IEEE Open Journal of the
Industrial Electronics Society

Received 6 December 2021; revised 22 January 2022; accepted 6 February 2022. Date of publication 14 February 2022;
date of current version 2 March 2022. The review of this paper was arranged by Associate Editor Yang Shi.

Digital Object Identifier 10.1109/OJIES.2022.3151528

A New Injection Threat on S7-1500 PLCs - Disrupting the Physical Process Offline

WAEL ALSABBAGH [1,2] (Member, IEEE), AND PETER LANGENDÖERFER [1,2]

[1]IHP – Leibniz-Institut für innovative Mikroelektronik, 15236 Frankfurt (Oder), Germany
[2]Brandenburg University of Technology Cottbus-Senftenberg, 03046 Cottbus, Germany

CORRESPONDING AUTHOR: WAEL ALSABBAGH (e-mail: Alsabbagh@ihp-microelectronics.com)

This work was supported by the Open Access Fund of Leibniz Association.

ABSTRACT Programmable Logic Controllers (PLCs) are increasingly connected and integrated into the Industrial Internet of Things (IIoT) for a better network connectivity and a more streamlined control process. But in fact, this brings also its security challenges and exposes them to various cyber-attacks targeting the physical process controlled by such devices. In this work, we investigate whether the newest S7 PLCs are vulnerable by design and can be exploited. In contrast to the typical control logic injection attacks existing in the research community, which require from adversaries to be online along the ongoing attack, this article introduces a new exploit strategy that aims at disrupting the physical process controlled by the infected PLC when adversaries are not connected neither to the target nor to its network at the point zero for the attack. Our exploit approach is comprised of two steps: 1) Patching the PLC with a malicious *Time-of-Day* interrupt block once an attacker gains access to an exposed PLC, 2) Triggering the interrupt at a later time on the attacker will, when he is disconnected to the system's network. For a real attack scenario, we implemented our attack approach on a Fischertechnik training system based on S7-1500 PLC using the latest version of S7CommPlus protocol. Our experimental results showed that we could keep the patched interrupt block in idle mode and hidden in the PLC memory for a long time without being revealed before being activated at the specific date and time that the attacker defined. Finally, we suggested some potential security recommendations to protect industrial environments from such a threat.

INDEX TERMS Programmable logic controllers, industrial control systems, injection attack, time-of-day block, offline attack.

I. INTRODUCTION

Industrial Control Systems (ICSs) are used to automate critical control processes such as production lines, electrical power grids, oil and gas facilities, petrochemical plants, and others. Each ICS environment consists of two main sites: a control site and a field site. Fig. 1 shows a typical ICS environment. The control center runs ICS services such as Human Machine Interfaces (HMIs) and engineering workstations. The field site has sensors, actuators, and Programmable Logic Controllers (PLCs) that are installed locally to monitor and control physical processes. The engineering workstation is used to configure and program PLCs. It has a PLC vendor-specific programming software to write control logic that defines how the PLC should control and maintain the physical process at a desired state. PLCs are offered by several vendors such as Siemens, Allen-Bradley, Mitsubishi, Schneider and Modicon. Each has its own proprietary firmware, programming language, communication protocols and maintenance software.

In the past, when PLCs were first introduced, it was uncommon for them to be connected to the outer world and they were often running independently i.e., the PLC-based ICS environments were air-gapped. This separation is no longer possible due to new demands such as maximizing the profits, minimizing the costs, and achieving a better efficiency [1]. Therefore, it is not surprising that most of modern ICS environments are increasingly connected to corporate networks and no longer controlled/monitored on-site. Unfortunately, this higher connectivity has also enlarged the attack surface, and brought its security challenges allowing attacks that were

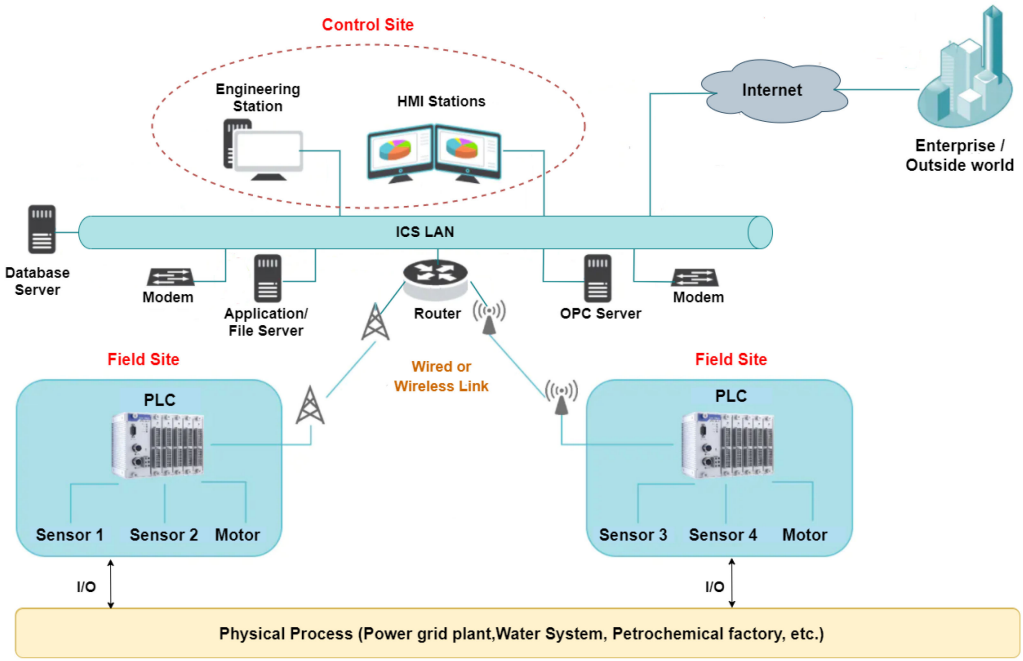

FIG. 1. An example of an industrial control system environment.

not existing in the times of the air-gapped industrial plants. Stuxnet [12], which targeted the Iranian uranium enrichment in 2010, played an important role in increasing awareness of security for industrial control systems. This attack showed that no plant is resilient to cyber-attacks and that PLCs could be potentially hacked causing disastrous damages. But since then, several other ICS have been successfully attacked, for example the Ukrainian power grid [17], the German steel Mill [22], TRITON [16], etc.

In this work, we show that modern PLC-based ICS environments are not fully protected against control logic injection attacks, and that these systems are still quite far from being completely secure. To this end, we present a new attack strategy that allows malicious adversaries to disrupt the physical process controlled by PLCs offline i.e., without being connected to the target or to its network at the point zero for the attack. The main focus of our investigations is on Siemens devices, precisely the latest PLC models i.e., devices from S7-1500 family, and the latest version of S7CommPlus protocol i.e., S7CommPlusV3. Our attack approach is structured into two main phases:

1) Patching the control logic program of a PLC with an interrupt, precisely with *Time-of-Day* (ToD) interrupt block using the specific Organization Block 10 (OB10). This is done online i.e., when the adversary has access to the target device. During this phase, the patch has no impact, neither on the physical process nor on the execution process of the control logic program i.e., the patch is in the idle mode.
2) Activating the patch injected in the target later at a certain date and time. This is done offline i.e., without the need of being connected to the target PLC at the point zero for the attack.

To conduct experiments for proving the research, a Fischertechnik training industry plant[1] controlled by an S7-1500 PLC was used to test our attack approach. Our new threat is network based, and can be successfully conducted by any attacker with network access to any S7-1500 PLC with a firmware V2.9.2 or lower.

A. MOTIVATION

The objective of this article is to introduce a new control logic injection on cryptographically secured PLCs that use sophisticated protection methods. The intention of discussing this new type of attack is to raise awareness for sophisticated attacks and to assist in determining new vulnerabilities and weaknesses existing in PLCs as they are running in millions of critical industrial plants, and play a major point of interacting between the cyber and physical world. Our main focus is to understand the attack vectors in the first place, and show the security research community, engineers, and industrial vendors what the consequence of the vulnerabilities would be if they are exploited. To conduct a real-world attack scenario, we chose a device from Siemens S7-1500 family. Our selection is based on two factors. First, Siemens is the leading provider of industrial automation components and their SIMATIC families have approximately 30-40% of the industry market [23]–[25]. Secondly, Siemens claimed that its newest PLCs generation is well-secure against various attacks,

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

and their new developed S7CommPlus protocol supports improved security measures like an advanced anti-replay mechanism and a sophisticated integrity check. These two factors 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 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 very well in advance, and do not need to be successful with the attack just at the right time.

B. PROBLEM STATEMENT

Most of the injection attacks have two critical challenges: First is that the typical injection attacks are designed to gain access to the target or its network in very specific circumstances i.e., when the security measure implemented is absent or disabled for a certain reason [2], [3], [5]–[7], [9]–[11], [13], [18], [29], [30], [34] for example, the security mean is being updated, the ICS operator is running some maintenance processes, other devices are being removed/replaced/added to the network, etc. The system is at high risk to get a malicious infection during these critical phases, but it is not operating in its normal state i.e., the physical process is more likely to be temporally off. Hence, if attackers manage successfully to gain access to the target device during these times, and perform their attacks right after that, they will, pretty likely, not impact the physical process. The second challenge is that after the ICS operator is done with the ongoing maintenance processes, he usually reactivates the security measure before re-operating the system once again. This allows him to reveal and prevent any attempt to inject the PLC if the attacker is still connected to the network. Our attack approach overcomes both challenges by patching the PLC with a malicious block at that point in time at which the attacker accesses the network successfully, keeping the infection hidden in the PLC's memory, and lunching the attack at a later time on his will. This ensures that the attack is not being 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 are still able to reveal any infection or 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 [10]. In this article, we also overcome this challenge by exploiting a vulnerability existing in the newest S7CommPlus protocol (explained in Section V) to hide the infection from the ICS operator i.e., who will always be shown the original code that runs on his engineering software whilst the PLC runs the attacker's code.

C. ATTACKER MODEL

Assumption: Our attacker model assumes that an attacker has access to the level-3 network of the Purdue Model [2] (i.e., control center network). This assumption is based on real-world ICS attacks e.g., TRITON [16] and Ukraine power grid attack [17] that gained access to the control center via a typical IT attack vector such as infected USB stick and social engineering attack. We also assume that the attacker has access to the PLC and its respective engineering software along with a packet-sniffing tool such as Wireshark.[3] 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 assumptions have already been reported to hold true in reports on real world attacks, we are convinced that our attack is a realistic one.

