IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 6 January 2024, accepted 16 January 2024, date of publication 19 January 2024, date of current version 25 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3356051

RESEARCH ARTICLE

Investigating the Security of OpenPLC: Vulnerabilities, Attacks, and Mitigation Solutions

WAEL ALSABBAGH, (Member, IEEE), CHAERIN KIM, AND PETER LANGENDÖRFER
IHP—Leibniz-Institut für Innovative Mikroelektronik, 15236 Frankfurt (Oder), Germany
Wireless Systems, Brandenburg University of Technology Cottbus–Senftenberg, 03046 Cottbus, Germany

Corresponding author: Wael Alsabbagh (alsabbagh@ihp-microelectronics.com)

This work was supported in part by the German Federal Office for Information Security as part of the Echtzeitfähige Machine-Learning-Lösungen für resiliente und sichere 5G/6G-Netze am Beispiel von Automatisierungsanwendungen (EMiL) project under Grant 01MO23014C.

ABSTRACT Open-source Programmable Logic Controller (OpenPLC) software is designed to be vendor-natural and run on almost any computer or low-cost embedded devices e.g., Raspberry Pi, Ardunio, and other controllers. The aim of this project is to introduce an affordable and practical alternative solution for the high-cost of real hardware PLCs, and has successfully gained substantial interest within both the research and industrial communities. Due to its popularity grows, understanding its security vulnerabilities and implementing effective mitigation strategies become crucial. Through a combination of threat modeling, vulnerability analysis, and practical experiments, this article provides valuable insights for developers, researchers, and engineers aiming to deploy OpenPLC securely in industrial environments. To this end, we first conducted an in-depth analysis aimed to shed light on various security challenges and vulnerabilities within the OpenPLC project. These encompass issues such as unauthorized access, vulnerabilities in communication protocols, concerns regarding data integrity, the absence of robust encryption mechanisms, etc. After that, we showed the research community what the consequences of those vulnerabilities would be if they are exploited. To this end, we performed a sophisticated control logic injection attack that maliciously modifies the user program run on the OpenPLC *Runtime*. Our injection was stealthy and not detected by the legitimate user. Finally, we introduced a security-enhanced OpenPLC software called OpenPLC Aqua. Our developed software is equipped with a set of security solutions designed specifically to address the vulnerabilities to which current OpenPLC versions are prone. All our attack codes as well OpenPLC Aqua software are publically available.

INDEX TERMS OpenPLC, OpenPLC Aqua, SCADA, vulnerabilities, cyberattacks, cybersecurity, mitigation solutions.

I. INTRODUCTION

Industrial Control Systems (ICSs) play a crucial role in automating critical and complex physical processes across various domains, including production lines, electrical power grids, oil and gas facilities, petrochemical plants, and others. These ICS environments typically consist of two fundamental components: a control center and a field site. Within the control center, a suite of ICS services operates, including Human Machine Interfaces (HMIs) and engineering workstations. In contrast, at the field site, a network of sensors, actuators, and Programmable Logic Controllers (PLCs) is deployed to locally monitor and control physical processes [1]. The engineering station contains an appropriate software to configure and program PLCs. It uses specialized programming software tailored to the specific PLC vendor. This software empowers engineers to create control logics, which determine how PLCs should operate and maintain the physical processes at their intended operational state [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

It is worth mentioning that there are two categories of PLCs in the industrial market as shown in figure 1. In the following, we explore each category in more detail.

Hardware PLCs: Hardware PLCs, or traditional PLCs, are physical devices designed for industrial control and automation. They consist of dedicated hardware components, including a processor, input/output modules, and communication interfaces. The hardware PLCs have a fixed proprietary hardware architecture and execute control logics that are completely coupled to the hardware. Such devices are programmed using vendor-specific engineering software which is also tightly coupled to a single or very limited type of hardware from a single vendor. Hardware PLCs are often employed in medium- and large-scale industrial environments and provide deterministic as well as real-time control for complex physical processes. A good example of such PLCs is Rockwell CompactLogix 5480. This device has a virtualized Logix control engine which is tightly coupled to the very certain hardware.

Software PLCs:

Software PLCs are classified into two sub-categories based on how hardware and software are partially/completely decoupled from each other as follows.

- "Hardware specific" software PLCs:

This refers to software-based PLCs designed to run on specific hardware configurations from the same vendor. Such PLCs execute control logics that are partially decoupled from the hardware, which allows users to run these control logics, seamlessly, on different hardware models produced by the same manufacturer. However, this type of PLCs still has some compatibility limitations when used with hardware devices from the same vendor e.g., devices from two PLC generations (non-cryptographically and cryptographically PLCs) [2], and not compatible with hardware devices from different vendors. A good example of such PLCs is S7-1500 PLCs provided by Siemens. They can run control logics on different devices from the same family and vendor.

- "Hardware agnostic" software PLCs:

The PLCs in this category are designed to be independent of the underlying hardware, offering greater flexibility in terms of hardware compatibility. These software PLCs can run on a variety of hardware platforms, regardless of the manufacturer. Users can choose hardware based on their requirements, and the software PLC is expected to adopt to different hardware configurations. Such PLCs are often designed to adhere to open communication and programming standards, promoting interoperability, and can be easily scaled and adapted to different system sizes and complexities. A good example of such PLCs is CODESYS software PLCs which have a control logic engine that can run on multiple hardware from different vendors.

Software-based PLCs, particularly the "hardware-agnostic" variety, have garnered widespread popularity and show promise in potentially supplanting traditional PLCs in specific applications. This is owing to their numerous advantages, including cost-effectiveness, flexibility, scalability, energy efficiency, rapid adaptability, realistic simulation capabilities, and seamless integration with IT systems—attributes often lacking in traditional PLCs.

As illustrated in Figure 2, the prevalence of automation systems relying on "hardware-agnostic" software PLCs is anticipated to double from 3.5% in 2019 to 7% by 2025 within the broader industrial PLCs market [5]. This shift is propelled by the utilization of diverse development environments and platforms in the industrial sector, such as CODESYS, TwinCAT (Beckhoff), Beremiz, PLCopen Control Code Library (CCL), OpenPLC, MatPLC, SoftPLC, among others. In the scope of this work, we deliberately selected the OpenPLC project to conduct a comprehensive security investigation. Our objectives encompass uncovering vulnerabilities, executing sophisticated attacks, and fortifying the overall security posture of the project.

A. OPENPLC PROJECT

In 2014, Alves introduced OpenPLC, an open-source software-based PLC, as a cost-effective alternative to expensive hardware PLCs [8]. Initially, the project aimed to support the research community, enabling scientists and academic researchers to establish small ICS environments for their research experiments [9]. However, due to its numerous advantages, OpenPLC has undergone significant development in recent years and has found application in small- to medium-scale industrial settings[1] where cost is a crucial factor. Moreover, OpenPLC has been adopted in diverse contexts, including home automation systems, traffic lights,[2] water treatment plants, field area netowrk,[3] and the control of Heating, Ventilation, and Air Conditioning (HVAC) systems [8]. Additionally, it has been utilized in various agricultural settings for tasks such as controlling irrigation systems, monitoring environmental conditions, and managing machinery operations.

The popularity and widespread use of OpenPLC can be attributed to its open-source nature, providing freely accessible source code, and its alignment with the common programming languages specified by the International Electrotechnical Commission (IEC) 61131-3 [10]. Moreover, OpenPLC exhibits compatibility with a diverse array of low-cost hardware devices, including Raspberry Pi, Arduino, BeagleBone, and others. This level of flexibility empowers engineers to both test and implement their control logic programs on various real hardware platforms without being confined to specific vendors or configurations.

However, despite these advantageous features, the software poses significant security risks that could potentially expose OpenPLC-based systems to various cyberattacks. These

[1]https://www.cs2ai.org/post/raspberry-pi-and-openplc-how-to-and-use-cases

[2]https://andisama.medium.com/towards-industry-4-0-4-plc-programming-a-traffic-light-controller-use-case-with-sfc-scada-a12398ae2762

[3]https://www.rad.com/resources/Demo-Videos/Edge-Computing-IIoT-OpenPLC-SCADA-Firewall

