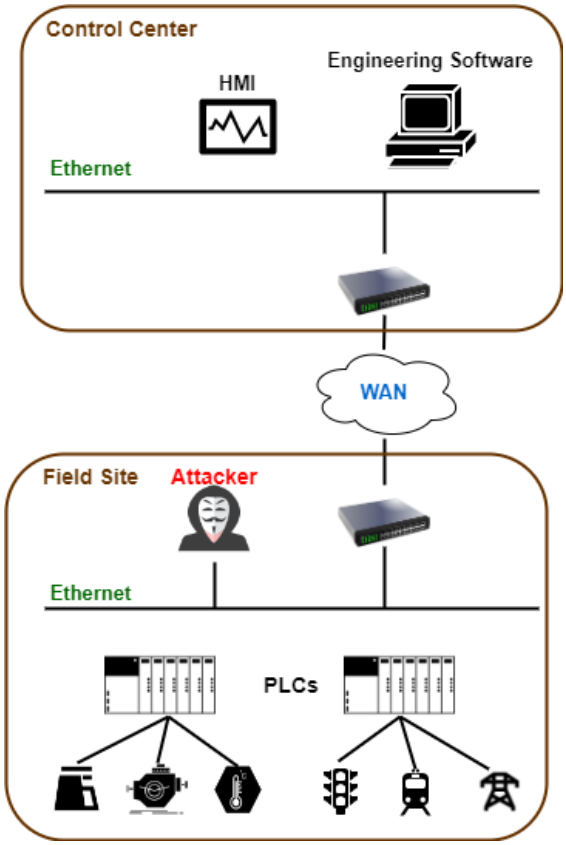


Figure 3.4: Attacker Model [12]

e) T1562 - Impair Defenses. Adversaries are capable of impairing preventive security solution e.g., authentication.

f) T0830 - Man in the Middle (MitM). Adversaries are placed between the EWS and a PLC by poisoning the Address Resolution Protocol (ARP) cache of both connected parties to maliciously modify commands and data.

g) T1565.002 - Transmitted data manipulation. Man-in-the-Middle (MitM) adversaries can sniff and manipulate critical data exchanged between two connected machines.

h) T1499 - Endpoint Denial of service. MitM adversaries alter the memory block containing the password and prevent legitimate users from accessing the PLC.

i) T0806 - Brute Force I/O. Adversaries repetitively or successively change I/O values to perform an action.

j) T0813 - Denial of Control. Adversaries cause a denial of control to temporarily prevent operators and engineers from interacting with process controls.

k) T0816 - Device Restart/Shutdown. Adversaries forcibly restart or shutdown a device in an ICS environment to disrupt and negatively impact physical processes.

l) T0877 - I/O Image. Adversaries seek to capture process values related to the inputs and outputs of a PLC.

m) T0831 - Manipulation of Control. Adversaries can change set point values, tags, or other parameters that will manipulate physical process control.

n) T0832 - Manipulation of View. Adversaries attempt to manipulate the information reported back to operators or controllers.

o) T0821 - Modify Controller Tasking. Adversaries modify the tasking of a PLC to allow for the execution of their own programs. This can allow to manipulate the execution flow and behavior of a PLC.

p) T0889 - Modify Program. Adversaries alter or add a program on a PLC to affect how it interacts with the physical process, peripheral devices and other hosts on the network.

q) T0843 - Program Download. Attackers perform a program download to transfer a user program to a PLC.

r) T0845 - Program Upload. Adversaries attempt to upload a program from a PLC to gather information about an industrial process.

- Retrieve the plain-text Password

We reverse engineered the substitution table using our decoding scheme depicted in figure 3.3 and presented all our results in table 3.1.

- Replay Attacks

To execute the replay attacks demonstrated in Section 3.1.6, we crafted a custom Python script utilizing the Scapy library. We opted for Scapy[4] over other libraries because of its robust functionality and diverse packet manipulation capabilities, such as packet sniffing, replaying in the network, network scanning, trace-routing, etc. Additionally, its implementation in Python makes it easily integrable into any Python script. Algorithm 3.2 depicts the core of our python script we used to perform different replay attack scenarios, taking into account that for each scenario we use specific packets stored already as a pcap file.

Our python program has been tested using two types of commands (upload and download) which require a password authentication. In the following we only illustrate the first scenario i.e., setting a new password, as the other scenarios are conducted in a similar way. Figure 3.5 shows the capture packet for setting a new password.

We selected "weatt@ck" as an 8-character ASCII password to upload to the target PLC. As illustrated, the last 8 bytes of the packet represent the encoded-text password, denoted as

[4]https://pypi.org/project/scapy/

Table 3.1: Decoding results of using our decoding scheme - Cha.: Character, Enc.: Encoded.

Cha.	Enc.	Cha.	Enc.	Cha.	Enc.	Cha.	Enc.	Cha.	Enc.	Cha.	Enc.
!	11	3	3	J	7a	Z	6a	p	40	:	a
@	70	4	4	K	7b	a	51	q	41	“	12
#	13	5	5	L	7c	b	52	r	42	<	c
$	14	6	6	M	7d	c	53	s	43	>	e
%	15	7	7	N	7e	d	54	t	44	?	f
^	6e	8	8	O	7f	e	55	u	45	{	4b
&	16	9	9	P	60	f	56	v	46	}	4d
*	1a	A	71	Q	61	g	57	w	47	|	4f
(	18	B	72	R	62	h	58	x	48	=	d
)	19	C	73	S	63	i	59	y	49	-	1d
–	6f	D	74	T	64	j	5a	z	4a	_	4c
+	1b	E	75	U	65	K	5b	[	6b	/	1f
~	4e	F	76	V	66	1	5c	]	6d	\	6c
0	0	G	77	W	67	m	5d	’	50		
1	1	H	78	X	68	n	5e	,	1c		
2	2	I	79	Y	69	o	5f	.	1e		

Algorithm 3.2: Replay Attack based on Per-recorded Packets

```
Function: Replay (pcapfile, eth_interface, src_ip, src_prt)
recv_seq_num = 0
syn = true
for pkt in rdpcap(pcapfile) do
    ip = pkt[ip]
    tcp = pkt[tcp]
    delete ip.checksum
    ip.src = src_ip
    ip.port = src_port
    if (tcp.flags == ack) or (tcp.flags == rstack) then
        tcp.ack = recv_seq_num + 1
        if syn or tcp.flags == rstack then
            sendp(pkt, iface=eth_interface)
            syn = false
            continue
        end
    end
    recv = scrp1(pkt, iface= eth_interface)
    recv_seq_num = rcv[tcp].seq
end
End Function
```

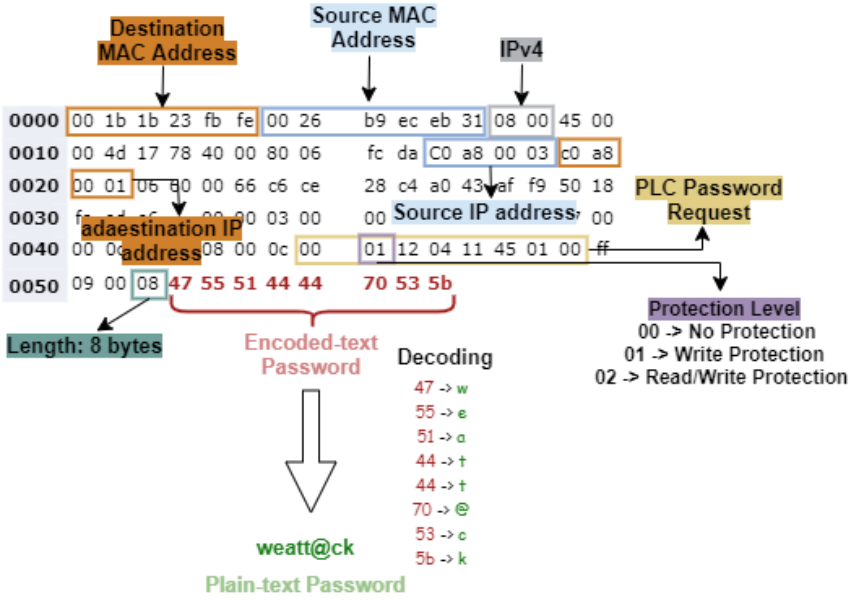


Figure 3.5: Password Authentication during an Upload Command in S7-300 PLCs

"0x4755514444470535b". It is important to note that if the PLC is not password-protected, these particular bytes would be filled with random values.

In the 8-byte PLC password request, the attacker can set the desired protection level by placing *0x00*, *0x01*, or *0x02* for no protection, write protection, and read/write protection levels, respectively. In this attack scenario, we set the write protection level by placing *0x01* in the second byte of the PLC password request, as shown in Figure 3.5. Consequently, an attacker can set this byte to *0x00*, making the PLC require no further authentication from the user to access the control logic program. Similarly, we tested other replay attack scenarios, all of which were successful.

In conclusion, an attacker located on the same network as the target PLC, without proper engineering software, can set a new password, update, or even remove an existing one in the PLC without knowing the plaintext password. This clearly prevents legitimate users from accessing the PLC, causing a denial of access condition. However, a factory reset or clearing the memory is required to regain access to the compromised PLC. Furthermore, we evaluated the processing times that authentication attacks consume to bypass the password of the tested PLC. To achieve this, we downloaded 8 different user programs to the test device, each configured with an individual password. To generate different passwords, we used a random password generator[5]. The 8 generated passwords include numbers, upper case, and lower case characters.

[5]https://www.passwordsgenerators.net/

All our experimental results are presented in Table 3.2. As evident from the table, retrieving the plaintext password consumes more processing time than the replay attacks. Our results indicate that an attacker can reveal the plaintext password within a maximum of 13.88 seconds, while requiring less time to remove the password from the PLC or update an old password with a new one, taking 2.37 and 3.94 seconds (s), respectively.

Table 3.2: The experimental results of authentication attacks

Plain-text Password	Encoded Password	Retrieve the Plain-text	Time (s)	Remove the Password	Time (s)	Update the Password	Time (s)
ro3dRRq	0x425f035311626241	100%	11.12	100%	2.27	100%	3.51
Sm23rP2	0x635d020342601102	100%	12.39	100%	2.32	100%	3.26
134fbhe	0x0103045652587055	100%	13.07	100%	2.29	100%	3.70
n[7'4kL"	0x5e66071c045b7c12	100%	13.87	100%	2.13	100%	3.34
ijknmMN^	0x595a5b5e5d7d7e6e	100%	10.09	100%	**2.37**	100%	3.25
TSRh&$pa	0x6463625816144051	100%	**13.88**	100%	2.18	100%	3.42
WeAy("LJ	0x6755714918127c7a	100%	10.76	100%	2.11	100%	3.13
CC29CRCk	0x737302097362735b	100%	09.82	100%	2.26	100%	**3.94**

3.1.8 Discussion

Our experimental results have identified two major security issues in the authentication protocol widely used in PLCs. Firstly, PLCs require a single password for user authentication without any additional identifications, such as user ID, email address, phone number, etc. From our perspective, this introduces a significant security vulnerability, as the PLC deems the communicating user authorized solely by knowledge of the correct password. In typical IT domains, the authentication process is more robust, typically involving both a user ID and password. We recommend addressing this issue in the future by designing PLCs to incorporate at least a pair of identifications rather than relying on only one.

The second issue pertains to the fact that the authentication protocol used by PLCs supports only the authentication of the connected client, for example, the TIA Portal based on the user password. This means that the PLC, functioning as a server, does not authenticate the engineering software or other client applications. Consequently, attackers could create a counterfeit PLC to mimic control center services, as demonstrated in the following section.

3.1.9 Summary

Our experiments revealed that the encryption scheme employed in PLCs is notably weak, with a limited key space of only 256 characters. Additionally, the authentication protocol lacks the utilization of a cryptographic nonce to encrypt passwords during transit. Another

noteworthy vulnerability in the authentication process is the absence of key rotation between sessions for PLCs. This significantly simplifies the task for potential attackers.

In light of these findings, it can be concluded that non-cryptographically protected PLCs are susceptible to password-based vulnerabilities and lack the implementation of security measures to prevent the use of old communications in potential replay attacks. Consequently, an adversary could either decrypt a transmitted password over the network using a substitution table containing plain-text/encoded-text pairs or intercept an authorized password set/update request through a conventional replay attack.

3.2 Stealthy Control Logic Injection Attack

Contents of this section are as follows:

One of the biggest challenges that an attacker encounters after injecting the target PLC is to hide the malicious infection from the EWS. Existing approaches such as Stuxnet [17] and *DEO* attacks [86] were designed to interrupt and manipulate packets transmitted from the EWS to the PLC and vice versa to conceal the malicious injection. Stuxnet [2] was designed to listen to network traffic, identify packets containing control logic, and then replace the "s7otbxdx.dll," which handles communication between the SIMATIC STEP 7 software and the PLC. To hide the infection from the ICS operator, Stuxnet used a particular masking approach. The *DEO* attacks [86] were designed similarly to Stuxnet, but the authors utilized the Ettercap[6] tool and performed a MitM attack based on ARP poisoning to interrupt network traffic. Meanwhile, they removed any changes in the user program before pushing the unchanged program (original) again to the EWS. Both approaches could successfully hide infections, but they have a few limitations. The Stuxnet attack required very advanced and sophisticated skills to overcome the security measures implemented in the target plant. On the other hand, *DEO* attacks were based on the Ettercap software. The problem with this software is that it does not provide a comprehensive filter that covers complex programs. Additionally, *DEO* attacks were not fully stealthy due to the duplicate messages generated by forwarding packets between connected stations. Consequently, the ICS operator could easily reveal the ongoing attack by using regular network sniffer and analyzer tools such as Wireshark. Another disadvantage that *DEO* attacks had is that the packet filter used was developed based on a single simple control logic program. Meaning that the concealment approach introduced in [86] was valid only for a specific control logic. Therefore, if the EWS updates the user program running in the victim PLC, their designed filter will fail to find the exact location in the packets where the attacker needs to make malicious changes.

In this section, we present a new attack scenario that conceals the malicious modification by employing a fake PLC approach, overcoming the limitations of existing attacks. The

[6] https://www.ettercap-project.org/

goal of our fake PLC is to achieve as stealthy an infection as possible by excluding all duplicate packets that might cause perturbations in the network. All the experiments are conducted on PLCs from the S7-300 family and their related S7Comm protocol. We chose these specific PLC models for two main reasons. First, Siemens is the leading vendor of industrial automation components [93, 94], and their SIMATIC devices hold approximately 30-40% of the industry market. Secondly, S7-300 PLCs are widely common and employed in millions of real-world industrial systems. Therefore, their security ultimately reflects the security of millions of ICSs worldwide.

3.2.1 Fake PLC Approach

The fake PLC we introduced in this work works as a server. It intercepts the transmitted messages between the victim PLC and the EWS, precisely the requests coming from the engineering software, and then responds to each request with a valid response based on pre-recorded packet captures from former communication sessions. This means that our fake PLC operates quite similarly to a real PLC from the EWS point of view. PLCs use specific communication protocols which are proprietary for the vendors, e.g., S7 protocol for SIMATIC S7 PLCs, PCCC for RSLogix PLCs, etc. As those protocols are undocumented, the information about any protocol is limited. Therefore, any further research requires reverse engineering.

Figure 3.6 depicts our systematic approach to building the fake PLC. Our approach encounters two challenges: First, when the EWS sends a request packet over the network, the fake PLC should reply with a valid response the EWS expects to receive. To this end, the fake PLC has to be able to search, as quickly as possible, for the corresponding response in old network sessions recorded prior to the attack. Secondly, the fake PLC needs to adjust certain dynamic fields in each response message appropriately. The values of such fields vary from one session to another. For this reason, we built our fake PLC based on our insight that the regular network traffic (except upload and download messages that contain control logic) between an EWS and a PLC consists of different packets that can be saved in a Communication Template (CT). Once the fake PLC identifies a request coming from the EWS, it scans the CT to select a valid response and forwards it to the EWS after adjusting the dynamic fields appropriately. The CT is a black box approach, which means that we do not need to have a full understanding of the request-response packet contents. Consequently, there is no need to reverse engineer the PLC communication protocol, which saves effort and time. Due to the fact that some dynamic fields vary from one session to another (e.g., sequence number, session ID, etc.), the fake PLC has to understand how to generate a valid

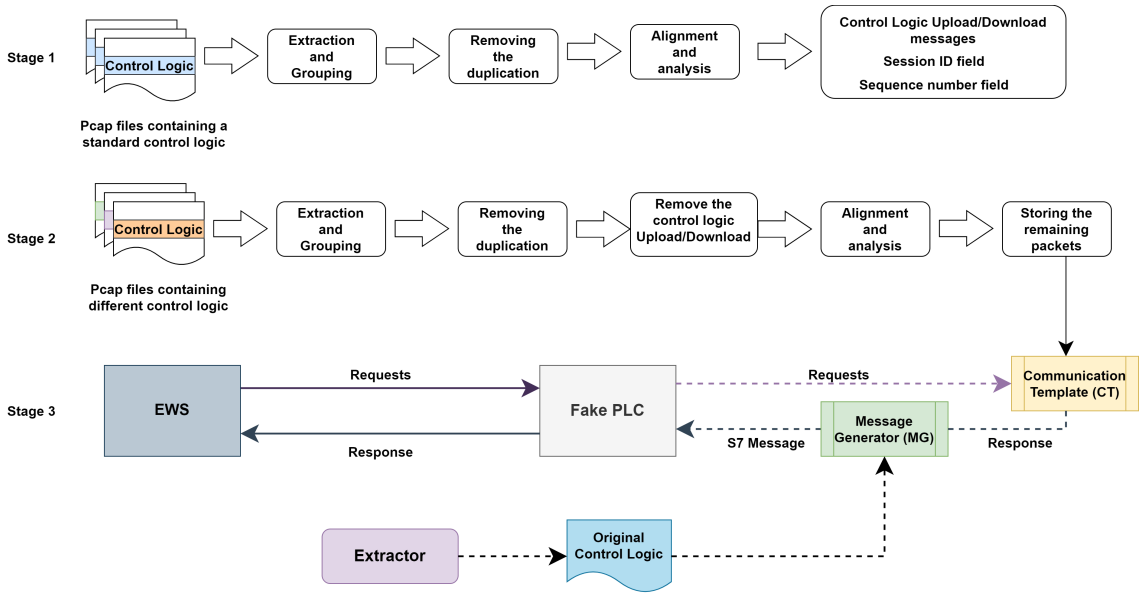


Figure 3.6: Systematic Approach to build a fake PLC

response message. To this end, we should have a basic understanding of the PLC protocol, such as the S7Comm protocol, referred to as Message Generator (MG). In the following, we elaborate on the three stages that our fake PLC consists of.

Stage /1/

In the first step, we collect different network streams using the same user program. After that, we handle each capture as follows: all S7 packets are first identified and then classified in pairs of request-response packets. Since the function ID of both S7Comm request and its corresponding response is always the same, the S7 request-response packets can be linked to each other with the help of the function ID. For example, for read and write functions, the S7 messages representing both requests and responses have the same function ID, which is in a network traffic *0x04* and *0x05*, respectively. After a successful pairing, all duplicated pairs are eliminated. Such duplicated messages can be in the network if a status-check process is running periodically between the EWS and PLC. The aforementioned process is reiterated for each capture. At the end of this stage, all messages in the captures are aligned, differences are analyzed, and the resultant messages are stored in an initial template to be used in the second stage.

The size of the aligned messages might be huge. This is due to the fact that some packets contain a user program whose size relies on the complexity and the length of the program that the PLC runs. Therefore, to ensure as quick a lookup process as possible, we need to remove the packets that contain the control logic and keep only the request-response pairs that do not contain any control logic, as explained in the next stage.

Stage /2/

This stage is quite similar to the former stage but differs in that different user programs are used here to collect network captures. Like the first stage, each capture is processed by identifying and classifying the packets in pairs of request-response. Then, we eliminate all duplicated pairs in the same capture. Afterwards, all packets containing user programs are removed. Since the control logic program is only transferred from the PLC to the EWS and vice versa over download and upload messages, we can easily recognize such messages and exclude them from the network captures saved in the CT. This is done by checking the S7 message header, precisely the function ID that both upload and download messages use, which are in network traffic *0x01E* and *0x1D* respectively. At the end of this stage, we align the remaining packets and then analyze the differences between them, ignoring the session ID field. Finally, all the remaining pairs (request-response) are stored in the CT.

Stage /3/

When the attack is running, the fake PLC looks up the CT to find the corresponding response to each request it receives. After that, it formats an appropriate S7 response by adjusting the dynamic fields, e.g., adding the current session ID, the next sequence number, etc. This is possible by using our MG, which is designed based on the Scapy library. It takes the corresponding response for the received request and adjusts the dynamic fields to match the current session. Moreover, it encapsulates the final response packet as an S7 message and sends it to the fake PLC. The latter receives the packet and transfers it to the connected EWS, e.g., the TIA Portal.

In case the fake PLC does not find the request packet in the CT, it is then a control logic upload/download request. For such a scenario, the fake PLC, precisely the MG, responds by generating a valid response packet containing the original (uninfected) user-program sniffed by the *Extractor* (illustrated later in section 3.2.2). Thus, if the legitimate user requests the program from the infected PLC, his request is dropped from reaching its destination, i.e., the infected PLC, and delivered to our fake PLC instead, which replies by sending the original control logic program.

It is worth mentioning that the fake PLC executes the first two stages offline, i.e., during the preparation of the attack, while the third stage is executed online, i.e., during an ongoing attack.

3.2.2 Attack Approach

A high-level overview of our full attack chain is shown in figure 3.7.

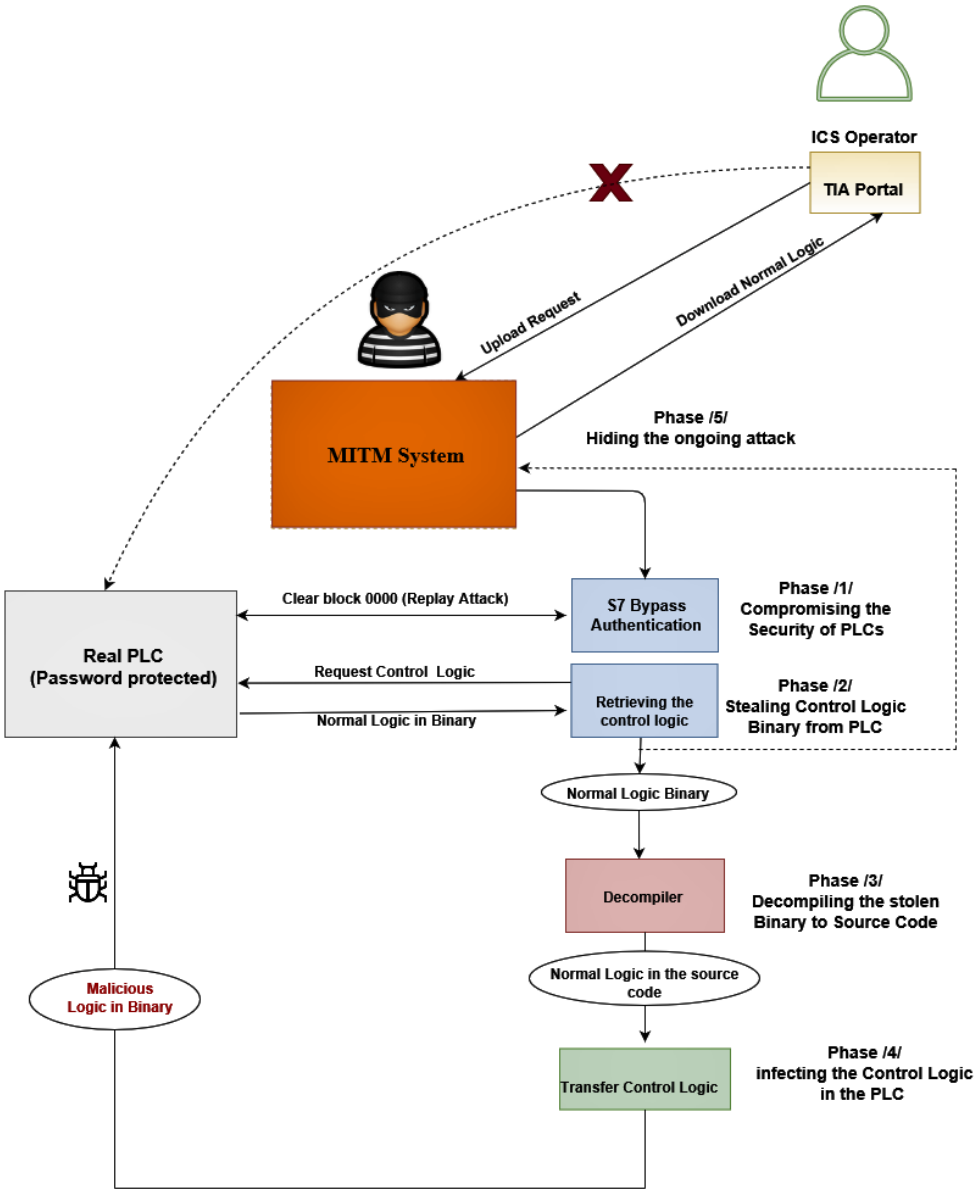


Figure 3.7: A high-level overview of the stealthy control logic injection attack [2]

It consists of five main phases as follows:
- Compromising the security of the PLC (already explained in Section 3.1).
- Stealing the control logic program from the target PLC.
- Decompiling the user-program from the low-level to high-level format.
- Injecting the target PLC with a malicious program.
- Concealing the infection from the operator.

After we gain access to the control network of the target ICS i.e., we can receive and send packets from/to the PLC and EWS, we launch our attack. Please note that compromising the ICS' network is out of the scope of our work, and can be done via typical IT attacks such as inserting an infected USB, exploiting a vulnerable web server, etc. Furthermore, the first step is not covered in this section as it is elaborated in detail in Section 3.1.

Stealing the Original Control Logic

After gaining access to the PLC, the adversary aims to steal the control logic over the network. As part of the full attack chain, we introduce our *Extractor* tool that retrieves the user program from the target PLC. It uses the communication protocol (in our case, the S7Comm protocol) that the victim PLC supports to connect and then requests the control logic from the PLC upon an upload request.

The control logic program is divided into blocks. The Organization Block 1 (OB1) contains the main program that the PLC reads and executes. But for a typical control logic program, it might more likely have Functions (FCs), Function Blocks (FBs), and Data Blocks (DBs) where values and parameters used in Timers, Counters, PID controllers, etc., are stored. In most cases, the size of the *Bytecode* significantly varies depending on the complexity of the program that the PLC runs. Therefore, the *Extractor* was designed to first parse the different blocks that the control logic consists of, i.e., how many blocks the user program comprises, what the kind and the size of these blocks are, etc. Afterwards, it requests the control logic program from the target PLC and identifies the exact byte shift where the control logic is placed. Finally, it extracts the *Bytecode* from the upload/download message and sends it to the *Decompiler* for further processing.

Decompiling the Bytecode to its Source Code

The user-program retrieved from the prior step is represented in a low-level programming language (*Bytecode*). Therefore, it is required for an attacker to decompile the *Bytecode* to its respective source code to understand the physical process that the target PLC controls.

Siemens provides engineers with four programming languages to develop control logic programs (LD, FBD, Statement List (STL), and Structured Control Language (SCL)). In this attack scenario, we are only interested in decompiling the *Bytecode* to STL source code as it is the most common programming language in the industrial community compared to the others.

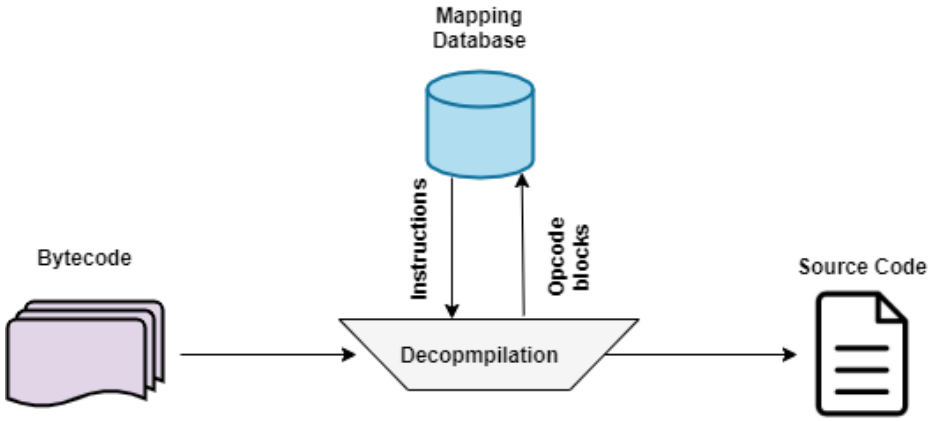


Figure 3.8: Mapping the user-program from Bytecode to Source Code format [2]

Figure 3.8 describes the decompilation process of our *Decompiler* used in this step. It takes the *Bytecode* block (the output of our *Extractor*) as input and utilizes a *Mapping Database* for the decompilation. Please note that the Data Block (DB) might contain additional configuration data or control parameters related to certain instructions. For example, to configure a *Timer*, the operator uses specific parameters such as pre-set, time base, the type of timer (e.g., Timer On (TON), Timer Off (TOF), Pulse Timer (PT), etc.) that are stored in a DB block. Our *Decompiler*, designed in this work, maps each hex-byte set (referred to as *Opcode* for the rest of this chapter) in the *Bytecode* to the *Mapping Database* to obtain the corresponding source code instruction and stores the resultant instructions in an *Instruction List* for future use.

Infecting the Control Logic based on Rules Approach

The next step involves infecting (altering) the control logic. To achieve this, our attack employs four rules to modify various control logic programs. This rule-based modification is conducted on the decompiled source codes stored in the *Instruction List*. We use a rule-based approach to automatically modify the control logic.

To identify the physical process, our attack reads the *Instruction List* to leverage hints that help the attacker speculate on the user-program, as well as critical variables, parameters, and set-points that the PLC controls. These hints play a significant role in generating infection

rules. For example, an attacker can search for particular instructions in the *Instruction List* to deduce the meaning of outputs, set-points, timer parameters, etc. They can also look for data that discloses critical variables in the target physical process, similar to the Stuxnet attack [17], where adversaries accessed data identifying different frequency values used in the control logic program of the nuclear plant.

In this work, we generated four appropriate infection rules to infect the control logic running on the victim device as follows.

- **Rule 1 – Substituting inputs or outputs with memory bits:** Infecting the PLC can be achieved by changing input/output bits that the PLC reads and writes with memory bits. In this rule, we aim to manipulate digital values in a user program, eventually leading to the introduction of new malicious outputs/inputs to the control logic.

- **Rule 2 – Manipulating control flow determinate:** Given that a variable X is defined as a control flow determinant, and X impacts the decision made at a conditional branch, an attacker can manipulate the control flow as follows: first, they seek control flow determinants in the user program, and then attempt to alter variables involved in the control flow with random values.

- **Rule 3 - Manipulating set-points:** Set-points are the values that the ICS operator desires to achieve during a control process (e.g., a certain temperature in a tank, motor frequency of a gas centrifuge, etc.). They play a critical role in any closed control loop used in user programs. Set-points are represented by the operator through two ways: either by instructions or by particular function blocks. Thus, an attacker can maliciously alter set-point values by first seeking instructions or function blocks that contain set points and then finding configured variables related to set points, allowing them to alter these values with random ones.

- **Rule 4 – Substituting operators in control equations:** Manipulating control equations can impact the execution of the user program, potentially leading to catastrophic consequences. An example of this rule is changing an operator with its opposite. If an attacker replaces the division operator (/) with the multiplication operator (*), the control flow would act completely opposite to what the user intended.