Attacker's goal: The attacker's goal is as follows: disrupting the physical process at a time when he is completely offline, i.e., without being connected to the target or its network at the point zero for the attack, while the physical process of the target network is controlled by the infected PLC. In order to ensure achieving the overall goal, the injection may not be revealed by the ICS operator in the time between infecting the PLC and the attack launch date. In this work, we assume that an attacker achieves these goals if the following three tasks are accomplished:

1) patching the malicious code when the attacker is connected to the target's network.
2) keeping this infection hidden in the PLC's memory without being revealed.
3) disrupting the physical process at a later time when the attacker is completely offline of the target's network.

Attacker's capabilities: The attacker can employ one or more of these capabilities to achieve the goals mentioned earlier:

1) Eavesdropping: read any messages between two communicating parties.
2) Fabrication: initiate conversation with any other party and compose/send a new message.
3) Interception: intercept messages, and block or modify/resend them.

D. CONTRIBUTIONS

In this article, we take the attack approach presented in our former paper [10] one step further in the direction of exploiting PLCs offline, and extend our experiences to involve the modern S7-1500 PLCs that use S7CommPlusV3 protocol. Our main contributions in this article are summarized as follows:

1) Extending our control logic injection attack approach presented in [10] from S7-300 to S7-1500 PLCs.
2) Hiding the malicious interrupt code in the PLC's memory until the very moment determined by the attacker.

[2]https://www.goingstealthy.com/the-ics-prude-model/
[3]https://www.wireshark.org/

3) Disrupting the physical process controlled by the compromised PLC offline i.e., when the attacker is not connected to the target or its network.
4) Demonstrating our attack using a real Siemens S7-1512SP controlling a Fischertechnik training factory.
5) Revealing two new vulnerabilities in the integrity protection method that S7-1500 PLCs and their S7CommPlus protocol use.

The rest of this work is organized as follows. Section II provides an overview of control logic injection attacks and related work. Section III presents the technical background, followed by the description of the protection mechanism of the latest S7CommPlus protocol in Section IV. Our attack approach is presented and explained in details in Section V. In Section VI, we evaluate and discuss the impact of our attack, as well as suggest some possible mitigation methods. Finally, we conclude our work in Section VII.

II. OVERVIEW AND RELATED WORK

One of the recent threats targeting ICSs is the control logic injection attack. Such an attack involves modifying the original control logic running on a target PLC by engaging its engineering software, typically employing the man-in-the-middle approach [3]–[5], [9], [10], [13], [30]–[32]. The main vulnerability exploited in this type of attacks is the lack of authentication measures in the PLC protocols. ICS vendors responded to this threat by providing their PLCs with passwords to protect the control logic from unauthorized access i.e., whenever an ICS supervisor attempts to access the control logic running in a PLC, the device first requires an authentication to allow him to read/write the code. This is normally done via proprietary authentication protocol. But, this solution is not fully preventing the controllers from being compromised. Previous academic efforts [2]–[5], [9], [35] managed successfully to bypass the authentication and to access the control logic in different password-protected PLCs. The authors of the abovementioned papers discussed two prime ways to bypass the authentication: either by extracting the hash of the password and then pushing it back to the PLC (known as a replay attack), or using a representative list of "plain-text password, encoded-text password" pairs to brute-force each byte offline. Overall, protecting the control logic by password authentication only failed. Attackers are still capable of accessing the PLCs' program and manipulating the physical processes controlled by the exposed devices.

In the research community there are two types of control logic injection attacks: traditional control logic injection and firmware injection. However, infecting a PLC firmware would be a challenging task in a real ICS environment as most PLC vendors protect their PLCs from unauthorized firmware updates by cryptographic methods e.g., digital signature, or allowing firmware updates only by local access (e.g., SD cards and USB). This work does not cover a firmware injection and only focuses on the traditional control logic injection attack. In the following, we classify the existing injection attacks aiming at disrupting the physical process into two groups.

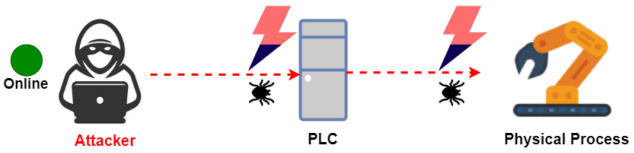

FIG. 2. Disrupting the physical process online.

A. DISRUPTING THE PHYSICAL PROCESS ONLINE

The attacks in this group are designed to modify the original control logic program by engaging its engineering software. The physical process controlled by the infected device is impacted right after the malicious code is successfully injected. Fig. 2 shows the attack sequence.

The most well-known attack representing this kind is the one that was conducted on Iranian nuclear facilities in 2010, named as Stuxnet to sabotage centrifuges at a uranium enrichment plant. The Stuxnet attack [12], [20], [21] used a windows PC to target Siemens S7-300 and S7-400 PLCs that were connected to variable frequency drives. It infects the control logic of the PLCs to monitor the frequency of the attached motors, and launches an attack if the frequency is within a certain range (i.e., 807 Hz and 1,210 Hz). More recent examples of such attacks on ICS occurred in Ukraine [17], [19]. These attacks targeted the electrical distribution grid causing widespread blackouts. In 2014, the German federal office for information security also announced a cyber-attack at an unnamed steel mill [22]. The hackers manipulated and disrupted control systems to such a degree that a blast furnace could not be properly shot down, resulting in a massive damage. McLaughlin [45] conducted a control logic injection attack on a train interlocking program. The malicious program he introduced was reverse engineered using a format program. With the help of the decompiled program, he extracted the field-bus ID that indicated the PLC vendor and model, and then retrieved clues about the process structure and operations. Afterwards he designed his own program that generates unsafe behaviors for the train e.g., causing conflict states for the train signals. As a real attack scenario, he targeted timing-sensitive signals and switches. In a follow up work, McLaughlin *et al.* [46] implemented SABOT. It required a high-level description of the physical process, for example, "the plant contains two ingredient valves and one drain valve". Such information could be got from public channels, and are similar for processes in the same industrial sector. With this information, SABOT generates a behavioral specification for the physical processes and used incremental model checking to search for a mapping between a variable within the program, and a specified physical process. Using this map, SABOT compiled a dynamic payload customized for the physical process. Both studies were limited to Siemens PLCs, without illustrating many details on reverse engineering. Valentine [48] introduced attacks that could install a jump to a subroutine command, and modify the interaction between two or more ladders in a program. This could be disguised as an erroneous use of scope and linkage by a novice programmer. In 2015, Klick *et al.* [6]

presented the injection of malware into the control logic of a SIMATIC PLC, 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 MC7 bytecode. In a follow on work, Spenneberg *et al.* [7] introduced a PLC worm. The worm spreads internally from one PLC to other target PLCs. During the infection phase, the worm scans the network for new target PLCs. A Ladder Logic Bomb malware written in ladder logic or one of the compatible languages was introduced in [8]. Such a malware is inserted by an attacker into existing control logic on PLCs. A group of researchers [9] demonstrated a remote attack on the control logic of PLCs. They were able to infect the PLC and to hide the infection from the engineering software at the control center. They implemented their attack on Schneider Electric Modicon M221, and its vendor-supplied engineering software SoMachine-Basic. Senthivel *et al.* [18] presented three control logic injection attacks where an attacker interferes with engineering operations of downloading and uploading PLC control logic. In the first attack scenario, an attacker, placed in a man-in-the-middle position between a target PLC and its engineering software, injects malicious control logic to the PLC and replaces it with original control logic to deceive the engineering software when the uploading operation is requested. The second scenario that their paper presented is very similar to the first scenario but differs in that an attacker uploads malformed control logic instead of the original control logic to crash the engineering software. The last scenario does not require a man-in-the-middle position, as the attack just injects crafted malformed control logic to the target PLC. Lei *et al.* [31] demonstrated a spear that can break the security wall of the S7CommPlus protocol that Siemens SIMATIC S7-1200 PLCs utilize. The authors first used the Wireshark software to analyze the communications between the TIA Portal software and S7 PLCs. Then, they applied the reverse debugging software WinDbg [4] 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 a serious damage for the physical process controlled by the infected PLC. In 2021, researchers in [3] also showed that S7-300 PLCs are vulnerable to such attacks and demonstrated that exploiting the control logic running in a PLC is feasible. After they compromised the security measures of PLCs, they conducted a successful injection attack and kept their attack hidden from the engineering software by engaging a fake PLC impersonating the real infected device. Researches behind Rogue7 [30] were able to create a rogue engineering station which can masquerade as the TIA Portal to S7 PLCs, and to inject any messages favorable to the attacker. By understanding how cryptographic messages were exchanged, they managed to hide the code in the user memory, which is invisible to the TIA Portal engineering station. In [44], a group of security researchers analyzed the anti-replay mechanism that the new S7 PLCs used, and managed successfully to steal an existing communication session and to make unauthorized changes to the PLC states. As a part of their experiments, they identified specific bytes necessary to craft valid network packets, and demonstrated a successful replay attack on S7 PLCs.

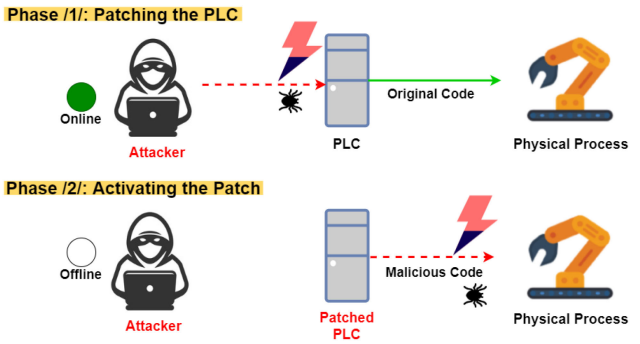

FIG. 3. Disrupting the physical process offline.

All the attacks mentioned above are limited and require that attackers are connected to the target at the point zero for the attack, which increases the possibility of being revealed by the ICS operators beforehand, or detected by security measures.

B. DISRUPTING THE PHYSICAL PROCESS OFFLINE

The attacks in this class are quite similar to the ones mentioned in the prior class, but differs in that an adversary does not aim at attacking the physical process right after gaining access to the target device. Meaning that, he patches his malicious code once he accesses an exposed PLC, then closes any live connection with the target keeping his patch inside the PLC's memory in idle mode. Afterwards, he activates his patch and compromises the physical process at a later time he wishes even without being connected to the system network (see Fig. 3).