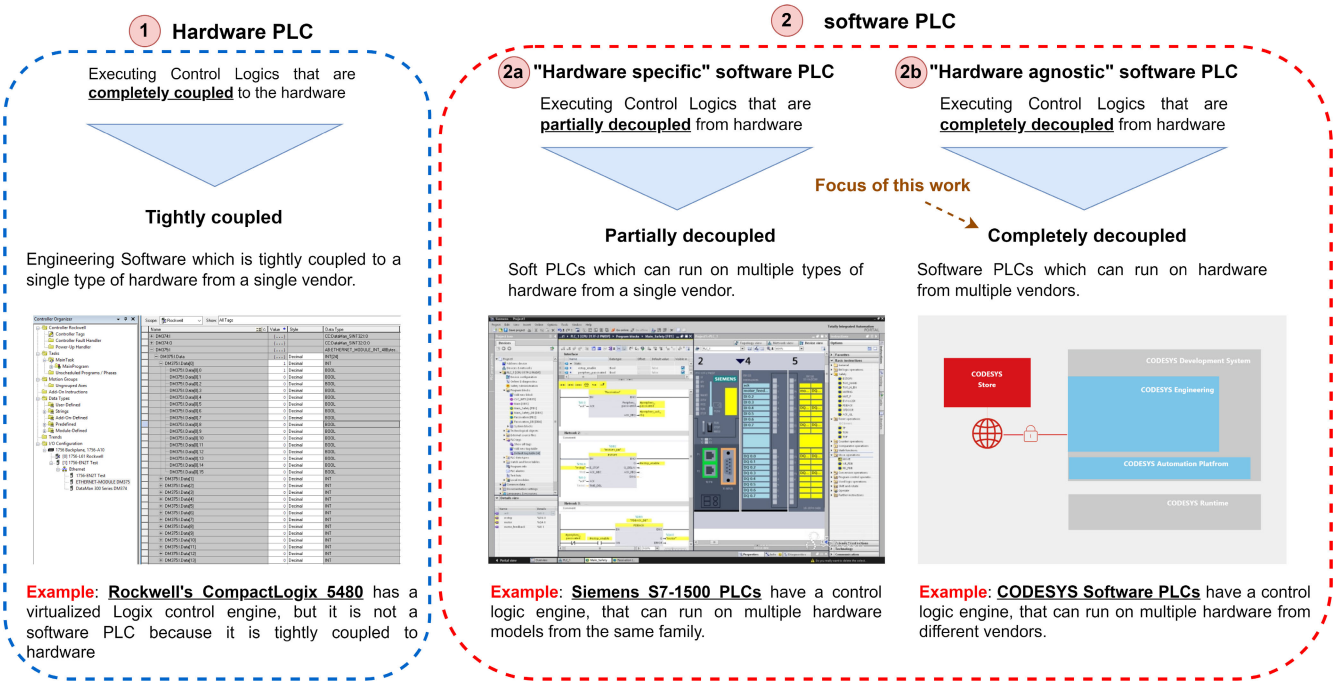

FIGURE 1. Definition of hardware and software PLCs in the industrial market.

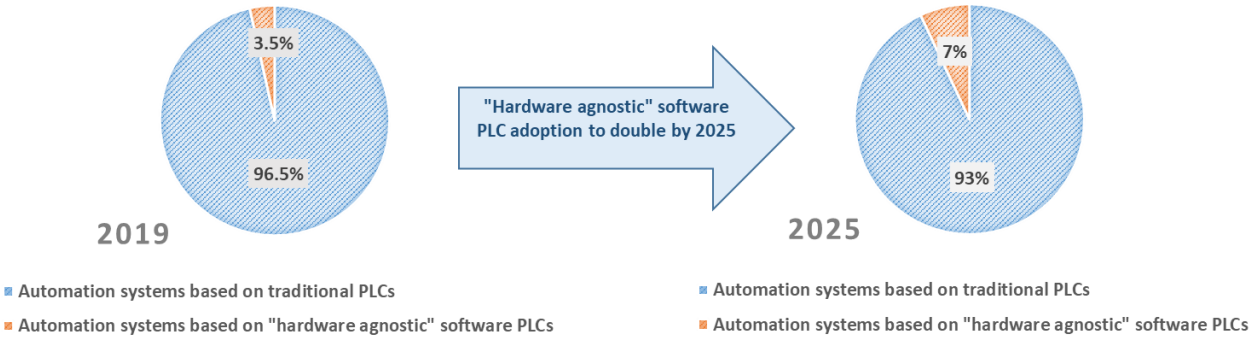

FIGURE 2. Global Industrial PLCs market: Percentages are based on PLCs employed in real-world automation systems [5].

vulnerabilities arise from OpenPLC's susceptibility to a multitude of software vulnerabilities, including but not limited to buffer overflows, injection flaws, insecure communication channels, privilege escalation, and more. Exploitation of these vulnerabilities by malicious attackers can lead to severe damage to the physical processes controlled by OpenPLC and its associated hardware devices. Hence, careful consideration and implementation of security measures are imperative to mitigate these potential risks and ensure the robustness of OpenPLC-based systems in industrial applications.

B. PROBLEM STATEMENT

Since the OpenPLC inception in 2014, the project has garnered increasing interest and enthusiasm. However, despite the considerable success achieved by the project, a critical issue remains unaddressed: its susceptibility to cyberattacks. This issue has significant concerns among engineers and scientists who not only require cost-effective open-source PLC solutions but also demand robust security measures to protect against potential threats. In light of these pressing concerns, urgent action is imperative. The OpenPLC must undergo through investigation to identify and rectify its vulnerabilities.

This article sheds light on the security vulnerabilities within the OpenPLC project, thereby alerting both founder and the boarder research community that the software is still susceptible to cyberattacks, and skilled adversaries armed with appropriate tools can inflict significant damages on OpenPLC environments. Our investigations aim to raise awareness about these security concerns within the project and contribute to the identification of new vulnerabilities and weaknesses inherent in the existing OpenPLC versions. Our main focus is on comprehending the potential attack vectors and illustrating the repercussions of exploiting these

vulnerabilities. In pursuit of real world scenarios, we run experiments against the last version of the OpenPLC, namely OpenPLCV3, and have detailed our findings in this article, bolstered by a proof-of-concept demonstration. Finally, we introduce an enhanced iteration of the OpenPLC, referred to as OpenPLC Aqua. This upgraded software is developed to provide a higher level of security, rendering it more resilient against a diverse array of cyberattacks. We hope that this new software will be a valuable resource for researchers and engineers, enabling them to conduct their experiments with confidence and free from security concerns.

C. SCOPE OF OUR WORK

In this work, we investigate the security of the OpenPLC project, and highlight design flaws within the project that allow attackers to pilfer executable programs, particularly those located in the *Webserver*, and subsequently carry out successful control logic injection attacks. We also emphasize that the communication channels employed by the OpenPLC software are susceptible to security risks. The absence of encryption leaves sensitive data, such as user credentials, control commands, configurations, etc., exposed to potential interception, manipulation and compromise by attackers with the ability to tamper with the integrity of data transmitted over the network. To support our findings, we conducted a sophisticated control logic injection attack that enables an external adversary not only to gain unauthorized access to the OpenPLC *Runtime*, but also to disrupt the controlled physical process. Remarkably, this scenario can be executed without prior knowledge of user credentials or specific details about the specific physical processes controlled by the OpenPLC.

To address and mitigate all the vulnerabilities revealed by our investigations, we introduce a more secure version of the OpenPLC software in this article, named OpenPLC Aqua. Our new software incorporates robust security measures to enhance the overall security and reliability of the OpenPLC project. One significant feature of OpenPLC Aqua is its utilization of the Advanced Encryption Standard (AES) algorithm to encrypt user credentials, including usernames and passwords. The outputs of AES algorithm are then encoded from binary to American Standard Code for Information Interchange (ASCII). This approach prevents potential attackers from intercepting user credentials in plaintext or attempting to retrieve them from the OpenPLC project files, such as the *openplc.db* database.

Furthermore, the OpenPLC Aqua enhances the security of the project by restricting access to the *Webserver* and *openplc.db* database, allowing only legitimate users with root permissions. Unauthorized users are thus prevented from accessing critical data e.g., control logics, user credentials, and configuration settings. Additionally, OpenPLC Aqua introduces a novel whitelisting function that permits only pre-approved or trusted users to upload a new program into the OpenPLC *Runtime*. This function aims to detect any malicious attempts to manipulate the running program by an unauthorized user.

Finally, our new software ensures the security of all data transmitted between the client (user) and server (OpenPLC) over the internet by implementing Secure Sockets Layer/Transport Layer Security (SSL/TLS) protocol. This safeguard guarantees the confidentiality of information, rendering it inaccessible to malicious attackers and preventing eavesdropping and data tampering.

D. COMPARING OUR WORK TO OTHERS

Reviewing the prior research works, as listed in table 1, showed that, until this point, there are no comprehensive papers have delved into the full spectrum of security aspects related to the OpenPLC project. This includes investigating new vulnerabilities, potential attacks and introducing corresponding security solutions. In the following subsections, we spotlight the differences between our work and the previous ones related to the OpenPLC theme.

1) INVESTIGATING THE SECURITY OF OPENPLC PROJECT

The security of the OpenPLC software was examined by Xinxin in [16]. The author analyzed the four phases of the OpenPLC's *Runtime* operation and monitored all the hook functions of linux kernel Security Modules (LSM). This investigation revealed several serious vulnerabilities in the software e.g., ST file integrity issues, susceptibility of associated memory to tampering, insecure network communications, and more. Based on this findings, the author developed a security-enhanced software named AESI-PLC. This new software offers improved resistance against MitM attacks, replay attacks, and command injection attacks when compared to the original OpenPLC software.

However, the author didn't consider the possibility of a control logic injection attack, precisely a stealthy attack scenario as outlined in Section IV. In our article, we conducted a comprehensive security assessment of the entire OpenPLC project, not just the OpenPLC *Runtime*, as done by Xinxin. Our investigation revealed additional vulnerabilities in the project files, communication protocols, and design flaws. These findings are detailed in Section III.

2) A SOPHISTICATED CONTROL LOGIC INJECTION ATTACK

Several research works introduced attack scenarios targeting the OpenPLC software and related systems. For instance, Alves and Morris [9] conducted a study wherein they demonstrated a Modbus command injection attack targeting PLCs from different vendors (Schneider, Siemens, Omron and OpenPLC). The authors evaluated the behavior of each device under such an attack. Their approach aimed to rapidly send write messages to the Modbus holding register "0" on the target PLC, with the intention of overwriting the internal count on the target PLC with the value "99". Another research group performed a command injection attack against OpenPLC [11]. The author exploited a security vulnerability in the "Hardware Layer Code Box" within the OpenPLC *Runtime*, successfully executing an arbitrary code via this box. By overwriting a specific code in the vulnerable code

TABLE 1. Related works discussing the security of OpenPLC project.