- Compiling the Source Code to the *Bytecode*

Once the original user program is successfully modified, an attacker needs to recompile it to its low-level format, for example, *Bytecode/Binary*, before pushing it to the target PLC. For this purpose, we design a *Compiler* that utilizes the same *Mapping Database* that our *Decompiler* uses for decompilation but in a reverse way. This means that the *Compiler* searches for the *Opcode* block for each STL instruction in the *Mapping Database* and substitutes the former with the latter.

- Patching the PLC with the infected control logic

After successful compilation, the modified user program *Bytecode* is then patched in the victim PLC using a crafted S7Comm *Download* request. Please note that to make the PLC accept updating its control logic, it must be in *STOP* mode before sending the payload. This holds true for other PLC vendors as well. Therefore, an adversary needs to switch the PLC *OFF* before the injection. However, this can be done by a typical replay attack that sends a specific packet containing a legitimate *STOP* command to turn the PLC to *STOP* mode [16]. After updating the control logic in the target PLC, the adversary turns it *ON* by sending a specific packet containing a legitimate *START* command. Furthermore, in case the PLC demands an integrity check for the user program, the integrity bytes for the modified program must be correctly generated. However, S7-300 PLCs use the S7Comm protocol which lacks integrity check mechanisms. This means that these PLCs execute all instructions and commands regardless of whether or not they are delivered from a legitimate EWS.

Concealing the Malicious Control Logic

To overcome the challenge of concealing an ongoing injection attack, we present a new method based on replacing the infected device with a fake PLC (introduced in Section 3.2.1) that the ICS operator communicates with. Technically, the engineering software provides the user with the ability to compare the online code running in the remote PLC with the offline one stored in the project files on their machine (e.g., Personal Computer (PC)). This process reveals any infection in the control logic. The main goal of our new approach is to prevent the operator from uploading the actual infected code from the remote PLC. This is achieved by redirecting their requests to our fake PLC, which sends the uninfected (original) version that we want the user to see. This approach hides our ongoing injection and accomplishes a fully stealthy attack.

3.2.3 Attack Implementation

Experimental Setup

We executed our full attack chain approach, as presented in Section 3.2.2, in a real-world industrial setting using a Siemens S7-300 PLC running the latest firmware version V3.2.8 and TIA Portal software version V16. Figure 3.9 illustrates the experimental setup we used to test our attack approach.

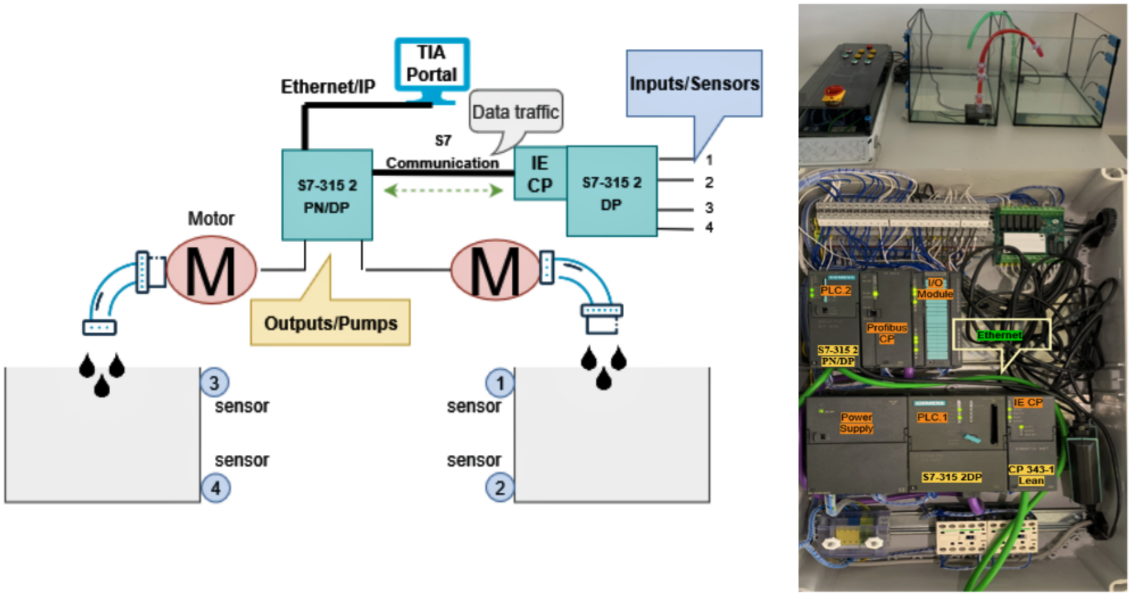


Figure 3.9: Experimental setup based on S7-300 PLCs [1–3, 5]

As can be seen from Figure 3.9, the example application consists of two aquariums filled with water that is pumped from one to the other until a certain level is reached, and then the pumping direction is inverted. The PLC is connected to the engineering station, i.e., the TIA Portal software, via an Ethernet cable, and exchanges data over the network to control the water level in each aquarium. The control process in this setup runs cyclically as follows: The PLC reads the input signals coming from sensors 1, 2, 3, and 4. The two upper sensors (Num. 1, 3) installed on both aquariums report to the device when the aquariums are full of water, while the two lower sensors (Num. 2, 4) report to the device when the aquariums are empty. Then, the PLC powers the pumps ON/OFF depending on the sensors' readings received.

The experimental setup consists of the following components: legitimate user, attacker machine, PLC, CP, sensors, and pumps. In the following, we provide a brief description for each component.

- **Legitimate User:** It is a device connected to the PLC using the TIA Portal software. Here, we use version 16 and Windows 10[7] as an operating system.

- **Attacker Machine:** It is a device that sneakily connects to the system without appropriate credentials. In our experiments, the attacker uses *LINUX* Ubuntu 18.04.1 *LTS*[8] as an operating system.

- **PLC S7-300:** The PLC used in this experimental setup is from the 300 family, precisely the S7 315-2 PN/DP CPU (*6ES7315-2EH14-0AB0*).

- **Four Capacitive Proximity Sensors:** In our testbed, there are four sensors from *SICK*, type *CQ35-25NPP-KC16*[9], with a sensing range of 25 millimeter (mm) and electrical wiring DC 4-wire.

- **Two Pumps:** Here, two DC-Runner 1.1 from *AquaMedic* with transparent pump housing, 0-10 Volts (V) connection for external control, maximum pumping output of 1200 liter/hour (l/h), and pumping height of 1.5 meter (m).

Stealing the Original Control Logic

The initial step that our *Extractor* aims at is discovering the local switched network to gain an overview of targetable PLCs in the target system's network. In order to collect sufficient data on the available devices, our *Extractor* utilizes a so-called Profinet-Input Output (PN-IO) Scanner based on Profinet Discovery and basic Configuration Protocol (PN-DCP). Technically, this scanner sends a Discovery and basic Configuration Protocol (DCP) identify-request over the network and waits for responses from all discovered devices, such as PLCs, Communication Processors (CPs), etc. It then sniffs the responses for a predefined time interval of 5 s. Finally, it saves all the results of the sniffing in a Python dictionary for future use. The output of executing our PN-IO scanner is shown in Figure 3.10 and can be broken down into the following steps: 1) get local IP, port, and subnet, 2) calculate IP addresses of the subnet, 3) set up a TCP connection, 4) send a DCP identify-request, 5) receive a DCP response, 6) save all responses in a Python response file, and 7) stop scanning and disconnect the TCP connection.

After a successful scan, all IP and MAC addresses of targets are known. In the next step, our *Extractor* gains an insight into the target PLC. This means that it starts collecting data

[7] https://www.microsoft.com/de-de/software-download/windows10
[8] https://old-releases.ubuntu.com/releases/18.04.1/?_ga=2.74542550.550971829.1670487059-1438380202.1670487059
[9] https://www.sick.com/de/de/kapazitive-naeherungssensoren/cq/cq35-25npp-kc1/p/p244267

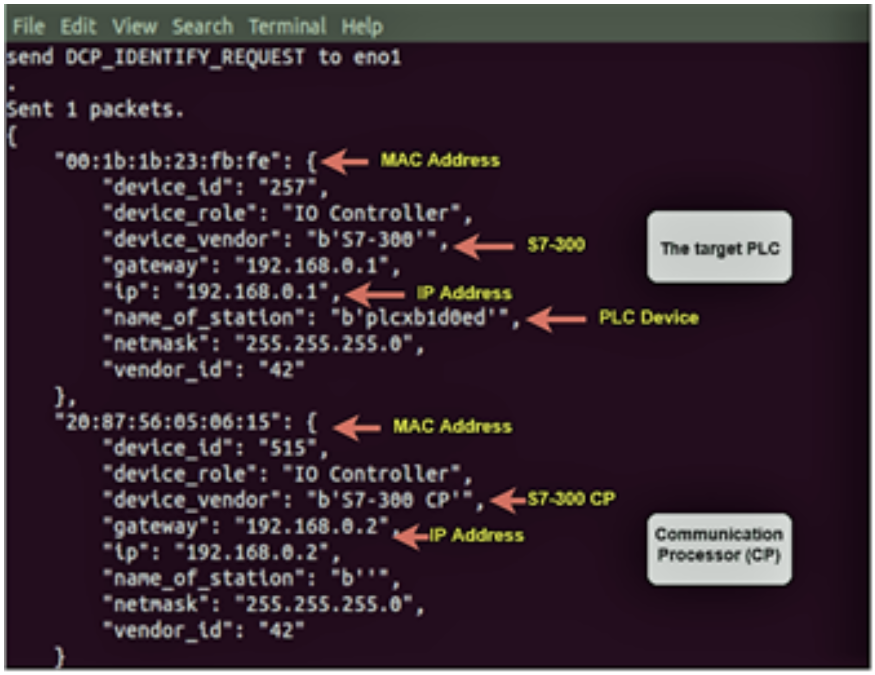


Figure 3.10: The output of executing PN-IO scanner [1]

on the PLC's control logic using specific commands supported by the Python-Snap7 library to determine which software blocks the PLC has, the complexity of the control logic program, the size of each block, etc. This helps our attack obtain a sufficiently deep knowledge of the target PLC from a software point of view, and then uses the information collected to retrieve the control logic program from the PLC accurately.

Figure 3.11 presents the output of executing a deeper scan on the tested PLC. As can be seen, the control logic program that our target PLC runs consists of an OB1, 15 System Function Blocks (SFBs), 77 System Function Calls (SFCs), a Data Block 1 (DB1), and 14 System Data Blocks (SDBs). Furthermore, since OB1 runs the main program and there are no other FBs or FCs, we can conclude that the entire control logic program exists in OB1 and has a *Bytecode* size of 130 Kilo Byte (KB), while the loaded memory size, i.e., the packet size, is 264 KB.

The *Extractor* takes the collected information and begins monitoring the network traffic between the TIA Portal software and the remote PLC using Wireshark software. Since each S7 message is identified by a unique protocol ID (*0x32*), the *Extractor* recognizes S7Comm packets transmitted over the network by checking each packet header, specifically the byte that represents the protocol ID. The protocol header size is fixed for all S7 packets, thus we can easily determine the location of the byte representing the job function of the captured

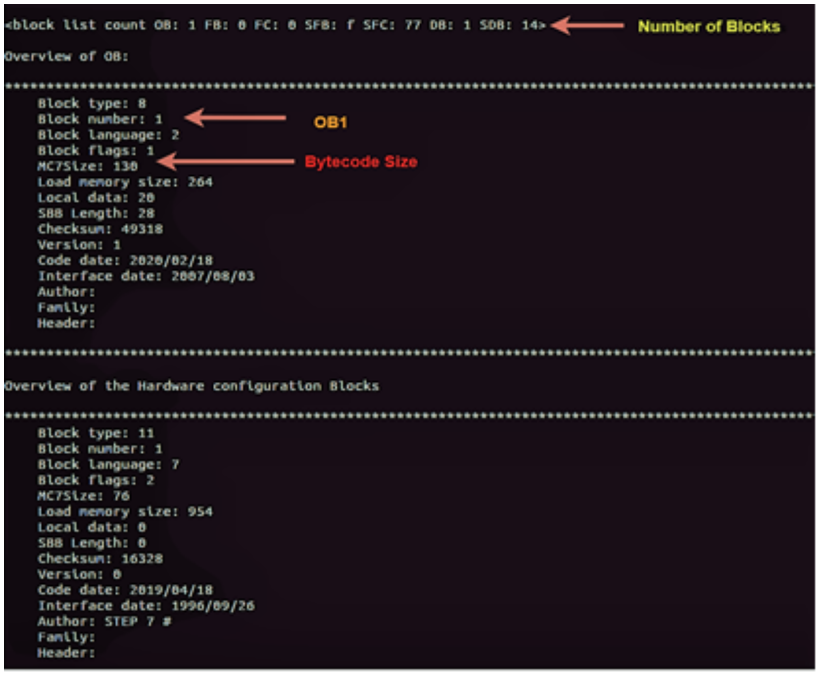


Figure 3.11: The output of executing a deeper scanner [1]

packet. This allows us to identify the specific operation for which each packet is sent. Our analysis showed that byte 13 contains the function code, which is unique for each operation. However, table 3.3 lists pairs of the function codes and their corresponding operations.

Table 3.3: Pairs of S7 function codes to their corresponding operations

Function Code	Operation
0x00	CPU Services
0xF0	Communication Setup
0x04	Read Variable
0x05	Write Variable
0x1A	Download Request
0x1B	Download Block
0x1C	Download End
0x1E	Upload Request
0x1D	Upload Start
0x1F	Upload End
0x28	PLC Control
0x29	PLC Stop

As the control logic program is transferred only through upload or download operations, we can identify upload and download requests by reading their function codes, i.e., *0x1A*

and *0x1E* for download and upload, respectively. Once an upload or download request is recognized, the *Extractor* records all the subsequently transferred messages with the same function code. Figure 3.12 shows a snippet of Python code that searches for the protocol ID *0x32* in the packet header and uses this position as the starting point of an S7Comm packet. Afterward, it calculates byte 13 to determine the packet type and saves only upload and download packets in a Pcap file.

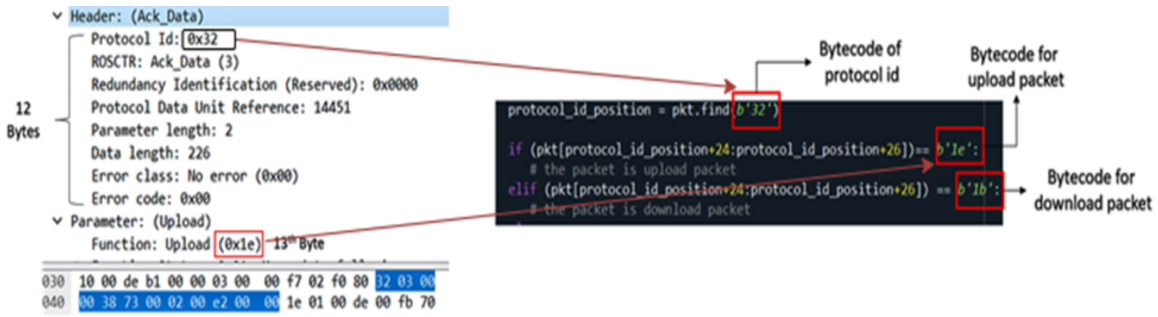


Figure 3.12: Identify an S7 Request Functionality [5]

After the PLC receives an upload or download request from the TIA Portal, it responds by sending either its code to the TIA Portal for an upload request or an acknowledgment packet, informing the TIA Portal that it is ready to receive the code for a download request. The TIA Portal then starts downloading the control logic into the remote PLC. Therefore, in the next step, our *Extractor* records the complete response stream for any identified upload/download request, which eventually contains the control logic *Bytecode* that the PLC runs.

Based on the collected information from the previous step (see Figure 3.11), our *Extractor* records and saves only the S7 packet with a size of 264 bytes, dropping all other packets. In the next step, the *Extractor* determines the *Opcodes* representing the raw data for the S7 packet saved and then filters this data to retrieve only the *Opcodes* representing the *Bytecode* program that the PLC executes.

Our findings showed that the control logic *Bytecode* is always located between two *Opcode* keys in the extracted raw data: *Start_Key* (*0x0082*) and *End_Key* (*0x6500*), as shown in Figure 3.13. By extracting only the bytes located between these two *Opcode* keys, we successfully obtained the control logic *Bytecode*, which serves as the input for the next step, i.e., the *Decompiler*.

Our *Extractor* ensures the accuracy of the extraction process by comparing the size of the extracted bytes to the size of the *Bytecode* obtained in the previous step, i.e., 130 Bytes, involving the *End_Key*. If both sizes match, the extraction process is successful, and the bytecode is sent to the *Decompiler*.

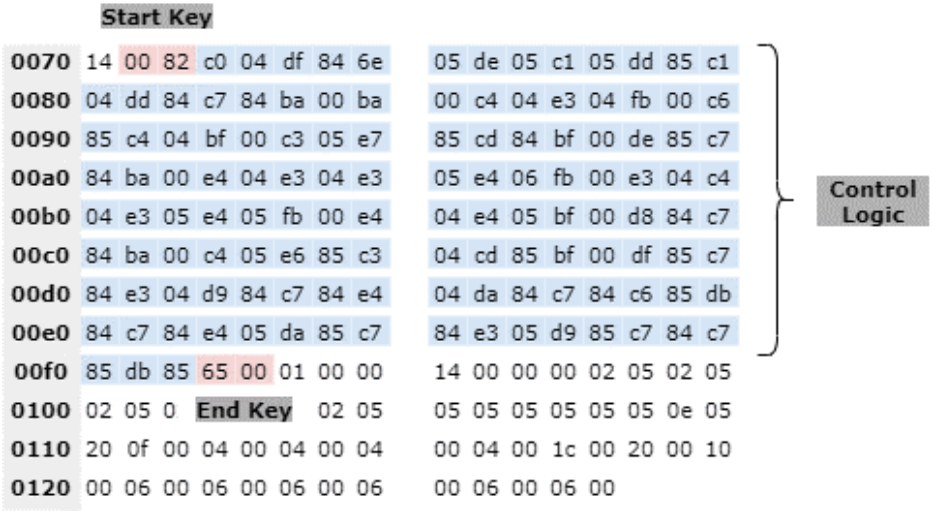


Figure 3.13: The location of the control logic in an S7 packet

Decompilation Process

In the following, we test the decompilation process on our given example application in Figure 3.9. This converts the control logic from its low-level format, i.e., *Bytecode*, to its high-level format, i.e., STL.

- STL *Decompiler*

We implement the decompiler that we developed in this work to take the extracted *Bytecode* as input and decompile it into STL source code. The *Decompiler* consists of two components: 1) The *Mapping Database* that maps each *Opcode* to its corresponding STL instruction. Our *Mapping Database* is created based on analyzing 108 different control logic programs and includes 3802 pairs of "Opcode - STL" for different instructions such as inputs, outputs, memory bits, relative branches, function block, operational block, etc. 2) The *Mapper Program* that uses the *Mapping Database* to match each *Opcode* from the control logic program with its STL instruction from the *Mapping Database* and then stores the resultant STL instruction in the *Instruction List*.

- ***Mapping Database:***

  In order to map the *Opcodes* with their corresponding STL instructions, we applied an offline division method to extract all the instructions of the 108 programs used in this work to create a sufficient database. We followed these steps: we opened the TIA Portal software and programmed our target PLC with a certain code consisting of 10 repetitions of the same instruction. Here, we used the instruction *NOP 0*, which

has no effect on the program. After that, we downloaded this code to our PLC and recorded the packets containing the bytecode, which is the representation of 10 *NOP 0* instructions (see Figure 3.14).

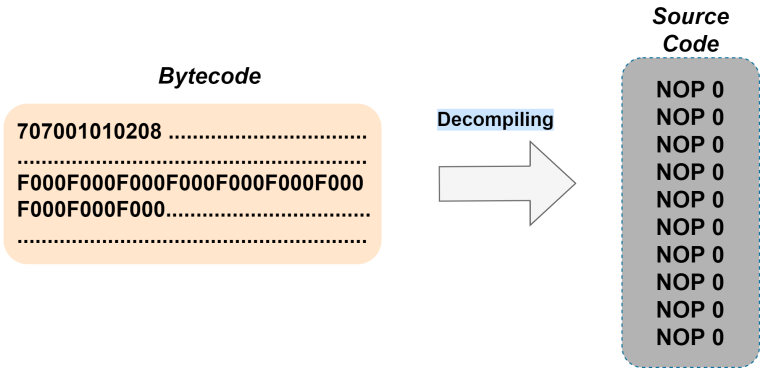


Figure 3.14: Decompiling 10 NOP 0 instructions from bytecode to its STL code [2]

We could identify that each *NOP 0* instruction is represented as *0xF000* in the *Bytecode* format. Afterwards, we opened each program in the TIA Portal software and inserted *NOP 0* before and after each instruction, and then downloaded the new program to the PLC. We recorded the packets that contain this new bytecode, and identified each *Opcode* representing each instruction as figure 3.15 shows. After extracting all the instructions with the corresponding *Opcodes*, we created a *Mapping Database* that contains 3802 pairs of *Opcodes* to their corresponding STL instructions, and used this *Mapping Database* to convert the original *Bytecode* to its STL source code online.

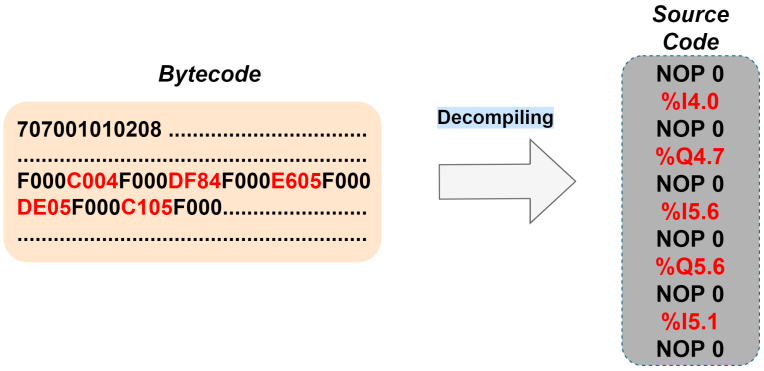


Figure 3.15: An example of the offline division method used to create the *Mapping Database* [2]

- ***Mapper Program:***

  In the first step of the decompilation process, the *Mapper Program* separates the user program into individual networks using the following two rules:

A network can be divided wherever there is an input instruction coming right after an output instruction. A network can be divided wherever there is an *END_BLK* instruction, which always comes right after using a control block instruction (e.g., timer, counter, controller, etc.).

After the program is successfully divided, and the resulting small networks are stored in an array, the *Mapper Program* starts converting the *Opcodes* representing each network to their corresponding STL instructions by looking up the *Mapping Database* and replacing each *Opcode* with an STL instruction. For instance, the *Opcodes 0x23* and *0x22*, in *Bytecode* format, represent the *AND* and *OR* bitwise operations in STL respectively. For a better understanding of the mapping process applied to an individual network, we take "0xC0042302DF84" as an example. The two instructions *0xC004* and *0xDF84* are in series connection (*0x23*), and the *0x02* indicates that the next instruction has two bytes.

In most cases, the user program includes additional operational blocks. For instance, when the user includes equations in their program, it may use memory variables (e.g., integers, floats, words, etc.). These memory variables are utilized as control flow determinants in the control process. In the case of using operational blocks, the *Bytecode* program uses a particular *Opcode* (*0x3DB4*) as a jump command to a sub-routine. Therefore, our *Mapper Program* is not sufficient for correct decompilation. To overcome this challenge, we added an additional program (*Operational Equations Decompiler*) exclusively for decompiling operational equations. This means that when our *Mapper Program* finds an operational equation, it calls the *Operational Equations Decompiler* program for processing. Figure 3.16 depicts a simple equation example. As seen after the byte *0x3DB4*, we jump to a sub-routine, and the equation starts with *0x2A1B00*, defining the type of operation. In our given example, the operation is division. Then, a byte *0x3232* defines the type of operands, followed by a byte *0x0792* representing the parameter on the left-hand side of the equation. Finally, two bytes *0x0288* and *0x0388* represent the two operands in the equation.

In case the equation is more complicated, i.e., many operands are linked to each other in the equation, then the *Bytecode* of the complex equation is divided into multiple simple equations with Temporary Variable (TMV). Figure 3.17 shows an example of a complex equation. As can be seen, we break down the entire equation into simpler equations, using a TMV that forms the final complex equation by connecting the simple equations to each other.

In this given example, the variable operand is represented as *0x32*, while the constant

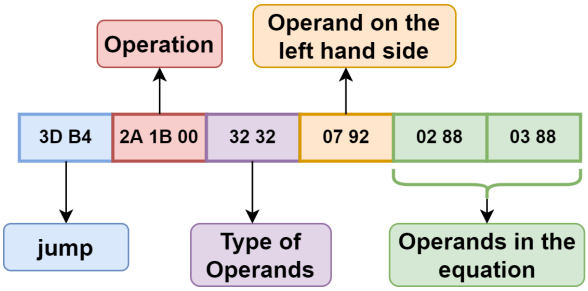


Figure 3.16: Decompiling a simple operation network (%A1 := %A2 / %A3) into STL format

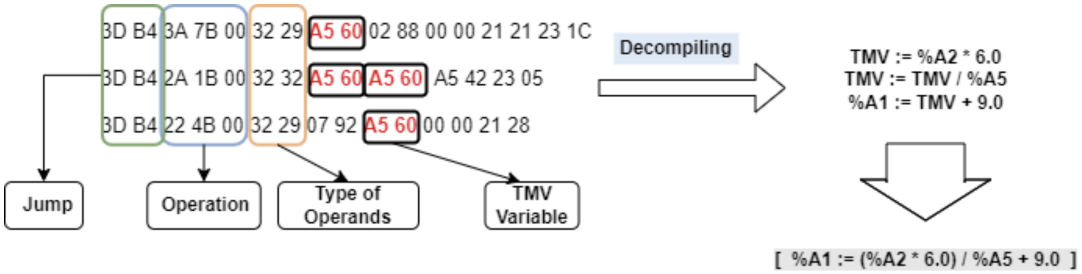


Figure 3.17: Decompiling a complex operational equation (%A1 := (%A2 * 6.0) / (%A5 + 9.0)) into STL format

operand is represented as *0x29*. Now, once our *Mapper Program* finds an operational block in the *Bytecode*, it sends the *Opcodes* of the operational block to the *Operational Equations Decompiler* program. The latter checks the *Opcodes* and identifies the complexity of the equation. If the equation is simple, then it can be decompiled directly, as shown in the prior example (see Figure 3.16). But if the equation is complicated, the program initializes a TMV. In the first step, it decompiles each individual equation with the help of the TMV and then replaces the TMV with its corresponding equation to format the final complex equation. The challenge of this process arises when the TMV is reassigned, as seen in Figure 3.17. Therefore, to replace the TMV with its corresponding equation step-by-step correctly, a top-down approach is used in this program.

Control Logic Modification

To infect the user program in its high-level format, we manipulate the resultant *Instruction List* of the decompilation process, i.e., the outcome of our *Decompiler*. This *Instruction List* contains all the STL instructions used in the original *Bytecode*. The manipulation can be achieved by modifying, removing, or inserting new instructions in the *Instruction List* to cause abnormal behavior in the target PLC. The reason behind manipulating the

Instruction List is that the *Opcode* representing a certain STL instruction might have a different length compared to other *Opcodes*. Meaning that the *Opcode* length varies depending on the instruction type. Our investigations showed that the longer *Opcode* length increases the likelihood that smaller *Opcodes* representing other instructions are included.

- Modification Rules: We conducted two modification scenarios on the control logic program designed to manage our example application (see Figure 3.9), using the rules mentioned earlier in Section 3.2.2. First, we applied *Rule 1* to replace an I/O instruction with a predefined one to control a pump via a memory bit. In the second scenario, *Rule 4* was used to reverse the condition that deactivates a pump when the desired water level is reached. These malicious modifications can result in serious damage to a real-world power plant. For example, consider a scenario where an infected PLC activates a radiator in a nuclear plant when a particular condition is met. If an attacker uses *Rule 4*, for instance, to reverse the operation of the PLC by deactivating the radiator instead, the plant and the surrounding area are all put in serious danger.

Compiling the Source Code to its Bytecode

After successfully modifying the control logic, we need to recompile the infected STL code into *Bytecode* format before pushing it back to the target PLC. To achieve this, we utilize our *Compiler*, which operates similarly to the *Decompiler* but in reverse. This means it reads the resulting output of the modification process and then recompiles each STL instruction to its corresponding *Opcode* using the *Mapping Database*. Figure 3.18 displays the output of our *Compiler*, which is the infected *Bytecode* intended for the PLC to read and process, while Figure 3.19 illustrates the original *Bytecode* that the real PLC runs.

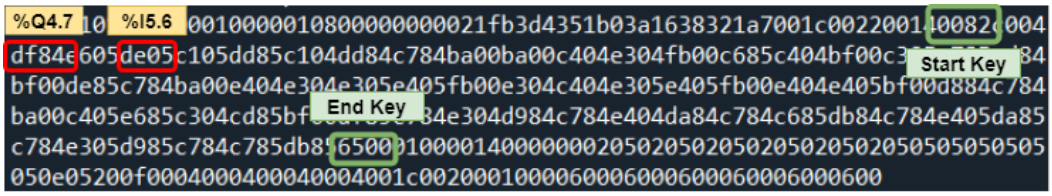


Figure 3.18: The infected control logic program in *Bytecode* format [3]

Transferring the Infected Control Logic to the PLC

In the final step, the attacker already possesses the malicious bytecode, and all that is needed to corrupt the target system is to push the infected control logic back to the PLC. Due to the lack of integrity checks in S7-300 PLCs, such controllers execute commands whether or

Figure 3.19: The original control logic program in *Bytecode* format [3]

not they are delivered from a legitimate user. Therefore, our attack crafts the full S7Comm packet that we want to send to the PLC by placing the malicious *Bytecode* obtained from the *Decompiler* as raw data, and then adding the parameters and the proper S7 packet header.

In this work, we use the same S7 packet that our *Extractor* already identified and replace only the original *Bytecode* located between the *Start* and *End* keys with the malicious one. Afterwards, it injects the crafted packet into the PLC using the well-known Python Snap7 library, precisely the function *download* as shown in listing 3.1. For our example application shown in figure 3.9, we managed successfully to alter the physical process controlled by the infected PLC, causing a water overflow.

Listing 3.1: Python snippet to upload the malicious Bytecode to an S7-300 PLC

```
if (search_upload_block >0):
        print (pkt[response_msg+2:])
        upload_res = (pkt[response_msg+2:])
        sample = b'\x70\x02\x30\x20'
        sample1 = bytearracy(sample)

        bx = b'\x70\x70\x01\x01\x08\x00\x01\x00\x00\x01' \
        b'\x08\x00\x00\x00\x00\x02\x1f\xb3\xd4\x35\x1b\x03\xa1\x63\x83' \
        b'\x21\xa7\x00\x1c\x00\x22\x00\x14\x00\x82\xc0\x04\xdf\x84\xe6' \
        b'\x05\xde\x05\xc1\x05\xdd\x85\xc1\x04\xdd\x84\xc7\x84\xba\x00' \
        b'\xba\x00\xc4\x04\xe3\x04\xfb\x00\xc6\x85\xc4\x04\xbf\x00\xc3' \
        b'\x05\xe7\x85\xcd\x84\xbf\x00\xde\x85\xc7\x84\xba\x00\xe4\x04' \
        b'\xe3\x04\xe3\x05\xe4\x05\xfb\x00\xe3\x04\xc4\x04\xe3\x05\xe4' \
        b'\x05\xfb\x00\xe4\x04\xe4\x05\xbf\x00\xdf\x85\xc7\x84\xba\x00' \
        b'\xc4\x05\xe6\x85\xc3\x04\xcd\x85\xbf\x00\xdf\x85\xc7\x84\xe3' \
        b'\x05\xe7\x85\xcd\x84\xbf\x00\xde\x85\xc7\x84\xba\x00\xe4\x04' \
        b'\xc4\x05\xe6\x85\xc3\x04\xcd\x85\xbf\x00\xdf\x85\xc7\x84\xe3' \
        b'\x04\xd9\x84\xc7\x84\xe4\x04\xda\x84\xc7\x84\xc6\x85\xdb\x84' \
        b'\xc7\x84\xe4\x05\xda\x85\xc7\x84\xe3\x05\xd9\x85\xc7\x84\xc7' \
        b'\x85\xdb\x85\x65\x00\x01\x00\x00\x14\x00\x00\x00\x02\x05\x02' \
        b'\x05\x02\x05\x02\x05\x02\x05\x05\x05\x05\x05\x05\x05' \
```

```
22          b'\x0e\x05\x20\x0f\x00\x04\x00\x04\x00\x04\x00\x04\x00\x1c\x00' \
23          b'\x20\x00\x10\x00\x06\x00\x06\x00\x06\x00\x06\x00\x06\x00\x06' \
24          b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' \
25          b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' \
26          b'\x00\x01\x00\xc0\xa6\x00\x00\x00\x00\x00\x00\x00\x00'
27
28 v = bytearray(bx)
29 plc.download (v, 1)
```

Concealing the Infected Control Logic

To test our fake PLC approach introduced in section 3.2.1, we initially register our counterfeit PLC in the user's TIA Portal and then establish a communication session between both parties. Finally, we transfer the original (uninfected) control logic program to the TIA Portal. In the following, we illustrate each step in detail.