To the best of our knowledge, only a few academic efforts discussing this new threat were published. Serhane *et al.* [47] focused on Ladder logic code vulnerabilities and bad code practices that may become the root cause of bugs and subsequently be exploited by attackers. They showed that attackers could generate uncertainly fluctuating output variables e.g., performing two timers to control the same output values could lead to a race condition. Such a scenario could result in a serious damage to the devices controlled, similar to Stuxnet [12]. Another scenario that the authors pointed out is that skilled adversaries could also bypass some functions, manually set certain operands to desired values, and apply empty branches or jumps. In order to achieve a stealthy modification, attackers could use array instructions or user-defined instructions, to log insert critical parameters and values. They also discussed that attackers could apply an infinite loop via jumps, and use nest timers and jumps to only trigger the attack at a certain time. We, in our former paper [10], presented a novel approach based on injecting the target PLC with a

[4] http://www.windbg.org/

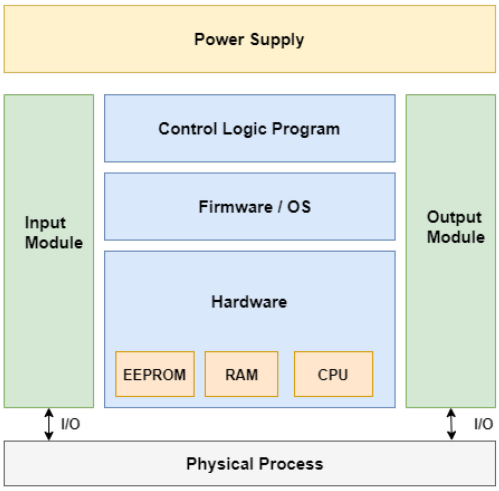

FIG. 4. A typical S7 PLC Architecture.

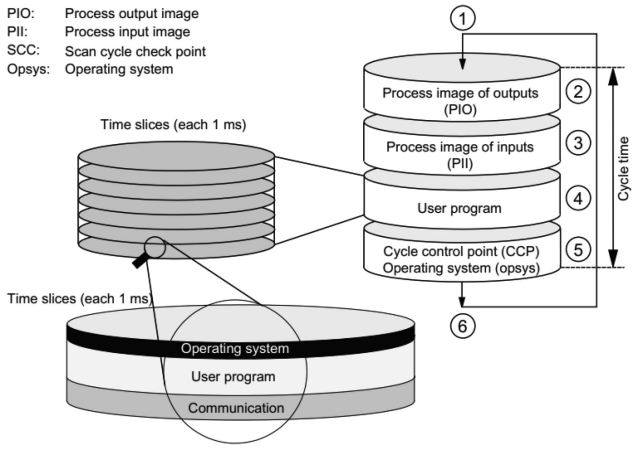

FIG. 5. Overview of program execution, extracted from [43].

Time-Of-Day interrupt code, which interrupts the execution sequence of the control logic at the time the attacker sets. Our evaluation results proved that an attacker could confuse the physical process even being disconnected from the target system. Although our research work was only tested on an old S7-300 PLC, and was just aiming at forcing the PLC to turn into stop mode, the attack was successful and managed to interrupt executing the original control logic code running in the patched PLC. Such attacks are severer than the online ones as the PLC keeps 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. The only realistic way to reveal this kind of attack is that the ICS operator requests the program from the PLC and compares the online code running in the infected device with the offline code that he has on the engineering station. But in this work, we overcome this challenge as illustrated later in Section V.

III. TECHNICAL BACKGROUND

In this section, we outline the architecture of a standard S7 PLC and its operating system, engineering software, user program, *Time-of-Day* interrupt, and S7Communication protocols.

A. SIMATIC S7 PLC ARCHITECTURE

Siemens produces several PLC product lines in the SIMATIC S7 family e.g., S7-300, S7-400, S7-1200, and S7-1500. All have the same architecture. Fig. 4 depicts a standard architecture of an S7 PLC that includes input and output modules, power supply, and memory such as Random Access Memory (RAM) and Electrically Erasable Programmable Read-only Memory (EEPROM). The firmware, known as Operating System (OS), as well as the user-specific program is stored in the EEPROM. Input and Output devices such as sensors, switches, relays, and valves are connected with the input and output modules. The PLC is connected to a physical process; the input devices provide the current state of the process to the PLC, which the PLC processes through its control logic, and controls the physical process accordingly via the output devices.

The control logic that an S7 PLC runs is programmed and compiled into a lower representation of the code i.e., to MC7 or MC7+ bytecode for S7-300/S7-400 or S7-1200/S7- 1500 PLCs respectively. After the code being compiled by the engineering station, its blocks, in MC7/MC7+ format, are downloaded and installed into the PLC via Siemens' S7Comm or s7CommPlus protocol for S7-300/S7-400 or S7-1200/S71500 PLCs respectively. Then, the MC7/MC7+ virtual machine in the S7 PLC will dispatch the code blocks, interpret and execute the bytecode.

B. OPERATING SYSTEM (OS)

Siemens PLCs run a real time OS, which initiates the cycle time monitoring. Afterwards, the OS cycles through four steps as shown in Fig. 5. In the first step, the CPU copies the values of the process image of outputs to the output modules. In the second step, the CPU reads the status of the input modules and updates the process image of input values. In the third step, the user program is executed in time slices with a duration of 1 ms (ms). Each time slice is divided into three parts, which are executed sequentially: The operating system, the user program and the communication. The number of time slices depends on the complexity of the current user program and the events interrupting the execution of the program.

In normal operation, if an event occurs, the block currently being executed is interrupted at a command boundary and a different organization block that is assigned to the particular event is called. Once the new organization block has been executed, the cyclic program resumes at the point at which it was interrupted. This holds true as the maximum allowed cycle time (150 ms by default) is not exceeded. In other words, if there are too many interrupt OBs called in the main OB1, the entire cycle time might be extended more than it is set in the PLC hardware configuration. Exceeding the maximum allowed execution cycle generates a software error, and the PLC calls a specific block to handle this error i.e., OB80.

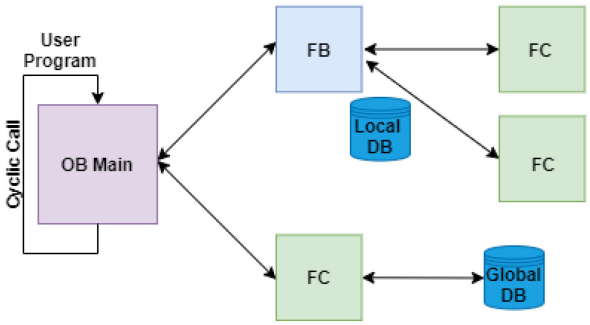

FIG. 6. S7 PLC's user program blocks.

There are two ways to handle with this error: 1) PLC turns to a stop mode if the OB80 is not loaded in the main program, 2) PLC executes the instructions that OB80 is programmed with e.g., an alarm.

C. ENGINEERING SOFTWARE

Siemens provides their Total Integrated Automation (TIA) Portal software to engineers for developing PLC programs. It consists of two main components. The STEP 7 as development environment for PLCs and WinCC to configure Human Machine Interfaces (HMIs). Engineers are able to program PLCs in one of the following programming languages: Ladder Diagram (LAD), Function Block Diagrams (FBD), Structured Control Language (SCL), and Statement List (STL).

D. USER PROGRAM

S7 PLC programs are divided into the following units: Organization Blocks (OBs), Functions (FCs), Function Blocks (FBs), Data Blocks (DBs), System Functions (SFCs), System Function Blocks (SFBs) and System Data Blocks (SDBs) as shown in Fig. 6.

OBs, FCs and FBs contain the actual code, while DBs provide storage for data structures, and SDBs for the current PLC configurations. The prefix *M*, memory, is used for addressing the internal data storage. A simple PLC program consists of at least one organization block called OB1, which is comparable to the main () function in a traditional C program. In more complex programs, engineers can encapsulate code by using functions and function blocks. The only difference is an additional DB as a parameter for calling an FB. The SFCs and SFBs are built into the PLC. However, the operating system calls OB cyclically and with this call it starts cyclic execution of the user program.

E. TIME-OF-DAY (TOD) INTERRUPTS

A *Time-of-Day* (TOD) interrupt is executed at a configured time, either one-time or periodically depending on the needs of interrupt e.g., every minute, hourly, daily, monthly, yearly, and at the end of the month. A CPU 1500 provides 20 organization blocks with the numbers OB10 to 0B17 and after OB 123 for processing a TOD interrupt.

To start a TOD interrupt, a user must first set the start time and then activate the interrupt. He can carry out both activities separately in the block properties, automatic configuration, or also with system functions, manual configuration. Activating the block properties means that the *Time-of-Day* interrupt is automatically started. However, in the following we illustrate both ways briefly:

1) *Automatic configuration:* The user adds an organization block with the event class *Time-of-Day* and enters the name, programming language, and number. He programs the OB10 with the required instructions to be executed when the interrupt occurs.

2) *Manual configuration:* In this method, the user uses system function blocks to set, cancel, and activate a *Time-of-Day* interrupt. He sets the necessary parameters for the interrupt in the main OB1, by using system function blocks while the interrupt instructions to be executed are programmed in OB10. [49] provides technical details to set and program *Time-of-Day* interrupts in S7-1500 PLCs.

F. S7COMMUNICATION PROTOCOLS

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 flavors: the older SIMATIC S7 PLCs implement an S7 flavor that is identified by the protocol number 0x32 (S7comm), while the new generation PLCs implement an S7 flavor that is identified by the protocol number 0x72 (S7CommPlus). The newer S7CommPlus protocol has three sub-versions: S7CommPlusV1, S7CommPlusV2, and S7CommPlusV3.

In this article, we only focus on the S7CommPlusV3 Protocol that is used in the newer versions of the TIA Portal from V13 on, and in the newer PLC S7-1500 firmware e.g., V1.8, 2.0, etc. This protocol requires that both the TIA Portal and the PLC support its features, and has more complex integrity protection mechanisms as illustrated in the next section. S7CommPlusV3 protocol is considered as the most secure protocol compared to the older S7CommPlus versions, i.e., S7CommPlusV1 and S7CommPlusV2.