Year	Reference	Use Case	Target	Vulnerabilities	Attacks	Security Solutions
2014	[8]	Commercial & Academic Use	OpenPLC	-	-	-
2018	[9]	Academic Use	OpenPLC	Insecure Communication	Modbus Injection	Embedded IDS & Encryption
2018	[15]	Commercial & Academic Use	OpenPLC	-	Modbus Injection	OpenPLC Neo
2021	[11]	Academic Use	OpenPLC	Software Flaw	Command Injection	-
2021	[24]	Academic Use	OpenPLC	-	Zero-day Attacks	Whitelisting Function
2022	[12]	Academic Use	OpenPLC	Insecure Communication	MitM & FDI Attacks	-
2022	[19]	Commercial & Academic Use	OpenPLC61850	-	-	[20]
2022	[17]	Academic Use	OpenPLC	-	Replay, Data Tempering, Delay Attacks	Anomaly Detection
2023	[13]	Academic Use	OpenPLC & HMI	Insecure communication	MitM & Modbus Injection	-
2023	[4]	Commercial & Academic Use	OpenPLC	Software Flaw & Lack of Encryption	Control Injection	-
2023	[14]	Commercial & Academic Use	OpenPLC	Software Flaw & Lack of Encryption	-	Embedded Security Features
2023	[16]	Academic Use	OpenPLC	Software Flaw	MitM, Replay, Command Injection Attacks	AESI-PLC

box, the author could establish a communication between the attacker's machine and OpenPLC. However, the founder of the OpenPLC project responded to this attack by discontinuing the "Hardware Layer Code Box" and replacing it with a Python Sub Module (PSM) Code Box. The authors of [12] introduced two attack scenarios against the OpenPLC. In the first scenario, they placed the attacker's machine in a Man-in-the-Middle (MitM) position between the OpenPLC *Runtime* and the HMI *Builder*. Subsequently, they initiated a False Data Injection (FDI) by injecting false data through the Manufacturing Message Specification (MMS) messages in the OpenPLC. In the second scenario, they impersonated the HMI and injected false commands into the PLC. Alsabbagh et al. [13] conducted a stealthy FDI attack scenario using a virtual system that incorporated OpenPLC and its HMI *Builder*. The authors created a database containing real Modbus request-response pairs (captured prior to the attack) exchanged between the OpenPLC and HMI stations. Their attack approach generated two independent communication channels: one between the OpenPLC Runtime and the attacker, and the other between the attacker and HMI *Builder*.

In opposite to all previous works, this article pioneers the examination of the OpenPLC's control logic program. Furthermore, it conducts a sophisticated attack scenario that combines various techniques, including unauthorized access, MitM, replay, and control logic injection attacks. Our injection technique is insidious, evading detection by typical security measures such as Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs), leaving the user unaware of the ongoing attack on the target system. Notably, our work marks the first instant of an attack that manipulates the information displayed on the OpenPLC *Runtime* dashboard, presenting users with falsified data, showing the user what he is expecting to see.

3) ADDRESSING THE DISCLOSED VULNERABILITIES - OPENPLC AQUA

Researchers have actively been developing the OpenPLC project in the recent years. Their developments include more security features, communication protocols, programming languages, etc. For example, in 2017, Alves et al. [15] introduced an enhanced version of the OpenPLC project called OpenPLC Neo. The new enhancement incorporated an AES Layer positioned between the *Webserver* and the external network. This AES encryption layer secures all messages transmitted from the OpenPLC to an external client (e.g., user, HMI, etc.) by employing a symmetric key, and forwards then the corresponding ciphertext to the external network. Conversely, it decrypts received messages using a symmetric key provided by the user, and then transmits them to the network layer for further processing. The OpenPLC Neo establishes a secure end-to-end encrypted channel between the PLC and user or HMI, eliminating the need for external hardware to encrypt data. However, the project's security hinges entirely on the confidentiality and integrity of the encryption keys. Therefore, improper key management, such as weak, compromised, or poorly stored keys, could undermine the overall security of the AES encryption, exposing OpenPLC Neo to cyberattacks. In 2021, Roomi et al. [19] introduced the OpenPLC61850

as another enhancement of the OpenPLC project. This updated version supports IEC 61850 protocols, including IEC 61131-3 standard for programming PLC logics, and uses the MatIEC compiler for logic program compilation. In a follow up work, Roomi et al. [12] assessed the vulnerability of the OpenPLC61850 software to network-based attacks e.g., false data and command injections. Their attack scenarios successfully manipulated different Circuit Breaker (CB) connected to OpenPLC. However, Roomi did not introduce particular security solutions to safeguard the OpenPLC61850 against cyberattacks and only recommended incorporating the security measures proposed in [20] as a part of forthcoming research. Zheng et al. [17] proposed an active real-time anomaly detection framework integrated with the OpenPLC software, precisely in the control logics. They implemented and tested their approach in an ICS virtual simulation platform called *GRFICS* (Graphical Realism Framework For Industrial Control Simulations) [18]. The authors managed successfully to identify multiple threats e.g., replay, data tempering and delay attacks. Another research group [24] proposed a test-bed using OpenPLC for control system security. The authors introduced a whitelisting function that register normal operations on the list and register operations that are not registered. Such operations includes communication commands, execution orders and configurations. Their proposed approach could detect zero-day attacks and illegal commands.

In this article, we present OpenPLC Aqua, an enhanced OpenPLC software, implementing a suite of security solutions tailored to address all the vulnerabilities present in current versions. The architecture of the OpenPLC Aqua encompasses four key security features: 1) encryption of user credentials, 2) restricted accessibility, 3) whitelisting approach, and 4) secure communication channels. To validate the security of our new software, it underwent rigorous testing against several attacks that were feasible against the older OpenPLC versions. Our results showed that the OpenPLC Aqua successfully thwarted all the conducted attacks.

E. CONTRIBUTION

Our main contributions in this work are summarized as follows:

- **Thorough Security Investigation of OpenPLC software:** in contrast to prior research on OpenPLC security, our work provides a comprehensive investigation, revealing several vulnerabilities within the design of the OpenPLC project.
- **Pioneering Control Logic Injection Attack against OpenPLC based Systems:** this article introduces the first control logic injection attack tailored for OpenPLC and related applications/systems. Our injection approach maliciously alters the ongoing user program, thereby causing confusion in the physical process controlled by the compromised OpenPLC.
- **Concealing of Malicious Injection:** to heighten the severity and stealthiness of our attack scenario, we exploit security flaws in project files, effectively masking the attack from legitimate users who remain unaware of the ongoing injection.
- **Development of Enhanced OpenPLC Software (OpenPLC Aqua):** building upon our findings, we have enhanced the OpenPLC software by introducing a more secure version known as OpenPLC Aqua. This new software addresses the vulnerabilities present in current OpenPLC versions while remaining compatible with the same hardware devices as the older versions. Our OpenPLC Aqua is publically available for academic researchers and industrial engineers.

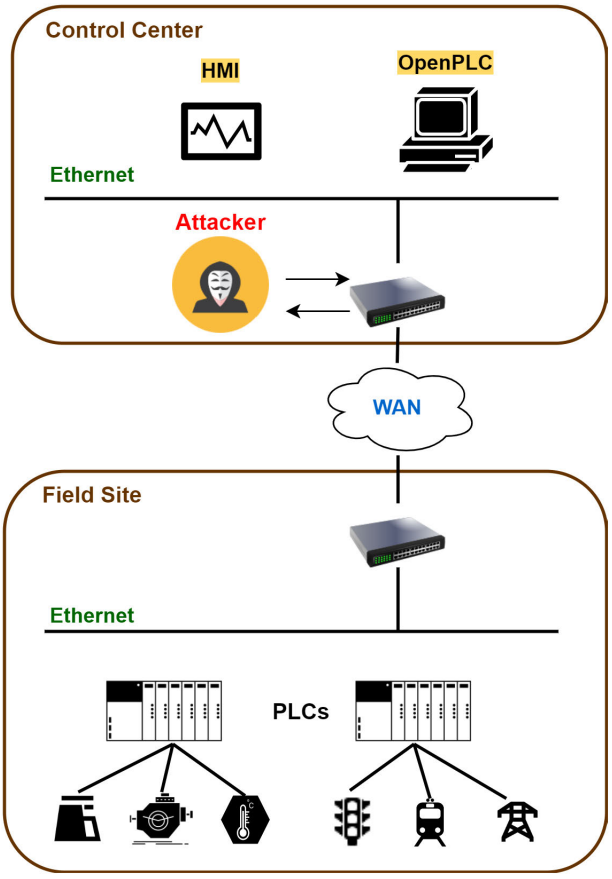

FIGURE 3. Attacker model [4].

F. METHODOLOGY

1) ATTACKER MODEL

Figure 3 depicts the attacker model we used in this article to conduct all our investigations and experiments.

As can be seen, the attacker is situated within the control center and possesses access to the same network as the OpenPLC.

2) ATTACK TECHNIQUES

All the attack scenarios presented in this work were conducted with the help of MITRE ATT@CK knowledge

base of adversary tactics and techniques [21]. They are discussed as follows under the given ICS context.