- Registering the Fake PLC on the TIA Portal

The TIA Portal software uses the PN-DCP protocol to discover new devices or configure devices' names, IP addresses, MAC addresses, etc. It requests all accessible devices in the network by broadcasting a certain packet called "identify all," and all S7 PLCs available will reply with a certain response packet called "identify ok." The payload of the response packet sent by each PLC contains all details of the device, e.g., the name, IP address, vendor's name, subnets, etc.

In order to trick the ICS operator, our attack prevents the legitimate TIA Portal software from reaching the remote PLC and connects it to our fake PLC instead. To this end, we first initiate our MitM system, which is based on the well-known ARP poisoning attack, between the TIA Portal and the target PLC. Thus, all packets transmitted between the two stations will first go through the attacker machine and then be redirected depending on the attacker's ARP cache table.

Afterwards, we broadcast an "identify all" over the network (see figure 3.20) and record the response from the PLC, i.e., the "identify ok" packet (see figure 3.21). This packet is then modified by replacing the IP address of the real PLC (in our example application *192.168.0.1*) with the IP address of our fake PLC (*192.168.0.3*), as shown in figure 3.22. Now, whenever the TIA Portal sends a new "identify all" packet over the network in an attempt to connect with the remote PLC, the attacker machine, which listens to the network, will identify the request and drop the packet to prevent the real PLC from responding to this request. It then sends the crafted "identify ok" packet back to the TIA Portal, which registers our fake PLC

as a remote PLC located at *192.168.0.3*.

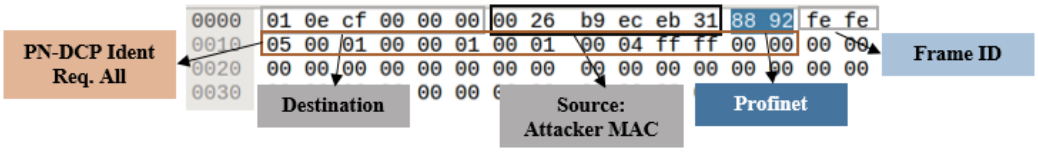


Figure 3.20: Identify all request message from the attacker to the PLC [2]

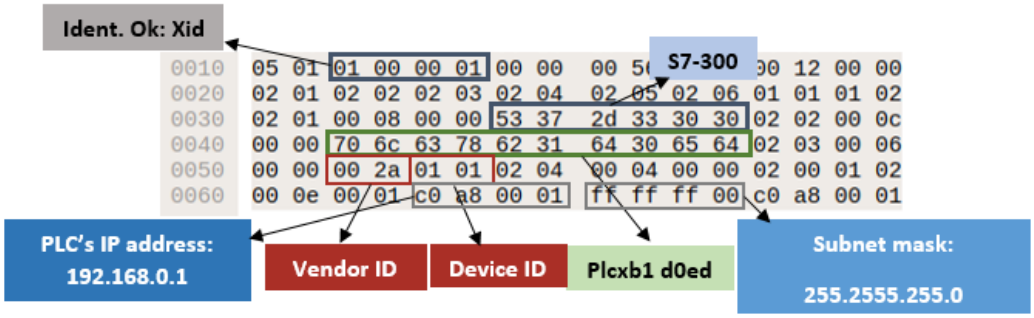


Figure 3.21: Identify ok respond message sent from the PLC to the attacker [2]

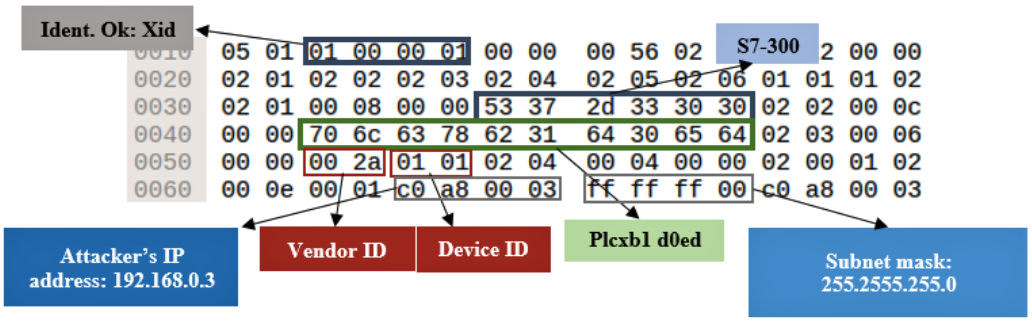
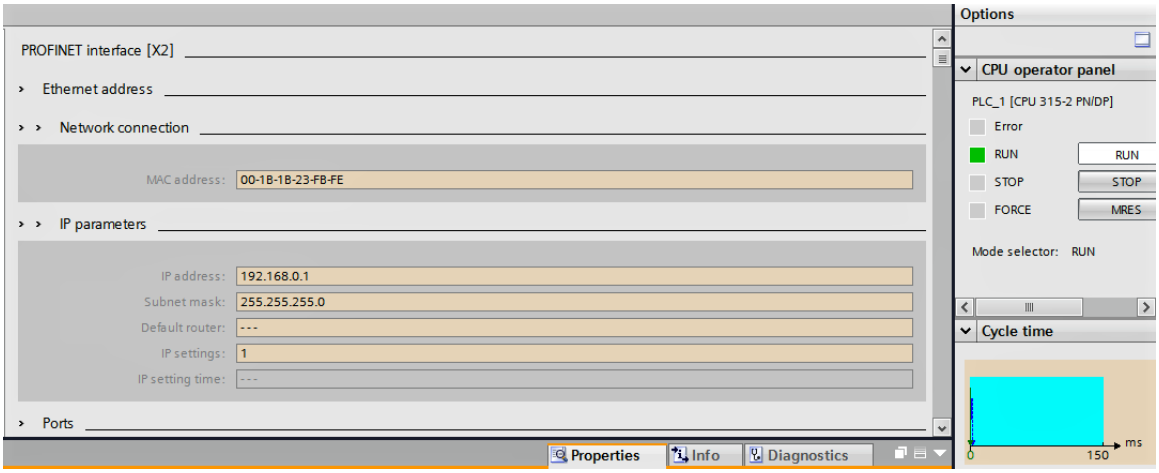


Figure 3.22: Identify ok respond message sent from the attacker to TIA Portal [2]

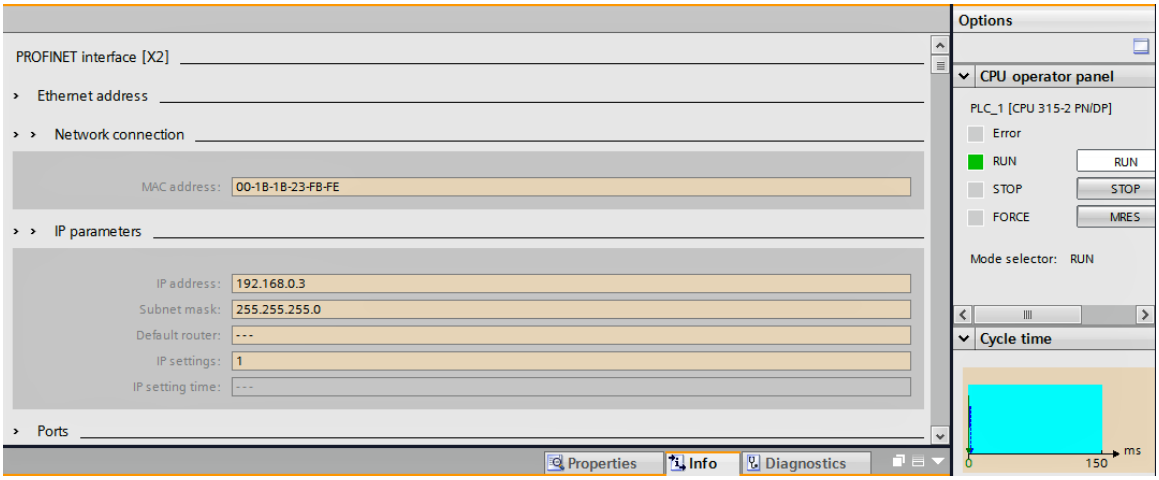
After the TIA Portal identifies an accessible PLC, which is, in fact, our counterfeit PLC, it attempts to establish an online session by initiating a TCP connection. Our MitM system redirects the connection request to the fake PLC, which then establishes TCP communication with the TIA Portal. After a successful connection, the attacker needs to maintain the session between the TIA Portal and the fake PLC. Our investigations revealed that S7-300 PLCs keep the online session with the TIA Portal active by exchanging four specific packets over the time the user is online (S7Comm: POSCTR: [Request], S7Comm POSCTR: [Response],

COTP TPDU (0), and TCP [Acknowledgment (ACK)]). Based on our findings, the fake PLC should continuously send the PLC's response packets, i.e., S7Comm [Response], and TCP [ACK], which are sufficient to keep the connection between the TIA Portal and the fake PLC alive.

Please note that the ICS operator could potentially spot the abnormality in case he checks the differences in the IP addresses between the real PLC and the fake PLC as they are clearly displayed in the TIA Portal see figure 3.23. However, in normal operation this impersonating might go undetected as the IP address is only shown if the operator explicitly checks details of the Profinet interface, which is not required during an ongoing operation.



(a) CPU's online diagnostic before the attack - the IP address of the connected PLC is 192.168.0.1



(b) CPU's online diagnostic after the attack - the IP address of the connected PLC is 192.168.0.3

Figure 3.23: Different IP addresses shown in TIA Portal [2]

- Transferring the Original Logic to the TIA Portal

When the fake PLC receives any request message from the TIA Portal, it looks up the corresponding response messages from the CT and dynamically generates a valid S7 response message using the MG that formalizes the appropriate message by adjusting the dynamic fields in the response message to match the current session. Afterwards, the fake PLC sends the crafted S7 response message to the TIA Portal software. This holds true for all packets except download and upload requests. The TIA Portal requires the control logic program from the fake PLC by sending an upload request. The fake PLC searches for the corresponding response message in the CT and finds no matching response. Therefore, the fake PLC realizes that the TIA Portal requests the control logic. It takes an old upload response from the *Extractor*, adjusts its dynamic fields using the MG, and forwards it to the TIA Portal, which will eventually upload the original program similar to the one stored in the project files at the EWS. Our results show that the fake PLC successfully managed to upload the original code whenever the TIA Portal required the control logic program running in the PLC. This stealthy scenario could cause significant harm in a real-world plant, as the offline and online programs fully match each other, and the engineer will not detect our ongoing injection attack unless they check the IP address of the connected device in the Profinet interface. Figure 3.24 shows that both the offline and online programs are identical during our ongoing injection attack.

3.2.4 Evaluation and Discussion

The Fake PLC Approach

To evaluate our fake PLC approach, we first downloaded our 108 control logic programs one by one to the real PLC and extracted the control logic blocks using the *Extractor*. For each program, we verified whether the fake PLC could successfully upload it to the TIA Portal software, thereby allowing the TIA Portal to decompile it and display its high-level version to the user. Table 3.4 shows the results of our experiments. All the programs used in this thesis have a size range between 150 and 630 KB. Therefore, we classified the programs based on their size into four groups (150 $\sim$ 200, 200 $\sim$ 250, 250 $\sim$ 300, and larger than 300 KB). The "Number of Upload Generation" indicates how many times response messages, upon upload requests, are generated by the fake PLC. Here, the TIA Portal sends a single upload request for each program, and the fake PLC replies by generating valid upload response messages containing the original control logic. Please note that the upload messages are not in the CT. The "Avg. Time" field in the table represents the total operation time that the fake

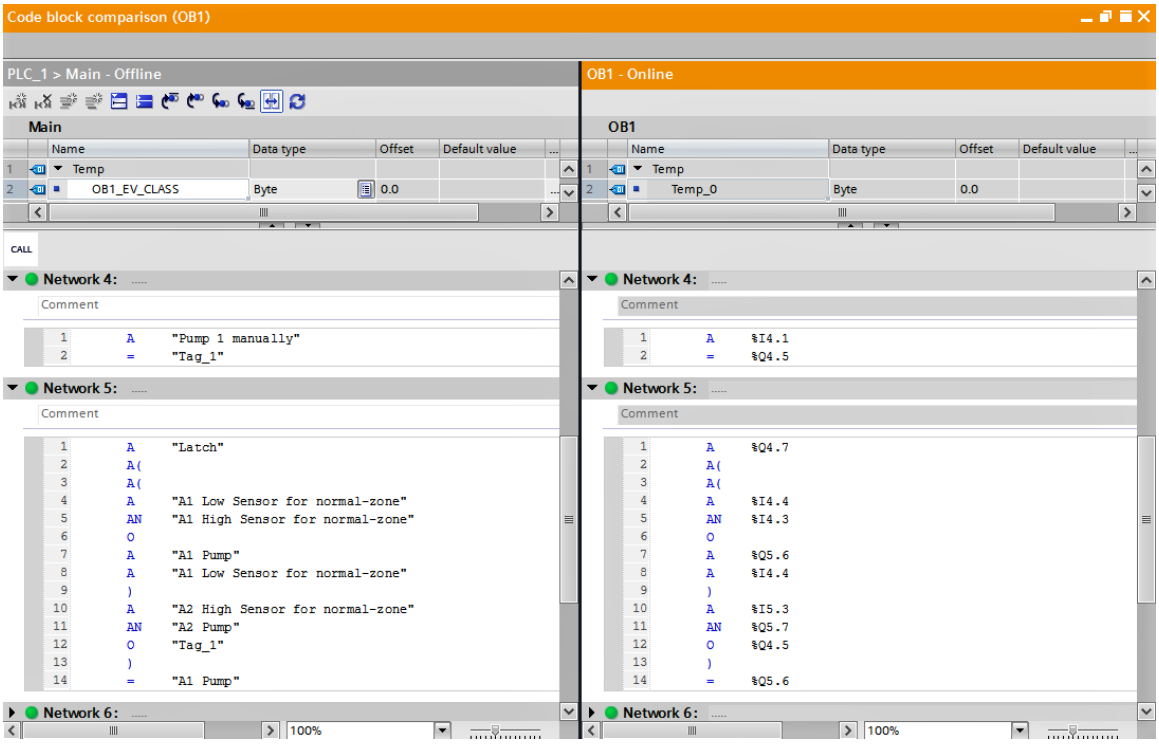


Figure 3.24: Online and Offline control logic comparison shown in TIA Portal [2]

PLC takes from receiving the upload request from the user until uploading and sending the control logic to the TIA Portal.

Table 3.4: The experimental results of the fake PLC

Block size (KB)	Num. of programs	Num. of CT lookup	Num. of Upload Generation	Avg. Time (s)	Success Rate
150 ∼ 200	18	305	18	10.82	100%
200 ∼ 250	27	512	27	12.12	100%
250 ∼ 300	34	716	34	12.09	100%
> 300	29	438	29	12.15	100%
Total	**108**	**1971**	**108**	**11.98**	**100%**

Our experimental evaluation clearly shows that the fake PLC could successfully upload every control logic program to TIA Portal with a 100% success rate. The average time the fake PLC needs to upload a control logic program is approximately 12 s. This is a bit longer than the average time that S7 PLCs need to send their programs to the TIA Portal upon upload requests, which is approximately 5.5 s. However, our investigations showed that Siemens does not provide their TIA Portal software with a specific expiration time for

upload/download sessions with S7 PLCs. This means that an upload/download process lasts as long as the online session between parties (PLC and TIA Portal) is alive. This significant security gap played a big role in the success of implementing our fake PLC and concealing our ongoing injection from the ICS operator.

The Full Attack Chain

To assess the full attack chain, we ran it in autonomous mode. We used six randomly selected real-world control logic programs for the final evaluation (different from the 108 programs utilized to create the *Mapping Database*). Each program was configured with a unique password. We reset our experimental setup (see Figure 3.9) before launching our attack each time. The evaluation results show that our approach successfully performs its five stages: compromising, retrieving, decompiling, infecting, and concealing for each control logic. Table 3.5 summarizes the evaluation results.

Table 3.5: The final evaluation results of our full attack chain - Stage/1/: Compromising, Stage /2/: Stealing, Stage /3/: Decompiling, Stage /4/: Infecting, Stage /5/: Concealing

Control Logic Program	Original Program size (KB)	Infected Program Size (KB)	Stage /1/	Stage /2/	Stage /3/	Stage /4/	Stage /5/	Success Rate
Traffic Light	168	169	100%	100%	100%	100%	100%	100%
Gas Pipeline	193	196	100%	100%	100%	100%	100%	100%
Water Tank	154	154	100%	100%	100%	100%	100%	100%
Bottle Detection	210	217	100%	100%	100%	100%	100%	100%
Car Parking	316	321	100%	100%	100%	100%	100%	100%
Fan Control	171	172	100%	100%	100%	100%	100%	100%

As seen in the table, the infected and original programs exhibit slight differences in terms of size. This is due to our four-rules modification approach, which involves making minor changes to the control logic program, such as replacing an input/output bit with a memory bit, changing a variable, altering a set point, etc. In summary, our modification is lightweight and can be successfully implemented even if the target PLC memory has limited space.

3.2.5 Mitigation Solutions and Security Recommendations

To mitigate the effects of our attack presented in this chapter, we highly suggest that Siemens implement a specific expiration time for its TIA Portal software, terminating any upload/download process that exceeds this designated time. We believe that if this solution were implemented, our fake PLC approach would fail to upload the original control logic to the target TIA Portal, as it requires a longer period compared to the time that S7 PLCs

need to upload their programs to the TIA Portal in response to legitimate upload requests. Additionally, we propose several countermeasures, such as protecting and detecting programs.

The initial step to safeguard our systems from various types of attacks involves enhancing isolation from other networks [121]. This, combined with standard security practices [122] and defense-in-depth security in control systems [123], contributes to a more secure environment. Furthermore, we recommend employing a digital signature not only for the firmware, as most PLC vendors do, but also for the control logic. Moreover, implementing a mechanism to inspect the protocol header, which contains information about the payload type, is suggested as a solution to detect and block any potential unauthorized transfer of control logic.

Siemens provides users with an Multi-Point Interface (MPI) adapter to securely upload and download control logic between the TIA Portal and PLC. The MPI Protocol is currently unsupported by any network sniffers. Considering the cost and convenience benefits of using Ethernet/Profinet connections, the MPI connection still offers better secure communication between the control center and remote devices. This helps prevent attackers from sniffing, consequently enhancing security, as listening and capturing packets transferred over the network form the main basis for attackers to execute most attacks against ICSs.

3.2.6 Summary

In this section, we have presented an advanced, stealthy control logic attack based on employing a fake PLC approach that mimics the behavior of the real PLC. Our fake PLC receives all requests coming from the EWS and responds according to the received requests. For practical implementation, we conducted our full attack chain on real hardware and software used in a small industrial setting. Our results showed that we successfully managed to inject the PLC with malicious code, keeping our infection hidden from the ICS operator. This stealthy attack scenario could cause significant harm in the relevant

CHAPTER 4

Investigating the Security of Cryptographically Protected PLCs

Contents of this chapter are as follows:

Parts of this chapter have already been published in the following papers:

- W. Alsabbagh and P. Langendörfer, "A New Injection Threat on S7-1500 PLCs - Disrupting the Physical Process Offline," in IEEE Open Journal of the Industrial Electronics Society, vol. 3, pp. 146-162, 2022, doi: 10.1109/OJIES.2022.3151528 [6].

- W. Alsabbagh and P. Langendörfer, "No Need to be Online to Attack - Exploiting S7-1500 PLCs by Time-Of-Day Block," 2022 XXVIII International Conference on Information, Communication and Automation Technologies (ICAT), 2022, pp. 1-8, doi: 10.1109/ICAT54566.2022.9811147 [7].

- W. Alsabbagh and P. Langendörfer, "You Are What You Attack: Breaking the Cryptographically Protected S7 Protocol," 2023 IEEE 19th International Conference on Factory Communication Systems (WFCS), Pavia, Italy, 2023, pp. 1-8, doi: 10.1109/WFCS57264.2023.10144251 [11].

Most injection attacks pose two critical challenges. The first is that typical injection attacks are designed to gain access to the target device or its network in very specific circumstances, i.e., when security measures are absent or disabled for a certain reason [31,66,70,74–76,86,112]. For example, security measures may be temporarily disabled during updates, maintenance processes by the ICS operator, removal/replacement/addition of other devices to the network,

etc. The system is at high risk of malicious infection during these critical phases, but it is not operating in its normal state, meaning the physical process is more likely to be temporarily OFF. If adversaries successfully gain access to the target device during these times and conduct their attacks right after, they are less likely to impact the physical process. The second challenge is that after the ICS supervisor completes ongoing maintenance processes, they usually reactivate security measures before re-operating the system. This helps in detecting and preventing any attempt to inject the PLC if the attacker is still connected to the network.

This chapter addresses exactly these two challenges by patching the PLC with a malicious block at the point in time when the attacker successfully accesses the network. This keeps the infection hidden in the PLC's memory, and the attack is launched at a later time at the attacker's will. This ensures that the attack is not performed when the system is not operating normally or being detected by an introduced or reactivated security measure. It is also important to highlight that ICS operators can still disclose any modification in the control logic program by uploading and comparing both programs, the one running on the PLC and the one running in the engineering software [3]. In this approach, we also overcome this challenge by exploiting a vulnerability existing in the newest S7CommPlus protocol to hide the infection from the ICS operator. They will always see the original code that runs on their engineering software, while the PLC runs the attacker's code. However, our attack approach is structured into two main phases.

- Patching the control logic program of a PLC with an interrupt, precisely with a ToD interrupt block using the specific Organization Block 10 (OB10). This is done online, i.e., when the attacker gains access to a target device. During this phase, the patch has no impact on either the physical process or the execution process of the control logic program. In other words, the patch is in idle mode.

- Activating the patch injected in the target at a later date and time. This is done offline, i.e., without the need to be connected to the target device at point zero for the attack.

The major benefit of our attack strategy is that the time running the attack and the point in time when it shall hit the victim can be fully decoupled. For example, if motivated adversaries want to collapse a certain system at a specific date/time, e.g., the day before elections or the day before going to the stock market to harm a country or a company respectively, they have sufficient time to inject their malicious code well in advance and do not need to be successful with the attack just at the right time. Our threat approach is network-based and can be successfully conducted by any attacker with network access to any S7-1500 PLC with firmware V2.9.2 or lower.

To conduct experiments and assess our attack approach for a real-world scenario, a Fischertechnik[1] training industry plant controlled by an S7-1500 PLC was used. Our tested-device selection is based on the fact that Siemens reportedly claimed that its newest PLCs generation is well-secured against diverse threats, and their newly developed S7CommPlus protocol supports improved security measures like an advanced anti-replay mechanism and a sophisticated integrity check. This motivated us to show how the most secure PLCs in Siemens SIMATIC lines can be exploited by external adversaries and how attackers can confuse the physical process even without being connected to the victim devices. This could lead to disastrous damages to the plants employing such compromised devices.

The rest of this chapter is structured as follows: Section 4.1 highlights the latest model of the S7 protocol and its security vulnerabilities. Section 4.2 describes the attack scenario presented in this chapter, and Section 4.3 demonstrates the implementation and evaluation of our attack approach. Following that, possible mitigation solutions are suggested in Section 4.4. Finally, we conclude this chapter with Section 4.5.

4.1 S7Communication Security Issues

4.1.1 S7 Protocols Background

The S7 protocol defines an appropriate format for exchanging S7 messages between devices. Its main communication mode follows a client-server pattern: the HMI or TIA Portal device (client) initiates transactions, and the PLC (server) responds by supplying the requested data to the client or by taking the action requested in the instruction. Siemens provides its PLCs with two different protocol versions: the older SIMATIC S7 PLCs (e.g., S7-300 and S7-400) implement an S7Comm protocol identified by the unique number *0x32*, while the new generation PLCs (e.g., S7-1200 and S7-1500) implement an S7CommPlus protocol identified by the unique number *0x72*. The newer S7CommPlus protocol, the focus of this chapter, has three sub-versions as follows:

- **S7CommPlus V1:** it is used by the older versions of TIA Portal and only in S7-1200 PLCs firmware. This protocol does not include any integrity protection.
- **S7CommPlus V2:** it is used in the TIA Portal up to V12 and in S7-1500 PLCs firmware up to 1.5. This protocol is integrity protected and has security features

[1] https://www.fischertechnikwebshop.com/de-DE/fischertechnik-lernfabrik-4-0-24v-komplettset-mit-sps-s7-1500-560840-de-de

against replay attacks (e.g., Hashed-based Message Authentication Code-Secure Hash Algorithm-256 (HMAC-SHA-256)).

- **S7CommPlus V3:** it is used in the newer versions of TIA Portal from V13 on, and in the newer PLCs S7-1500 firmware e.g., V1.8, 2.0, etc. This protocol requires that both TIA Portal and PLC to support the features of this protocol e.g., support a newer variant of HMAC-SHA-256. Since the S7CommPlus V3 has a more complex integrity protection method, it is considered as the most secure protocol among the other versions.

In this chapter, we focus only on the S7CommPlus V3 protocol, as it is the most complex and sophisticated request-response protocol on one hand. On the other hand, it involves an improved cryptographic mechanism for exchanging an integrity check compared to the mechanism used in the earlier S7CommPlus protocol, i.e., V2.

4.1.2 S7CommPlus V3 Protocol

The S7CommPlus V3 protocol is used only by the newer versions of TIA Portal and S7-1500 PLCs, and supports various operations that are performed by the TIA Portal software as follows:

- Start/Stop the control program currently loaded in the PLC memory.
- Download a control program to the PLC.
- Upload the current control program from the PLC to the TIA Portal.
- Read the value of a control variable.
- Modify the value of a control variable.

The aforementioned operations are first translated by the TIA Portal software into S7CommPlus messages before they are transmitted to the PLC. Then, the PLC acts upon the messages it receives, executes the control operations, and responds back to the TIA Portal accordingly. The messages are transmitted in the context of a session, each with a session ID (chosen by the PLC). Each communication session begins with a four-message handshake used to select the cryptographic attributes of the session, including the protocol version and keys. After the handshake, all messages are integrity-protected using a very complex cryptographic protection mechanism (illustrated in the following subsection (4.1.3)).

S7CommPlus Protocol Structure

S7CommPlus V3 is a request-response protocol. Each message consists of a protocol header, data and trailer as shown in figure 4.1.

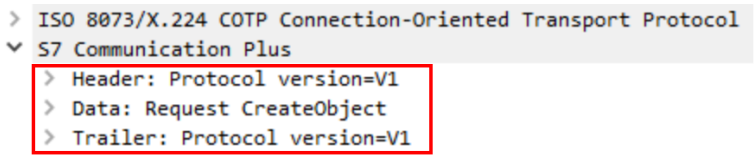


Figure 4.1: The structure of S7CommPlus protocol [11].

The header and trailer have always the same structure including the following components: 1-byte protocol version, 1-byte protocol ID and 2-byte data length as shown in figure 4.2.

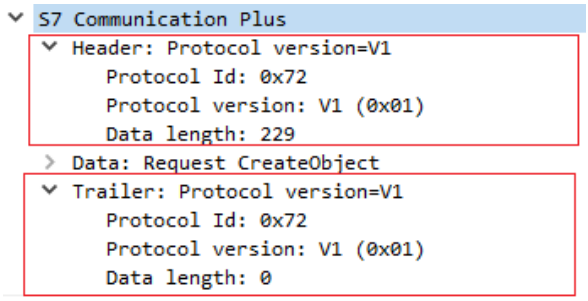


Figure 4.2: S7CommPlus protocol: header and trailer have the same structure [11].

The Protocol Data Unit (PDU) type determines the version of the S7CommPlus protocol, i.e., V1, V2, or V3 for the values *0x01*, *0x02*, or *0x03*, respectively. If the PDU type has the value *0x01* or *0x02*, it indicates the absence of the *Integrity Part* in the *Data* field. Conversely, when the PDU type is *0x03*, an additional *Integrity Part* (refer to the red block in Figure 4.3) is included with the *Data* field, as illustrated in Figure 4.3.

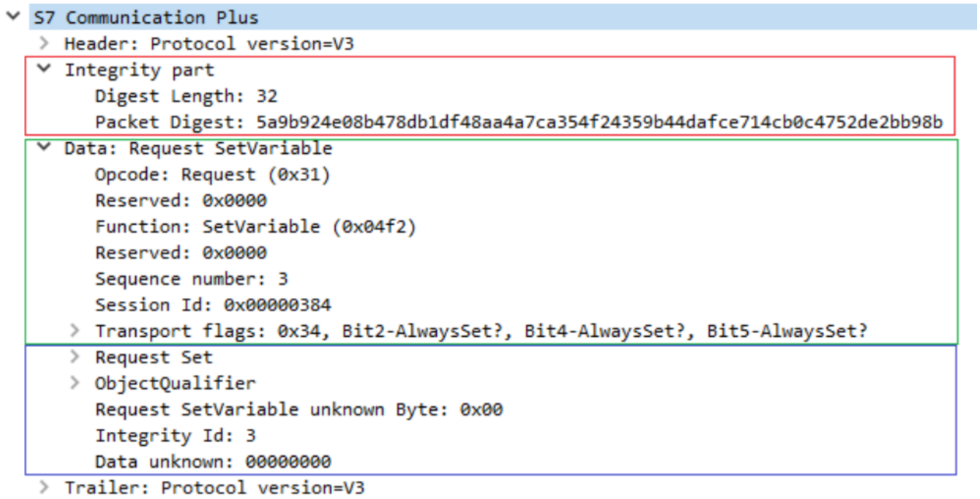


Figure 4.3: S7CommPlusV3 protocol - Data field components [11].

The *Data* block comprises 14 bytes (see the green block in figure 4.3). Starting from the top, we see a 1-byte *Opcode* which identifies the purpose of the S7CommPlus packet, e.g., *0x31* if the packet is a request, *0x32* if the packet is a response, or *0x33* if the packet is a notification. After the *Opcode*, we see a 2-byte field that has a fixed value of *0x0000*. Then, there is a 2-byte field, called *Function*, which determines the functionality of the packet, e.g., *0x04ca* for *CreateObject*, *0x0542* for *SetMultiVariable*, *0x04f2* for *SetVariable*, etc. In the next field, we find again a fixed value of *0x0000* followed by a 2-byte field representing the sequence number of the packet. The *Session* ID length is 4-byte and always has the format of *0x000003xx*. The *xx* in the *Session* ID is a combination of *ObjectID* and *0x80*. Finally, we have the transport flag that is a 1-byte field generated randomly without any use in either encryption or authentication methods.