IV. S7COMMPLUSV3 PROTOCOL

The S7CommPlusV3 protocol is used only by the newer version of the TIA Portal, and the S7-1500 PLCs. It supports various operations that are performed by the TIA Portal software as follows:

1) Start/Stop the control program currently loaded in the PLC memory.
2) Download a control program to the PLC.
3) Upload the current control program from the PLC to the TIA Portal.
4) Read the value of a control variable.
5) Modify the value of a control variable.

The above-mentioned operations are translated by the TIA Portal software to S7CommPlus messages before they are

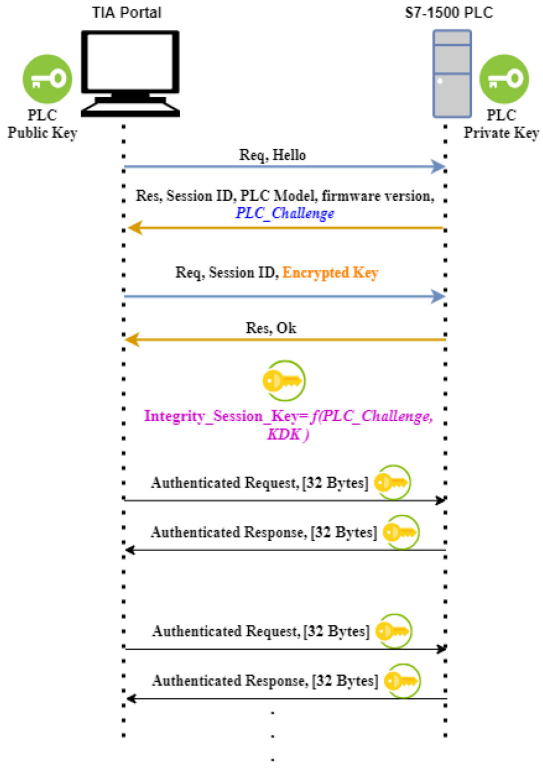

FIG. 7. The S7 Session Key Establishment Mechanism.

transmitted to the PLC. The PLC acts then on 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 session has a session ID chosen by the PLC. A 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 cryptographic protection mechanism as illustrated in the next subsection.

A. THE S7 INTEGRITY PROTECTION MECHANISM

Siemens integrated cryptographic protection in its newer S7 proprietary protocol in order to protect its PLCs from unauthorized access. The new mechanism uses two main modules:

1) **A session key exchange protocol** that the two parties (PLC and TIA Portal) use to establish a secret shared key in each session.

2) **Per-fragment message protection** that calculates a Message Authentication Code (MAC) value.

1) S7 KEY EXCHANGE PROTECTION

Siemens improved its S7CommPlus protocol by replacing the key generation process in the prior version, i.e., the S7CommPlusV2, by a more complex process in the newer version S7CommPlusV3. The new mechanism involves a new key exchange technique, that uses elliptic-curve public-key cryptography [33] as depicted in Fig. 7.

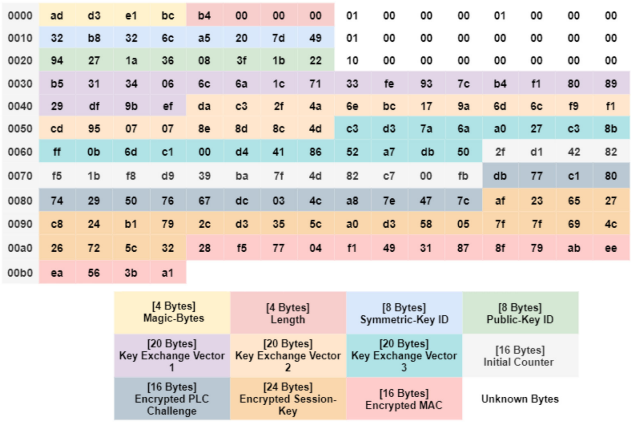

FIG. 8. The Structure of the *SecurityKeyEncriptedKey* BLOB Data.

The first request message is a *Hello* message that the TIA Portal sends to initialize a new session. Then, the PLC responds back with sharing its firmware version, model, Session ID, and specific 20-bytes known as *PLC_Challenge*. The PLC firmware version determines the elliptic-curve public-key pair to be used in the key exchange. After the TIA Portal receives the second message from the PLC, it activates a derivation algorithm to randomly select a *key Derivation Key* (*KDK*), and to generate the session key from the *PLC_Challenge* and the selected *KDK*. Afterwards, the TIA Portal transmits the key encrypted using Elliptic-Curve Cryptography (ECC) to the PLC over the third message. The third message contains, among other things, two main parts:

a) A data structure called *SecurityKeyEncryptedKey* shown in Fig. 8, which contains the selected key encrypted with the PLC's public key.

b) Two 8-bytes key fingerprints (additional key), of the PLC public key ID and the selected key, respectively.

Finally, the PLC verifies the third message. If this is done successfully, it returns OK in the fourth message, and from this point on, all the following messages in the session are integrity protected with the derived Session Key.

2) PER-FRAGMENTATION MESSAGE PROTECTION

When the TIA Portal downloads/uploads the control logic program to/from an S7-1500 PLC, the assigned S7CommPlus messages are fragmented to many small fragments sent over the TCP/IP packets. All messages exchanged between the two parties are integrity protected *HMAC-SHA256* [27]. This integrity protection is applied at the fragment level. Meaning that, it replaces the signal MAC value at the end of each message, and a cryptographic digest is placed at each fragment between the fragment header and the fragment data as shown in Fig. 9. [27] presents more technical details about this protection mechanism.

Although fragmenting the S7 messages was more challenging for attackers, they eventually overcame this protection mechanism and compromised the PLCs using this technique. The vulnerability reported in [28] shows that attackers

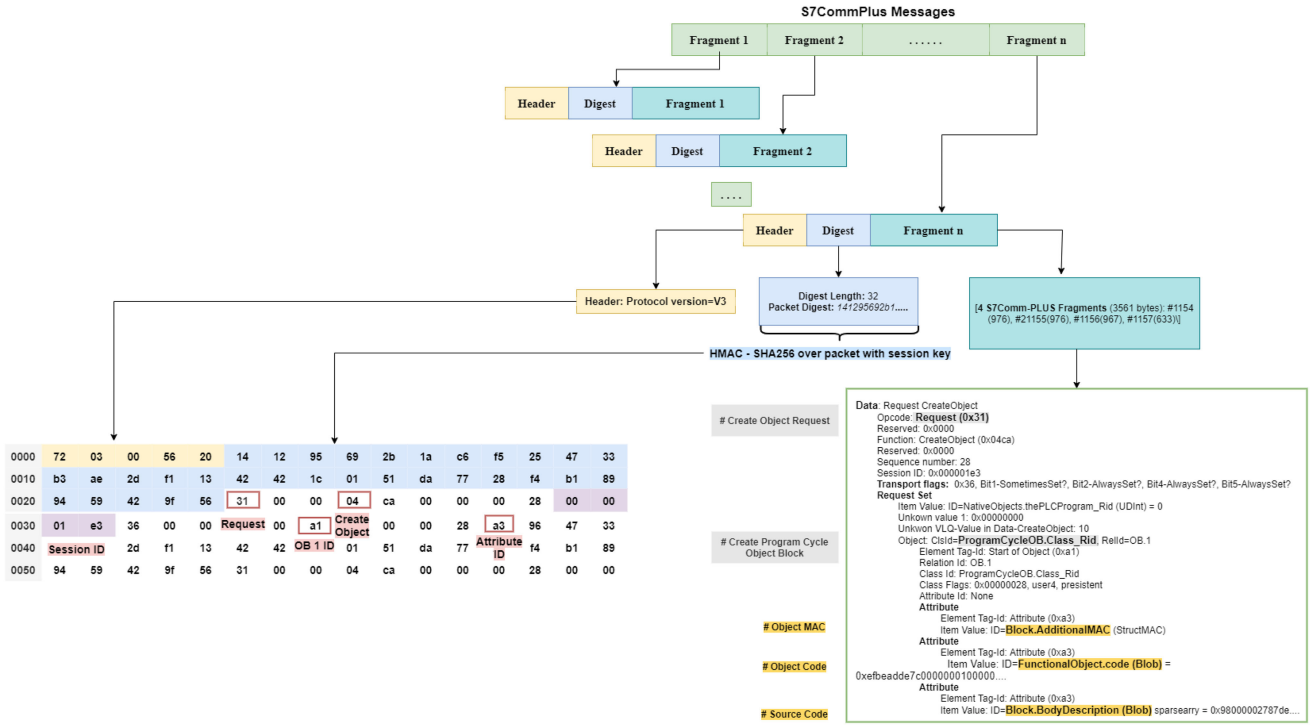

FIG. 9. **S7CommPlus message with integrity protection at fragment level.**

could implement man-in-the-middle approach and managed successfully to modify the network traffic exchanged on port 102/TCP due to the certain properties in the calculation used for this integrity protection.

B. S7COMMPLUS DOWNLOAD MESSAGES - OBJECTS AND ATTRIBUTES

S7 is a request response protocol. Each request message consists of a request header, and a request set. The header contains a function code, which identifies the requested operation e.g., *0x31* for a download message (see Fig. 9). A single S7CommPlus message might contain multiple objects, each containing multiple attributes. All objects and attributes have unique class identifiers. However, the *CreateObject* request builds a new object in the PLC memory with a unique ID (in our example, *0x04ca*). The program download message then creates an object of the class *ProgramCycleOB*. This object contains multiple attributes, each one having values dedicated to a specific purpose. For instance, the *FunctionalObject.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 MAC value in the integrity process, and both *Block.OptimizedInfo* and *Block.BodyDescription* are equivalent to the program written by the ICS operator which are stored in the PLC and can be later uploaded, upon request, to a TIA Portal project.

From the security point of view, these attributes are critical data that is transmitted over the S7CommPlusV3 protocol. Meaning that, if an attacker can intercept the S7 packets containing these attributes, and manage successfully to modify them independently, he is able to cause a source-binary inconsistency as explained in detail in the next section.