- **T1555 - Credentials from Password Stores.** In this scenario, the attacker initiates a crafted request to retrieve the most recently entered password.
- **T1040 - Network Sniffing.** The attacker engages in network traffic interception during authentication and when uploading a new program.
- **T1040 - Unauthorized Password Reset.** In this case, the attacker sends a crafted request to reset the password without proper authorization.
- **T1110.002 - Password Cracking.** The attackers exploit vulnerabilities to crack a password, particularly when they can intercept and analyze network traffic.
- **T830 - Man in the Middle (MitM).** In this tactic, the attacker positions himself between the machine running the engineering software and the PLC. This is achieved by poisoning the Address Resolution Protocol (ARP) cache of the two machines to manipulate data flow.
- **T0831 - Manipulating the Control.** Here, the attacker makes changes to set points value, tags, or other parameters to manipulate the control of physical processes.
- **T0889 - Modify Program.** In this scenario, the attacker alters or introduces a program on a PLC to affect its interactions with physical processes, peripheral devices and other network hosts.
- **T0843 - Program Upload.** Here, the attacker initiates a program upload, transferring a user-program to a PLC.

3) SOFTWARE AND TOOLS

In this work, we used the following software and tools to analyze the security of the OpenPLC project and conduct our experiments.

- **OpenPLC software:** we utilized the last version of OpenPLC, namely OpenPLCv3.[4]
- **ScadaBR[5] software:** to create an HMI for the OpenPLC, making it easier to monitor and control the physical process.
- **Network Analyzer:** we used the Wireshark[6] to monitor all the packets transmitted between the connected stations i.e., between the OpenPLC and User machine.
- **Python Libraries:** sqlite3[7] for storing the captured data. Fnmatch[8] for identifying the location of the OpenPLC files. Requests[9] to send HTTP packets. OS[10] to access the project files and modify them. hashlib[11] to create message digest for the Whitelisting function. Base64[12] to convert bytes into srting for data storage. Pycrypto[13] to encrpyt password with AES algorithm. Os.urandom[14] to create random keys for the encryption process.

The rest of the article is organized as shown in figure 4. Section II provides an architecture overview of the OpenPLC project, while Section III shows our security investigations and the vulnerabilities we found in the current OpenPLC software. In Section IV, we demonstrate a control logic injection attack against an OpenPLC based environment, followed by security solutions to close the vulnerabilities as well as the architecture of our new OpenPLC Aqua software in Section V. In Section VI we evaluate our new software and discuss our results, while Section VII concludes this article. A proof-of-concept, along with the attack codes and the OpenPLC Aqua software are found in Section VIII.

II. ARCHITECTURE OVERVIEW

A. OPENPLC OVERVIEW

The OpenPLC project is comprised of three main components: *Editor*, *Runtime* and HMI *Builder* as shown in figure 5 (highlighted in blue). In the following, we provide more technical details of each component.

1) OPENPLC EDITOR

The OpenPLC *Editor* is a programming software environment, providing users with set of tools to develop control logic programs using graphical and textual programming languages specified by the IEC 61131-3 standard e.g., Function Block Diagram (FBD), Ladder Diagram (LD), Structured Text (ST), Instruction List (IL), and Sequential Function Chart (SFC). While users have the flexibility to choose their preferred programming language, the OpenPLC *Editor* ultimately compiles all programs into ST files. Each compiled program is then stored on the *Webserver* before being transmitted to the OpenPLC *Runtime* upon an upload request. To compile the programs into ST files, the *Editor* incorporates an integrated module known as PLCopen XML. When a user wishes to execute a new program on the OpenPLC *Runtime*, the software recalls the ST file copy of the program from the *Webserver* and sends it to the OpenPLC *Runtime* through the port 8080.

2) OPENPLC RUNTIME

The OpenPLC *Runtime* serves as the central component of the OpenPLC project, functioning as an open-source platform that emulates a real hardware PLC. In essence, it executes and monitors control logic programs developed by users, operating in real-time, the OpenPLC *Runtime* boasts the capability to swiftly respond to input signals and manage output actions with minimal latency. This attribute renders it highly suitable for a diverse sorts of industrial applications. Within the OpenPLC *Runtime*, a built-in compiler, known as *MatIEC*, is designed to compile the ST files received from the *Webserver* into C files. This is necessary because the OpenPLC *Runtime* is designed to exclusively read and

[4]https://github.com/thiagoralves/OpenPLC_v3
[5]https://github.com/ScadaBR
[6]https://www.wireshark.org/
[7]https://docs.python.org/3/library/sqlite3.html
[8]https://docs.python.org/3/library/fnmatch.html
[9]https://pypi.org/project/requests/
[10]https://docs.python.org/3/library/os.html
[11]https://docs.python.org/3/library/hashlib.html
[12]https://docs.python.org/3/library/base64.html
[13]https://pypi.org/project/pycrypto/
[14]https://docs.python.org/3/library/os.html

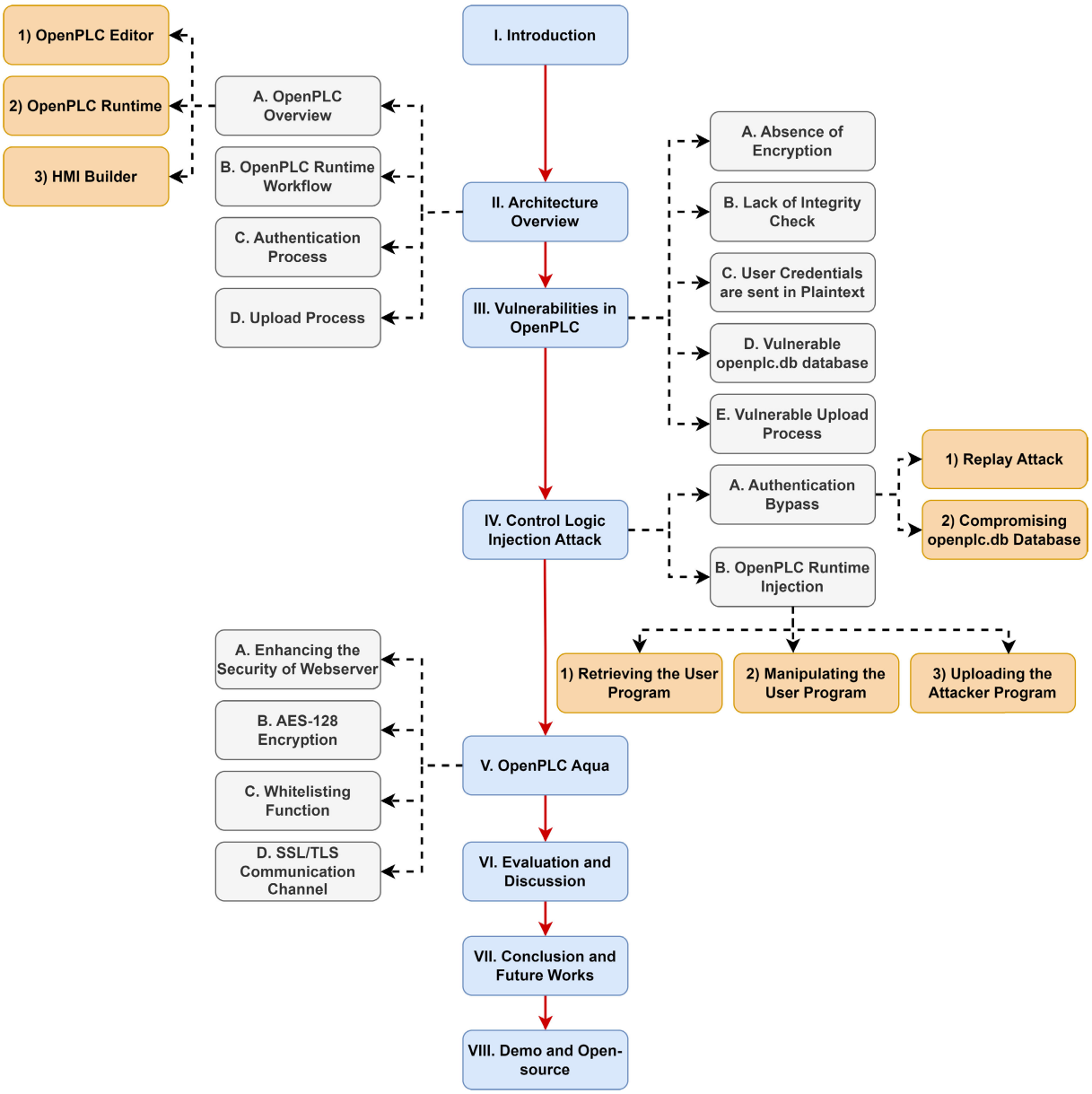

FIGURE 4. Article structure diagram: boxes in blue indicate the sections, boxes in grey indicate the subsections and boxes in orange indicate the sub-subsections.

execute C files. It is worth mentioning that the *MatIEC* compiler operates internally, converting user programs from their high-level code formats (ST files) to executable code (C files) within the OpenPLC *Runtime*. Consequently, users do not have direct access to the resulting C files. This compilation process occurs whenever the user presses the "Upload" button on the OpenPLC *Runtime*. Following a successful compilation, the final C file is uploaded and ready for execution. In contrast to typical PLCs, the OpenPLC *Runtime* does not automatically enter a "START" mode upon completing the uploaded process. Instead, it necessitates manual activation by the operator to execute the new program and starts the OpenPLC *Runtime*.