The structure and content of the *Set* block (see the blue block in figure 4.3) are related to the PDU type and *Opcode*. This block has many diverse types and is quite complex. For more details, please see the S7comm Wireshark dissector plugin project[2].

S7CommPlus Message Structure

A single S7CommPlus message might contain multiple objects, each containing multiple attributes (see Figure 4.4). All objects and attributes have unique class identifiers. However, for a download message, the *CreateObject* request builds a new object in the PLC memory with a unique ID (in our example, *0x04ca*). Then, it creates an object of the class *ProgramCycleOB*. This object contains multiple attributes, each one having values dedicated to a specific purpose. For instance, the *FunctionObject.Code* contains the binary executable code that the PLC runs, i.e., the compiled program in the PLC's machine language (MC7+). The *Block.AdditionalMac* is used as an additional Message Authentication Code (MaC) value in the integrity process, and both *Block.OptimizedInfo* and *Block.BodyDescription* are equivalent to the program written by the ICS operator, which is stored in the PLC and can be later uploaded, upon a request, to a TIA Portal project.

Communication Process

The TIA Portal and PLCs exchange four kinds of packets: S7 *Request*, *Challenge*, *Response*, and *Function* packets, see figure 4.5.

As can be seen, at the beginning of each new communication session, the TIA Portal sends an S7 *Request* to establish a connection with the PLC. After the PLC receives the S7 *Request*, it sends a 20-byte array, namely *Challenge*, that significantly differs from one session

[2]https://sourceforge.net/projects/s7commwireshark/

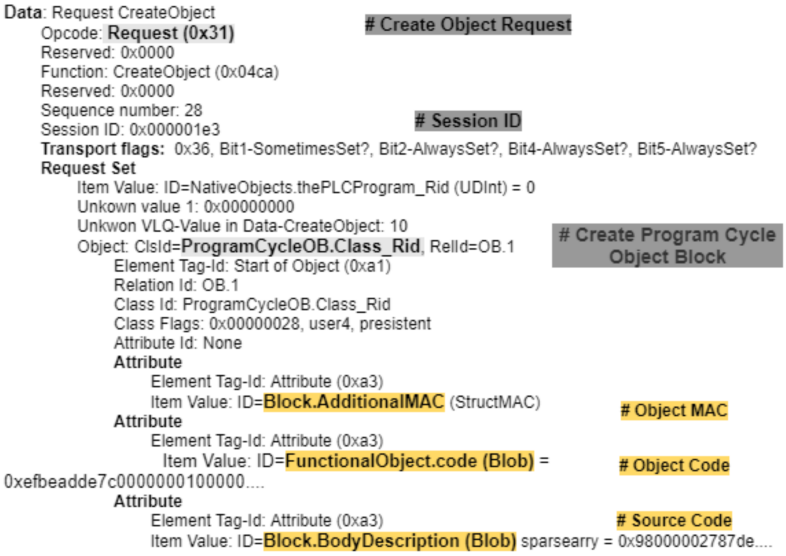


Figure 4.4: S7CommPlus download request - objects and attributes: *Block.AdditionalMAC* represents the *Object MAC*, the *FunctionObject.Code* represents the *Object Code* and *Block.BodyDescription (Blob)* represents the *Source Code* [6].

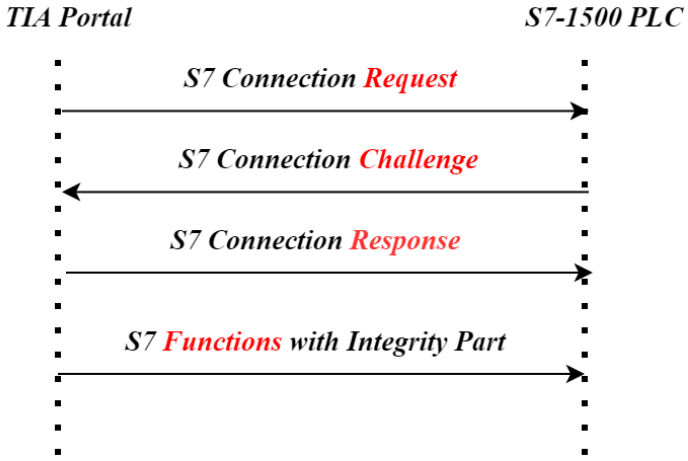


Figure 4.5: S7CommPlus Communication Process [11]

to another. These 20 bytes are generated by a hash or pseudo-random function. After the TIA Portal receives the *Challenge*, it generates a *Response* that contains, among many bytes, three interesting blocks: "block 1" is a 9-byte array, "block 2" is an 8-byte array, and "block 3" is a 132-byte array, as shown in Figure 4.6.

The PLC examines the integrity of those blocks and sends a TCP message, along with a reset flag, if the content of the blocks is different from what the PLC expects to receive. In

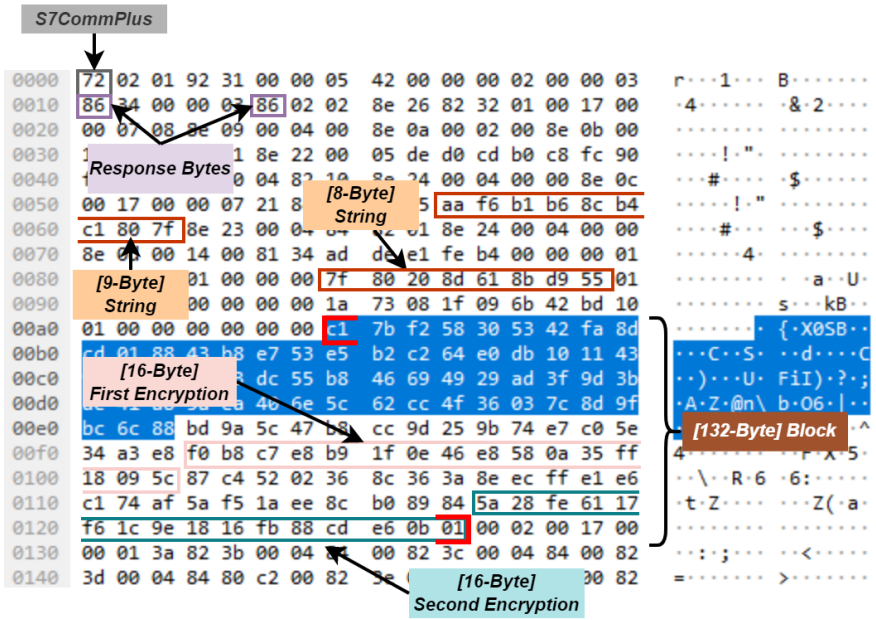


Figure 4.6: S7CommPlus Response Packet from TIA Portal to the PLC [11].

other respects, the establishment of the S7 session continues by sending an "*OK*" packet to the TIA Portal. The *Integrity Part* is generated by specific algorithms illustrated in the following subsection 4.1.3. Once the communication session is approved by the PLC, all subsequent packets exchanged between the TIA Portal and PLC are protected with an *Integrity Part* related to the functions provided by the TIA Portal. In the next section, we will investigate this communication process in more detail.

4.1.3 Investigating the Communication Process

In order to understand the encryption algorithms used in the S7CommPlus V3 protocol and explore possible exploits, we need first to analyze the communication process between the TIA Portal and PLC. To this end, a manual analysis was conducted using helpful tools such as Scapy[3] and WinDbg[4], and a number of different communication sessions. First, we open the TIA Portal and press on the "*go online*" button, then capture all the packets and save them in a pcap file for further analysis. To support our study, we use the WinDbg software that allows us to set several breakpoints during the communication session, which is comprised of four packets (see figure 4.5). In the following, we present our analysis results for each packet in detail.

[3]https://scapy.net/
[4]https://windbg.org/

S7 *Request* Packet

The TIA Portal initializes a new session by sending a *Request* packet to the PLC. This packet contains no encryption bytes; therefore, an attacker can reuse this packet "as-is" without making any appropriate adjustments.

S7 *Challenge* Packet

After the PLC receives the S7 *Request* from the TIA Portal, it responds by sending an S7 packet (which we call S7 *Challenge*). Our investigation showed that this packet has a 20-byte array that varies significantly every time the TIA Portal sends a new *Request*, i.e., every time the user presses the "go online" button. This 20-byte array is called *ServerSessionChallenge* and is always located in the 26th byte position of any S7 *Challenge* (as shown in figure 4.7).

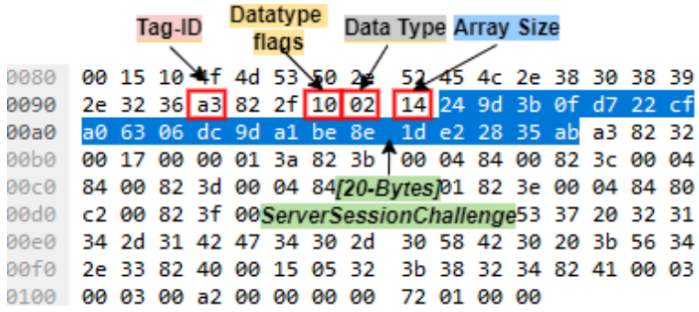


Figure 4.7: S7 *Challenge* Packet - *ServerSessionChallenge* Array [11].

Through further investigation and by inserting several breakpoints at the memory address where this array is located, we discovered that only 16 bytes—from byte 3 to byte 18 in the *ServerSessionChallenge*—were copied and stored at another address. This final 16-byte block, referred to as the *challenge* in [76], plays a crucial role in generating specific encryption bytes for the subsequent S7 *Response* and *Function* packets, as illustrated later in the following subsections (4.1.3 and 4.1.3).

S7 Response Packet

The *Response* packet is sent from the TIA Portal to the PLC as a response to the *Challenge* packet. It is quite complex and can be divided into several parts as shown in figure 4.6.

- Encryption Bytes can be manipulated:

Our investigations into this packet showed that the Secure Hash Algorithm-256 (SHA-256) is utilized two times to generate two hashes. The inputs for the SHA-256 algorithm are gen-

erated randomly using the Application Programming Interface (API) cryptography functions, specifically the "*CryptGenRandom*" function. See figure 4.8.

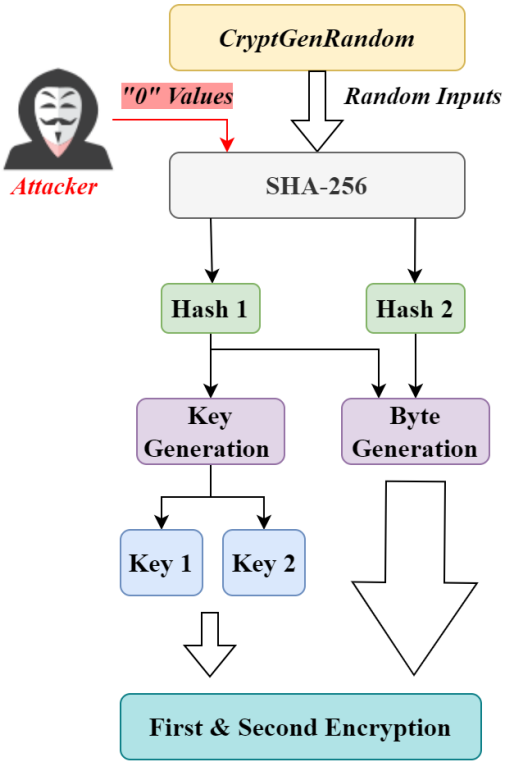


Figure 4.8: Generating Keys and Bytes for the First and Second Encryption [11].

The two resulting hashes are then used as a part of generating specific encryption bytes in the S7 *Response* packet. Figure 4.6 shows these bytes, which are as follows:

Block 1, 9-byte: located between the byte 91 and 99.

Block 2, 8-byte: located between the byte 136 and 143.

Block 3, 132-byte: Located between byte 168 and 299, this block is also divided into sub-blocks as follows: the first 76-byte block of the 132-byte block (located between byte 168 and 243), the "First Encryption" (16 bytes located between byte 244 and 259), and the "Second Encryption" (16 bytes located between byte 284 and 299).

Since the three encryption blocks are generated based on the two hashes that the SHA-256 introduces as outputs, adversaries can maliciously manipulate those blocks by manipulating the generation process of the two hashes. To this end, an attacker can use the WinDbg software to feed constant inputs to the hash function of the SHA-256 algorithm, which will eventually result in fixed hashes rather than random ones when the "*CryptGenRandom*" function is used. For instance, when we feed the hash function with "0" values as inputs, the

bytes representing the hashes generated by the SHA-256 algorithm remain constant in every session. This is a very serious vulnerability, as an attacker can subsequently generate the three encryption blocks, craft the entire S7 *Response* packet, and send it finally to the PLC without the need to have a TIA Portal software installed on his machine, as [76] assumed.

One of the two hashes, precisely "*Hash 1*," is used in a computation to generate two keys, each with a length of 16 bytes (referred to as "*Key 1* *Key 2*" for the remainder of this chapter). The resulting keys are then utilized in two symmetric-key encryption processes, specifically the AES-128 algorithm (referred to as "First Encryption Second Encryption" in [77]), as depicted in Figures 4.9 and 4.10, respectively. Since the inputs of the two hashes can be manipulated, the two keys can also be manipulated by attackers. Consequently, both encryption processes in the S7 *Response* packet could be manipulated. In the following, we explain in detail how an attacker can manipulate the First and Second Encryption processes.

- First Encryption Process:

Figure 4.9 depicts the "First Encryption" algorithm that the *Response* packet implements.

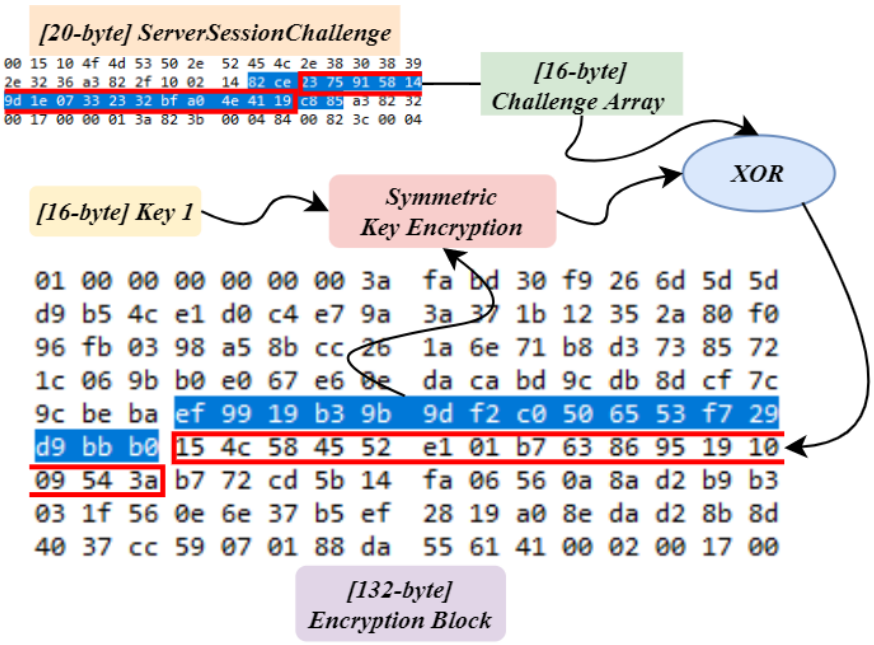


Figure 4.9: First Encryption in the S7CommPlus *Response* Packet [11].

As can be seen, the output of this encryption is placed between byte 77 and byte 93 of "block 3" (the 132-byte block in the S7 *Response* packet) and has a 16-byte length; see figure 4.6. Our analysis showed that the "First Encryption" process uses two inputs: 1) the bytes located between byte 61 and 76 (in "block 3") as plaintext, and 2) an encryption key, precisely

"*Key 1*". Afterwards, the output of the encryption process, which is a 16-byte block, will be XOR-ed with the 16-byte *challenge* array. The resulting output of the XOR operation is finally stored at a certain address before being sent to the PLC. Considering all of this, we can conclude that the "First Encryption" process is an XOR process of a fixed 16-byte block with the *challenge* array. Therefore, to manipulate this encryption, an attacker only needs to manipulate the hashes used to generate "*Key 1*," as mentioned earlier.

- Second Encryption Process:

The algorithm here is similar to the one used in the previous encryption process ("First Encryption"), namely AES-128. However, it differs in that the plaintext in the "Second Encryption" is generated by a sophisticated algorithm that utilizes the 16-byte output of the "First Encryption" as part of the inputs in the "Second Encryption." Figure 4.10 illustrates the complete algorithm used in this encryption process.

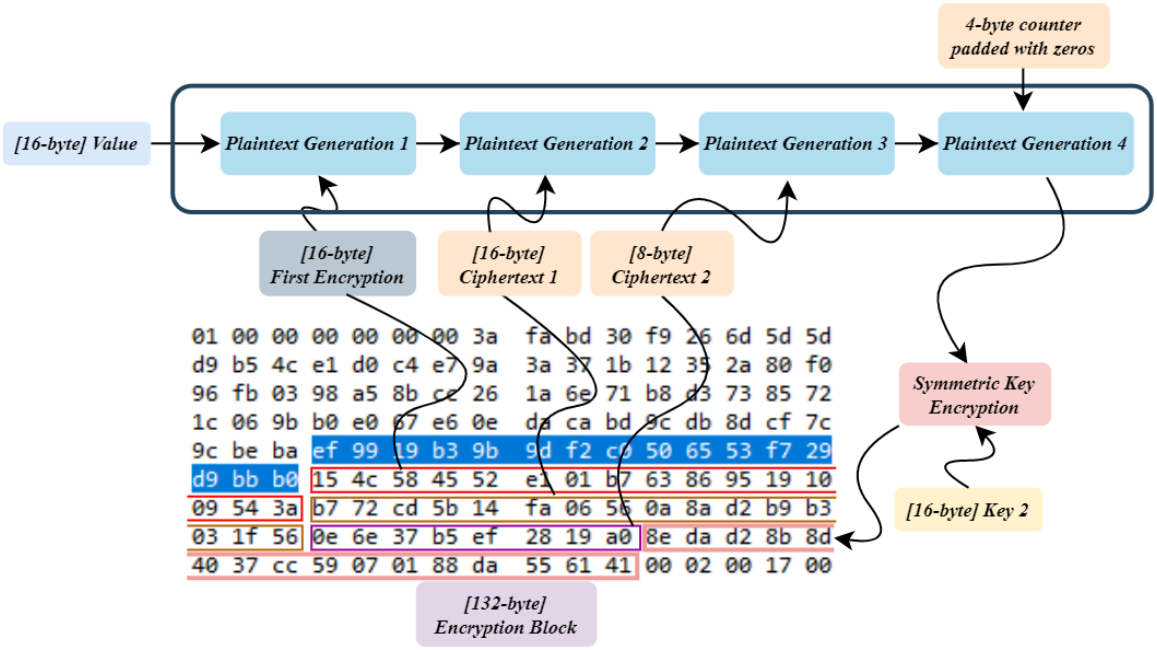


Figure 4.10: Second Encryption in the S7CommPlus *Response* Packet [11].

As can be seen, the 'Second Encryption' contains a four-stage 'Plaintext Generation' fed with five inputs, which are as follows: 1) a value with a 16-byte length, 2) the 'First Encryption' output, 3) a 16-byte ciphertext, 4) an 8-byte ciphertext value diminished from another ciphertext, and 5) a 4-byte value generated by a counter and padded with '0'. Our investigations showed that the two hashes we already identified are involved in the inputs of the 'Second Encryption', except for the 16-byte output of the 'First Encryption'. Furthermore,

each stage of generating plaintexts is an XOR operation of two inputs, and the result of each is fed as input to the next plaintext generation, as depicted in Figure 4.10. Once the last plaintext is produced, its value is then encrypted with the help of 'Key 2' using a similar algorithm to the 'First Encryption' algorithm. The output of the encryption algorithm is finally placed in the last 16 bytes of 'block 3' in the S7 *Response* packet, see Figure 4.6.

S7 Function Packet

Once the encryption processes (in the *Response* packet) are calculated, the PLC approves the connection with the TIA Portal when everything is correct. Subsequently, S7 *Function* packets containing the required data/operations will be sent from the TIA Portal to the PLC (see Section 4.1.2). Figure 4.11 shows one of these packets, which contains control information.

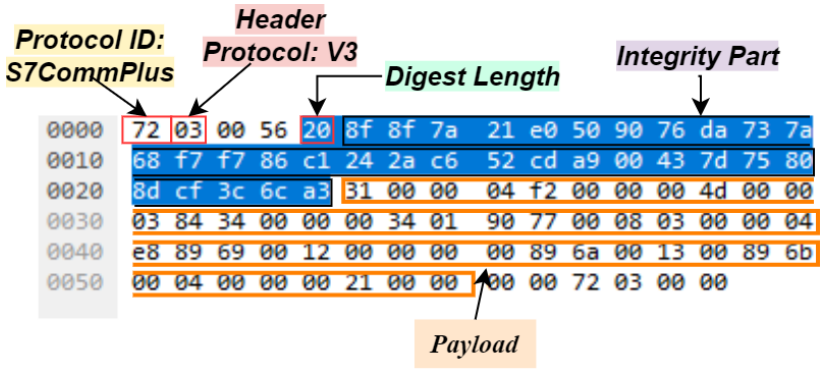


Figure 4.11: S7 *Function* Packet from the TIA Portal [11].

Each *Function* packet contains a 32-byte encryption block called the *Integrity Part* (as named in [75]) before the payload. Our analysis of the *Integrity Part* shows that this block is an Hash-based Message Authentication Code (HMAC) designed to examine the integrity of the *Function* packet. This examination aims to ensure that the payload has not been maliciously modified and to authenticate the TIA Portal, as the encryption keys used in the HMAC are only known by the connected parties in an ongoing S7 session. To calculate the *Integrity Part* block, two HMAC algorithms are called. The "First HMAC" is used to create an encryption key that is used in all subsequent HMAC operations, while the "Second HMAC" is used to digitally fingerprint all the following *Function* packets. These two HMAC algorithms are designed based on the same hashing algorithm, as follows.

- First HMAC:

The first HMAC is called before sending the S7 *Response* packet from the TIA Portal

to the PLC. The plaintext here consists of an 8-byte value (generated by a very specific algorithm critical to the S7 integrity check [76]) and the 16-byte *challenge* array, as shown in figure 4.12. The collective value is 24 bytes and is digitally signed using an encryption

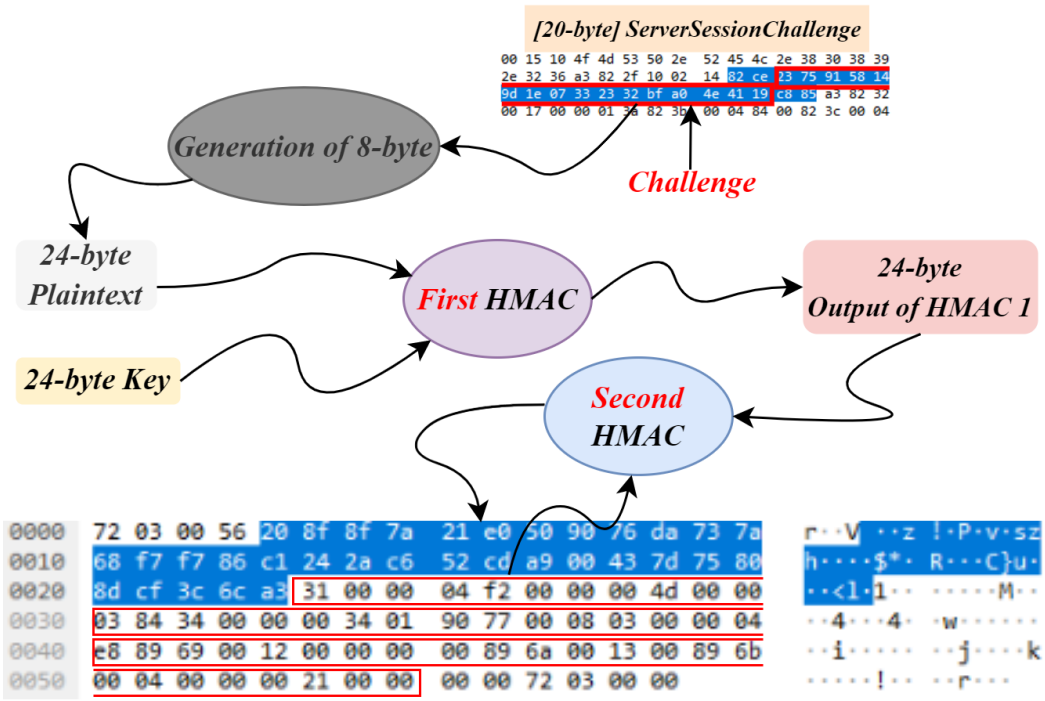


Figure 4.12: Integrity Part Encryption Process [11].

key (also 24 bytes) produced with the help of the two hashes identified earlier in 4.1.3. The output of the first HMAC is a 32-byte value, but it is diminished to only a 24-byte value that is eventually saved and utilized as an essential key in the "Second HMAC" computation.

- Second HMAC:

It is the actual algorithm that calculates the 32-byte *Integrity Part*. Please note that the length of the S7 *Function* packet varies significantly based on the purpose of the packet. However, the HMAC output (32 bytes) always starts at byte 5 of any *Function* packet; see figure 4.12. The "Second HMAC" takes all the bytes after the 32-byte *Integrity Part* as input, i.e., starting from byte 38, eliminating the packet's footer, which is usually the last six bytes (e.g., in the packet shown in figure 4.12, these six bytes are "*00 00 72 03 00 00*" at the end of the packet). Since the length of each *Function* packet and its payload can vary, the footer also varies from one *Function* packet to another. However, because attackers are familiar with the key generation process, a simple trial-and-error method could easily determine which bytes are used as input in the second HMAC computation.

Appendix B provides more technical elaborations on the four-handshake messages as well as the algorithms used to establish a successful communication session in S7CommPlus V3 protocol.

4.2 Attack Approach

As with any typical injection attack, we patch malicious code, ToD interrupt block OB10, into the original control logic of the target PLC. The CPU checks whether the interrupt condition is met in each single execution cycle. This means that the attacker's interrupt block will always be checked but only executed if the date and time of the CPU's clock match the date and time set by the attacker. Hence, we have two cases:

- The date of the CPU's clock matches the date set in the OB10 (the date of the attack). The CPU immediately halts executing OB1, stores the breaking point's location in a dedicated register, and jumps to execute the content of the corresponding interrupt block OB10.
- The date of the CPU's clock does not match the date set in OB10. The CPU resumes executing OB1 after checking the interrupt condition without activating the interrupt and without executing the instructions in OB10.

Our attack approach presented in this chapter comprises two main phases: patching the PLC (online phase) and attacking the physical process (offline phase). Please note that obtaining the IP address, MAC address, and model of the victim PLC is an easy task accomplished by running our PN-DCP protocol-based scanner presented in [1] or other network scanners that can gather all the information the attacker needs to communicate with the target device.

4.2.1 Patching Phase

Figure 4.13 presents a high-level overview of this phase. Our goal is to inject the PLC with malicious instructions programmed in the interrupt block OB10. This phase comprises four steps:

- Uploading and downloading the user's program.
- Modifying and updating the control logic program.
- Crafting the S7CommPlus download message.
- Pushing the attacker's message to the victim PLC.

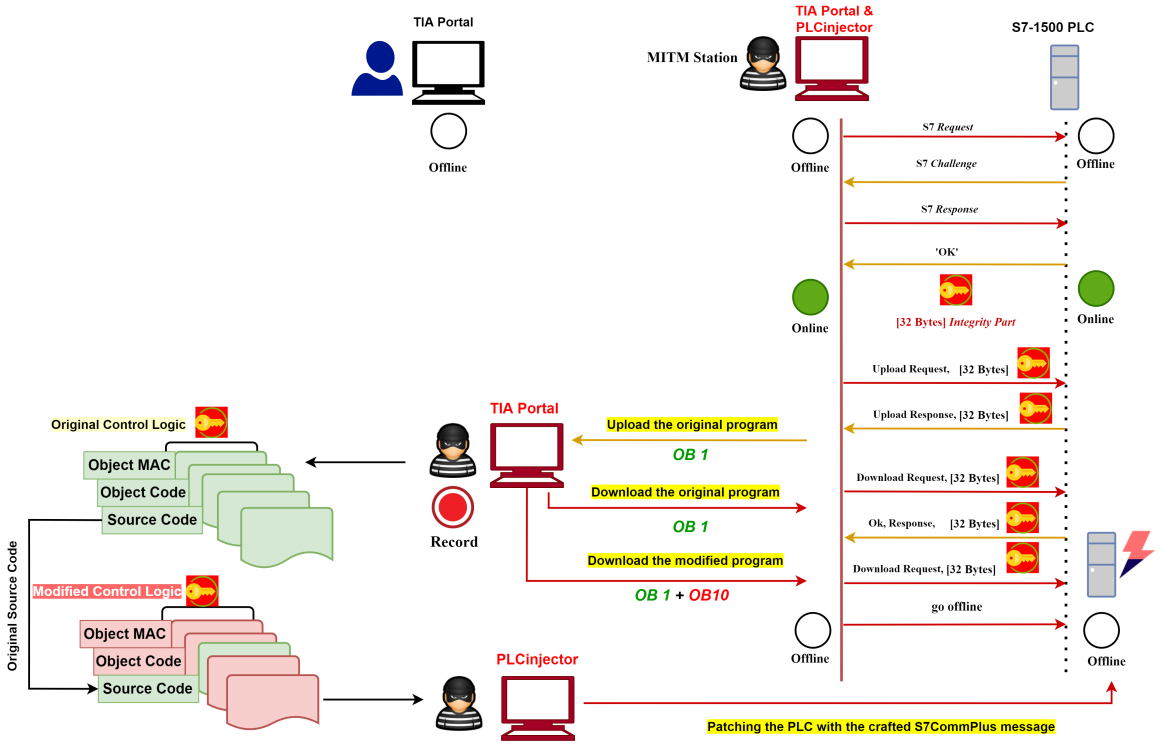


Figure 4.13: High-level Overview of the patching phase - Step 1: an attacker records the upload/-download network streams. Step 2: Alter the control logic program. Step 3: Craft the download message. Step 4: Push the crafted message to the PLC [6]

To compromise the target PLC, we employ our MitM station, which consists of two main components:

- **A TIA Portal:** used to retrieve and modify the current control logic program that the PLC is running.
- **A PLCinjector:** employed to generate a malicious *Integrity Part* and download the attacker's program to the PLC. In this work, we developed a Python script based on the Scapy library for this purpose.

For a realistic scenario, an attacker may encounter one of two possible cases after gaining access to the network.

Case_1: Inactive S7 Session

In this scenario, the legitimate TIA Portal is offline, and only communicates with the PLC if an upload process is required.

- Step 1: Uploading and Downloading the Original Control Logic Program:

In this step, our objective is to obtain the decompiled control logic program that the PLC runs, as well as the S7CommPlus V3 messages that the TIA Portal sends to download the original user program into the PLC. To achieve these goals, we first open the attacker's TIA Portal and establish a direct connection with the victim PLC, as shown in Figure 4.14.