V. ATTACK DESCRIPTION

As in any typical injection attack, we patch our malicious code, *Time-of-Day* interrupt block OB10, in the original control logic of the target PLC. The CPU checks whether the condition of the interrupt is met in each single execution cycle. Meaning that, the attacker's interrupt block will be always 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:

1) The date of 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.
2) The date of the CPU's clock does not match the date set in OB10. The CPU resumes to execute OB1 after checking the interrupt condition without activating the interrupt and without executing the instructions in OB10.

Our attack approach presented in this paper is comprised of two main phases: the patching phase (online phase), and the attack phase (offline phase). Please note that, getting the IP address, MAC address, and model of the victim PLC is an easy task by running our PN-DCP protocol based scanner presented in [5] or other network scanners that can obtain all the information that the attacker needs to communicate with the target device.

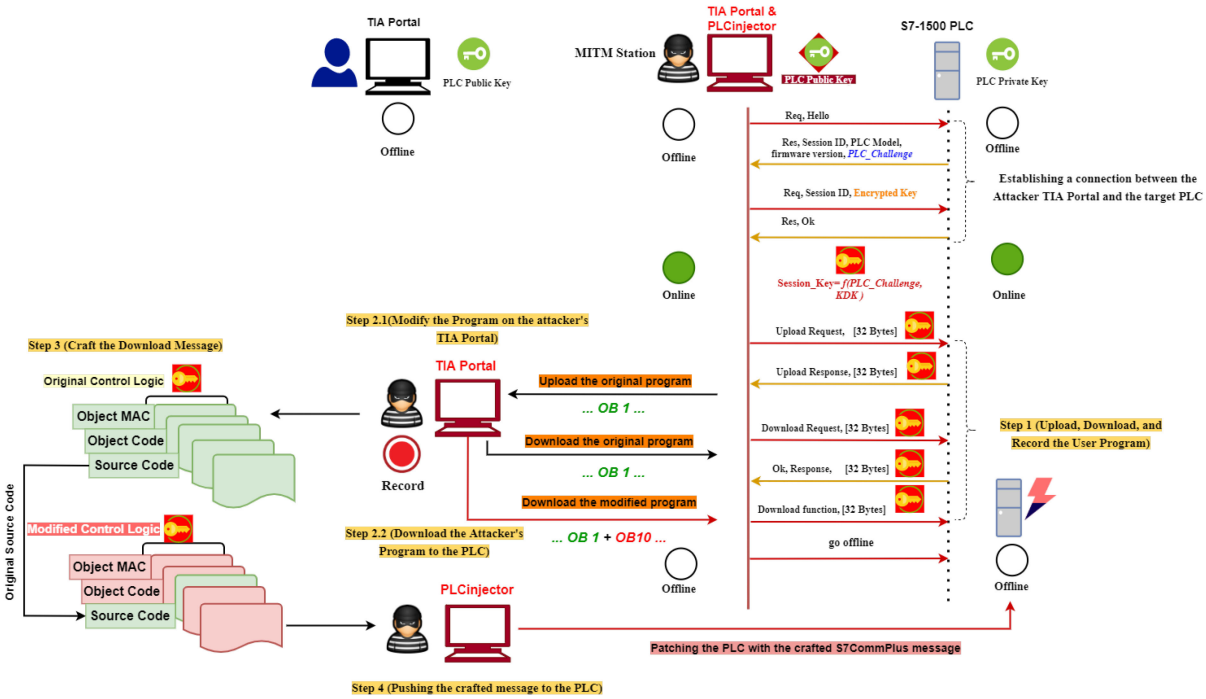

FIG. 10. **High-level overview of the patching phase.**

A. PATCHING PHASE

Fig. 10 shows a high-level overview of this phase. We aim at injecting the PLC with our malicious instructions programmed in the interrupt block OB10. This phase consists of four steps:

a) Uploading and downloading the user's program.
b) Modifying and updating the control logic program.
c) Crafting the S7CommPlus download message.
d) Pushing the attacker's message to the victim PLC.

To patch the target PLC, we utilize our MITM station which has two main components:

1) *A TIA Portal:* to retrieve and modify the current control logic program that the PLC runs.

2) *A PLCinjector:* to download the attacker's code to the PLC. In this work, we developed a python script based on the Scapy[5] library for this purpose.

For a realistic scenario, there are two possible cases that an attacker might encounter after accessing the network.

1) 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 & Downloading the User's Program: In this step, we aim at obtaining the decompiled control logic program that the PLC runs, and the S7CommPlus message that the TIA Portal sends to download the original user program into the PLC. For achieving these goals, we open first the attacker's TIA Portal and establish a connection with the victim PLC directly. This is possible due to a security gap in the S7-1500 PLC design. In fact, the PLC does not introduce any security check to ensure that the currently communicating TIA Portal is the same TIA Portal that it communicated with in an earlier session. For this, any external adversary provided with a TIA Portal on his machine can easily communicate with an S7 PLC without any effort.

After successfully establishing the communication, we upload the control logic program on the attacker's TIA Portal. Then we re-download it once again to the PLC and sniff the entire S7CommPlus messages flow exchanged between the attacker's TIA Portal and the victim PLC using the Wireshark software. At the end of this step, the attacker has the program on his TIA Portal, and all the captured download messages saved in a Pcap file for a future use (explained in step 3).

Step 2. Modifying & Updating the PLC's Program: After retrieving the user program that the target PLC runs, the attacker's TIA Portal displays it in one of the high-level programming languages that it was programmed with (e.g., SCL). Based on our understanding to the physical process controlled by the PLC, we configure and program our *Time-of-Day* interrupt block OB10 to force certain outputs of the system to switch off once the interrupt is being activated (shown later in Fig. 13). Although our malicious code differs from the original code with only an extra small size block (OB10), it is sufficient to confuse the physical process of our experimental set-up.

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. But, the ICS operator could easily reveal the modification by uploading the program from the infected PLC, and

[5]https://scapy.net/

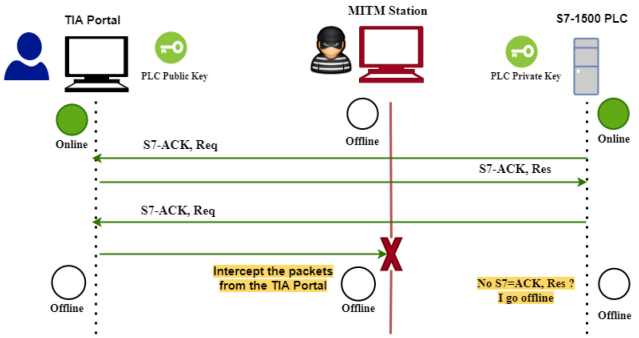

FIG. 11. Closing the online session using MITM Approach.

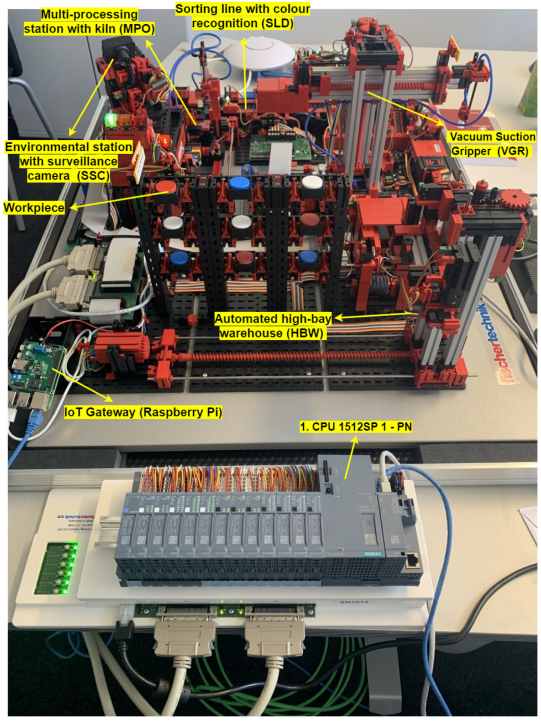

FIG. 12. Experimental Set-up.

comparing the offline and online programs running on his legitimate TIA Portal and the remote PLC respectively.

Step 3. Crafting the S7CommPlus Download Message: To hide our infection from the legitimate user, we first recorded the S7CommPlus messages exchanged between the attacker's TIA Portal and the PLC while downloading the modified program. As mentioned earlier in Section IV.B, each download message has objects and attributes see Fig. 9. The *ProgramCycleOB* object is dedicated to create a program cycle block in the PLC's memory and has three different attributes:

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

The *Object Code* is the code that the PLC reads and processes, whilst the *Source Code* is the code that the TIA Portal

```
1   // reset VerticalAxis Up and Down
2   "QX_VGR_M1_VerticalAaxisUP_Q1" := FALSE
3   "QX_VGR_M1_VerticalAaxisDown_Q2" := FALSE
4   // reset HorizontalAxis Forward and Backward
5   "QX_VGR_M2_HorizontalAxisBackward_Q3" := FALSE
6   "QX_VGR_M2_HorizontalAxisBackward_Q4" := FALSE
7   // reset Axis_rotate
8   "QX_VGR_M3_RotateClockwise_Q5" := FALSE
9   "QX_VGR_M3_RotateCounterclockwise_Q6" := FALSE
10  // turn off compressor
11  "QX_VGR_Compressor_Q7" := FALSE
12  // turn off vacuum
13  "QX_VGR_ValveVacuum_Q8" := FALSE
```

FIG. 13. The malicious instructions in OB10.

decompiles, reads, and displays for the user. Therefore, all what is required to show the user the original code is to modify the S7CommPlus 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 precaution measure. Hence, it can detect any manipulation and refuses to update its program in case the attributes do not belong to the same session. But surprisingly, this holds true only for the *Object MAC* and the *Object Code* attributes. Meaning that, to make the PLC accept the crafted message, our crafted S7CommPlus download message must always have the *Object MAC* and the *Object Code* attributes from the same session, whilst the *Source Code* attribute could be substituted with another attribute from a different session i.e., from a pre-recorded session. All the captured packets containing the attributes of the *ProgramCycleOB* for both the user and attacker programs are presented in the Appendix.

Step 4. Pushing the crafted message to 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. As S7CommPlusV3 exchanges a shared session key between the TIA Portal and the PLC to prevent performing replay attacks, we first need to bundle the packet with a correct key before we push the crafted message to the PLC. However, exploiting the shared key is out of the scoop of this paper, but it is explained in details in [30]. Once the malicious key exchange is completed, we can easily bundle the key bytecodes with our crafted message. Taking into consideration the appropriate modification to the session ID and the integrity fields, we store the final S7 message (the attacker's message) in a pcap file for pushing it back to the PLC as a replay attack. Algorithm 1 describes the main core of our PLCinjector tool that we use to patch the PLC with the attacker's download message.