Operators can upload, modify, update, execute, and remove control logic programs within the OpenPLC *Runtime* environment. To do so, user authentication is required, involving a correct username and password. For the first login, the project is protected by a default account with the username and password both set as "openplc". After the initial login, users can change the user credentials, create new user accounts, and upload new programs in ST file format. The *Webserver* operates on port 8080, and users can access the OpenPLC *Runtime* by entering "localhost:8080" into their web browser. Among many functions, the *Webserver* stores user credentials in a database named *openplc.db*, located within the internal OpenPLC project' files. It is essential to

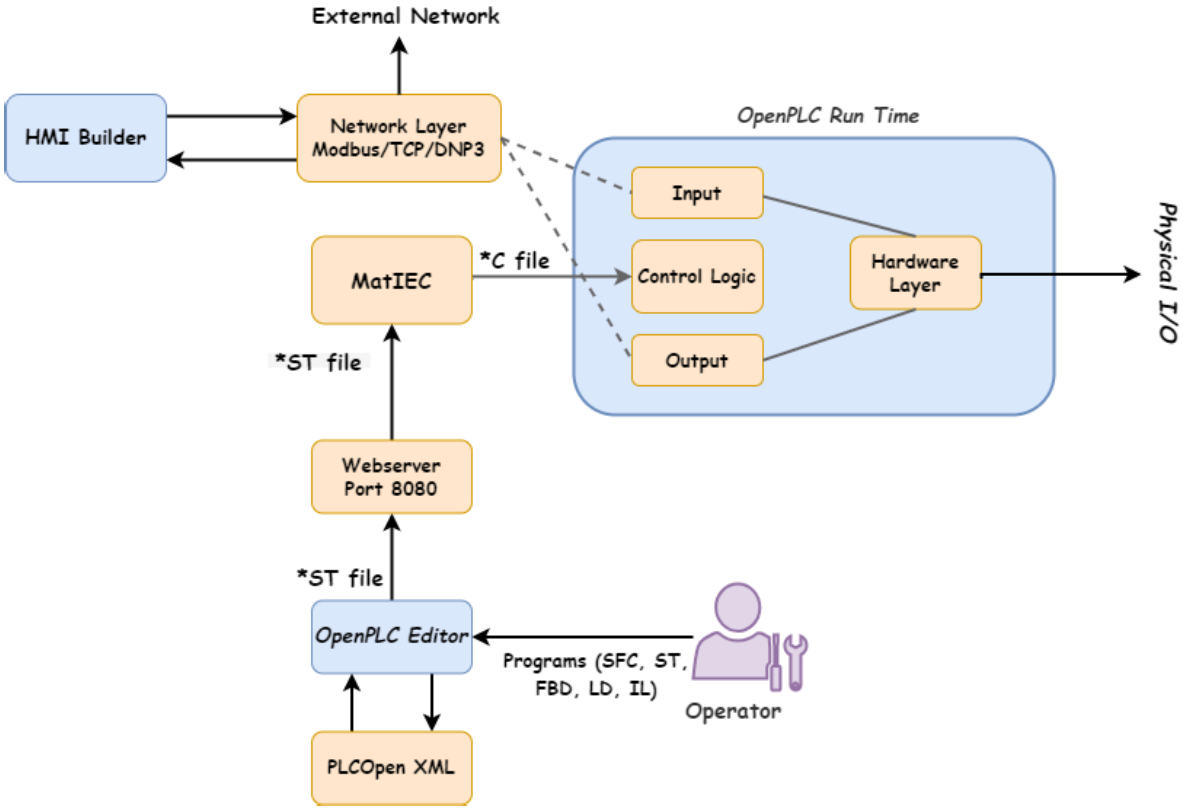

FIGURE 5. OpenPLC architecture [21].

note that the *openplc.db* also stores critical information e.g., upload programs, registered user accounts, configurations, and others.

User accounts are uniquely identified by a "*user_id*" assigned by the OpenPLC, which is used to access account information such as user credentials, programs, configurations, and more. Similarly, each uploaded program to the OpenPLC *Runtime* is assigned by a unique identifier known as "*prog_id*", along with general information including the program's name, date, and location. Both "*prog_id*" and "*table_id*" identifiers are employed by the OpenPLC *Runtime* to locate the specific program that the user intends to execute. The "*user_id*" is used when accessing the details of a registered account. As more files are uploaded, the unique identification number incrementally increases. Furthermore, within the OpenPLC project, there is a so-called "*active-program*" index, which denotes the copy of the compiled program (ST file) stored in the *Webserver*. If the "*active-program*" is empty or its value does not correspond to any of the existing copies in the *Webserver*, the upload process fails, and the OpenPLC *Runtime* remains inactive. It is worth mentioning that all the communication sessions between the user and OpenPLC *Runtime* are established through the Hypertext Transfer Protocol (HTTP).

3) HMI BUILDER

Like most other real hardware PLCs, the control logic in OpenPLC *Runtime* operates in a cyclical manner. Meaning that, during each execution cycle, inputs are initially read and then fed to the control logic, which executes the user program. Afterwards, the outputs are updated accordingly and transmitted to the physical Inputs/Outputs (I/O) devices through the hardware layer provided by the software.

Operators have the ability to interact with the system using the HMI *Builder* e.g., ScadaBR which is supported by the project. The ScadaBR is an open-source Supervisory Control and Data Acquisition (SCADA) system used to monitor and control complex processes and systems. It offers real-time visualization through a Graphical User Interface (GUI). Users can create customized dashboards to oversee and manage the physical processes controlled by the OpenPLC *Runtime*. ScadaBR also supports alarm notifications and event handing, alerting operators when predefined thresholds or conditions are met. The communication protocols employed for data exchange between OpenPLC and ScadaBR are Modbus and Distributed Network Protocol (DNP3), running on the default ports 502 and 20,000, respectively.

B. OPENPLC RUNTIME WORKFLOW

Figure 6 shows the four stages that the OpenPLC *Runtime* goes through upon an upload request. In the following, we illustrate each phase in more detail.

1) COMPILE STAGE

During the compilation stage, an ST program and runtime data are amalgamated along with hardware configuration and

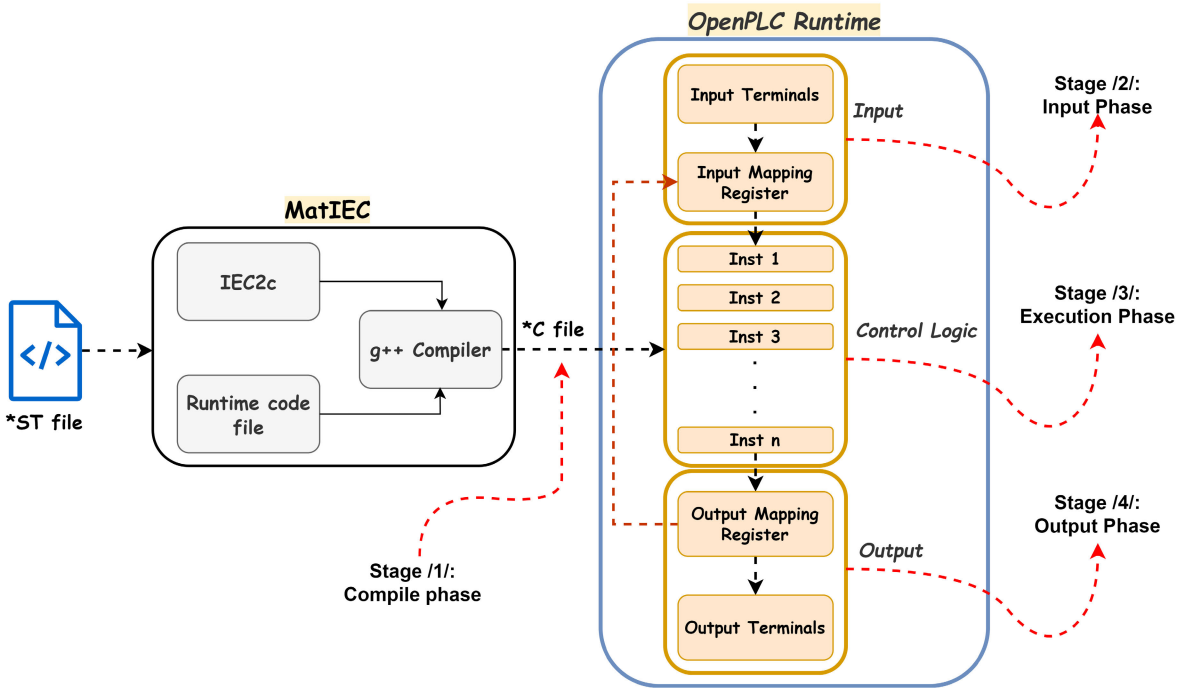

FIGURE 6. OpenPLC runtime workflow.

network details, transforming them into a runtime program in OpenPLC i.e., formatted as a C program.

2) INPUT STAGE

At the beginning of every scan cycle, the OpenPLC *Runtime* initiates by retrieving external input signals originating from sensors, switches, and various external devices. These input signals depict the system's present condition. The software then samples these signals, capturing the real-time state of the system, and subsequently stores this information within its internal input registers. This stage serves the fundamental purpose of accurately reflecting the current state of the system in real time.