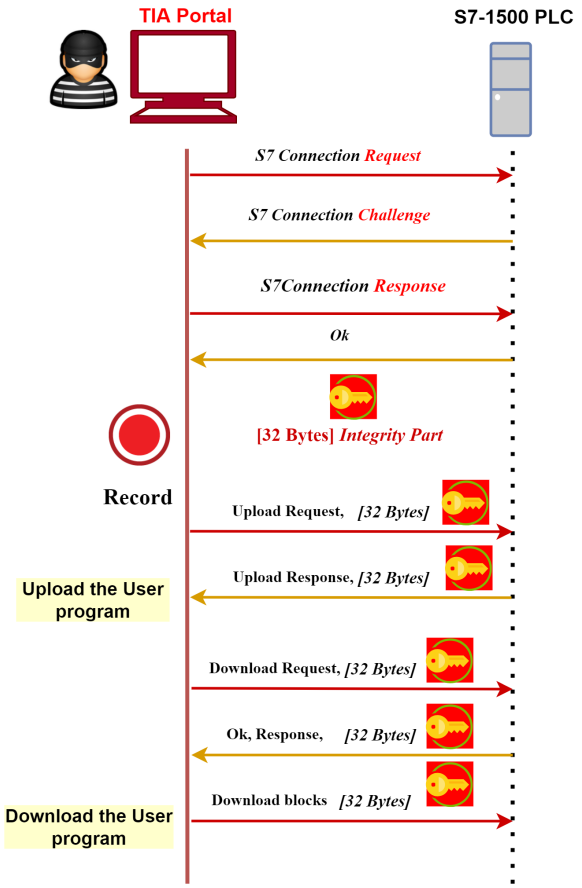


Figure 4.14: Step 1: Upload, Download and record the program [7]

This is possible due to a security gap in the S7-1500 PLC design. In fact, the PLC does not introduce any security checks to ensure that the currently communicating TIA Portal is the same TIA Portal that it communicated with in an earlier session. Therefore, any external adversary provided with a TIA Portal on their machine can easily communicate with an S7 PLC without any effort.

After successfully establishing communication, we upload the control logic program to the attacker's TIA Portal. Then, we re-download it once again to the PLC and sniff the entire S7CommPlus V3 message flow exchanged between the attacker's TIA Portal and the victim

PLC using Wireshark. At the end of this step, the attacker has the program on his TIA Portal, and all the captured download messages are saved in a pcap file for future use (as explained in step 3).

- Step 2: Modifying and Updating the Control Logic Program:

After retrieving the program that the target PLC runs, the attacker's TIA Portal displays it in one of the high-level programming languages in which it was programmed (e.g., SCL). For the given experimental setup in Section 4.3.1, and based on our understanding of the physical process controlled by the PLC, we configure and program our ToD interrupt block (OB10) to force certain outputs of the system to switch off once the interrupt is activated, as shown in Figure 4.15.

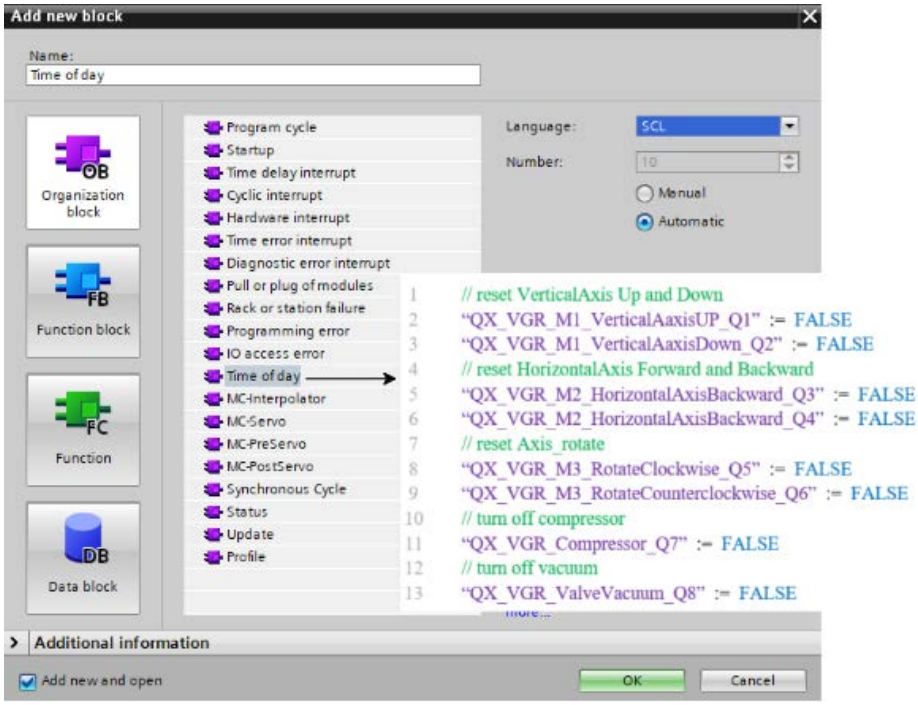


Figure 4.15: Programming the OB10 with malicious instructions - Setting the mini-motors of the industrial modules at the value '0' [6]

As can be seen, all eight mini-motors existing in the fischertechnik are set to the value '0'. This forces the industrial modules to stop operating once the interrupt is activated. Although our malicious code differs from the original code by only an extra small-sized block (OB10), it is sufficient to confuse the physical process of our experimental setup. The easiest way to update the program running in the PLC is to use the attacker's TIA Portal. When we downloaded the modified control logic, the PLC updated its program successfully. However,

the ICS operator could easily reveal the modification by uploading the program from the infected PLC and comparing the offline and online programs running on his legitimate TIA Portal and the remote PLC, respectively.

- Step 3: Crafting the S7CommPlus Download Messages:

To conceal our infection from the legitimate user, we initially recorded the S7CommPlus V3 messages exchanged between the attacker's TIA Portal and the PLC during the download of the modified program. As mentioned earlier in Section 4.2.2.2, each download message contains objects and attributes (see Figure 4.4). The *ProgramCycleOB* object is dedicated to creating a program cycle block in the PLC's memory and has three different attributes:

a) Object MaC: with the item value ID: *Block.AdditionalMac.*
b) Object Code: with the item value ID: *FunctionalObject.code.*
c) Source Code: with the item value ID: *Block.BodyDescription.*

The *Object Code* is the code that the PLC reads and processes, while the *Source Code* is the code that the TIA Portal decompiles, reads, and displays for the user. Therefore, all that is required to show the user the original code is to modify the S7CommPlus V3 message that the attacker sends; by replacing the *Source Code* attribute of the *ProgramCycleOB* object of the attacker's program with the *Source Code* attribute of the *ProgramCycleOB* object of the original program. Our investigation showed that the newest model of the SIMATIC PLCs has a serious design vulnerability. The PLC checks the session freshness by running a precautionary measure. Hence, it can detect any manipulation and refuses to update its program in case the attributes do not belong to the same session. Surprisingly, this holds true only for the *Object MaC* and the *Object Code* attributes. Meaning that, to make the PLC accept the forged messages, our crafted S7CommPlus download message must always have the *Object MaC* and *Object Code* attributes from the same session, while the *Source Code* attribute could be substituted with another attribute from a different session, i.e., from a pre-recorded session.

- Step 4: Injecting the PLC:

The crafted S7CommPlus download message contains the following attributes: The *Object MaC* and *Object Code* attributes of the attacker's program, and the *Source Code* attribute of the user program. See Figure 4.16.

As S7CommPlus V3 exchanges a 32-byte *Integrity Part* between the TIA Portal and the PLC to prevent replay attacks, we first need to calculate the *Integrity Part* correctly before pushing the crafted message to the PLC. To achieve this, our developed *PLCinjector* tool [6] generates a malicious *Integrity Part* first. Once the *Integrity Part* is generated, we can

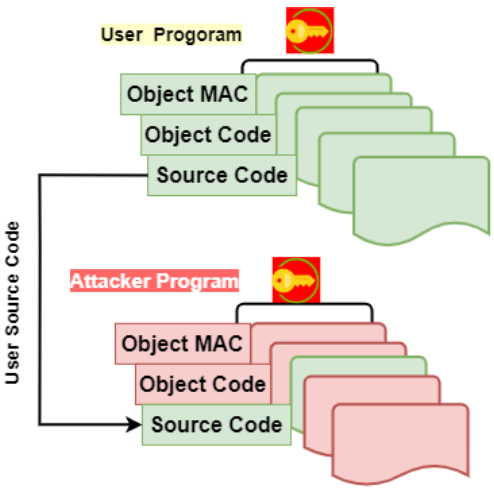


Figure 4.16: Crafting the S7CommPlus download messages [7]

easily bundle the malicious 32-byte *Integrity Part* with our crafted messages. Taking into consideration the appropriate modifications to the session ID and other integrity fields, we store the final S7 messages (the attacker messages) in a pcap file to push it back to the PLC as a reply attack. Please note that all the algorithms used to generate the *Integrity Part* in S7CommPlus V3 protocol are illustrated in detail in Appendix B.

After a successful injection, the PLC updates its program, processing the *Object Code* attribute of the attacker's program. Meanwhile, the *Source Code* of the user's program is saved in the PLC memory. Now, whenever the user requests the program from the infected device (upon an upload request), the PLC will send the *Source Code* attribute from its memory to the EWS. The TIA Portal will decompile the *Source Code* and display it to the user, who will always see the original program, i.e., the program that he wrote and that is already on his TIA Portal. This keeps our injection hidden inside the PLC, and the user will not detect any differences between the online (currently running on his TIA Portal) and offline (currently running in the PLC) programs.

Case_2: Active S7 Session

In this scenario, there is an ongoing active S7 session between the legitimate TIA Portal and the PLC during the patch. As the PLC allows only one active online session by default, an attacker is unable to communicate with the PLC. In other words, the PLC will immediately reject any attempt to establish a connection as it is already in communication with the user. In such a scenario, the attacker needs to first close the current online session between the legitimate user and the PLC before patching their malicious code. However, a user can

establish an online session with an S7 PLC by enabling the "*go online*" feature in the TIA Portal software. Then, they can remotely control, monitor, diagnose, download, upload, start, and stop the PLC. Once the user has established an online connection with the PLC, the two parties (TIA Portal and PLC) start exchanging a specific message along the session regularly. This message is known as S7-Acknowledgment (S7-ACK) and is in charge of keeping the session alive. The TIA Portal must always respond to any S7-ACK request sent by the PLC with an S7-ACK reply message. Therefore, to close the current online session, we run our MitM station, allowing us to intercept and drop all packets sent from the TIA Portal right after sending an acknowledgment request. This action closes the ongoing connection between the TIA Portal and PLC, and both go offline. Figure 4.17 depicts this scenario.

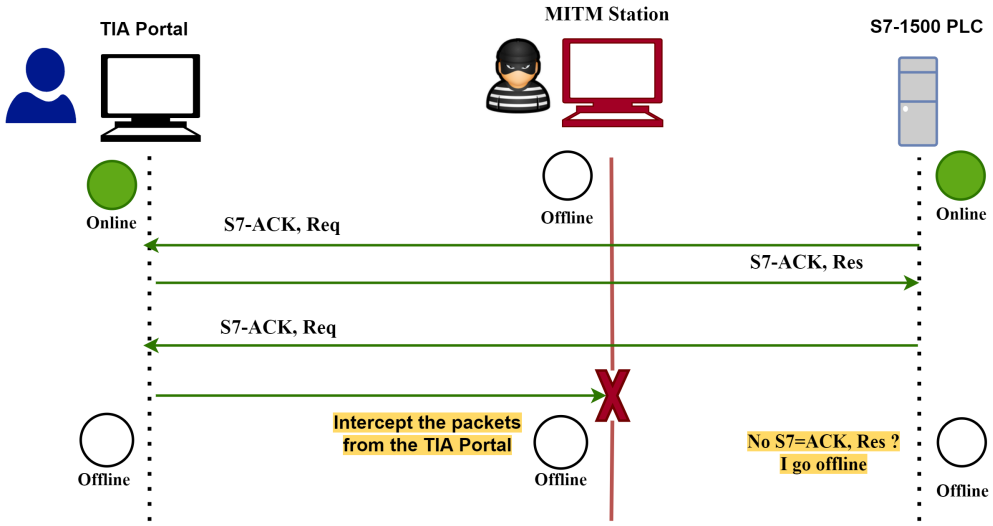


Figure 4.17: Closing the online session using MitM approach [6]

It is worth mentioning that an attacker can also use different methods to close the connection, such as port stealing, replay attacks with "go offline" packets, etc. After both the legitimate TIA Portal and the victim PLC go offline, the attacker can easily establish a new session with the PLC using their own TIA Portal. Then, they patch the victim device following the same four steps explained in the previous case.

For this scenario, our patching approach has limitations. The legitimate TIA Portal is forced to close the live session with the PLC, meaning that the user can clearly see that they have lost connection with the remote device. If they attempt to reconnect to the PLC while it is connected to the attacker's TIA Portal, the PLC will refuse their connection request. Our investigations showed that there is no way to reconnect the legitimate TIA Portal to the victim PLC after patching the PLC, unless the ICS operator enables "go online" on their

TIA Portal. This abnormal disconnection between the two parties is the only effect of our patch in this scenario.

4.2.2 Attack Phase

After a successful injection, the attacker goes offline and closes the current communication session with the target PLC. In the next execution cycle, the attacker's program will be executed in the PLC. This means that the interrupt condition of the malicious interrupt block OB10 will be checked in each execution cycle and will remain in idle mode (hidden) in the PLC's memory as long as the interrupt condition is not met. Once the configured date and time of OB10 matches the date and time of the CPU, the interrupt code will be activated, i.e., the execution process of the main program (OB1) is suspended, and the CPU jumps to execute all the instructions that the attacker programmed into his malicious OB10. In our experimental setup, as seen in figure 4.18, we programmed OB10 to force certain motors to turn off at a specific time and date when we are completely disconnected from the target's network.

4.3 Implementation and Evaluation

In this section, we present the implementation of our attack approach and assess the service disruption of the physical process caused by our patch.

4.3.1 S7-1500 PLC based Experimental Setup

For testing our attack approach, we used the Fischertechnik training factory shown in figure 4.18. It consists of five industrial modules: Vacuum Suction Grippers (VGR), High-Bay Warehouse (HBW), Multi-Processing Station with Klin (MPO), Sorting Line with Color-Detection (SLD), and an environment Station with Surveillance Camera (SSC). The entire factory is controlled by a SIMATIC S7-1512SP with firmware V2.9.2 and programmed by TIA Portal V16. The PLC connects to a so-called *TXT* controller[5] via an Internet of Things (IoT) gateway. The *TXT* controller serves as a Message Queuing Telemetry Transport (MQTT) broker and an interface to the Fischertechnik cloud.

The factory we used in our experiments provides two industrial processes: sorting and ordering materials. The default process cycle begins with storing and identifying the material, i.e., workpiece. The factory has an integrated Near Field Communication (NFC) tag sensor storing production data that can be read out via an Radio Frequency IDentification (RFID)

[5]https://www.fischertechnik.de/en/service/elearning/teaching/txt-40-controller

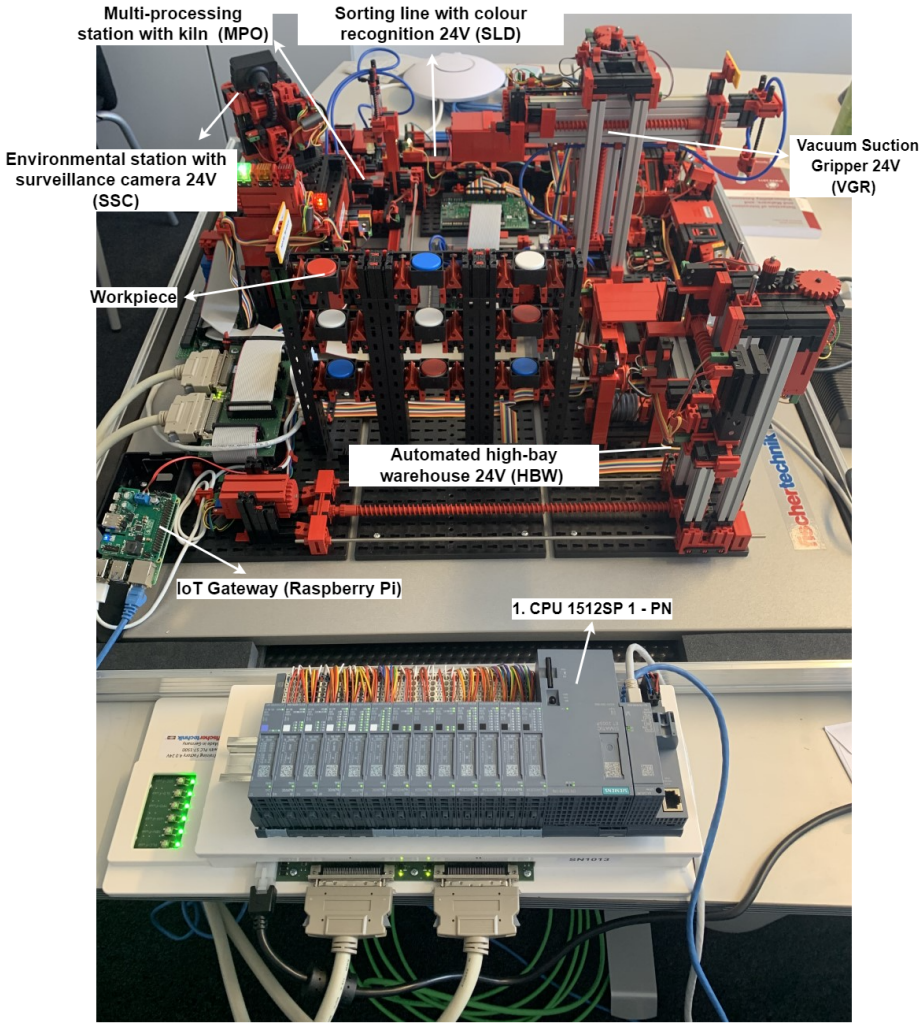


Figure 4.18: S7-1500 PLC based Experimental setup [6, 7]

module. This allows the user to track the workpieces digitally. The cloud displays the part's color and its ID-number. Afterwards, the VGR places suction on the material and transports it to the HBW, which applies a first-in-first-out principle for outsourcing. All goods that were stored could be ordered again online using a dashboard. The desired product and the corresponding color are selected by the user and then placed in the shopping cart. The suction gripper passes the workpiece from one step to the next and then moves back to the sorting system once the production is complete. The sorting system receives the allocation command as soon as the color sorter detects the proper color. The material is sorted using pneumatic cylinders. Finally, the production data are written on the material at the end of the production process, and the finished product will be provided for collection.

4.3.2 Attack Implementation

In our experimental setup, as shown in Figure 4.18, we discovered that the VGR module is integral to all the industrial processes carried out by the Fischertechnik system. Consequently, if we were able to disrupt its functionality, the entire system would be affected. The VGR module is propelled by 8 mini-motors, operating as follows:

- Vertical motor up: "QX_VGR_M1_VerticalAxisUP_Q1" (%Q2.0)
- Vertical motor down: "QX_VGR_M1_VerticalAxisDown_Q2" (%Q2.1)
- Horizontal motor backwards: "QX_VGR_M2_HorizontalAxisBackword_Q3" (%Q2.2)
- Horizontal motor forwards: "QX_VGR_M2_HorizontalAxisForward_Q4" (%Q2.3)
- Turn motor clockwise: "QX_VGR_M3_RotateClockwise_Q5" (%Q2.4)
- Turn motor anti-clockwise: "QX_VGR_M3_RotateCounterclockwise_Q6" (%Q2.5)
- Compressor: "QX_VGR_Compressor_Q7" (%Q2.6)
- Valve vacuum: "QX_VGR_ValveVacuum_Q8" (%Q2.7).

Therefore, for exploiting the VGR, we programmed our OB10 to force all these 8 motors to switch off at the point zero for the attack as shown in listing 4.2.

Listing 4.1: Malicious Instructions inserted in OB10

```
#Reset Vertical-Axis Up and Down
 "QX_VGR_M1_VerticalAxisUP_Q1" := False;
 "QX_VGR_M1_VerticalAxisDown_Q2" := False;
#Reset Horizontal Axis Forward and Backward
 "QX_VGR_M2_HorizontalAxisBackword_Q3" := False;
 "QX_VGR_M2_HorizontalAxisForward_Q4" := False;
#Reset Axis Rotate
 "QX_VGR_M3_RotateClockwise_Q5" := False;
 "QX_VGR_M3_RotateCounterclockwise_Q6" := False;
#Turn off compressor
 "QX_VGR_Compressor_Q7" := False;
#Turn off Vacuum
 "QX_VGR_ValveVacuum_Q8":= False;
```

After patching the PLC with our malicious block and before the ToD interrupt was activated, we did not record any physical impact, and the Fischertechnik system continued operating normally. Once the CPU clock matched the attack time we set, we noticed that the VGR module stopped moving. Furthermore, the workpiece being transported by the gripper fell down, as the compressor, which provides the appropriate airflow to carry the goods, was turned off. This led to an incorrect operation, and the movement sequence of the workpiece

was disrupted. In a real-world heavy factory, such as the automobile manufacturing industry, such an attack scenario might be seriously dangerous and could even cost human lives.

4.3.3 Evaluation

To accurately assess the impact of our patch on the physical process controlled by the infected device, we measured and analyzed the differences in the execution cycle times for the control logic program that the PLC runs in three different scenarios:

- Normal Operation: before patching the PLC as a baseline.

- Idle Attack: after patching the PLC and before the interrupt is activated, i.e., the PLC is running the attacker's program.

- Activated Attack: after the interrupt is executed.

Siemens PLCs, by default, store the time of the last execution cycle in the local variable of OB1 (called *OB1_PREV_CYCLE*). Therefore, we added a small STL code snippet to our control program, which stores the last cycle time in a separate data block (see listing 4.3 [34]).

Listing 4.2: STL code to switch the output in every execution cycle

```
1  PROGRAM PLC_PRG
2   VAR
3     Output: INT := 1; #Variable for Output, "1"
4   END_VAR
5
6   IF Output = 0 THEN #Compare Output Variable with "0"
7     Output1 := False; #Set Output1 Low
8     Output := 1; #Set Output Variable to "1"
9   ELSE
10    Output1 := True; #Set Output1 high
11    Output := 0; #Set Output Variable to "0"
12  END_IF;
13
14 END PROGRAM
```

Then we recorded 4096 execution cycle times for each scenario, calculated the arithmetic median value and used the Krulsal-Wallis [6] and the Dunn's Multiple Comparison test[7] for statistical analysis. All the results are presented as box-plots in figure 4.19.

[6] https://www.statology.org/kruskal-wallis-test/
[7] https://www.statology.org/dunns-test/

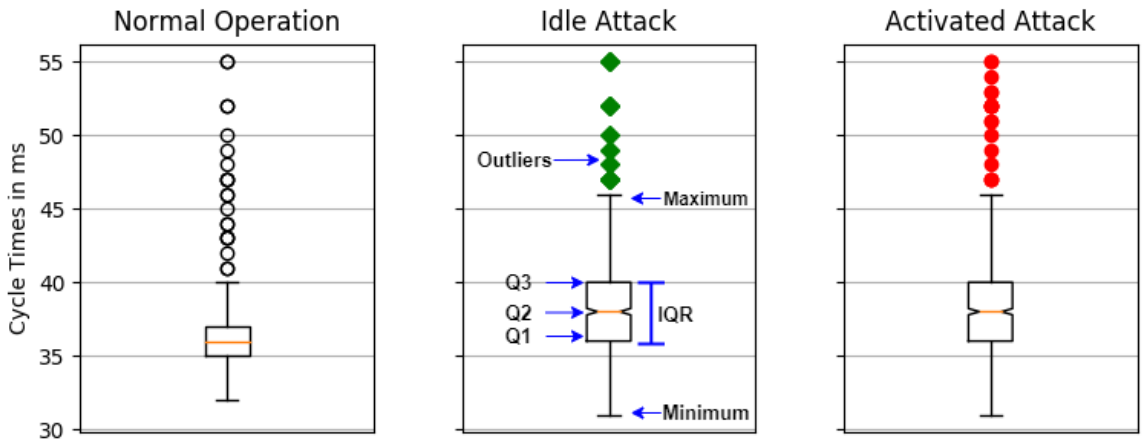


Figure 4.19: Box-plot presenting the measured execution cycle times of OB1 [6]

To make our resulting box-plots clearer and easier to read, we define the following parameters:

a) First Quartile (Q1): represents the middle value (Cycle Time) between the smallest value and the median of the total recorded values (4096 execution cycle times).

b) Median (Q2): represents the middle value of the total recorded values.

c) hird Quartile (Q3): represents the middle value between the highest value and the median of the total values recorded.

d) Interquartile Range (IQR): represents all the values between 25% and 75% of the total values recorded.

e) Maximum: represents Q3 + 1.5* IQR.

f) Minimum: represents Q1 – 1.5* IQR.

g) Outliers: represents all the values that they are higher and lower than maximum and minimum values respectively.

Our measurements show that the calculated Q2 for executing the OB1 for the infected program is approximately 38 ms, slightly differing from the median value of executing the OB1 for the original program, which is almost 36 ms. The Q1 and Q3 values for the infected program are as high as 36 ms and 40 ms, respectively. They are a bit higher compared to the recorded ones for the original program, i.e., 35 ms and 37 ms for Q1 and Q3, respectively. This implies that checking the interrupt condition of our malicious block in each execution cycle does not disrupt the execution of the control logic, and the Fischertechnik system keeps operating normally. Please note that executing the attacker's program should not exceed the overall maximum execution time of 150 ms that Siemens allows its PLCs to process user

programs. Our measurements clearly show that the injection did not trigger this timeout, as we recorded a maximum value as high as 47 ms, which is still quite small compared to 150 ms.

Once the CPU's date and time match the date and time that we set to trigger our attack, the CPU jumps to execute the malicious instruction existing in OB10, and the attack is activated. Our measurements for this scenario did not record any higher median values in the execution cycles compared to the prior scenario, i.e., when the attack is idle. This is because we set the OB10 to occur only once, so the PLC processes the instructions existing in OB10 and resumes executing OB1 from the last point before the interrupt. However, it keeps checking the condition of the interrupt in each cycle as long as OB10 exists in the control logic program. Our approach allows attackers to adjust the frequency of the interrupt and program the interrupt block at will, causing different impacts on the physical process of the target system.

4.3.4 Discussion

Based on our experiments conducted in this chapter, we can conclude that when our patch is in idle mode, the execution cycle times of the infected program are almost as high as the execution cycle times of the original program. Therefore, the ICS operator would not record any abnormality in executing the control logic, as the TIA Portal software will not report any differences before and after the patch. Furthermore, our attack approach always shows the original program to the ICS operator, despite the fact that the infected PLC is running a different one. We found that when the user requests the program from the PLC upon an upload request, the PLC sends only a copy of the *Source Code* attribute saved in its memory to the TIA Portal. Additionally, it is noticed that Siemens provides its S7-1500 PLCs with a sophisticated integrity-checking algorithm that verifies the validity of any S7 message received, i.e., checking the 32-byte *Integrity Part* of each function packet. In addition, S7-1500 PLCs check the integrity of specific attributes created during a download operation, precisely the attributes existing in the *ProgramCycleOB* object. Unfortunately, this does not hold true for the entire *ProgramCycleOB* object. Meaning that, the CPU checks only the integrity of the *Object MaC* and the *Object Code*, and has no integrity check for the *Source Code*. So, if an attacker replaces the *Source Code* from another session with a new one, the PLC will authenticate the download message and run the attacker's program. This is a significant security gap in the design of the integrity mechanism for S7-1500 PLCs, as it keeps the injection hidden inside the memory. Please note that attackers need to provide the correct *Integrity Part* in the crafted S7 messages, otherwise the PLC will detect that the expected S7

message received has been modified and will refuse to update its program. Siemens claimed that the newest PLCs are resistant against replay attacks, but we could maliciously update the PLC's program by sending a crafted S7 download message. Our attack is capable of staying in the device in idle mode for a long time without being revealed, and the only way to remove it is to re-program the device once again by the ICS operator. However, in critical facilities and power plants, re-programming the PLCs is not a common case unless there is a specific reason to do so.

Another vulnerability we detected during our investigations is that there is no security pairing between the TIA Portal and the PLC, i.e., the PLC does not ensure that the TIA Portal currently communicating with it is the same TIA Portal from a previous session. This allows an attacker who has a TIA Portal installed on his machine to easily access the PLC without any efforts. Although this holds true as long as the target PLC is not already connected online to the legitimate TIA Portal. Our results showed that an attacker can still communicate and inject the victim after closing the current session between the TIA Portal and the PLC.

4.4 Mitigation Solutions and Security Recommendations

The fundamental solution would involve completely redesigning the integrity check mechanism that the newest S7CommPlus protocol uses. The enhanced mechanism should include security pairing and mutual authentication between the PLC and TIA Portal. However, we are aware of the fact that such a solution would also incur an extremely high cost and may have backward compatibility issues. Furthermore, ICS devices are usually not updated on time, and they have a very long life-cycle compared to common IT devices. Despite this, we should expect that insecure devices will continue to be employed in real-world ICS environments for a long time. In this regard, monitoring the control network can be seamlessly integrated into the existing ICS setting. In particular, control logic detection [87], verification [65], [116], and remote code attestation can be utilized to alleviate the current situation. Since our injection was hidden in the PLC memory, partitioning the memory space and enforcing memory access control [117] could also be a reasonable solution. Other suggestions include employing standard cryptography methods such as digital signatures (for messages like control logic manipulation) and using network monitoring tools like Snort [109], AepAlert[8], and ArpWatchNG[9] to reveal any attacks, including MitM attacks. Furthermore, a mechanism to check the protocol header, which contains information about the payload, is also recommended

[8] https://www.arpalert.org/arpalert.html
[9] https://www.kali.org/tools/arpwatch/

as a solution to detect and block any potential unauthorized transfer of control logic. However, from our perspective, the best solution to prevent injection attacks is to separate the IT domain from operational technology networks by using a Demilitarized Zone (DMZ).

4.5 Summary

This chapter discusses a new control logic injection attack scenario on cryptographically protected PLCs. As a testing device, we selected the latest S7 PLC model, i.e., S7-1500 PLCs. Our investigation shows that S7-1500 PLCs and its S7CommPlus V3 protocol have serious vulnerabilities. Based on our findings, we performed a sophisticated injection attack scenario that infects an exposed PLC with a ToD block. The malicious interrupt block allows adversaries to trigger the patch at a certain time and date, and eventually disturb the industrial process without being connected to the PLC or its network at point zero for the attack. Our investigations proved the concept that the original control logic program is always displayed on the legitimate TIA Portal, while the infected PLC runs another program. On the other hand, our malicious program does not exceed the overall maximum execution time of 150 ms. Hence, the industrial process is not interrupted/disturbed when the patch is in idle mode, and the malicious infection will not be detected. Finally, we provide some possible security recommendations to secure our ICS environments from such a severe threat.

CHAPTER 5

Blind False Data Injection against Profinet I/O based Systems

Contents of this chapter are as follows:

Parts of this section have already been published in the following papers:

- W. Alsabbagh and P. Langendörfer, "A Fully-Blind False Data Injection on PROFINET I/O Systems," 2021 IEEE 30th International Symposium on Industrial Electronics (ISIE), 2021, pp. 1-8, doi: 10.1109/ISIE45552.2021.9576496 [4].

- W. Alsabbagh, S. Amogbonjaye, D. Urrego and P. Langendörfer, "A Stealthy False Command Injection Attack on Modbus based SCADA Systems," 2023 IEEE 20th Consumer Communications Networking Conference (CCNC), Las Vegas, NV, USA, 2023, pp. 1-9, doi: 10.1109/CCNC51644.2023.10059804. [10].

This chapter introduces an FDI attack against PLCs and their industrial field-buses, i.e., Profinet I/O. To this end, we present a new attack approach based on implementing an I/O *Database* created prior to launching the attack, i.e., offline. This I/O *Database* allows adversaries to conduct a fully-blind FDI scenario and exploit the Profinet I/O communication between connected stations without any prior knowledge of the target system or its network. The full attack chain presented in this chapter consists of two main phases: Offline, which includes sniffing and collecting data prior to our attack, and Online, which involves injecting and forwarding false data to the victim(s) in real-time. For real-world attack scenarios, we test our attack on a real Profinet I/O setup based on S7-300 PLCs.