The PLCinjector tool has two functions. The first one is utilized to exploit the integrity protection session key that S7CommPLusV3 uses. The session key exchanged in each session between the TIA Portal and S7-1500 PLCs originates from combining 16 bytes of the PLC's *ServerSessionChallange*, precisely the ones located between the bytes 2 and 18, with a random 24-byte *KDK* that the TIA Portal chooses. Afterwards, a fingerprinting function *f ()* is used within the *sessionKey* calculation. Line 5 generates a 24 bytes random quantity (*M*), and maps it to the elliptic curve's domain donated as *PreKey*. From the random point *PreKey*, we use a *Key Derivation Function (KDF)* to derive 3-16 bytes quantities identified as follows: *Key Encryption Key (KEK), Checksum Seed (CS)* and *Checksum Encryption Key (CEK)*. In line 7, the *CS* generates 4096 pseudo-random bytes organized as four 256-word, namely *LUT*. This *LUT* is used to calculate a checksum over the *KDK* and *PLC_Challenge*. Lines from 8 to 13 depict the elliptic curve key exchange method similar to the one that the TIA Portal uses to encrypt the random generated *PreKey*. After that, we mask the elliptic curve calculations with 20 bytes chosen randomly (donated to *x* in the algorithm). Line 19 provides an authenticated encryption for the encrypted *KDK*. Here a non-cryptographic checksum is computed, then encrypted by *AES-ECP* function. Finally, we add 2 header fields including key fingerprints i.e., 8-byte truncated *SHA256* hashes of the relevant key with some additional flags see line 20.

After establishing a successful session with the victim, the PLC exchanges the malicious generated *Session_Key* with the attacker machine along the current communicating session. In the next step, our tool executes function 2 to send the attacker's crafted S7 message that contains both the malicious code combined with the generated *Session_Key*. Our attacking tool can be also used against all the S7-1500 PLCs sharing the same firmware. This is due to the fact that Siemens has designed its new S7 key exchange mechanism assuming that all devices running the same firmware version use also the same public-private key pair mechanism [30].

After a successful injection, the PLC updates its program, processing the *Object Code* of the attacker's program while it saves the *Source Code* of the user's program in its memory. Therefore, whenever the user uploads the program from the infected PLC, the TIA Portal will recall, decompile, and display the original program. This kept our injection hidden inside the PLC and the user could not detect any difference between the online and offline programs.

Algorithm 1: PLCinjector Tool.

Function 1 Get_Session_Key ((*ServerSessionChallenge*))

1: Checksum = 0
2: *PLC_Challenge* = *ServerSessionChallenge*[2:18]
3: *KDK* = prng (24)
4: *Session_Key* = HMAC-SHA256 (*f* (*Challenge*,8)) [:24]
5: *PreKey* = *M* (prng(24))
6: *KEK,CEK,CS* = *KDF (PreKey)*
7: *LUT[4][256]* = *hash-init (CS)*
8: **while** point == ∞**do**
9: *x* = prng(20)
10: *point* = *fx* (G, y, Nonce)
11: *EG2* = *y* (point)
12: **end while**
13: *EG1* = *add* (s, *PreKey*)
14: **for** block in E(*KDK*)**do**
15: Checksum = *hash (checksum)* ⊕ *block, LUT[4][256]*
16: **end for**
17: Checksum[12] = Checksum[12] ⊕ 40
18: final_Checksum = *hash* (Checksum, *LUT[4][256]*)
19: key = *AES-ECB* (final_Checksum)
20: KEY = *SHA256(key[:24]* || *"DERIVE" [:8])*
21: Return KEY

END Function 1

Function 2 Replay (pcapfile, Ethernet_interface, SrcIP, SrcPort)

22: RecvSeqNum = 0
23: SYN = TRUE
24: **for** pkt in rdpcap (Pcapfile)**do**
25: IP = packet [IP]
26: TCP = packet[TCP]
27: delete IP.checksum
28: IP.src = SrcIP
29: IP.Port = SrcPort
30: **if** TCP.flags == Ack or TCP.flags == RSTACK**then**
31: TCP.ack = RecvSeqNum+1
32: **if** sendp(packet, iface=Ethernet_interface)**then**
33: SYN = False
34: Continue
35: **end if**
36: **end if**
37: Recv = Srp1(packet, iface=Ethernet_interface)
38: RecvSeqNum = rcv[TCP].seq
39: **end for**

END Function 2

2) 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 S7 PLC, by default, allows only one active online session, an attacker is not able to communicate with the PLC. It will immediately refuse any attempt to establish a connection as it is already communicating with the user. For such a scenario, the attacker needs first to close the current online session between the legitimate user and the PLC, before patching his malicious code.

A user can establish an online session with an S7 PLC by enabling the "go online" feature in the TIA Portal software. Then he can control, monitor, diagnose, download, upload,

start, and stop the CPU remotely. Once the user has established an online connection with the PLC, the two parties (the TIA Portal and PLC) start exchanging a specific message along the session regularly. This message is known as *S7-ACK*, and in charge of keeping the session alive. The TIA Portal must always respond to any *S7-ACK* request sent by the PLC with a *S7-ACK* replay message. Therefore, for closing the current online session we run our MITM station (presented in [3]) that allows us to intercept and drop all packets sent from the TIA Portal, by performing the well-known ARP poisoning approach. If the PLC does not receive a response from the TIA Portal right after sending an acknowledgment request, it will close the connection with the connected TIA Portal and both go offline. Fig. 11 describes this scenario.

It is worth mentioning that, an attacker can also use different ways to close the connection e.g., port stealing, replay attack with "go offline" packets, etc. After both the legitimate TIA Portal and the victim PLC turned offline, the attacker can easily establish a new session with the PLC, using his own TIA Portal. Then he patches 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 was forced to close the session with the PLC. Meaning that, the user can see obviously that he lost the connection with the remote device. In case he attempts to re-connect to the PLC while it is connected to the attacker's TIA Portal, the PLC will refuse his connection request. Our investigations showed that there is no way to re-connect the legitimate TIA Portal to the victim PLC after patching the PLC, unless the ICS operator himself enables "go online" on his TIA Portal. This abnormal disconnection between the two parties is the only effect of our patch in this scenario.

B. ATTACK PHASE

After a successful injection, the attacker goes offline and closes the current communication session with the target PLC. With the next execution cycle, the attacker's program will be executed in the PLC. Meaning that, the interrupt condition of the malicious interrupt block OB10 will be checked in each execution cycle. This block remains in idle mode, and hidden in the PLC's memory as long as the interrupt condition is not met. Once the configured date and time of the attack 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 OB10 with. In our application example, we programmed the OB10 to force certain motors to turn off at a certain time and date when we are completely disconnected from the target's network.

VI. EVALUATION, DISCUSSION, AND MITIGATION

In this section, we present the implementation of our attack approach, and assess the service disruption of the physical process due to our patch. Afterwards, we discuss our results and suggest possible mitigation methods to protect systems from such a threat.

A. LAB SETUP

For evaluating our attack approach, we used the Fischertechnik training factory shown in Fig. 12. It consists of industrial modules such as storage and retrieval stations, vacuum suction grippers, high-bay warehouse, multi-processing station with kiln, a sorting section with color detection, an environment sensor and a pivoting camera. The entire factory is controlled by a SIMATIC S7-1512SP with a firmware V2.9.2, and programmed by TIA Portal V16. The PLC connects to a TXT controller via an 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 experiment provides two industrial processes. Storing and ordering materials. The default process cycle begins with storing and identifying the material i.e., workpiece. The factory has an integrated NFC tag sensor storing production data that can be read out via an RFID NFC module. This allows the user to trace the workpieces digitally. The cloud displays the part's colour and its ID-number. Afterwards, the vacuum gripper places suction on the material and transports it to the high bay warehouse which applies a first-in first-out principle for the 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 is written on the material at the end of the production process, and the finished product will be provided for collection.

B. IMPLEMENTATION

In our experiment, we found that the vacuum suction gripper (VGR) is involved in all the industrial processes that the Fischertechnik system operates. Therefore, if we could disrupt its functionality, then the entire system would be impacted. The VGR module moves with the help of 8 mini motors: vertical motor up (%Q2.0), vertical motor down (%Q2.1), horizontal motor backwards (%Q2.2), horizontal motor forwards (%Q2.3), turn motor clockwise (%Q2.4), turn motor anticlockwise (%Q2.5), compressor (%Q2.6), and valve vacuum (%Q2.7). Therefore, for exploiting the VGR, we programmed our OB10 to force all the 8 motors to switch off at the point zero for the attack as shown in Fig. 13.

After patching the PLC with our malicious block, and before the *Time-of-Date* interrupt being activated, we did not record any physical impact and the Fischertechnik system keeps operating normally. Once the CPU clock matches the attack time that we set, we noticed that the VGR module stopped moving. Furthermore, the workpiece that is being transported by the gripper has fallen down, as the compressor, which provides the appropriate air flow to carry the good,

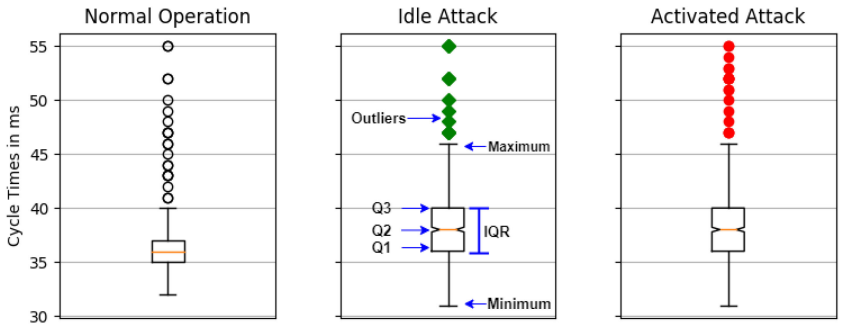

FIG. 14. **Boxplot presenting the measured execution cycle times of OB1.**

was turned off. This led to an incorrect operation, and the movement sequence of the workpieces was disrupted. For a real-world heavy factory e.g. automobile manufacturing industry, such an attack scenario might be seriously dangerous and even cost human lives.