3) EXECUTION STAGE

After capturing and storing the input signals, the OpenPLC *Runtime* engages in executing control programs developed by the user. These programs commonly employ logic, arithmetic operations, and conditional statements to analyze the input signals and produce corresponding output signals. During the execution stage, the program's objective is to compute the subsequent operation of the system by evaluating the current state of the input signals.

4) OUTPUT STAGE

Upon completion of the program execution stage, the OpenPLC *Runtime* updates the output registers with the calculated signals. These signals are then transmitted to actuators, drives, and external devices, regulating the system's operations. Refreshing the output signals guarantees the system functions in accordance with the control program's specifications.

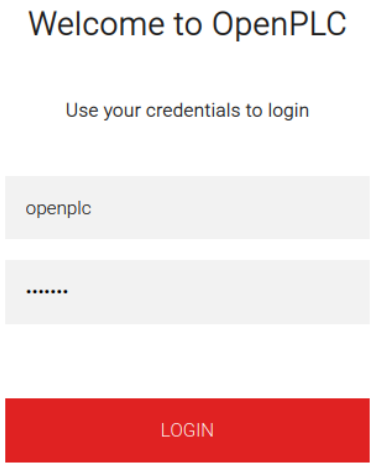

FIGURE 7. Login web page to authenticate the operator.

C. AUTHENTICATION PROCESS

The OpenPLC Runtime is in constant operation on the *Webserver*, specifically on port 8080, and can be accessed using various web browsers. To access the OpenPLC *Runtime*, one simply needs to enter "localhost:8080" into the address bar of a web browser. This action will bring up a login web page for the operator. To gain entry, the operator must provide a username and password, as depicted in figure 7.

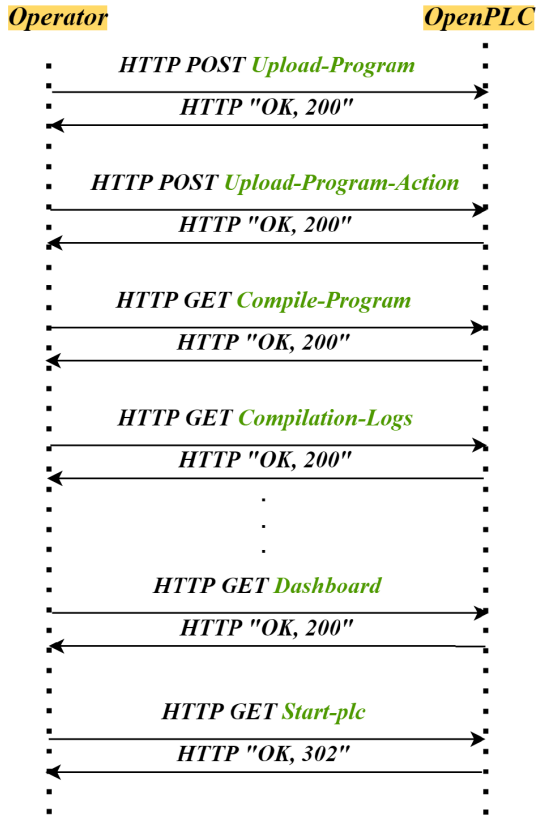

FIGURE 8. Upload program sequence diagram.

The default login credentials are set to ''openplc'', but users have the flexibility to change their credentials after their initial logic.

D. UPLOAD PROCESS

When a user initiates the upload of a new program file (ST file) into the OpenPLC *Runtime*, the program is transmitted from the *Webserver* to the OpenPLC *Runtime* through a series of HTTP packets. Figure 8 illustrates the complete upload program process executed by OpenPLC, showing the order of the transmitted packets.

Here is a step-by-step description of this process:

- The operator initiates the upload process by sending an ''upload-program'' packet, indicating the intention to upload a new program (ST file) into the OpenPLC *Runtime*.
- In response, The OpenPLC *Runtime* sends back an ''OK'' packet with a status code ''200'' to acknowledge the request.
- Afterwards, an ''upload-program-action'' packet is dispatched to the OpenPLC *Runtime*. Prompting it to generate a new ''prog_id'', ST file name, and a new upload date.
- Once these details are generated and stored in the *openplc.db*, the OpenPLC *Runtime* acknowledges with another ''OK'' packet.
- Thereafter, a ''compile-program'' packet is sent to the OpenPLC, marking the commencement of the compilation process from the ''ST file'' to the ''C file''.
- Throughout the compilation process, several ''compilation-logs'' packets are transmitted to the OpenPLC *Runtime*, their frequency dependent on the complexity of the compiled program.
- Upon successful completion of the compilation process, the OpenPLC *Runtime* sends a final ''OK'' packet, accompanied by a ''compilation finished successfully'' message.
- If all steps proceed without issues, the user can start executing the uploaded program, and the OpenPLC responds with an ''OK'' packet containing a status code ''302''.
- Otherwise, the OpenPLC sends an error message with a status code ''500'', and terminates the upload process.

III. VULNERABILITIES IN OPENPLC

While the OpenPLC Project has undoubtedly brought significant advantages to researchers and engineers, it is noteworthy that it currently lacks essential security measures. This deficiency renders the entire project and, related systems built upon it, vulnerable to various cyberattacks. In our effort to point and address these vulnerabilities within the OpenPLC software, we conducted a comprehensive investigation, and reported all our findings in the following subsections.

A. ABSENCE OF ENCRYPTION

The OpenPLC software currently lacks encryption mechanisms to safeguard the confidentiality and integrity of data exchanged between the user's machine, ScadaBR, and OpenPLC *Runtime*. Our investigation, conducted using the Wireshark tool, revealed that various packets transmit sensitive data in plain-text over the HTTP protocol. This exposed critical information related to user programs, credentials, accounts, and more, making the OpenPLC *Runtime* vulnerable to several cyberattacks, including MitM, eavesdropping, replay, unauthorized access, and control logic injection attacks. Consequently, malicious adversaries with access to the system's network can intercept and manipulate the transmitted packets between these stations with minimal effort.

B. USER CREDENTIALS ARE SENT IN PLAINTEXT

After conducting a thorough analysis of the authentication packets exchanged between the OpenPLC *Runtime* and the operator's machine, it became evident that user credentials are transmitted in plaintext without any form of encryption, as shown in figure 9.

Therefore, this vulnerability allows an attacker to easily capture the required credentials from the ''login'' packets sent from the legitimate user's machine to the OpenPLC *Runtime*, potentially compromising security.

Please note that both aforementioned vulnerabilities (i.e., absence of encryption and user credentials are sent in

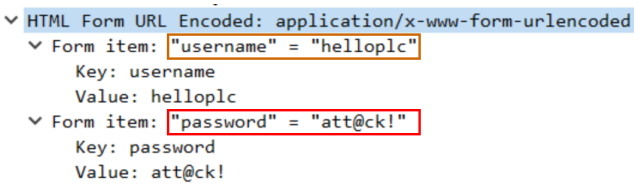

FIGURE 9. Login packet - user credentials are sent in plaintext.

plaintext) share the overarching concern of jeopardizing data confidentiality. But the ''absence of encryption'' encompasses a broader range of potential vulnerabilities in data transmission, while ''user credentials sent in plaintext'' zooms in on the specific risk related to authentication information. Recognizing these differences is crucial for comprehensive threat analysis and the development of targeted mitigation strategies. For instance, addressing ''absence of encryption'' involves implementing cryptographic measures across all data transmissions, whereas mitigating the risk of ''user credentials sent in plaintext'' concentrates on securing authentication-related traffic, restricting the access to certain system files, and other sources where attackers might be able to sniff and steal the user credentials.

C. LACK OF INTEGRITY CHECK

Our analysis of the captured HTTP packets transmitted between the user and the OpenPLC *Runtime* revealed a critical security gap: the absence of integrity check or anti-replay mechanisms to protect the control logic programs from unauthorized tampering. Consequently, attackers have the capability to inject their malicious programs into the OpenPLC *Runtime* by replaying previously recorded packets from older sessions.

In our given example, the user credentials are ''helloplc'' as a username, and ''att@ck!'' as a password. Subsequently, an attacker can exploit these credentials to authenticate himself with the OpenPLC *Runtime* and proceed to launch further attack.

D. VULNERABLE OPENPLC.DB DATABASE

As previously discussed in Section II, the OpenPLC uses a database called *openplc.db* to store critical information, including user programs, settings, secondary devices, and user credentials. Our research showed a concerning vulnerable: this database lacks access restrictions, making it susceptible to unauthorized access by potential attackers, who can exploit this security gap by creating customized scripts such purposes, as shown in figure 10.

It is worth mentioning that the construction of the *openplc.db* database relies on the *SQLite*[15] Python library. Therefore, an attacker can employ a python script to both access and modify the data existing in the *openplc.db*. As a result, we can conclude that all the information displayed on the OpenPLC *Runtime* is accessible and susceptible to manipulation scenarios. What exacerbates the situation is that the exposed database is in plaintext and contains, among many identifiers (IDs) and values, a very interesting ID called *user_id*. This *user_id* is dedicated to providing information (including operator credentials) about all the accounts registered in the OpenPLC *Runtime*. Each account is identified by a unique *user_id*, which always has the value ''10'' if the operator uses the default account i.e., ''openplc'' as the username, and ''openplc'' as password.