The rest of this chapter is structured as follows. In Section 5.1, we provide a brief overview of the Profinet I/O systems, while Section 5.2 illustrates our false data injection approach. In Section 5.3, we present the results of implementing our approach on the given experimental setup and propose some mitigation solutions and security recommendations in Section 5.4. Section 5.5 concludes this chapter.

5.1 Profinet I/O Background

Modern ICS components are increasingly connected over Profinet I/O communication to exchange control data between engineering stations, PLCs, and other industrial devices. This concept is based on the Ethernet standard provided by Institute of Electrical and Electronics Engineers (IEEE) 802.3, allowing the connected stations to establish and maintain connectivity through three different channels: Real-Time (RT), Non Real-Time (NRT), and Isochronous Real-Time (IRT), as shown in figure 5.1. These channels coexist in AR between nodes and satisfy all the requirements for industrial automation. A minimal Profinet I/O system comprises, at least, one PLC and one or more devices as peripheral equipment connected over Ethernet [107].

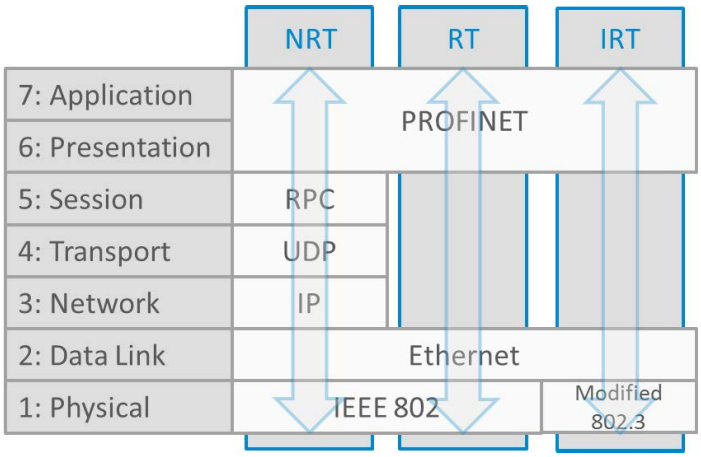


Figure 5.1: PROFINET I/O communication channels [124]

5.1.1 Profinet I/O Classes

Profinet I/O defines three types of devices based on their roles in the network as follows:

- **IO-Supervisor**: This device is used for project engineering, diagnostics, and troubleshooting. More likely, it is a PC, HMI, or a programming device.

- **IO-Controller**: This is typically a PLC, which is responsible for controlling the automation process.
- **IO-Device**: This is a field device that exchanges data (e.g., sensor/actuator values) with one or more IO-Controllers.

5.1.2 Profinet I/O Configuration

Figure 5.2 depicts the configuration process to establish a Profinet I/O communication.

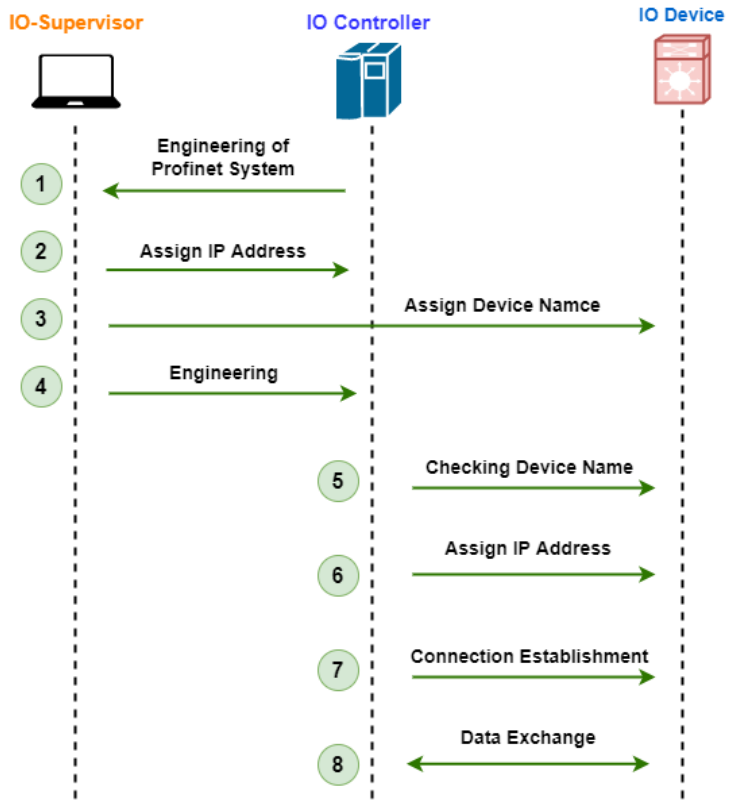


Figure 5.2: Profinet I/O Configuration Process

At the beginning, (1) the IO-Supervisor plans the entire system, i.e., the engineering software models both the automation process and the desired network topology. Afterwards, (2) the IP address of the IO-Controller and the (3) device name are then set by the IO-Supervisor. Thereafter, (5) the name of the device is checked, and (6) the IP address is assigned by the IO-Controller. To be able to transmit data between the IO-Controller and IO-Device, (7) a logical channel called AR has to be established. Within the AR channel, further Communication Relation (CR) is configured, as shown in figure 5.3. In order to set parameters and diagnostics, a Record Data (RD) is utilized over the NRT channel, whereas

cyclic data exchange and alarms are set to be sent over the RT channel. The connection is established, and (8) transmitting data in real-time begins [107].

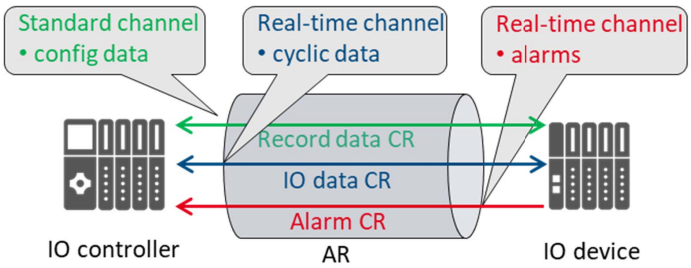


Figure 5.3: Data Exchange within an Application Relationship [125]

5.1.3 Profinet I/O Security Issues

Although integrating Profinet I/O with ICSs provides better network connectivity and a more streamlined control process, it also comes with its security challenges. This is due to the fact that Profinet I/O nodes do not have any endpoint security functionality, which eventually exposes them to a variety of attacks such as MitM, DoS, FDI, control logic injection, replay attack, etc.

One of the well-known vulnerabilities that might compromise the communication integrity of Profinet I/O systems is fake sensor readings or actuator values exchanged between the connected stations, i.e., between IO-Controller, IO-Device, and IO-Supervisor. These threats are known as deception attacks [106] or false data injection attacks in the IT world and occur when the physical values of a hardware device are manipulated by sending fake values for signals to a victim device. Such manipulations are severe due to the fact that PLCs keenly rely on reading accurate sensor measurements to safely control critical processes in real-time. Meaning that any successful FDI attack might eventually cause significant damage to the infected system.

In Profinet I/O systems using RT channels, the nodes normally exchange I/O process data through specific frames, namely Profinet Real-Time (PNIO-RT) (see figure 5.4). These frames have fixed structured features, such as packet size, packet type, frame identifier, data size, etc. The RT data field in each PNIO-RT frame contains bytes that represent either sensor readings or actuator values, depending on the transmission direction between the nodes, for example, from IO-Controllers to IO-Devices for actuator values and from IO-Devices to IO-Controllers for sensor readings.

The approach presented in this chapter involves manipulating specific RT data bytes

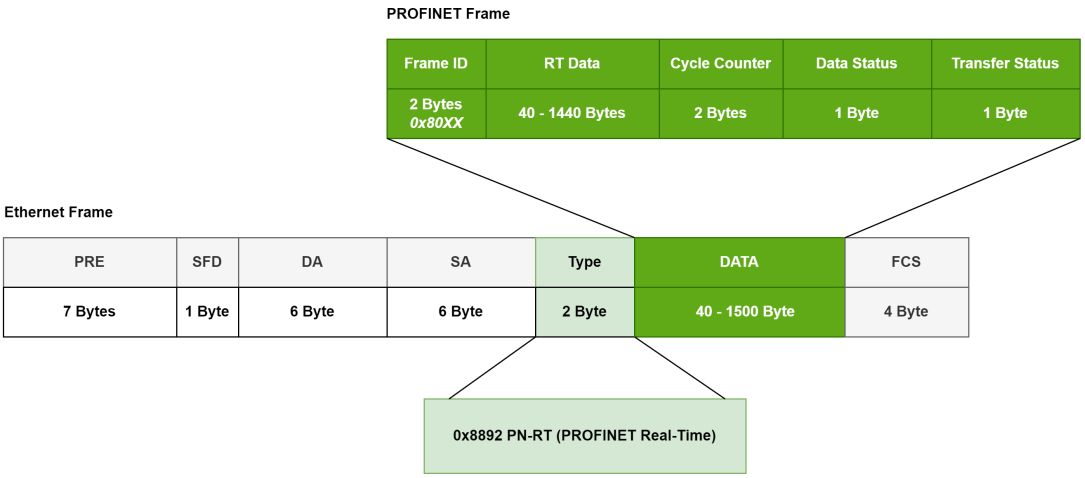


Figure 5.4: Ethernet Profinet message structure - Real-Time frame [4]

through the execution of an FDI attack scenario without any prior knowledge of the target system, the data exchanged between stations, the physical process, or even the system parameters. To achieve this goal, we introduce a new attack approach based on integrating a so-called I/O *Database* during the attack scenario. The newly presented I/O *Database* in this chapter is created by collecting network packets that contain actual sensor and actuator values from the target system before launching the FDI attack. Therefore, adversaries can interrupt, compare, and then replace the correct RT data bytes with false ones with the help of our I/O *Database*. Please note that our attack approach does not require attackers to map the I/O bytes to their readable version, as assumed in most previous works [63, 96–105]. In our opinion, this is not practical, as adversaries should not be familiar with the system they aim to attack. However, our new approach is more realistic and easy to implement, as we use a Python script based on Scapy to filter, extract, and store the PNIO-RT frames in the I/O *Database*.

5.2 Blind False Data Injection Approach

Figure 5.5 shows a high-level overview of the attack scenarios we perform to inject false data into the network traffic exchanged between the Profinet I/O nodes. To achieve fully-blind FDI attack scenarios, we need to first discover the network topology of the target Profinet I/O system and then collect PNIO-RT packets from the traffic to create our I/O *Database*, which eventually contains actual sensor and actuator values. However, these two steps are done prior to our injection attack, i.e., offline.

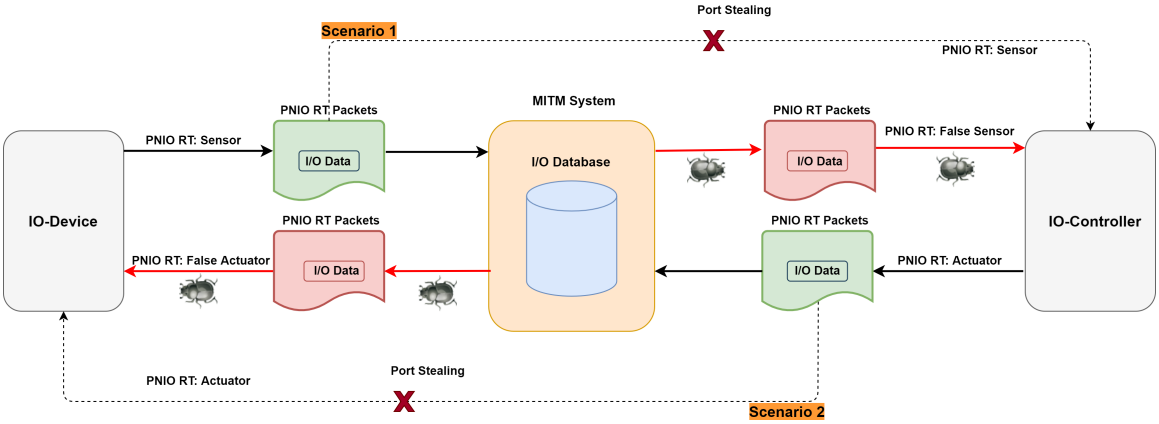


Figure 5.5: High-level overview of the FDI attack presented in this chapter: scenario 1 manipulating sensor data - the upper part of the figure; scenario 2 manipulating control commands – the lower part of the figure [4].

After collecting the required data from the target, we initiate our main attack by hijacking the ports on the nodes and sending false data packets to the victim devices using our I/O *Database* created earlier. In the following, we illustrate the two phases of our full attack chain in detail.

5.2.1 Pre-Attack Phase (Offline)

Here, the attacker aims to gain an overview of the network and the roles of the devices connected in the target system as a first step. Subsequently, the attacker focuses on sniffing and collecting data packets that the stations exchange cyclically using the Profinet I/O frames.

Discovering the target Network

For obtaining all the required information about the target system for our attack, we use our PN-IO scanner introduced earlier in Chapter 3 (Section 3.2.3) [1]. Our scanner sends a DCP identify request via multicast to the network. Each connected device returns its identifying parameters. Table 5.1 shows information gathered about all Profinet-enabled devices in our experimental setup, as depicted in Figure 5.10. This information includes critical data assisting the adversary in running further attacks, such as devices' names, MAC addresses, IP addresses, vendors, etc. As shown in Table 5.1, our scanner managed to find two devices

(nodes). The first node is an IO-Controller located at the IP address *192.168.0.1*, using the MAC address *00:1b:1b:23:fb:fe*. The second one is an IO-Device located at the IP address *192.168.0.2*, using the MAC address *20:87:56:05:06:15* to connect with the other station.

Table 5.1: Output of executing our PN-IO scanner

Parameter	Device 1	Device 2
MAC Address	00:1b:1b:23:fb:fe	20:87:56:05:06:15
Device ID	257	515
Device Role	IO-Controller	IO-Device
Device Vendor	b' S7-300	b' S7-300 CP
IP Address	192.168.0.1	192.168.0.2
Network Mask	255.255.255.0	255.255.255.0
Vendor ID	42	42

Sniffing and Collecting Data

After discovering and determining the role of each device in our target Profinet I/O system, the next step is to collect sensor and actuator values. To create an I/O *Database*, we first sniff and record the entire network stream between the Profinet I/O stations using sniffing network software, e.g., Wireshark. For our example application (see figure 5.10), the nodes exchange I/O process data through PNIO-RT frames, precisely class 1 (RTC1) frames, as shown in figure 5.6. These PNIO-RT frames have the same structure, which makes them easy to be recognized and extracted during ongoing network traffic.

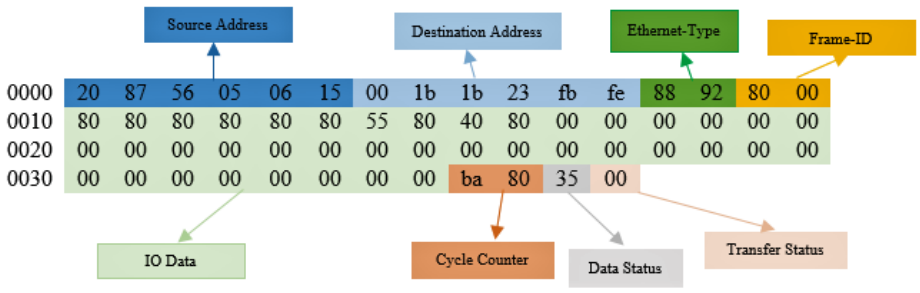


Figure 5.6: PNIO-RT class 1 frame structure [4]

For collecting a sufficient number of sensor measurements and actuator values, the sniffing process should last for a reasonably long period of time. For example, in this work, we sniff the network for approximately 30 minutes. Then, the captured stream is filtered to retrieve only the PNIO-RT frames using the unique packet type (*0x8892*) and frame ID (*0x80xx*) bytecode. See Figure 5.7.

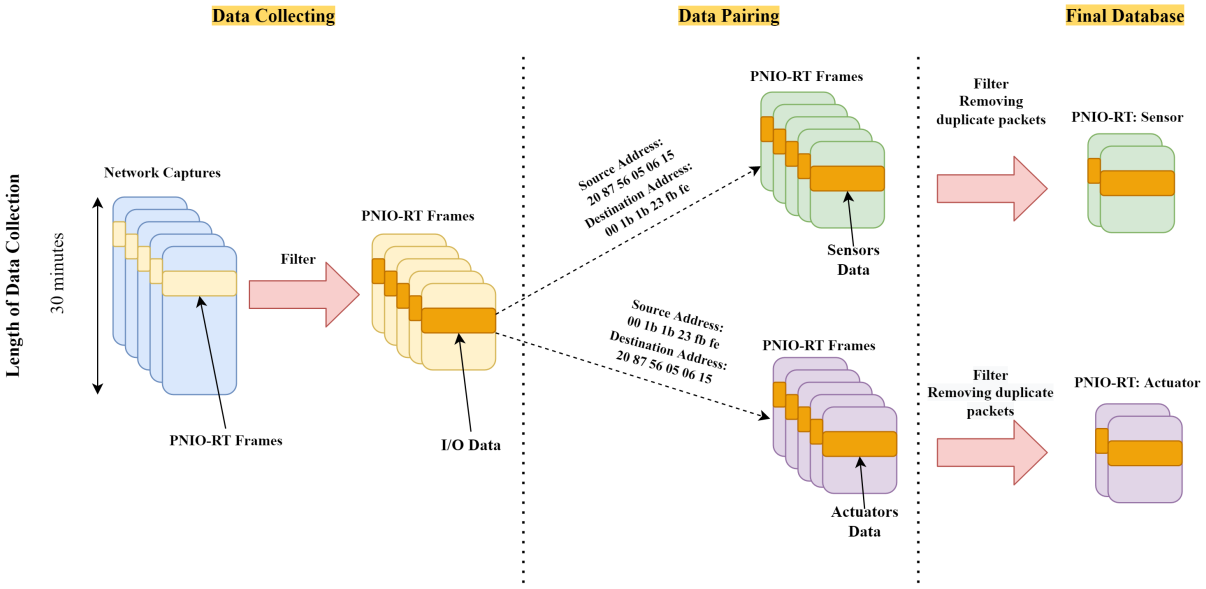


Figure 5.7: Scheme of creating I/O *Database* [4]

According to our prior knowledge of each device's role (from the previous step), we can check the source and destination MAC addresses of each PNIO-RT frame and group the captured frames into pairs of pcap files (sensor data and control command) accordingly.

To expedite the comparison process during our injection, we need to remove the duplicated packets received during the sniffing period, i.e., those containing similar I/O data for each pcap file, keeping only the packets that differ from each other. This is done by comparing the I/O data bytes of each PNIO-RT frame with those of the other frames using a bytes comparison tool (Burp Suite Comparator in our case). Please note that the I/O data bytes are located between byte numbers 17 and 56, as can be seen in figure 5.6.

For our example application presented in fig 5.10, we managed to create an I/O *Database* containing 7 sensor reading frames as inputs and 2 actuator value frames as outputs. It's worth mentioning that pairing the captured frames in our I/O *Database* into input and output pcap files helps us compare and replace the specific I/O data bytes with false ones online and win the strict race condition that Profinet I/O nodes must meet (as illustrated in the next subsection) before replying to the forged PNIO-RT packets to the network.

5.2.2 Attack Phase (Online)

In this phase, false data is injected into the network traffic based on our I/O *Database* approach. To achieve this, we first hijack the port from the victim, such as the IO-Controller or IO-Device, and then inject false packets that affect the process using our I/O *Database*.

Port Stealing Approach

Before pushing incorrect data into the network, the AR between the IO-Controller and IO-Device has to be intercepted. Technically, a typical industrial Ethernet switch controls and manages the binding of each MAC address to a certain switch port in an ARP mapping table. Once the MAC address at any port is changed because a new device has been added to the network, the switch updates its mapping table, and the old entry is removed. Therefore, we just need to flood the switch with forged gratuitous ARP packets, registering the attacker's MAC address in place of the victim host to achieve successful port stealing, as shown in figure 5.8.

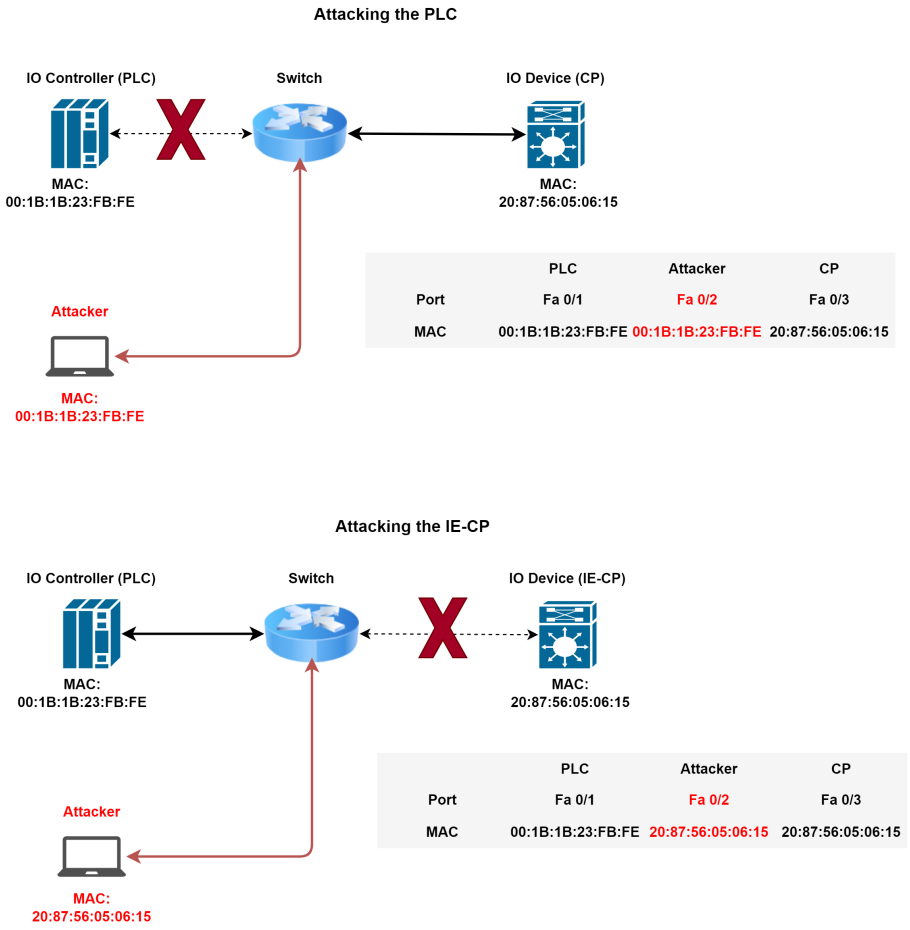


Figure 5.8: Data exchange configuration after ARP Poisoning attack: Scenario 1 stealing the port from the IO-Controller - the upper part of the figure; Scenario 2 stealing the port from the IO-Device - the lower part of the figure [4]

This technique is widely used in MitM attacks in traditional IT switched networks, where the switch assumes that the victim device is currently using another switch port and forwards

the packets to the new port [107]. This attack presents a critical challenge. The frequency of sending ARP packets to the victim must be sufficiently high. This means that if the target device sends ARP packets before the attacker, the switch constantly updates the binding of the port to the victim's MAC address back and forth. To overcome this issue, it is essential for the attacker to send ARP packets at a much higher frequency than the victim does. However, in our attack, a frequency of up to 1 ms was sufficient to prevent the switch from updating its mapping table. Now, if the packets of two devices are redirected through port stealing to the attacker, they only need to forward the packets accordingly to achieve a full MitM attack.

Injecting and Forwarding False Data

The final step is to replace the I/O data exchanged between the stations with false data included in one of the already recorded frames. This is done using our I/O *Database* approach explained in the former section. Algorithm 5.1 presents the main core of our attack script, which is employed to inject false data. The algorithm is implemented through a Python script and utilizes the third-party library Scapy.

As seen in Algorithm 5.1, after interrupting the AR between the stations in the previous step, the attacker listens and receives a PNIO-RT frame in the very next Profinet update cycle. The I/O data field of the received frame is then compared to the data fields of the already recorded frames existing in our I/O *Database*, considering only frames recorded for the same communication direction. This comparison process aims to find an appropriate frame with different I/O data bytes compared to the one captured and is repeated starting from the first frame in the *Database*. Once a different frame is found, its I/O data field, precisely from byte 17 to 56, will be used as a new I/O data field in our forged PNIO-RT packet, and the port-stealing attack is then stopped. Finally, the forged malicious packet that contains the false I/O data is forwarded to its final destination.

Please note that forwarding the crafted packet back to the network is challenging due to security features discussed below. First, the malicious packet cannot be directly forwarded due to the fact that each PNIO-RT frame has a cycle counter (see Figure 5.6). The 2-byte value that the cycle counter has is always read, and the number of missing packets between the consecutive cycles is set inside the TIA Portal software. To overcome this security challenge, the forged packet should always have the cycle counter value of the next expected PNIO-RT packet to be received at the final destination. However, this is easily solved, as the cycle counter values always differ by a constant number (e.g., in our system, the cycle counter number increases by 256 per cycle).

Algorithm 5.1: FDI Attack based on I/O *Database* using Scapy

```
Function: Inject (iface = eno1, src_prt)
packet = sniff(iface = eno1, time_out = cfg_sniff_time)
 save_pcap(packet, filter = "0x8892", frame_id = "0x8000", sniff.pcap)
for pkt in rdpcap(sniff.pcap) do
    src_mac = pkt[1:6], dest_mac = pkt[7:12], data = pkt[17:56], coun_cyc = pkt
     [57:58], data_status = pkt [59]
    if (src_mac ≠ plc_src_mac) then
        for p in rdpcap(inputs_pcapfile) do
            if data ≠ load_packet(p[17 : 56]) then
                fgd_data = load_packet(p[17 : 56]) break
            end
            p = p + 1
        end
    else
        for p in rdpcap (outputs_pcapfile) do
            if data ≠ load_packet(p[17 : 56]) then
                fgd_data = load_packet(p[17 : 56]) break
            end
            p = p + 1
        end
    end
    if data ≠ fgd_data then
        break
    end
    pkt = pkt + 1
end
fgd_pkt = padd_pkt (raw (PNIO_Real_Time (CycleCounter= 'coun_cyc + 256',
 fgd_data, data_status, len = '60')
stop_port_stealing ()
while time_slot() do
    sendp(fgd_pkt, iface=eno1)
end
End Function
```

Another challenge is that after stopping the port stealing, the attacker should always win the race condition by sending the malicious PNIO-RT frame to the victim before the correct data is sent from the original source. In Profinet I/O systems, the transmission interval (Profinet update time cycle) is divided into four phases, namely *Send Clock* (as shown in Figure 5.9). This parameter represents the frequency of exchanging data between the IO-Device and IO-Controller. In fact, the Profinet update time cycle results from the *Send Clock* x *Reduction Ratio*. Therefore, a *Send Clock* of 1 ms and a *Reduction Ratio* of

4 mean that I/O data is sent every 4 ms. However, the *Send Clock* is normally set from 2 to 512 ms and differs from one system to another based on their requirements. For most industrial Profinet I/O systems, the *Send Clock* is set at 128 ms to avoid extreme network traffic overloads. This means that each Profinet node gets updated every 0.5 s. Assuming that the *Send Clock* in our example application is set to 128 ms, the attacker needs to send his false data in less than 0.5 s to avoid updating the target Profinet node with the correct I/O data.

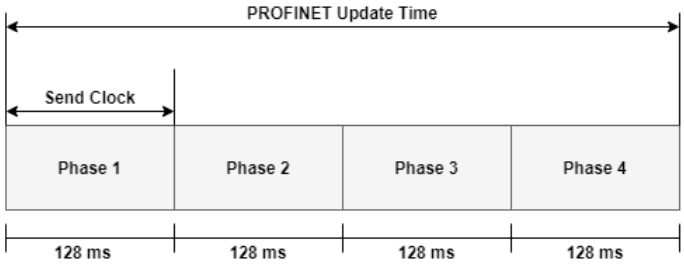


Figure 5.9: Profinet update time cycle [4]

5.3 Attack Implementation and Evaluation

In this section, we test our false data injection attack approach, as introduced in Section 5.2, on our Profinet I/O-based experimental setup presented in Figure 5.10. To this end, we conduct two attack scenarios. In the first one, the attacker aims to manipulate the sensor readings sent from the IO-Controller to the IO-Device, while the second scenario aims to alter the actuator values sent from the IO-Device to the IO-Controller.

5.3.1 Profinet I/O System Setup

To test our attack approach on real hardware, we utilized the same example application introduced in Chapter 3, namely the water control level system (refer to Figure 3.9). However, in this scenario, the network configuration differs, as the entire system is governed by a PLC (S7-315 PN/DP) connected to a remote I/O module through the Profinet I/O standard. Data, including sensor and actuator values, is exchanged cyclically over the network via Industrial Ethernet-Communication Processor (IE-CP).

The physical process is monitored by the TIA Portal software installed on the EWS, which is here a normal PC. However, in our example application, we have three stations. In the first one, an S7 315-2 PN/DP CPU was set as an IO-Controller. The second station is the IO-Device. It consists of an S7 315 DP CPU and an IE-CP 343-1 Lean. This station is connected

directly to an external I/O module, which is attached to the inputs and outputs hardware, i.e., the two pumps and the four digital sensors, as shown in Figure 5.10. Both stations exchange input and output data cyclically in real-time using the Profinet I/O protocol. The third station is the IO-Supervisor, which represents the engineering station in this example, and all three stations are connected to a 100 Megabits/Second (Mbit/s) industrial switch.

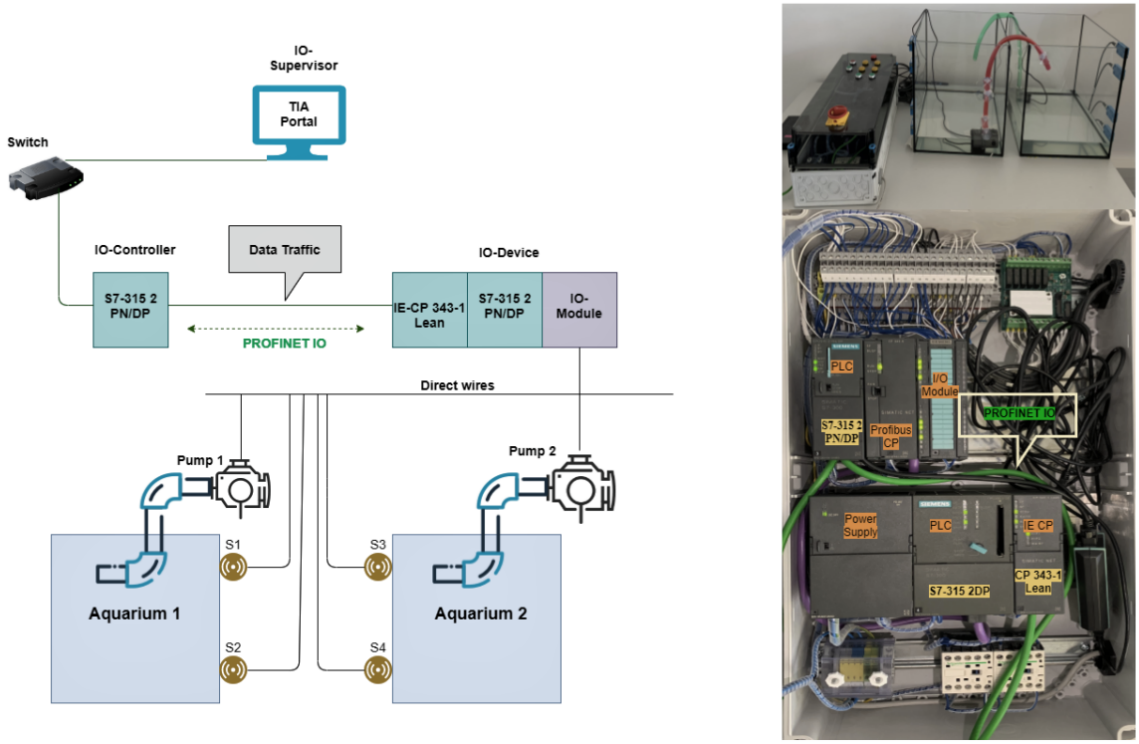


Figure 5.10: Profinet I/O Configuration – Experimental Set-up [4]

5.3.2 Injecting False Sensor Data to the IO-Controller

Figure 5.11 describes this scenario. First the port is stolen from the PLC. As a consequence, the PLC stops receiving any real-time data from the IE-CP and the data is redirected to the attacker. The packet received on the attacker's machine is then compared, and the I/O data bytes are replaced with false ones based on our I/O *Database*. The cycle counter value is then read and increased by 256 to match the expected counter value of the next PNIO-RT frame before the port-stealing attack is stopped. Finally, our forged packet is sent at the next Profinet update cycle, taking into account the race condition, i.e., in less than 0.5 s.