C. EVALUATION

To assess the impact of our patch on the physical process controlled by the infected device accurately, we measured and analyzed the differences of 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 being activated i.e., the PLC is running the attacker's program.
- *Activated Attack:* after the interrupt is being executed.

Siemens PLCs, by default, store the time of the last execution cycle in local variable of OB1 called *OB1_PREV_CYCLE*. Therefore, we added a small SCL code snippet to our control program which stores the last cycle time in a separate data block. Then we recorded 4096 execution cycle times for each scenario, calculated the arithmetic median value, and used the *Kruskal-Wallis* and the *Dunn's Multiple Comparison* test for statistical analysis. All the results are presented as boxplots in Fig. 14. In order to make our resulting boxplots clearer and easier to read, we define the following parameters:

1) 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).
2) Median (Q2): represents the middle value of the total recorded values.
3) Third quartile (Q3): represents the middle value between the highest value and the median of the total values recorded.
4) Interquartile Range (IQR): represents all the values between 25% to 75% of the total recorded values.
5) Maximum: represents *Q3 + 1.5*IQR*
6) Minimum: represents *Q1 - 1.5*IQR*
7) Outliers: represents all the values that they are higher and lower than the maximum and minimum values respectively.

Our measurements show that the calculated median value (Q2) of executing the OB1 for the infected program is approx. 38 ms, and differs slightly 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. Meaning that, checking the interrupt condition of our malicious block in each execution cycle does not disrupt executing 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. Our measurements clearly show that our 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. But it keeps checking the condition of the interrupt in each cycle as long as OB10 is existing in the control logic program. However, our approach allows attackers to adjust the repeating of the interrupt (see Section III), as well as to program the interrupt block on their will causing different impacts in the physical process of the target system.

D. DISCUSSION

Based on our analysis, 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 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 PLC is running a different one. This is due to the fact that the original *Source Code* attribute is always sent back to the TIA Portal whenever the user requires the program from the infected PLC. Due to all that, 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 certain reason to do so.

The success of our attack approach on S7-1500 PLCs is, indeed, based on serious design vulnerabilities in the newest model of S7 PLCs and security issues in the integrity mechanism used in the latest version of the S7CommPlus protocol. We found that the PLC does not authenticate the TIA Portal as we expected, and only confirms the session freshness. This allows an external attacker to perform replay attacks against the PLC, keeping in mind that he has always to provide the correct *Session_Key* in his 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 resilient against replay attacks, but unfortunately we could maliciously update the PLC's program by sending a crafted S7 download message.

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 it is currently-communicating with, is the same TIA Portal than in 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. It is also noticed that Siemens provides its 1500 CPUs with a sophisticated integrity checking algorithm which checks the validity of any S7 message received. But 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.

E. MITIGATION

The fundamental solution would be completely redesigning the integrity check mechanism that the newest S7 PLCs use. The new mechanism should include a security pairing and mutual authentication between the PLC and TIA Portal. But 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 software updated on time, and have a very long life-cycle compared to common IT devices. For all that, we should expect that insecure devices will keep employed in real-world ICS environments for a long time. In this term, network detection can be seamlessly integrated into the existing ICS setting. In particular, control logic detection [36], and verification [41], [42] can be utilized to alleviate current situation. As our injection was hidden in the PLC memory, so partitioning the memory space and enforcing memory access control [37] could also be a reasonable solution. Other suggestions include employing standard cryptography methods such as digital signatures (for messages like control logic manipulation), but also using network monitoring tools like snort [38], ArpAlert [39], and ArpWatchNG [40] for revealing any attack involving MITM attacks. Furthermore, a mechanism to check the protocol header which contains information about the type of the payload is also recommended as a solution to detect and block any potential unauthorized transfer of the control logic. However, from our perspective the best solution to prevent injection attacks is to separate the information technology (IT) domain from operational technology networks by using a Demilitarized Zone (DMZ).

VII. CONCLUSION

This paper presented a new threat on the newest SIMATIC PLCs. Our attack approach is based on injecting the attacker's malicious code once he gains access to the target's network, but activating his patch later without a need to be connected at the time of the attack. Our investigation identified a few design vulnerabilities in the new integrity method that the S7-1500 PLCs use. Based on our findings, we managed successfully to conduct an injection attack, by patching the tested PLC with a *Time-of-Day* interrupt block (OB10). This block allows us to activate our patch, and to confuse the physical process without being connected to the victim at the point zero for the attack. We analyzed and evaluated the possibility of revealing our injection by the ICS operator. Our experimental results showed that the original control logic program is always shown to the user, whilst the PLC runs the attacker's program. In addition, our injection does not increase the execution times of the control logic. Hence, the physical process is not impacted when our patch is in idle mode. To summarize, our attack is a very serious threat targeting ICSs, as attackers need to be only online during the patch and can close all the connections to the target's network afterwards. Therefore, they will not be detected even if the ICS operators re-activate the security measure. Finally, we provided some recommendations to secure ICSs from such a severe threat.

Our attack approach is feasible for all S7-1500 PLCs with a firmware 2.9.2 or lower. However, Siemens updated the firmware for all S7-1500 CPUs in December, 2021 to the newer version 2.9.4. Therefore, a further investigation is required to test the security of the latest firmware version. Furthermore, a deeper analyzes of the advanced S7CommPlus protocol aiming at understanding the private key mechanism that PLCs implement can be also be a part of future works. We believe that, if attackers manage successfully to extract the

private key from an S7-1500 PLC, then stronger attacks e.g., fully man in the middle, session-hijacking, and impersonation PLC attacks might become possible for the entire products line.

VII. APPENDIX. PACKETS CAPTURE

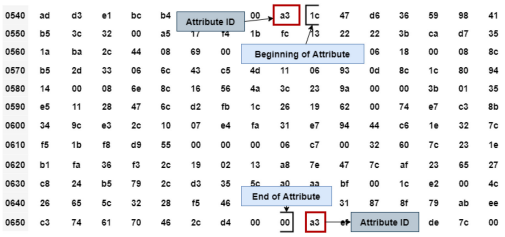

FIG. 15. Object MAC Attribute - User Program.

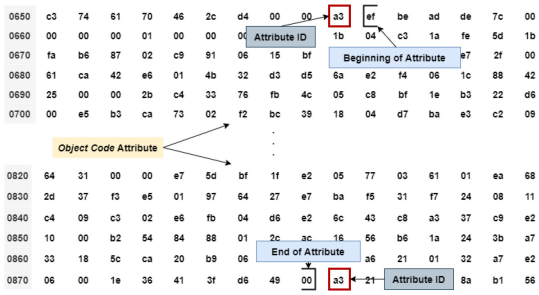

FIG. 16. Object Code Attribute - User Program.

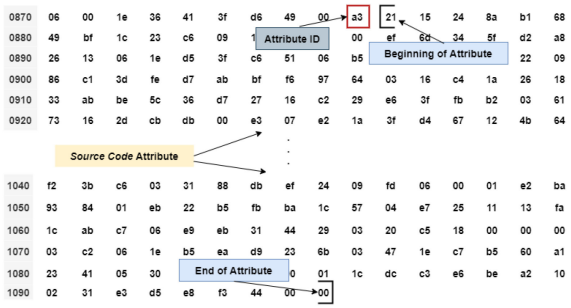

FIG. 17. Source Code Attribute - User Program.

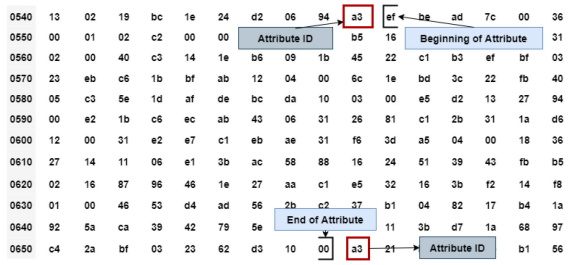

FIG. 18. Object MAC Attribute - Attacker Program.

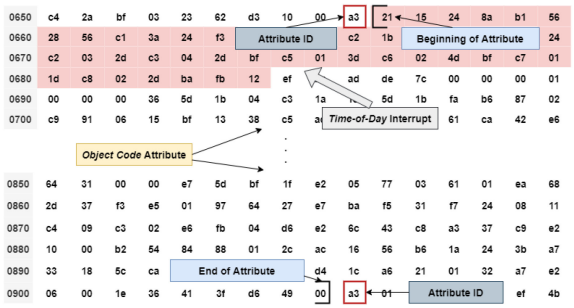

FIG. 19. Object Code Attribute - Attacker Program.

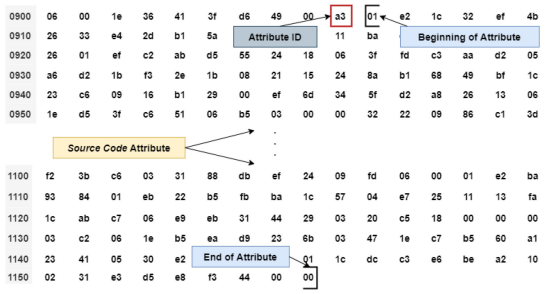

FIG. 20. Source Code Attribute - Attacker Program.

REFERENCES

[1] W. Alsabbagh and P. Langendörfer, "A fully-blind false data injection on PROFINET i/o systems," in *Proc. IEEE 30th Int. Symp. Ind. Electron.*, 2021, pp. 1–8.

[2] H. Wardak, S. Zhioua, and A. Almulhem, "PLC access control: A security analysis," in *Proc. World Congr. Ind. Control Syst. Secur.*, 2016, pp. 1–6.

[3] W. Alsabbagh and P. Langendörfer, "A stealth program injection attack against S7-300 PLCs," in *Proc. 22nd IEEE Int. Conf. Ind. Technol.*, 2021, pp. 986–993.

[4] D. Beresford, "Exploiting siemens simatic S7 PLCs," in *Black Hat USA*, 2011, pp. 723–733.

[5] W. Alsabbagh and P. Langendörfer, "A remote attack tool against siemens S7-300 controllers: A practical report," in *11. Jahreskolloquium Kommunikation in der Automat.*, 2020.

[6] J. Klick, S. Lau, D. Marzin, J. Malchow, and V. Roth, "Internet-facing PLCs-a new back orifice," in *Black Hat USA*, 2015, pp. 22–26.