Our analysis showed that even if an operator initially uses the default account and subsequently changes the username and password to different values, the *user_id* value remains ''10''. This value is only changed if the operator completely removes the default account from the OpenPLC *Runtime* and registers an entirely new account. Therefore, if an attacker gains access to the *user_id* value(s), he can potentially access all the accounts registered in the OpenPLC *Runtime*. This scenario is quite severe because attackers can exploit the *openplc.db* database and change the operator account(s) credentials at will, thereby denying the operator access to the OpenPLC *Runtime*.

E. VULNERABLE UPLOAD PROCESS

Our analysis to the upload-program process revealed a significant security vulnerability in the OpenPLC software. During this process, the software consistently generates a copy of the uploaded program and automatically stores it in the OpenPLC project folders, precisely in the *Webserver*. These copies are associated with unique *prog_ids*. Notably, we have observed that these copies, once stored in the *Webserver*, cannot be removed or deleted manually and persist in the folder forever. Even if an operator removes programs from the OpenPLC *Runtime* dashboard, the corresponding copies within the *Webserver* folder remain unaffected. This poses a critical security risk as adversaries with appropriate attacking tools can potentially access and read all programs uploaded to the OpenPLC *Runtime*, including the currently running one. It is important to note that these stored copies are essentially ready-to-execute ST files. Attackers only need to maliciously modify a copy and then re-upload it to the OpenPLC *Runtime* using a patching tool, or they can replace the currently running program with any other copy of an old program stored in the *Webserver*. This attack scenario is highly effective and not detectable using traditional control logic detection methods, as outlined in [23], [25], [26], and [27].

IV. CONTROL LOGIC INJECTION ATTACK

In accordance with the findings detailed in Section III, we successfully performed a sophisticated stealthy control logic injection attack against the OpenPLC, precisely against OpenPLC *Runtime* component. Figure 11 depicts a high-level overview of our control logic injection attack introduced in this article. This attack consists of two primary phases:

- **Authentication Bypass:** This initial phase focuses on circumventing the authentication mechanisms.

[15]https://docs.python.org/3/library/sqlite3.html

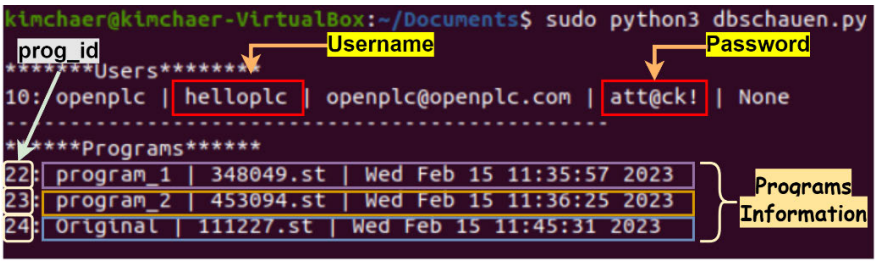

FIGURE 10. Reading information from the openplc.db database.

- **OpenPLC Runtime Infection:** The second phase involves infecting the OpenPLC *Runtime*.

A. AUTHENTICATION PYPASS

The OpenPLC *Runtime* is protected through a username and password authentication. Thus, any potential adversary must initially acquire the correct login credentials to gain access to the OpenPLC *Runtime* and subsequently carry out further actions. There are two primary methods by which this can be achieved: through a typical replay attack, or by retrieving user credentials from the *openplc.db* database.

1) TYPICAL REPLAY ATTACK

As OpenPLC currently lacks any encryption measures for the data transferred via the HTTP protocol, it leaves a vulnerability where an attacker can readily intercept user credentials, including username and password, exchanged during an authentication session. To this end, we utilized our Wireshark tool to capture the ''login'' packet transmitted from the operator to the OpenPLC *Runtime*, as shown in figure 9.

In our given example, the user credentials are ''hello'' as a username, and ''att@ck!'' as a password. After that, an attacker could potentially access to the OpenPLC *Runtime* and carry out further malicious attacks. However, it is important to note that this scenario comes with certain limitations. To successfully obtain the correct credentials, an attacker would need to be in a situation where the user logs into the OpenPLC *Runtime* while the attacker already has access to the target system and is actively monitoring the network. In essence, this means that if the attacker begins monitoring the network after the ''login'' packet has been sent, he would not be able to intercept the user credentials through any other HTTP packets. Consequently, the attacker must patiently wait until the operator performs another authentication to capture their credentials.

2) RETRIEVING USER CREDENTIALS FROM OPENPLC.DB DATABASE

In Section II, we discussed that the OpenPLC relies on a vulnerable database named *openplc.db* to store critical data, including uploaded programs, settings, secondary devices and user credentials. This database is susceptible to unauthorized access as proved in Section III. As a result, there is opportunity to bypass the previous scenario and obtain user credentials without waiting for a ''login'' packet to be sent. To address this issue, we developed a python script capable of reading the *openplc.db* database. Remarkably, we successfully extracted, among other information, the user credentials as figure 10 shows. With this information in hand, an attacker can authenticate himself, and engage in further malicious actions by sending a crafted HTTP ''login'' packet, containing the correct username and password. Furthermore, the attacker can prevent the legitimate user from accessing the OpenPLC *Runtime* by manipulating the user information stored in the *openplc.db*. Thus, when a user attempts to provide his own credentials to access the OpenPLC, there will be no match between the provided and stored credentials. This lack of authentication results in a denial of service situation, preventing the legitimate user from accessing the OpenPLC *Runtime*.

B. OPENPLC RUNTIME INFECTION

To modify the program running in the OpenPLC *Runtime*, an attacker must first identify the currently executed program. Subsequently, he proceeds to extract this program, make alterations to it, and then compel the OpenPLC *Runtime* to execute the malicious program. In the following, we illustrate each step of our infection phase in more detail.

1) RETRIEVING THE USER PROGRAM (ST FILE)

Accessing the *openplc.db* database provides an opportunity for attackers to obtain vital information concerning all programs uploaded to the OpenPLC, as shown in figure 10. Nonetheless, it is crucial for the attacker to identify the specific program currently being executed by the OpenPLC *Runtime*. Our investigations showed that the attacker can disclose the name of the running program by intercepting specific packets exchanged between the user and the OpenPLC *Runtime*. Precisely, this information can be obtained from different packets. For instance, the response packets when the user requested ''*users*'' packet to call the list of registered users or ''Modbus'' packet that calls the list of secondary devices. Figure 12 shows a response packet of the ''*users*'' request that reveals the name of the running program.

Our investigation revealed that the OpenPLC *Runtime* employs an index called ''*active-program*'' to denote the presently executing program (see Section II-A2). This index

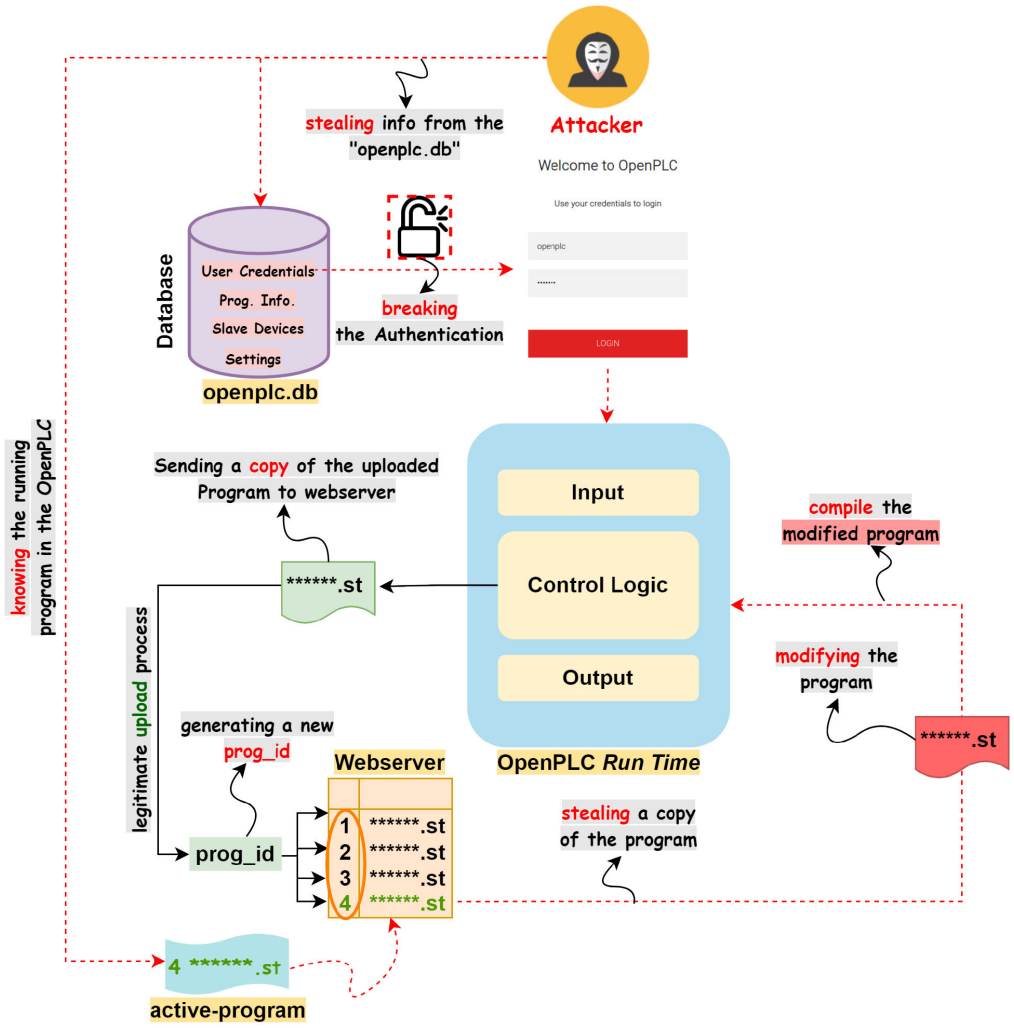

FIGURE 11. High-level overview of our attack scenario.