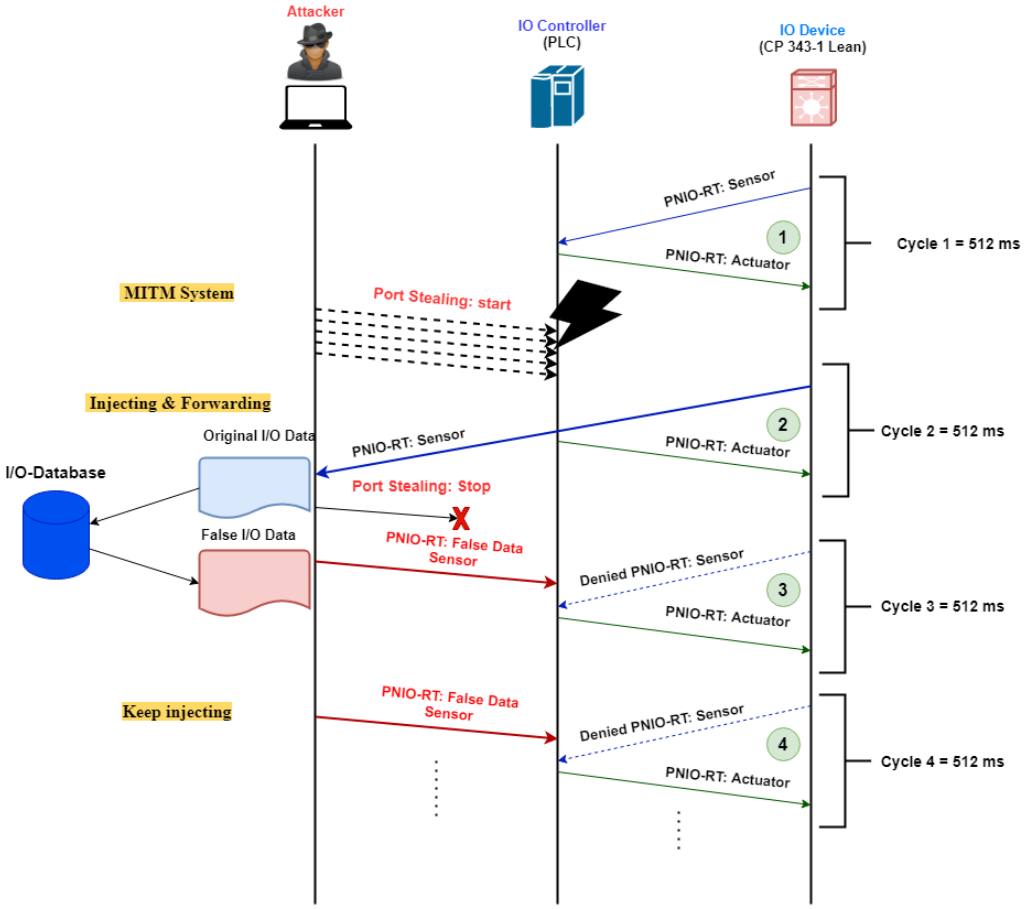


Figure 5.11: False Data Injection Attack against IO-Controller [4]

5.3.3 Injecting False Actuator Value to the IO-Device

Figure 5.12 depicts this scenario. Our goal here is to manipulate the control command sent from the PLC to the IE-CP. First, the port is hijacked from the IO-Device, diverting real-time data to the attacker's machine. The data bytes of the PNIO-RT packet received from the PLC are then compared to those in our I/O *Database* and replaced with false data bytes in the forged packet. The cycle counter value is increased by 256, and the malicious packet is then sent back to the IO-Device after the port hijacking is reversed.

As a consequence of executing our injection attack chain against the Profinet I/O setup shown in figure 5.10, we successfully managed to trick the PLC by reading false sensor measurements (in the first scenario) and the IE-CP by receiving false actuator values (in the second scenario). Our attack approach leads the tested Profinet I/O system in both scenarios to operate the physical process incorrectly, depending on the false data frames chosen from our I/O *Database*, keeping the infected system running at a certain operational state as long

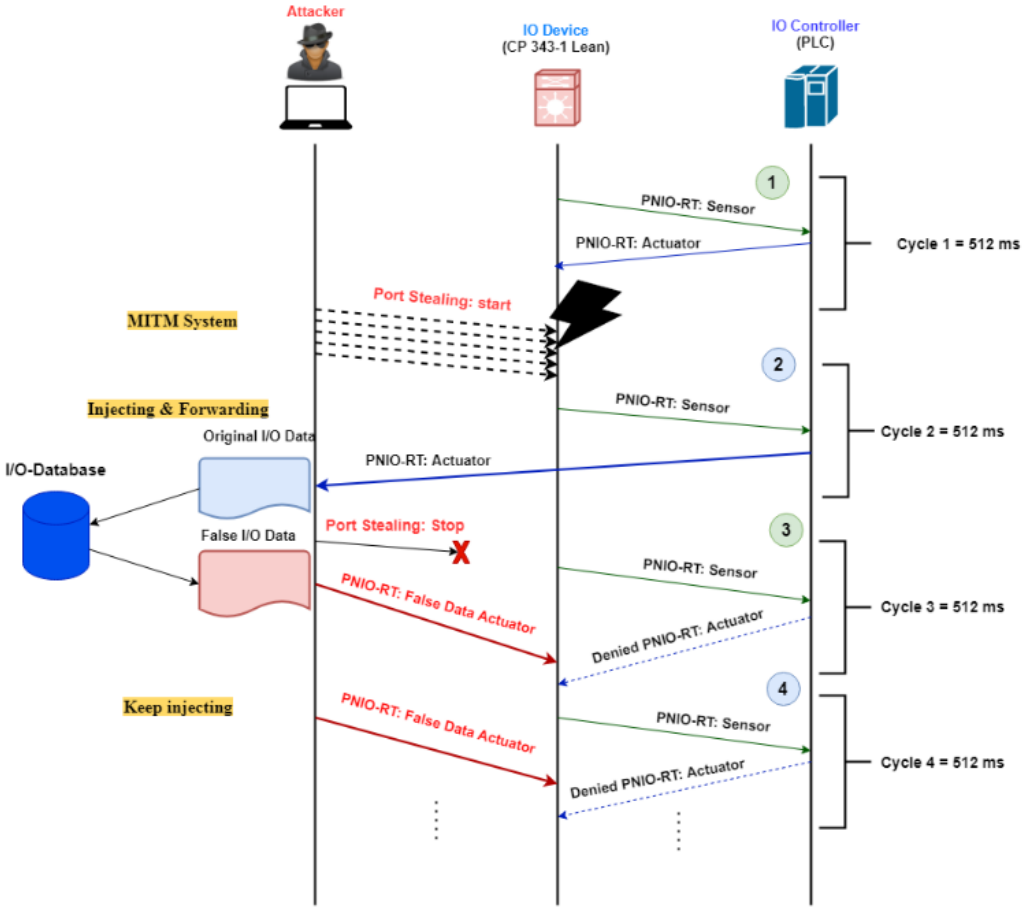


Figure 5.12: False Data Injection Attack against IO-Device [4]

as the injection attack lasts. For example, the water may exceed the limits, causing water overflow due to reading or acting on false data values.

However, to improve the success probability of such an attack, we need to continually deliver our forged data to the PLC/IE-CP before the original data. Thus, we send each false data for more than one update cycle, as seen in figures 5.11 and 5.12. Furthermore, if the attacker keeps stealing the port for a long period, they will disturb the AR communication between the PLC and the IE-CP, and our attack turns out to be a DoS attack, not an FDI attack.

5.4 Mitigation and Security Recommendations

Different security solutions may be implemented to prevent or, at least, detect our attack scenario. One solution is presented in [108]. The authors suggested an Intrusion Detection

System (IDS) that uses anomaly detection based on connecting the switch mirror port to the IDS to monitor all network packets transmitted through the switch. In a training phase, deviations can be identified as anomalies. The authors of [108] also introduced a signature-based IDS using Snort [109]. Rules describe undesired communications and trigger an alarm if suspicious packets are detected. However, both solutions presented in [108] are designed to detect the ongoing attack after it was conducted successfully.

To avoid severe damages an FDI attack might cause to Profinet I/O systems, we highly recommend improving the isolation from other networks [110], combined with standard security practices [111]. However, in our opinion, the best solution to make the industrial network more resistant to FDI attacks is when different prevention mechanisms are in place, e.g., a DMZ and network segmentation to improve attack prevention, and a layered defense-in-depth strategy to further improve the detection of successful malicious injections.

5.5 Summary

Due to the increasing industrial connectivity demands, the use of Ethernet in automation processes has become common. However, security threats have also been rising. The main reasons for this are, first, the easier access for potential adversaries, and second, the already disclosed vulnerabilities existing in the Ethernet standard. In this chapter, we present FDI attack scenarios on the most common industrial communication protocol used in automation processes, i.e., Profinet I/O. Our attack had a severe impact on the affected Profinet I/O setup used in this work. It allows adversaries to interrupt the AR between the IO-Controller and IO-Device(s) connected over Ethernet. The method of injecting the IO-Controller and IO-Device is developed by using a pre-created I/O *Database* storing pairs of sensor readings and their corresponding actuator values. Therefore, the attack could sniff the exchanged packets between the stations and replace the valid data with malicious one to cause physical damage in the target system. The result of our comprehensive evaluation showed that the attack can be successfully conducted on a real Profinet I/O setup. We found that the IO-Controller, i.e., PLC, was tricked by reading false sensor readings in the first scenario, and the IO-Device by executing false control commands in the second scenario. This led to operating the physical process controlled by the infected devices inappropriately. The affected system kept operating at a certain state as long as the attacker keeps sending false data, depending on the data packet chosen from our I/O *Database*. Since Profinet I/O is widely used in critical infrastructures, this attack scenario may cause considerable damages in the target systems.

CHAPTER 6

Chapter 6: Summary and Future Work

Contents of this chapter are as follows:

Parts of this chapter have been already published in the following papers:

- W. Alsabbagh and P. Langendörfer, "A Flashback on Control Logic Injection Attacks against Programmable Logic Controllers," Automation. 2022; 3(4):579-595. https://doi.org/10.3390/ automation3040029 [8].
- W. Alsabbagh and P. Langendörfer, "Security of Programmable Logic Controllers and Related Systems: Today and Tomorrow," in IEEE Open Journal of the Industrial Electronics Society, vol. 4, pp. 659-693, 2023, doi: 10.1109/OJIES.2023.3335976 [13].

In this chapter, a summary of this work is provided. This is followed by an outlook on open and further research.

6.1 Summary

Since Stuxnet [17] occurred in 2010, a lot of research investigating security issues in PLCs and suggesting various mitigation solutions has been conducted. Nevertheless, more than a decade later, many open research questions in the area of PLCs security remain unanswered. In this work, a part of these research questions was addressed, and current problems of PLCs with regard to the security of such critical devices have been analyzed. Furthermore, different successful attack scenarios were conducted to show the research community and engineers that our PLCs-based systems are still not completely secure, even after industrial vendors have improved their PLCs by integrating advanced integrity methods and developing communication protocols. To achieve realistic results, we conducted all our attack approaches on real hardware in industrial settings using S7 PLCs from different SIMATIC families. Our selection is based on the fact that Siemens is the leading automation manufacturer, and its

devices are employed in millions of systems all over the world. Therefore, their security reflects the big picture of ICSs security as a whole. In this work, we analyzed the security issues and vulnerabilities in both cryptographically and non-cryptographically protected PLCs and showed the consequences if successful attacks are conducted against exposed devices.

First, we investigated the authentication mechanism used in non-cryptographically protected PLCs (e.g., S7-300) and found that the authentication protocol utilizes a weak encryption algorithm. Thus, such PLCs are vulnerable to brute-force attacks. Furthermore, our experiments showed that the PLCs utilize a key space as small as 8-byte characters and use the same secret key for multiple sessions. Due to these vulnerabilities, we managed to perform several attacks such as retrieving the password in plain-text format, removing the password, setting a new password, and updating an old password with a new one. After that, we introduced a stealthy control logic injection attack that injects a victim PLC with a modified program, keeping the injection undetected by the ICS operator. To this end, we implemented a fake PLC approach that hinders the operator from uploading the actual infected code from the real PLC by redirecting the communication to the fake PLC. As part of the full attack chain, we also introduced decompiler and compiler approaches that convert control logic programs to their source code format (here STL) and vice versa. Eventually, we presented some security recommendations to mitigate the impact of such a threat.

In the second part, the security of cryptographically protected PLCs was investigated. As a case study, we analyzed the very new integrity check method of the S7-1500 PLCs and their S7CommPlus V3 use. Our investigation showed that such a method is still vulnerable, and an attacker can manipulate the context of the exchanged messages in different ways. We also disclosed many vulnerabilities. Based on this, we performed a control logic injection attack scenario that aims at disrupting the physical process controlled by PLCs. The approach we introduced in this work allows adversaries to cause damages when they are not connected either to the target or its network at the point zero for the attack. To conduct real-world experiments, we performed our attack approach against a Fischertechnik training factory based on the modernist model of S7 PLCs, i.e., S7-1500 CPU that uses the latest and most secure S7 protocol, i.e., S7CommPlus V3. In the end, we suggested a few security recommendations that help in preventing/detecting such an attack.

In the third part, a false data injection attack against Profinet I/O systems was introduced. Our attack scenario presented in this work showed that adversaries without any prior knowledge of the target system or its network are able to manipulate the sensor readings and actuator values, causing serious damages. As part of the attack, we integrated the typical FDI scenario with a new I/O Database approach that contains pairs of real sensor readings and their corresponding actuator values. For a realistic scenario, we configured an experimental

setup (based on S7-300 PLCs) to exchange data cyclically over Profinet I/O messages. Our results showed that both Profinet I/O nodes were tricked, i.e., the IO-Controller by reading false sensor readings, and the IO-Device by executing false control commands. Eventually, the physical process controlled by the infected devices ran incorrectly, and the system remained operating at a certain state as long as the attacker keeps sending false data, depending on the data packet chosen from the I/O Database. Finally, some mitigation solutions and security practices were introduced.

On the whole, this dissertation identified several security problems and vulnerabilities in the area of PLCs and their related protocols' security, which eventually led to a new awareness of severe attacks in industrial systems. Furthermore, we suggested possible mitigation solutions to increase the security level of PLCs, contributing significantly to securing millions of running PLCs all over the world. Overall, the presented results in this dissertation assist the state of research in the area of PLC security..

6.2 Future Work

Further industrial plants are increasingly prone to more and more threats due to the increased demands in network connectivity. To withstand this, more research efforts and innovative security solutions will be required. In the following, we recommend some additional research directions for the research community.

6.2.1 Source Code Injection Attacks

In Chapter 3, we discussed a malicious control logic injection attack at the source code level. Our investigations showed that an attacker could conduct a stealthy injection attack by implementing a fake PLC approach. Therefore, we recommend further research in terms of comparing the original program behavior with unsafe behaviors and performing automatic cleaning of the program in case unsafe behavior is detected. Furthermore, continuous measuring of execution times may also be a promising solution, as it hints at manipulations if the execution cycle time is larger. Another solution might be updating all PLCs periodically with the original code, i.e., overwriting malicious ones. We believe such solutions might overcome stealthy injections where ICS operators are not able to detect injections in traditional security practices.

6.2.2 Bytecode Injection Attacks

The research community still lacks open-source libraries, similar to [118], aimed at disassembling PLC programs from different vendors into high-level source code programs and vice versa. Such libraries would help security researchers better understand how attackers can manipulate PLC programs in low-level formats, thus formalizing a detection plant model that checks the compiled code sent from the EWS with the one received at the PLC side. We also recommend further studies on reverse engineering function blocks, parameters lists, timers, and counters that each type of PLC supports. We believe that such open-source libraries and research studies can provide a more resistant PLC-based environment against malicious attacks at the bytecode level.

6.2.3 False Data Injection Attacks

Our investigations showed that FDI attacks are widely adopted by attackers. This means that they are still capable of conducting FDI attacks even with no prior knowledge about the system they target. Thus, we recommend more future research in the direction of consistently checking program behaviors. A possible solution might be formalizing a plant model to detect FDI attacks by considering instruments on the input and output variables of the programs and comparing the values with those from the plant model. Another possible solution might be implementing a plausibility check mechanism that verifies measured values and reveals any obvious inaccuracy in the sensor readings. For instance, in nuclear plants, a sudden spike in temperature might indicate manipulation. We believe such a solution would mitigate the impact of FDI attacks and accurately monitor program behaviors.

6.2.4 Lightweight Run-Time Formal Verification

PLCs are embedded devices, meaning that they have only limited memory to store the control logic blocks and any additional security solutions. Furthermore, PLCs are increasingly employed in SCADA systems and different implementations that transfer data over possibly insecure networks. For all that, an appropriate security solution could be to introduce lightweight run-time formal verification that shares memory with the PLC. The proposed verification should have authorized access to the inputs and outputs of the PLC and reveal any abnormal changes. Moreover, the control logic blocks should also be concurrently integrated with the verification model and scanned each time the user attempts to upload or download new control logic.

6.2.5 Secure Communication Protocols

The vulnerabilities existing in most PLCs are due to flaws and defects in their communication protocols. Such vulnerabilities include plain-text transmission, lack of authorization and authentication mechanisms, and the absence of an integrity check algorithm. Our investigations showed that PLCs are prone to various kinds of attacks such as MitM, replay, and injection attacks. As a consequence, manufacturers have updated their industrial communication protocols, integrating more security features, e.g., encryption, anti-replay, authorization, integrity, etc. However, to a large extent, the newly variant protocol versions are not suitable for all employed PLCs in current systems. For instance, the latest version of the S7CommPlus protocol is not supported by old S7 PLC families, e.g., S7-300 and S7-400. However, our analyses showed that even advanced protocol versions are not fully secure and can still be manipulated by skilled adversaries. For example, the newest S7CommPlus protocol uses certain sophisticated algorithms that are supposed to effectively prevent replay attacks from being performed against modern S7 PLCs and their TIA Portal software. But we proved that adversaries are still capable of launching protocol-oriented attacks. Therefore, further research in terms of improving industrial communication protocols or designing secure ones should be conducted.

Overall, our research examples, presented in this work, clearly show that the research in the PLC security field is still far from complete and that there is a necessity to keep researching in this area.

APPENDIX A

Parameters used in different search engines

In this appendix, we list all the parameters as well as the URL that we used in the four research engines *Shodan*, *ZoomEye*, *Ditecting* and *Censys*.

The queries that we used to discover Internet-facing PLCs with the help of *Shodan* search engine are presented in Table A.1.

Table A.1: Parameters used for *Shodan* search engine

Protocol	Query	URL
Siemens S7	Port:102	https://www.shodan.io/search?query=port%3A502
Modbus	Port:502	https://www.shodan.io/search?query=port%3A502
BACnet	Port:47808	https://www.shodan.io/search?query=port%3A47808
DNP3	Port:20000 source address	https://www.shodan.io/search?query=port%3A20000+source+address
Ethernet/IP	Port:44818	https://www.shodan.io/search?query=port%3A44818
Niagara Fox	Port:1911, 4911	https://www.shodan.io/search?query=port%3A1911%2C4911+product%3ANiagara
Codesys	Port:2455	https://www.shodan.io/search?query=port%3A2455+operating+system
Phonix/PCWorx	Port:1962	https://www.shodan.io/search?query=port%3A1962+PLC

The queries that we used to discover Internet-facing PLCs with the help of *ZoomEye* search engine are presented in Table A.2.

Table A.2: Search engine parameters for *ZoomEye*

Protocol	Query	URL
Siemens S7	+port: 102	https://www.zoomeye.org/searchResult?q=%2Bport%3A%22102%22&t=all&is_dork=0
Modbus	+port: 502	https://www.zoomeye.org/searchResult?q=%2Bport%3A%22502%22&t=all&is_dork=0
BACnet	+port: 47808	https://www.zoomeye.org/searchResult?q=port:47808
DNP3	+port: 20000 +service: "dnp"	https://www.zoomeye.org/searchResult?q=port:20000%20%2Bservice:%22dnp%22&t=all&is_dork=0
Ethernet/IP	+port: 44818	https://www.zoomeye.org/searchResult?q=port:44818
Niagara Fox	+port: 1911	https://www.zoomeye.org/searchResult?q=port:1911
Codesys	+service: "CoDeSyS"	https://www.zoomeye.org/searchResult?q=%2Bservice%3A"CoDeSyS"&t=all&is_dork=0
Phonix/PCWorx	+port: 1962	https://www.zoomeye.org/searchResult?q=port:1962

The queries that we used to discover Internet-facing PLCs with the help of *Ditecting* search engine are presented in Table A.3.

Table A.3: Search engine parameters for *Ditecting*

Protocol	Query	URL
Siemens S7	service: Siemens S7	https://www.ditecting.com/index.php/home/Result/index.html?query=service%3ASiemens%20s7?query=service%3ASiemens%20s7
Modbus	service: Modbus	https://www.ditecting.com/index.php/home/Result/index.html?query=service%3AModbus?query=service%3AModbus
BACnet	service: BACnet	https://www.ditecting.com/index.php/home/Result/index.html?query=service%3ABACnet?query=service%3ABACnet
DNP3	service: DNP3	https://www.ditecting.com/index.php/home/Result/index.html?query=service%3ADNP3?query=service%3ADNP3
Ethernet/IP	service: EtherNet/IP	https://www.ditecting.com/index.php/home/Result/index.html?query=service%3AEtherNet/IP?query=service%3AEtherNet/IP
Niagara Fox	port: 1911	https://www.ditecting.com/index.php/home/Result/index.html?query=port%3A1911?query=port%3A1911
Codesys	port: 2455	https://www.ditecting.com/index.php/home/Result/index.html?query=port%3A2455?query=port%3A2455
Phonix/PCWorx	service: PCWorx	https://www.ditecting.com/index.php/home/Result/index.html?query=service%3APCWorx?query=service%3APCWorx

The queries that we used to discover Internet-facing PLCs with the help of *Censys* search engine are presented in Table A.4.

Table A.4: Search engine parameters for *Censys*

Protocol	Query	URL
Siemens S7	Protocols: "102/s7"	https://censys.io/ipv4?q=protocols%3A+%22102%2Fs7%22
Modbus	Protocols: "502/modbus"	https://censys.io/ipv4?q=protocols%3A+"502%2Fmodbus"2
BACnet	Protocols: "47808/bacnet"	https://censys.io/ipv4?q=protocols%3A+%2247808%2Fbacnet%22
DNP3	Protocols: "20000/dnp3"	https://censys.io/ipv4?q=protocols%3A+%2220000%2Fdnp3%22
Ethernet/IP	Not Available	Not Available
Niagara Fox	Protocols: "1911/fox"	https://censys.io/ipv4?q=protocols%3A+%221911%2Ffox%22
Codesys	Not Available	Not Available
Phonix/PCWorx	Not Available	Not Available

APPENDIX B

Technical Details of the Communication Process in S7CommPlus Protocol

B.1 S7 *Request* Message:

The first message is a '*Hello*' message sent from the TIA Portal to the PLC in order to initialize a new session.

B.2 S7 *Challenge* Message

The PLC communicates with the TIA Portal by providing information such as its firmware version and a specific set of 20 bytes known as the *ServerSessionChallenge*. The firmware version of the PLC determines the elliptic-curve public key pair to be employed in the subsequent key exchange process. Once the TIA Portal receives the second message from the PLC, it initiates a derivation algorithm (designated as algorithm B.1) to randomly generate a Key Derivation Key (KDK). This selected KDK is then utilized in a fingerprinting function (f) to produce a 32-byte *Integrity Part* from the received *Challenge*.

Algorithm B.1: *Integrity Part Derivation*

Input: *ServerSessionChallenge* ▷[20*bytes*]

$challenge = ServerSessionChallenge$ [2:18]

$KDK = prng(24)$

$Integrity\ Part = HMAC - SHA - 256_{KDK}(f(challenge,8)||challenge)$ [:24]

Return: *KDK, Integrity Part*

B.3 S7 *Response* Message:

The designated KDK is sent from the TIA Portal to the PLC through the *Response* message. The TIA Portal employs the public-key scheme in this process. The *Response* message is intricate and includes, among other elements, a notable structure referred to as *SecurityKeyEncryptedKey*, illustrated in Figure B.1. The length of this structure is 180 bytes,

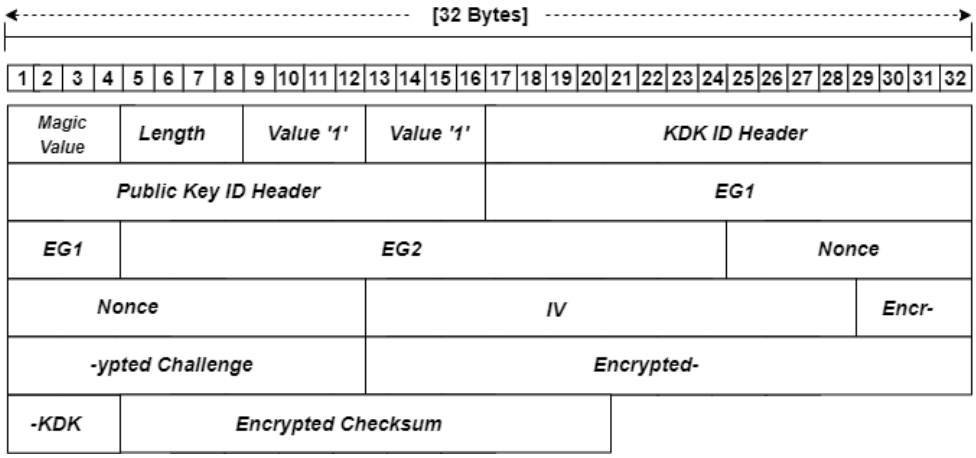


Figure B.1: The [180 Bytes] *SecurityKeyEncryptedKey* Structure in the third message.

always commencing with a fixed 4-byte magic value, denoted as *0xFEE1DEAD*. To initialize the cryptographic elements, the TIA Portal triggers algorithm B.2, which operates in the following manner:

Algorithm B.2: *Initialization*

$R = prng(24)$
$PreKey = EC\text{-}MAP(R)$ ▷$[60bytes]$
$KEK, CEK, CS = KDF(PreKey)$
$LUT[4][256] = TB\text{-}HMAC\text{-}INIT(CS)$
Return: *Prekey, LUT, KEK, CEK*

The initial step involves generating a 24-byte quantity, denoted as (R), through a random process. Subsequently, this R is mapped to the domain of the elliptic curve, resulting in a curve point referred to as *PreKey*. The length of the resulting *PreKey* is 60 bytes. Following this, the TIA Portal proceeds to pad the *PreKey* with the *Response* message using an elliptic-curve El-Gamal public key exchange. Algorithm B.3 illustrates the procedure that the TIA Portal employs to encrypt the *PreKey*.

- At the beginning, the TIA Portal randomly selects a 20-byte sequence known as the *Nonce*. These 20 bytes are transmitted unchanged within the S7 *Response* message and serve as a masking factor for the elliptic-curve calculation. Subsequently, another set of 20 bytes is chosen at random and stored in an array named y. Once y is generated, the algorithm computes yG using the base point G and places its x coordinate into a designated *EG2* field within the S7 *Response* message (refer to Figure B.1).

In the ensuing step, the TIA Portal employs Q (the public key) to compute yQ, and subsequently utilizes the resulting yQ to encrypt the *PreKey* point, i.e., $(yQ + PreKey)$. The

Algorithm B.3: *Elliptic-Curve El-Gamal Key Exchange*

Procedure *EC-ECN(PreKey, Q)*
point = ∞
Nonce = *prng*(20)
while *point* > 0 **do**
    y = *prng*(20)
    point = *EC-MULT*(G,y,*Nonce*)
    EG2 = *pointx*
end
s = *EC-MULT*(Q,y,*Nonce*)
EG1 = *EC-ADD*(s,*PreKey*)x
return *EG1*, *EG2*, *Nonce*
End Procedure

result of this encryption process, referred to as *EG1*, is then placed in the x-coordinate field of the resulting point within the S7 *Response* message.

- In the subsequent stage, the TIA Portal employs the *PreKey* to generate a Key Derivation Function (KDF) and derive three sets of 16-byte quantities, as illustrated in the third line of algorithm B.2. These three quantities are designated as follows:

- Key Encryption Key (KEK): Signifying an AES key, the TIA Portal utilizes it to encrypt both the KDK and the *challenge*.

- Checksum Encryption Key (CEK): Representing another AES key, it is employed by the TIA Portal for encrypting the checksum.

- Checksum Seed (CS): Used by the TIA Portal to generate pseudo-random bytes (4096 bytes). These bytes are organized into four groups of 256 bytes each, referred to as Look Up Table (LUT) in the fourth line of algorithm B.2. The LUT is employed in conjunction with the KDK and the *challenge* to compute the encrypted checksum (the final 16 bytes located in the *SecurityKeyEncryptedKey*, as depicted in Figure B.1).

B.3.1 KDK Key ID Header & Public Key ID Header

The S7 *Response* message, specifically the 180-byte *SecurityKeyEncryptedKey* structure, incorporates two 16-byte header fields situated between byte numbers 17 and 48, as illustrated in figure B.1. The KDK utilized by the TIA Portal to generate the *Integrity Part* is identified through the KDK ID header. Simultaneously, the ID header for the public key Q, utilized in the encryption process of the *PreKey*, is identified by the public key ID header. Both headers encompass 8-byte key fingerprints (SHA-256 hashes) along with additional flags, as

depicted in algorithm B.4. Our investigations have revealed that the public key ID header is stored in the TIA Portal memory, specifically within the key-chain data located in the *DLL* file (*OMSp_core_managed.dll*). This implies that the public key ID header also plays a crucial role in selecting the public key that the TIA Portal software utilizes, based on the PLC model and firmware version.

Algorithm B.4: *Deriving Key ID*

Procedure *DerKeyID(Key)*
return *SHA-256(Key*[:24] || "*Derive*")[:8])
End Procedure

The PLC utilizes the KDK key ID header to scrutinize the authenticity of the KDK it derives, as well as the authenticity of the 16-byte encrypted checksum field.

B.3.2 Encrypted *Challenge* & Encrypted KDK

The TIA Portal utilizes the "*AES-CTR*(*challenge*//*KDK*)" algorithm to encrypt both the *challenge* and the KDK. For this purpose, the TIA Portal randomly selects a 16-byte value, referred to as the Initialization Vector (IV) (Initialization Vector), which is employed in the AES Counter Mode (CTR) algorithm[1]. This IV is transmitted in plaintext within the *Response* message, as depicted in figure B.1. The outcomes of the encryption process are stored in the corresponding positions of the S7 *Response* message. As illustrated in figure B.1, the encrypted challenge occupies byte numbers 124 to 140, while the encrypted KDK resides between byte numbers 141 and 164.