[7] A. Spenneberg, M. Brüggemann, and H. Schwartke, "PLC-blaster: A worm living solely in the PLC," in *Black Hat Asia Marina Bay Sands*, 2016, pp. 1–16.

[8] N. Govil, A. Agrawal, and N. O. Tippenhauer, "On ladder logic bombs in industrial control systems," in *Proc. Int. Workshop Secur. Ind. Control Syst. Cyber-Physical Syst.*, 2018, pp. 110–126.

[9] K. Sushma, A. Nehal, Y. Hyunguk, and A. Irfan, "CLIK on PLCs! Attacking control logic with decompilation and virtual PLC," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, [Online]. Available: https://ruoyuwang.me/bar2019/pdfs/bar2019-final74.pdf.

[10] W. Alsabbagh and P. Langendörfer, "Patch now and attack later—exploiting S7 PLCs by time-of-day block," in *Proc. 4th IEEE Int. Conf. Ind. Cyber-Phys. Syst.*, 2021, pp. 144–151.

[11] W. Alsabbagh and P. Langendörfer, "A control injection attack against S7 PLCs—manipulating the decompiled code," IECON 2021 Proc. 47th Annu. Conf. IEEE Ind. Electron. Soc., Toronto, ON, Canada, Oct., 2021, pp. 1–8.

[12] N. Falliere, "Exploring Stuxnet's PLC infection process," in *Virus Bulletin Covering Global Threat Landscape Conf.*, Sep. 2010, [Online]. Available: http://www.symantec.com/connect/blogs/exploringstuxnet-s-plc-infection-process.

[13] Y. Hyunguk and A. Irfan, *Control Logic Injection Attacks on Industrial Control Systems*. Berlin, Germany: Springer, 2019.

[14] L. Garcia *et al.*, "Hey my malware knows physics! Attacking PLCs with physical model aware rootkit," *Proc. 24th Ann. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–15, doi: 10.14722/ndss.2017.23313.
[15] Z. Basnight *et al.*, "Firmware modification attacks on programmable logic controllers," *Int. J. Crit. Infrastructure Protection*, vol. 6, pp. 76–84, 2013.
[16] *Attackers Deploy New ICS Attack Framework "TRITON," and Cause Operational Disruption to Critical Infrastructure*. Accessed: Apr. 12, 2021. [Online]. Available: https://www.fireeye.com/blog/threat-research/2017/12/attackers-deploy-new-ics-attack-framework-triton.html
[17] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the cyber-attack on the ukrainian power grid," Tech. Rep., SANS E-ISAC, Mar. 18, 2016. [Online]. Available at: https://ics.sans.org/media/ESAC_SANS_Ukraine_DUC_5.pdf
[18] S. Senthivel *et al.*, "Denial of engineering operations attacks in industrial control systems," in *Proc. 18th ACM Conf. Data Appl. Secur. Privacy*, 2018 pp. 319–329.
[19] G. liang, S. R. Weller, J. Zhao, F. Luo, and Z. Y. Dong, "The 2015 Ukraine blackout: Implications for false data injection attacks," *IEEE Trans. Power Syst.*, vol. 32, no. 4, pp. 3317–3318, Jul. 2017.
[20] N. Falliere, L. O. Murchu, and E. Chien, "W32. Stuxnet Dossier," Symantec Corp., Security Response, Tempe, AZ, USA, White Paper, 2011.
[21] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Secur. Privacy*, vol. 9, no. 3, pp. 49–51, May/Jun. 2011.
[22] T. De Maiziére, "*Die Lage Der IT-Sicherheit in Deutschland* 2014," The German Federal Office for Information Security, *German Federal Office Inf. Secur.*, 2014. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2014.pdf
[23] Siemens ProductCERT and Siemens CERT, "Security advisory," *Pers. commun.*, 2019. [Online]. Available: https://new.siemens.com/global/en/products/services/cert.html
[24] IPCS Automation, "Market share of different PLCs," 2018, [Online]. Available: https://ipcsautomation.com/blog-post/market-share-of-different-plcs/
[25] S. Frances, "Top 20 programmable logic controller manufacturers," *Robotics Automation News*, 2020. [Online]. Available: https://roboticsandautomationnews.com/2020/07/15/top-20-programmable-logic-controller-manufacturers/33153/
[26] Statista Research Department, "Programmable logic controllers: Global manufacturer market share 2017," 2021, [Online]. Available: https://www.statista.com/statistics/897201/global-plc-market-share-by-manufacturer/
[27] 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.
[28] National Institute of Standards and Technology, "CVE-2019-10929," National Vulnerability Database, 2019, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-10929
[29] T. Wiens, "S7comm wireshark dissector plugin," *SourceForge*, 2011. [Online]. Available: http://sourceforge.net/projects/ s7commwireshark
[30] E. Biham, S. Bitan, A. Carmel, A. Dankner, U. Malin, and A. Wool, "Rogue7: Rogue engineering-station attacks on S7 simatic PLCs," in *Black Hat USA*, 2019, [Online]. Available: https://i.blackhat.com/USA19/Thursday/us-19-Bitan-Rogue7-Rogue-Engineering-Station-AttacksOn-S7-Simatic-PLCs-wp.pdf.
[31] C. Lei, L. Donghong, and M. Liang, "The spear to break the security wall of S7CommPlus," in *Black Hat USA*, 2017, [Online]. Available: https://www.blackhat.com/docs/eu-17/materials/eu-17-Lei-The-Spear-ToBreak%20-The-Security-Wall-Of-S7CommPlus-wp.pdf.
[32] H. Hui and K. McLaughlin, "Investigating current PLC security issues regarding siemens S7 communications and TIA poral," in *Proc. Ind. Control Syst. Cyber Secur. Res.*, 2018, pp. 67–73.
[33] A. Menezes and S. Vanstone, "Elliptic curve cryptosystems and their implementation," *J. Cryptol.*, vol. 6, pp. 209–224, 1993.
[34] F. Weißerg, "Analysis of the S7CommPlus protocol in terms of cryptography used," (in German), Mar. 26, 2018. [Online]. Available: https://www.os-s.net/publications/thesis/Bachelor_Thesis_Weissberg.pdf
[35] A. Ayub, H. Yoo, and I. Ahmed, "Empirical study of PLC authentication protocols in industrial control systems," in *Proc. IEEE Secur. Privacy Workshops*, 2021, pp. 383–397.
[36] H. Yoo, S. Kalle, J. Smith, and I. Ahmed, "Overshadow PLC to detect remote control-logic injection attacks," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, 2019, pp. 109–132.
[37] 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.
[38] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks," *Lisa*, vol. 99, no. 1, pp. 229–238, 1999.
[39] C. H. Kim *et al.*, "Securing real-time microcontroller systems through customized memory view switching," *Network Distributed Syst. Security (NDSS) Symp.*, 2018, doi: 10.14722/ndss.2018.23117.
[40] C. Leres *et al.*, "arpwatch Description," KaliTools, 2021, [Online]. Available: https://en.kali.tools/?p=1411.
[41] S. Zonouz, J. Rrushi, and S. McLaughlin, "Detecting industrial control malware using automated PLC code analytics," *IEEE Secur. Privacy*, vol. 12, no. 6, pp. 40–47, Nov./Dec. 2014.
[42] M. Zhang *et al.*, "Towards automated safety vetting of PLC code in real-world plants," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 522–538.
[43] Siemens, "S7-300 CPU 31xC and CPU 31x: Technical specifications," 2011. [Online]. Available: https://cache.industry.siemens.com/dl/files/906/12996906/att_70325/v1/s7300_cpu_31xc_and_cpu_31x_manual_en-US_en-US.pdf
[44] H. Hui, K. McLaughlin, and S. Sezer, "Vulnerability analysis of S7 PLCs: Manipulating the security mechanism," *Int. J. Crit. Infrastructure Protection*, vol. 35, 2021, Art. no. 100470.
[45] S. Mclaughlin, "On dynamic malware payloads aimed at programmable logic controllers," in *HotSec*, 2011, [Online]. Available: http://www.stephenmclaughlin.org/hotsec-2011.pdf.
[46] S. McLaughlin and P. McDaniel, "SABOT: Specification-based payload generation for programmable logic controllers," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 439–449.
[47] A. Serhane, M. Raad, R. Raad, and W. Susilo, "PLC code-level vulnerabilities," in *Proc. Int. Conf. Comput. Appl.*, 2018, pp. 348–352.
[48] S. E. Valentine, "PLC code vulnerabilities through scada systems," Ph.D. dissertation, Univ. South Carolina, 2013. [Online]. Available: https://scholarcommons.sc.edu/etd/803
[49] Siemens, "SIMATIC STEP 7 Basic/Professional V16 and SIMATIC WinCC V16," 2019, [Online]. Available: https://support.industry.siemens.com/cs/document/109773506/simatic-step-7-basic-professional-v16-and-simatic-wincc-v16?dti=0&lcn-WW

WAEL ALSABBAGH (Member, IEEE) received the B.S. and M.S. degrees in automatic control and computer engineering from Al-baath University, Homs, Syria, in 2012 and 2015, respectively. He is currently working toward the Ph.D. degree in computer science with the Technical University of Cottbus, Cottbus, Germany. Since 2018, he has been a Scientist with the IHP-Leibniz-Institut für Innovative Mikroelektronik, Frankfurt (Oder), Germany. His research interests include the cyber-attacks and security, mitigation methods of the attacks targeting industrial control systems, and supervisory control and data acquisition.

PETER LANGENDÖERFER received the Diploma and Ph.D. degrees in computer science. Since 2000, he has been with the IHP-Leibniz-Institut für Innovative Mikroelektronik, Frankfurt (Oder), Germany. In the IHP-Leibniz-Institut für Innovative Mikroelektronik, he is leading the Wireless Systems Department. From 2012 to 2020, he was leading the Chair for security in pervasive systems with the Technical University of Cottbus-Senftenberg, Cottbus, Germany. Since 2020, he owns the chair wireless systems with the Technical University of Cottbus-Senftenberg. He has authored or coauthored more than 150 refereed technical articles, filed 17 patents of which ten have been granted already. His research interests include security for resource constraint devices, low power protocols, and efficient implementations of AI means and resilience. He was a Guest Editor of many renowned journals, such as *Wireless Communications and Mobile Computing* (Wiley) and *ACM Transactions on Internet Technology*.