B.3.3 Encrypted Checksum

The *TIA* Portal employs authenticated encryption for both the encrypted *challenge* and encrypted KDK by calculating the final checksum. In this process, the TIA Portal utilizes the AES algorithm with assistance from the CEK. The resulting 16-byte checksum is positioned at the conclusion of the *SecurityKeyEncryptedKey* structure within the S7 *Response* message (refer to Figure B.1). Algorithm B.5 outlines the steps involved in the computation of the final checksum.

B.3.4 Decryption of the S7 *Response* Message in the PLC

When the PLC receives the S7 *Response* message from the TIA Portal, it first decrypts the *PreKey* by using its private key. Afterwards, the PLC derives each of KEK, CEK and the

[1]Refer to https://www.rfc-editor.org/rfc/rfc3686 for more information on AES CTR algorithm

Algorithm B.5: *Computing the final Checksum*

Procedure *Checksum(ENC, CEK, LUT[4][256])*
current_checksum = 0
for *block in ENC* **do**
  | *current_checksum* = *TB-HASH*(*current_checksum* ⊕(*block*, *LUT*))
end
current_checksum[12] = *current_checksum*[12] ⊕40
final_checksum = *TB-HASH(current_checksum)*, *LUT*
return AES-ECB(*final_checksum*)
End procedure

CS hashes from the delivered *PreKey*. Once the hashes are derived, the PLC extracts both KDK and *challenge* with the help of the KEK, and finally verifies the TIA Portal software and the integrity of the KDK.

B.4 'Ok' Message:

If the validation process is successful, the PLC returns '*OK*' to the TIA Portal and from this point on, all the S7 messages transmitted in the current communication session are protected with the calculated *Integrity Part*.

APPENDIX C

Acronyms

ACK	Acknowledgment
AES	Advanced Encryption Standard
API	Application Programming Interface
AR	Application Relationship
ARP	Address Resolution Protocol
CaFDI	Controller-aware False Data Injection
CC	Command and Control
CEK	Checksum Encryption Key
CIA	Confidentiality, Integrity and Availability
CIs	Critical Infrastructures
CP	Communication Processor
CPs	Communication Processors
CPU	Central Processing Unit
CR	Communication Relation
CS	Checksum Seed
CTR	Counter Mode
CT	Communication Template
CVE	Common Vulnerabilities and Exposures
DB	Data Block
DBs	Data Blocks
DB1	Data Block 1
DCP	Discovery and basic Configuration Protocol
DES	Data Encryption Standard

DMZ	Demilitarized Zone
DNP3	Distributed Network Protocol 3
DoS	Denial of Service
DPI	Deep Packet Inspection
EEPROM	Electrically Erasable Programmable Read-only Memory
ERP	Enterprise Resource Planning
EWS	Engineering Work Station
FBD	Function Block Diagram
FBs	Function Blocks
FCs	Functions
FDI	False Data Injection
FIN	Finish
HBW	High-Bay Warehouse
HEX	Hexadecimal
HMAC	Hash-based Message Authentication Code
HMAC-SHA-256	Hashed-based Message Authentication Code-Secure Hash Algorithm-256
HMI	Human Machine Interface
Hz	Hertz
ICS	Industrial Control System
ICSs	Industrial Control Systems
ICS-CERT	Industrial Control System Computer Emergency Response Team
ID	Identifier
IDEs	Integrated Development Environments
IDS	Intrusion Detection System
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IE-CP	Industrial Ethernet-Communication Processor
IKSen	Industriellen Kontrollsystemen

IL	Instruction List
IoT	Internet of Things
IP	Internet Protocol
IQR	Interquartile Range
IRT	Isochronous Real-Time
IT	Information Technology
IV	Initialization Vector
I/O	Input/Output
JMP	Jump
JTAG	Joint Test Action Group
KB	Kilo Byte
KDF	Key Derivation Function
KDK	Key Derivation Key
KEK	Key Encryption Key
KIen	kritischen Infrastrukturen
LD	Ladder Diagram
LLB	Ladder Logic Bombs
LUT	Look Up Table
l/h	liter/hour
m	meter
MaC	Message Authentication Code
MAC	Media Access Control
Mbit/s	Megabits/Second
MC7	Machine Code 7
MD5	Message Digest 5
MES	Manufacturing Execution System
MG	Message Generator
MitM	Man-in-the-Middle
mm	millimeter

MPI	Multi-Point Interface
MPO	Multi-Processing Station with Klin
MQTT	Message Queuing Telemetry Transport
ms	milliseconds
MW	Megawatt
NAS	Network Attached Storage
NDAAO	Non-Deterministic Autonomous Automation with Output
NFC	Near Field Communication
NRT	Non Real-Time
NVD	National Vulnerability Database
OB	Organization Block
OB1	Organization Block 1
OB10	Organization Block 10
OS	Operating System
PC	Personal Computer
PCCC	Programmable Controller Communication Commands
PDU	Protocol Data Unit
PID	Proportional-Integral-Derivative
PLC	Programmable Logic Controller
PLCs	Programmable Logic Controllers
PNIO-RT	Profinet Real-Time
PN-DCP	Profinet Discovery and basic Configuration Protocol
PN-IO	Profinet-Input Output
PT	Pulse Timer
Q1	First Quartile
Q2	Median
Q3	hird Quartile
RAM	Random Access Memory
RAT	Remote Access Trojan

RC4	Rivest Cipher 4
RD	Record Data
RFID	Radio Frequency IDentification
RT	Real-Time
RTN	Return
s	seconds
SCADA	Supervisory Control and Data Acquisition
SCL	Structured Control Language
SDB	System Data Block
SDBs	System Data Blocks
SDB0	System Data Block 0
SFBs	System Function Blocks
SFC	Sequential Function Chart
SFCs	System Function Calls
SHA	Secure Hash Algorithm
SHA-256	Secure Hash Algorithm-256
SLD	Sorting Line with Color- Detection
SNMP	Simple Network Management Protocol
SPS	Speicherprogrammierbare Steuerung
SPSen	Speicherprogrammierbare Steuerungen
SSC	Station with Surveillance Camera
ST	Structured Text
STL	Statement List
S7Comm	S7 Communication
S7CommPlus	S7 Communication Plus
S7-ACK	S7-Acknowledgment
TCP	Transmission Control Protocol
TIA	Totally Integrated Automation
TMV	Temporary Variable

ToD	Time of Day
TOF	Timer Off
TON	Timer On
URL	Uniform Resource Identifier
USB	Universal Serial Bus
U.S.	United State
V	Volts
VGR	Vacuum Suction Grippers

Bibliography

[1] W. Alsabbagh and P. Langendörfer, "A Remote Attack Tool Against Siemens S7-300 Controllers: A Practical Report," In: Jasperneite, J., Lohweg, V. (eds) Kommunikation und Bildverarbeitung in der Automation. Technologien für die intelligente Automation, vol 14. Springer Vieweg, Berlin, Heidelberg. [online] available: https://doi.org/10.1007/978-3-662-64283-2_1.

[2] W. Alsabbagh and P. Langendörfer, "A Stealth Program Injection Attack against S7-300 PLCs," 2021 22nd IEEE International Conference on Industrial Technology (ICIT), 2021, pp. 986-993, doi: 10.1109/ICIT46573.2021.9453483.

[3] W. Alsabbagh and P. Langendörfer, "Patch Now and Attack Later - Exploiting S7 PLCs by Time-Of-Day Block," 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS), 2021, pp. 144-151, doi: 10.1109/ICPS49255.2021.9468226.

[4] W. Alsabbagh and P. Langendörfer, "A Fully-Blind False Data Injection on PROFINET I/O Systems," 2021 IEEE 30th International Symposium on Industrial Electronics (ISIE), 2021, pp. 1-8, doi: 10.1109/ISIE45552.2021.9576496.

[5] W. Alsabbagh and P. Langendörfer, "A Control Injection Attack against S7 PLCs -Manipulating the Decompiled Code," IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society, 2021, pp. 1-8, doi: 10.1109/IECON48115.2021.9589721.

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

[7] W. Alsabbagh and P. Langendörfer, "No Need to be Online to Attack - Exploiting S7-1500 PLCs by Time-Of-Day Block," 2022 XXVIII International Conference on Information, Communication and Automation Technologies (ICAT), 2022, pp. 1-8, doi: 10.1109/ICAT54566.2022.9811147.

[8] W. Alsabbagh and P. Langendörfer, "A Flashback on Control Logic Injection Attacks against Programmable Logic Controllers," Automation 2022, 3(4), 596-621; https://doi.org/10.3390/automation3040030.

[9] P. Langendörfer, S. Kornemann, W. Alsabbagh and E. Hermann, "Information Security: The Cornerstonefor Surviving the Digital Wild," In: Madsen, O., Berger, U., Møller, C., Heidemann Lassen, A., Vejrum Waehrens, B., Schou, C. (eds) The Future of Smart Production for SMEs. Springer, Cham, doi 10.1007/978-3-031-15428-7_29.

[10] W. Alsabbagh, S. Amogbonjaye, D. Urrego and P. Langendörfer, "A Stealthy False Command Injection Attack on Modbus based SCADA Systems," 2023 IEEE 20th Consumer Communications Networking Conference (CCNC), Las Vegas, NV, USA, 2023, pp. 1-9, doi: 10.1109/CCNC51644.2023.10059804.

[11] W. Alsabbagh and P. Langendörfer, "You Are What You Attack: Breaking the Cryptographically Protected S7 Protocol," 2023 IEEE 19th International Conference on Factory Communication Systems (WFCS), Pavia, Italy, 2023, pp. 1-8, doi: 10.1109/WFCS57264.2023.10144251.

[12] W. Alsabbagh, C. Kim and P. Langendörfer, "Good Night, and Good Luck: A Control Logic Injection Attack on OpenPLC," IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society, Singapore, Singapore, 2023, pp. 1-8, doi: 10.1109/IECON51785.2023.10312570.

[13] W. Alsabbagh and P. Langendörfer, "Security of Programmable Logic Controllers and Related Systems: Today and Tomorrow," in IEEE Open Journal of the Industrial Electronics Society, vol. 4, pp. 659-693, 2023, doi: 10.1109/OJIES.2023.3335976.

[14] W. Alsabbagh, C. Kim and P. Langendörfer, "Investigating the Security of OpenPLC: Vulnerabilities, Attacks, and Mitigation Solutions," in IEEE Access, doi: 10.1109/ACCESS.2024.3356051.

[15] D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything," Technical Report, CISCO Internet Business Solutions Group (IBSG), 2011.

[16] D. Beresford, "Exploiting Siemens Simatic S7 PLCs," Black Hat USA, 2011. [Online] available: https://media.blackhat.com/bh-us-11/Beresford/BH_US11_Beresford_S7_PLCs_WP.pdf

[17] N. Falliere, L. O. Murchu, and E. Chien, "W32. Stuxnet dossier," Symantec Corp., Secur. Response, Mountain View, CA, USA, White Paper, 2011, p. 29, vol. 5, no. 6.

[18] D. Hentunen and A. Tikkanen, "Havex hunts for ics/scadasystems," InF-Secure. 2014.

[19] Michael J Assante. Confirmation of a coordinated attack on the ukrainian power grid.SANS Industrial Control Systems SecurityBlog, 207, 2016.

[20] R. M. Lee, M. J. Assante and T. Conway, “German Steel Mill Cyber Attack,” 2014. [Online] available: https://ics.sans.org/media/ICS-CPPE-case-Study-2-German-Steelworks_Facility.pdf.

[21] “Attackers Deploy New ICS Attack Framework “TRITON” and Cause Operational Disruption to Critical Infrastructure,” [Online] available: https://www.fireeye.com/blog/threat-research/2017/12/attackers-deploy-new-ics-attack-framework-triton.html

[22] L. O. M. Nicolas Falliere and E. Chien, “W32.Stuxnet Dossier (Version 1.4),” White Paper, Symantec Security Response, 2011.

[23] L. of Cryptography and S. S. (CrySyS), “Duqu: A Stuxnet-like malware found in the wild v0.93,” Technical Report, 2011.

[24] S. Miller, N. Brubaker, D. K. Zafra, and D. Caban, "TRITON Actor TTP Profile, Custom Attack Tools, Detections, and ATTACK Mapping," April 10, 2019. [online] available: https://www.fireeye.com/blog/threat-research/2019/04/triton-actor-ttp-profilecustom-attack-tools-detections.html

[25] Z. Durumeric, E. Wustrow and J A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," In: USENIX Security Symposium. Vol. 8. 2013, pp. 47–53. [Online] available: https://zmap.io/paper.pdf

[26] R. Bodenheim, J. Butts, S. Dunlap and Barry Mullins, "Evaluation of the Ability of the Shodan Search Engine to Identify Internet-facing Industrial Control Devices,". In: International Journal of Critical Infrastructure Protection 7.2 (2014), pp. 114–123. Doi: 10.1016/j.ijcip. 2014.03.001.

[27] J. C. Matherly, "SHODAN the Computer Search Engine," 2009. [Online] available: https://www.shodanhq.com/

[28] "Knownsec Inc." ZoomEye – Cyberspace Search Engine. 2016. [Online] available: https://www.zoomeye.org

[29] "Ditecting." Detecting everything among the industrial control cyberspace, probing malicious vulnerability then mending the ”heaven”. 2015. [Online] available: https://www.ditecting.com

[30] "Censys.io." Frequently Asked Questions (FAQ). [Online] available: https://support.censys.io/en/articles/1294848-frequently-asked-questions-faq

[31] J. Klick, S. Lau, D. Marzin, J. -O. Malchow and V. Roth, "Internet-facing PLCs as a network backdoor," 2015 IEEE Conference on Communications and Network Security (CNS), 2015, pp. 524-532, doi: 10.1109/CNS.2015.7346865.

[32]] R. Radvanosky and J. Brodsky, "Project Shine (SHodan INtelligence Extraction) Findings Report," USA. 10th SANS ICS Security Summit, Orlando, FL, February 23-24, 2015.

[33] International Electrotechnical Commission and others. "IEC 62264-1 Enterprise-control system integration–Part 1: Models and terminology". In: IEC, Genf (2003). [Online] available: https://www.iso.org/standard/57308.html

[34] M. Niedermaier, "Security Challenges and Building Blocks for Robust Industrial Internet of Things Systems," Doctoral dissertation, Fakultät für Elektrotechnik und Informationstechnik Technische Universität München, 2020.

[35] A. -R. Sadeghi, C. Wachsmann and M. Waidner, "Security and privacy challenges in industrial Internet of Things," 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2015, pp. 1-6, doi: 10.1145/2744769.2747942..

[36] E. K. Hemsley and R. E. Fisher, "History of Industrial Control System Cyber Incidents," December 2018. doi: 10.2172/1505628.

[37] N. Maskelyne, "Electrical syntony and wireless telegraphy," The Electrician, 51, pp. 359–360, June 19, 1990.

[38] M. Abrams and J. Weiss, "Malicious Control System Cyber Security Attack Case Study–Maroochy Water Services, Australia," [Online] available: https://www.mitre.org/sites/default/files/pdf/08_1145.pdf, Jul 2008.

[39] Global Energy Cyberattacks, "Night Dragon," 2011. [Online] available: https://www.mcafee.com/wp-content/uploads/2011/02/McAfee_NightDragon_wp_draft_to_customersv1-1.pdf

[40] E. Chien, L. OMurchu, and N. Falliere, "W32.Duqu: The precursor to the next StuxNet," Symantec Secur. Response, Mountain View, CA, USA, p. 2, vol. 4, no. 4.

[41] Symantec, "The Shamoon Attacks." [Online] available: https://www.symantec.com/connect/blogs/shamoon-attacks, Aug 2012.

[42] F-Secure, “Havex Hunts For ICS/SCADA Systems,” [Online] available: https://www.f-secure.com/weblog/archives/00002718.html, Jun 2014.

[43] S. I. Response, "Dragonfly: Cyberespionage attacks against energy suppliers," AI Symantec Corp., Mountain View, CA, USA, Jul. 2014, p. 18, vol. 7.

[44] N. Nelson, "The impact of dragonfly malware on industrial control systems," SANS Inst., Bethesda, MD, USA, p. 27.

[45] Dragos Inc, “TRISIS Malware Analysis of Safety System Targeted Malware, Version 1.20171213.” [Online] available: https://www.dragos.com/wp-content/uploads/TRISIS-01.pdf, December 2017.

[46] R. M. Lee, M. J. Assante and T. Conway, “Analysis of the Cyber Attack on the Ukrainian Power Grid,” 2016. [Online] available: https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf.

[47] Dragos Inc, “CRASHOVERRIDE Analysing the Threat to Electric Grid Operations Version 2.20170613.” [Online] available: https://www.dragos.com/wp-content/uploads/CrashOverride-01.pdf, June 2017.

[48] A. Greenberg, “The Untold Story of NotPetya, the most Devastating Cyberattack in History,” 2019. [Online] available: https://www.wired.com/story/notpetya-cyberattackukraine-russia-code-crashed-the-world.

[49] M. Giles, "Triton is the world’s most murderous malware, and it’s spreading," [Online] available: https://www.technologyreview.com/2019/03/05/103328/cybersecurity-critical-infrastructure-triton-malware/, March 2019.

[50] R. Thomas, “Triton malware spearheads latest generation of attacks on industrial systems.” [Online] available: https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/triton-malware-spearheads, December 2018.

[51] W. Largent, "New VPNFilter malware targets at least 500k networking devices worldwide," [Online]. Available: http://blog.talosintelligence.com/2018/05/VPNFilter.html

[52] [Online]. Available: https://www.darktrace.com/en/blog/what-the-ekans-ransomware-attack-reveals-about-the-future-of-ot-cyber-attacks/

[53] [Online]. Available: https://www.msn.com/en-gb/news/world/chinese-cyberattack-almost-shut-off-power-to-three-million-australians/ar-AARz9G6?ocid=uxbndlbing

[54] K. John and M. Tiegelkamp, "IEC 61131–3: Programming Industrial Automation Systems," In Springer-Verlag Berlin Heidelberg 2001. [Online]. Available: https://link.springer.com/book/10.1007/978-3-662-07847-1

[55] ICS-CERT. CVE-2017-13997. https://nvd.nist.gov/vuln/detail/ CVE-2017-13997.

[56] ICS-CERT. CVE-2018-10619. https://nvd.nist.gov/vuln/detail/ CVE-2018-10619.

[57] ICS-CERT. CVE-2017-12739. https://nvd.nist.gov/vuln/detail/ CVE-2017-12739.

[58] ICS-CERT. CVE-2017-12088. https://nvd.nist.gov/vuln/detail/ CVE-2017-12088.

[59] ICS-CERT. CVE-2019-10922. https://nvd.nist.gov/vuln/detail/ CVE-2019-10922.

[60] C. Perrin, "The CIA Triad" 2008, [Online]. Available: http://www.techrepublic.com/blog/security/the-cia-triad.

[61] A. Serhane, M. Raad, R. Raad, and W. Susilo, “PLC code-level vulnerabilities,” in Proc. Int. Conf. Comput. Appl. (ICCA), Aug. 2018, pp. 348–352. [Online]. Available: https://ieeexplore.ieee.org/document/8460287/

[62] S. E. Valentine, Plc code vulnerabilities through scada systems, 2013.

[63] S. McLaughlin and S. Zonouz, "Controller-aware false data injection against programmable logic controllers," 2014 IEEE International Conference on Smart Grid Communications (SmartGridComm), 2014, pp. 848-853, doi: 10.1109/SmartGridComm.2014.7007754.

[64] S. McLaughlin, S. Zonouz, D. Pohly and P. McDaniel, " A Trusted Safety Verifier for Process Controller Code," Network and Distributed System Security Symposium, 2014, doi:10.14722/ndss.2014.23043.

[65] S. Zonouz, J. Rrushi and S. McLaughlin, "Detecting Industrial Control Malware Using Automated PLC Code Analytics," in IEEE Security & Privacy, vol. 12, no. 6, pp. 40-47, Nov.-Dec. 2014, doi: 10.1109/MSP.2014.113.

[66] S. Senthivel et al., “Denial of Engineering Operations Attacks in industrial Control Systems”, Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy March 2018 Pages 319–329https://doi.org/10.1145/3176258.3176319.

[67] S. Mclaughlin, "On dynamic malware payloads aimed at programmable logic controllers," In HotSec, 2011.

[68] 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, pages 439–449, 2012.

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

[70] S. Kalle, N. Ameen, H. Yoo, and I. Ahmed, "CLIK on PLCs! Attacking Control Logic with Decompilation and Virtual PLC," 2019, doi: 10.14722/bar.2019.23xxx

[71] S. A. Qasim, J. M. Smith and I. Ahmed, "Control Logic Forensics Framework using Built-in Decompiler of Engineering Software in Industrial Control Systems," Forensic Science International: Digital Investigation, 2020, doi: 10.1016/j.fsidi.2020.301013.

[72] A. Ferrari, G. Magnani, D. Grasso, and A. Fantechi, "Model checking interlocking control tables," In FORMS/FORMAT 2010, pages 107–115. Springer, 2011.

[73] RX Family User's Manual:Software, Renesas Electronics, 2013.

[74] R. Spenneberg, M. Bruggemann, and H. Schwartke, "Plc-Blaster: A worm living solely in the plc," Black Hat Asia, Marina Bay Sands, Singapore, 2016.

[75] C. Lei, L. Donghong, and M. Liang, "The spear to break the security wall of S7CommPlus", Black Hat USA 2017, 2017.

[76] E. Biham, S. Bitan, A. Carmel, A. Dankner, U. Malin, and A. Wool, "Rogue7: Rogue Engineering-Station attacks on S7 Simatic PLCs", Black Hat USA 2019, 2019.

[77] H. Hui and K. McLaughlin, "Investigating Current PLC Security Issues Regarding Siemens S7 Communications and TIA Portal," In 5th International Symposium for ICS & SCADA Cyber Security Research 2018: Proceedings (pp. 67-73), doi: 10.14236/ewic/ICS2018.8.

[78] H. Hui, K. McLaughlin, and S. Sezer, "Vulnerability analysis of S7 PLCs: Manipulating the security mechanism," International Journal of Critical Infrastructure Protection, Volume 35, 2021, 100470, ISSN 1874-5482, https://doi.org/10.1016/j.ijcip.2021.100470.

[79] Z. Basnight, "Firmware counterfeiting and modification attacks on programmable logic controllers, " Air Force Institute of Technology, Ohio, 2013.

[80] Z. Basnight, J. Butts, J. L. Jr., and T. Dube, “Firmware modification attacks on programmable logic controllers,” International Journal of Critical Infrastructure Protection, vol. 6, no. 2, pp. 76 – 84, 2013.

[81] D. Peck and D. Peterson, “Leveraging ethernet card vulnerabilities in field devices,” in SCADA Security Scientific Symposium, 2009, pp. 1–19.

[82] C. Schuett, J. Butts, and S. Dunlap, "An evaluation of modification attacks on programmable logic controllers," International Journal of Critical Infrastructure Protection, 2014, doi: 7. 10.1016/j.ijcip.2014.01.004.

[83] M. H. Rais, R. A. Awad, J. Lopez, and I. Ahmed, " JTAG-based PLC memory acquisition framework for industrial control systems," Forensic Science International: Digital Investigation, Volume 37, Supplement, 2021, 301196, ISSN 2666-2817, doi: 10.1016/j.fsidi.2021.301196

[84] L. A. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. Mohammed, and S. A. Zonouz, “Hey, my malware knows physics! attacking PLCs with physical model aware rootkit,” in Proceedings 2017 Network and Distributed System Security Symposium. Internet Society. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/hey-my-malware-knows-physics-attacking-plcs-physical-model-aware-rootkit/

[85] B. Lim, D. Chen, Y. An, Z. Kalbarczyk and R. Iyer, "Attack Induced Common-Mode Failures on PLC-Based Safety System in a Nuclear Power Plant: Practical Experience Report," 2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC), 2017, pp. 205-210, doi: 10.1109/PRDC.2017.34.

[86] H. Yoo and I. Ahmed, "Control Logic Injection Attacks on Industrial Control Systems," (2019), doi: 10.1007/978-3-030-22312-0_3.

[87] H. Yoo, S. Kalle, J. M. Smith, and I. Ahmed, "Overshadow plc to detect remote control-logic injection attacks," In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2019, pp. 109-132. [Online]. Available: http://www.people.vcu.edu/~iahmed3/publications/dimva_2019_shade.pdf

[88] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," In: International Conference on Recent Advances in Intrusion Detection (RAID), 2006, doi:10.1007/11856214_12

[89] Y. Wang, J. Liu, C. Yang, L. Zhou, L. Shuangfei, and X. Zhaoyan, "Access Control Attacks on PLC Vulnerabilities," (2018), in Journal of Computer and Communications, Vol.6, No.11, pages 311-325, doi: 10.4236/jcc.2018.611028.

[90] N. Govil, A. Agrawal, and N. O. Tippenhauer, "On ladder logic bombs in industrial control systems," In: Katsikas S. et al. (eds) Computer Security. SECPRE 2017, CyberICPS 2017. Lecture Notes in Computer Science, vol 10683. Springer, Cham. https://doi.org/10.1007/978-3-319-72817-9_8

[91] M. Xiao, J. Wu, C. Long and S. Li, "Construction of false sequence attack against PLC based power control system," 2016 35th Chinese Control Conference (CCC), 2016, pp. 10090-10095, doi: 10.1109/ChiCC.2016.7554953.

[92] A. Abbasi, and M. Hashemi, " Ghost in the PLC designing an undetectable programmable logic controller rootkit via pin control attack," In Black Hat Europe (pp. 1-35). Black Hat.

[93] [Online]. Available: https://ipcsautomation.com/blog-post/market-share-of-different-plcs/

[94] [Online]. Available: https://roboticsandautomationnews.com/2020/07/15/top-20-programmable-logic-controller-manufacturers/33153/

[95] [Online]. Available: https://support.industry.siemens.com/cs/document/45531107/simatic-programming-with-step-7-v5-5?dti=0&lc=en-US

[96] D. Urbina, J. Giraldo, N.O. Tippenhauer, and A. Cardenas, "Attacking Fieldbus Communications in ICS: Applications to the SWaT Testbed," 2016, doi:10.3233/978-1-61499-617-0-75.

[97] S. Sridhar, and G. Manimaran, "Data integrity attacks and their impacts on SCADA control system," in IEEE PES General Meeting, 2010, pages 1-6. doi: 10.1109/PES.2010.5590115.

[98] G. Bernieri, E. Etchevés Miciolino, F. Pascucci, and R. Setola, "Monitoring system reaction in cyber-physical testbed under cyber-attacks," Computers Electrical Engineering, vol. 59, pp. 86 – 98, 2017.

[99] A. P. Mathur and N. O. Tippenhauer, "SWAT: a water treatment testbed for research and training on ICS security," in 2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater), pp. 31–36, 2016.

[100] S. Adepu and A. Mathur, "Distributed attack detection in a water treatment plant: Method and case study," IEEE Transactions on Dependable and Secure Computing, pp. 1–1, 2018. 14 V.

[101] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "A dataset to support research in the design of secure water treatment systems," in Critical Information Infrastructures Security (G. Havarneanu, R. Setola, H. Nassopoulos, and S. Wolthusen, eds.), (Cham), pp. 88–99, Springer International Publishing, 2017.

[102] C. M. Ahmed, V. R. Palleti, and A. P. Mathur, "Wadi: A water distribution testbed for research in the design of secure cyber physical systems," p. 25–28, Association for Computing Machinery, 2017.

[103] V. K. Mishra, V. R. Palleti, and A. Mathur, "A modeling framework for critical infrastructure and its application in detecting cyber-attacks on a water distribution system," International Journal of Critical Infrastructure Protection, vol. 26, p. 100298, 2019.

[104] F. Zhang, H. A. D. E. Kodituwakku, J. W. Hines, and J. Coble, "Multilayer data-driven cyber-attack detection system for industrial control systems based on network, system, and process data," IEEE Transactions on Industrial Informatics, vol. 15, no. 7, pp. 4362–4369, 2019.

[105] X. Li, C. Zhou, Y.-C. Tian, N. Xiong, and Y. Qin, "Asset-based dynamic impact assessment of cyberattacks for risk analysis in industrial control systems," IEEE Transactions on Industrial Informatics, vol. 14, pp. 608– 618, 2018.

[106] M. Dibaji, M. Pirani, D. Flamholz, A. M. Annaswamy, K. H. Johansson, and A. Chakrabortty, "A Systems and Control Perspective of CPS Security," 2019, doi: 10.1016/j.arcontrol.2019.04.011.

[107] S. Mehner, and H. König, "No Need to Marry to Change Your Name! Attacking Profinet IO Automation Networks Using DCP," In: Perdisci, R., Maurice, C., Giacinto, G., Almgren, M. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2019. Lecture Notes in Computer Science(), vol 11543. Springer, Cham, doi: 10.1007/978-3-030-22038-9_19.

[108] S. Pfrang and D. Meier, "Detecting and preventing replay attacks in industrial automation networks operated with profinet IO," J Comput Virol Hack Tech 14, 253–268 (2018). Available online: https://doi.org/10.1007/s11416-018-0315-0

[109] M. Roesch, "Snort-lightweight intrusion detection for networks," In: Proceedings of the 13th USENIX Conference on System Administration. November, 1999, Pages 229–238. Available online: https://dl.acm.org/doi/10.5555/1039834.1039864.

[110] M. Stouffer and V. Pillitteri, “Guide to industrial control systems (ics)security,” NIST special publication, 2015.

[111] “Framework for improving critical infrastructure cybersecurity version1.1,” National Institute of Standards and Technology, Tech. Rep," 2018. Available online: https://doi.org/10.6028/NIST.CSWP.04162018.

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

[113] M. Menezes and S. Vanstone, “Elliptic Curve Cryptosystems and Their Implementation”, Journal of Cryptology, pp. 209-224, 1993.

[114] G. Benmocha, E. Biham, and S. Perle, “Unintended features of APIs: Cryptanalysis of incremental HMAC,” in Selected Areas in Cryptography. (Lecture Notes in Computer Science 12804) O. Dunkelman, M. J. Jacobson, Jr, and C. O’Flynn, Eds. Berlin, Germany: Springer, 2021.

[115] K. Rabah, "Elliptic Curve ElGamal Encryption and Signature Schemes," Information Technology Journal, 2005, 4: 299-306, doi: 10.3923/itj.2005.299.306. Available online: https://scialert.net/abstract/?doi=itj.2005.299.306

[116] M. Zhang et al., "Towards Automated Safety Vetting of PLC Code in Real-World Plants," 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 522-538, doi: 10.1109/SP.2019.00034.

[117] C. H. Kim et al., “Securing real-time microcontroller Systems through customized memory view switching,” in Proc. Netw. Distrib. Syst. Secur. Symp., 2018, doi: 10.14722/ndss.2018.23117.

[118] https://github.com/dotnetprojects/DotNetSiemensPLCToolBoxLibrary

[119] R. Sun, A. Mera, L. Lu and D. Choffnes, "SoK: Attacks on Industrial Control Logic and Formal Verification-Based Defenses," 2021 IEEE European Symposium on Security and Privacy (EuroSP), 2021, pp. 385-402, doi: 10.1109/EuroSP51992.2021.00034.

[120] M. Tiegelkamp and K. John, "IEC 61131-3: Programming Industrial Automation Systems; Springer," Berlin/Heidelberg, Germany, 2001; Volume VI, p. 376.

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

[122] "Framework for improving critical infrastructure cybersecurity version1.1," National Institute of Standards and Technology, Tech. Rep., 2018, [Online]. Available: https://doi.org/10.6028/NIST.CSWP.04162018.

[123] "Recommended practice: Improving industrial control system cybersecurity with defense-in-depth strategies," Department of Homeland Security, Tech. Rep., 2016.

[124] https://us.profinet.com/technology/profinet/

[125] https://reference.opcfoundation.org/PROFINET/docs/4.1.4/

[126] “MITRE ATTCK,” https://attack.mitre.org/, 2020