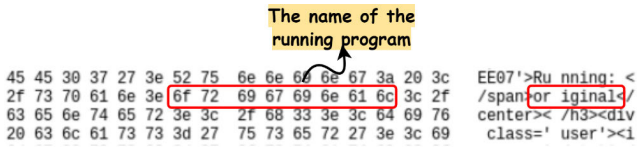

FIGURE 12. Users response packet contains the running program name.

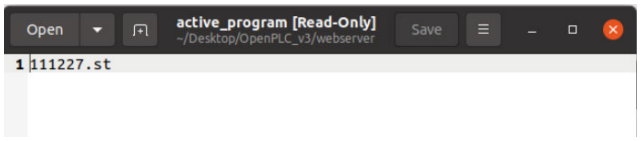

FIGURE 13. Reading the content of the "active-program" Index.

possesses a single value, represented by the ST file name of the program currently in operation, as can be seen in figure 13.

By utilizing the data stored in the *openplc.db*, we can retrieve the *prog_id*, allowing us to access the corresponding program's copy stored on the *Webserver*. It is important to note that an attacker can retrieve any copy from the *Webserver* by simply knowing the ST file name and its corresponding *prog_id*. In our example, the currently active program is identified by the *prog_id* "24", and has an associated ST file name of "111227".

2) MANIPULATING THE USER PROGRAM

The attacker possesses the original program that the OpenPLC *Runtime*, stored in an ST format. This ST file is exclusively executable within the OpenPLC *Runtime* environment. Consequently, the attacker cannot access the high-level source code, which is written in one of the IEC 61131-3 programming languages. As a result, the attacker must alter the ST file in its currently format, either manually or automatically as a part of our attack by applying a based-rules modification approach [28]. The modification includes overwriting the original program by inserting/removing instructions, modifying set-points, altering operators within equations, and changing the statuses of inputs and outputs.

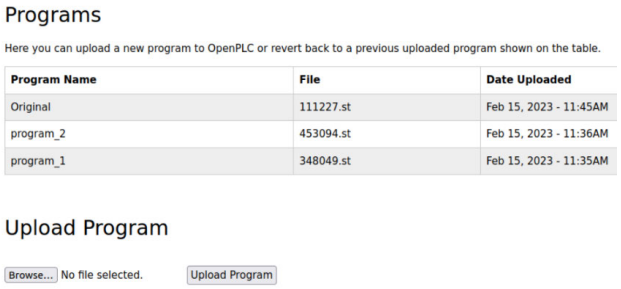

FIGURE 14. OpenPLC runtime - program list.

3) UPLOADING THE ATTACKER PROGRAM

Once we have successfully modified the program, the next step involves uploading and executing the malicious program within the OpenPLC *Runtime*. This can be achieved by conducting a completely new upload-program process as depicted in figure 8. However, simply using the upload process "as-is" would expose our infection. When utilizing this method, the OpenPLC *Runtime* generates a fresh copy of the uploaded malicious program, assigning it a new *prog_id* in the *Webserver*. Furthermore, observant user will easily discern the presence of a new program in the "Program List", complete with a new ST file name and date, as shown in figure 14.

This holds true even if we were to re-upload the same program to the OpenPLC *Runtime* i.e., each time there is an upload-program process executed, the OpenPLC *Runtime* adds a new program to the "Program List" with different entries. Consequently, patching our modified program through the upload-program process proves inadequate, as our objective is to execute our attack as stealthy as possible. Our attack strategy aims to obtain a copy of the current program running on the OpenPLC *Runtime* (see figure 11). This program has already been uploaded to the OpenPLC *Runtime* by a legitimate user, meaning it possesses a specific *prog_id*, ST file name, and an upload date. To achieve this, we simply need to compile our modified program to generate its "C file" version and then force the OpenPLC to switch to "START" mode, thus executing the attacker program as depicted in figure 15.

This scenario is feasible due to two facts. First, the OpenPLC *Runtime* generates a new *prog_id*, ST file name and upload date at the very beginning of the upload process and before the compilation process starts, precisely after sending "*upload-program-action*", and the uploading file is copied into the *Webserver* during the transmission of "upload-program". Therefore, an attacker can skip these packets since the program that he modified is already registered in the OpenPLC *Runtime*, and has an assigned *prog_id*. Secondly, the compilation of the program proceeds without the need to inspect the arrival of previous packets to the OpenPLC *Runtime*. For all this, compiling the modified "attacker" program is sufficient to update the user's program. Our attack tool initiates the process by first sending a

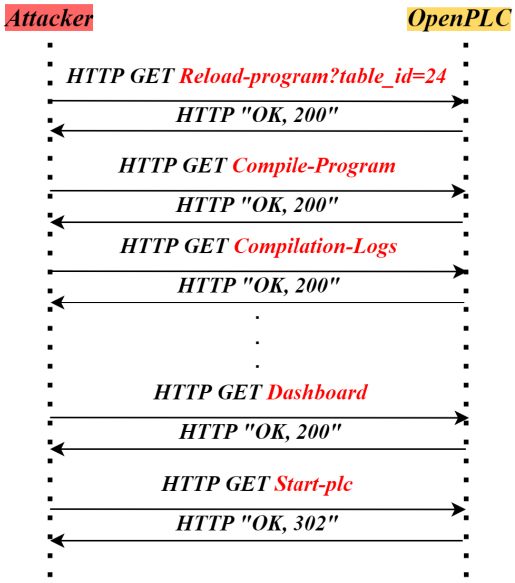

FIGURE 15. Sequence diagram of injecting the OpenPLC runtime.

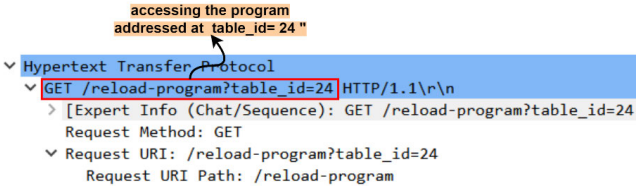

FIGURE 16. Reload program packet.

"reload-program" packet, as shown in figure 16. This packet serves as the gateway to accessing the "Programs Information" in the OpenPLC *Runtime*. Through this access, we gain the capability to execute different operations, such as launching, updating, or removing the currently running program from the OpenPLC *Runtime*.

In our example, we have three programs labeled as "*program_1*", "*program_2*" and "*Original*", with corresponding *prog_ids* of 22, 23 and 24, respectively see figure 10. To craft our "*reload-packet*", we just need to use the *prog_id* (called also *table_id*) associated with the currently running program, which, in this example, is "24". Once the attacker has chosen the program he wishes to modify, the next step is to compile the malicious program from its "ST file" format into "C file". This conversion process is achieved by sending a "compile-program" packet that includes the original ST file name, such as 111227.st in our example, see figure 17.

The compilation process begins by retrieving a copy of the "111227.st" file from the *Webserver*. This copy has already been tampered with by the attacker. Subsequently, the OpenPLC *Runtime* compiles it into an executable "C file". Once the compilation is completed successfully, OpenPLC sends a final "OK" packet, containing a "compilation finished successfully" message. It is worth mentioning that the OpenPLC *Runtime* does not run automatically after

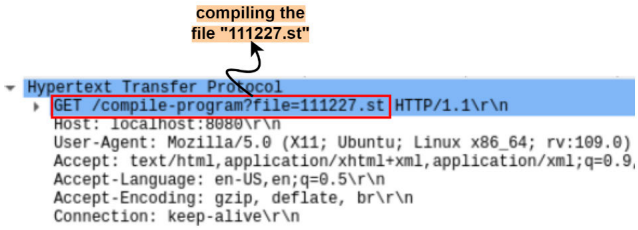

FIGURE 17. Compile program packet.

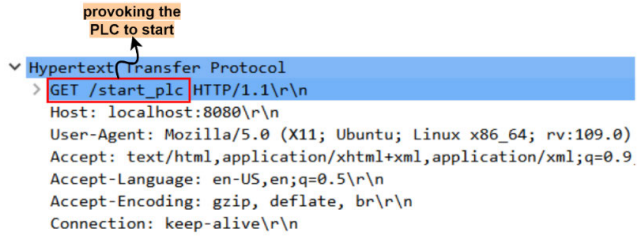

FIGURE 18. Start PLC packet.

the compilation process concludes. Therefore, to trigger the OpenPLC to commence its operations and execute the injected malicious program, the attacker must take deliberate action. Figure 18 shows a "start-plc" packet that the attacker sends to initiate the OpenPLC and execute his malicious program.

V. SECURITY ENHANCED OPENPLC AQUA

In Section III and IV, we demonstrated that the current version of OpenPLC is susceptible to vulnerabilities that enable interception, manipulation, and comprise of data integrity transmitted over the network. In addition, attackers can alter control logic programs executed by the OpenPLC *Runtime* runs without detection by legitimate users. In this section, we introduce a more secure OpenPLC software (called OpenPLC Aqua), seamlessly integrated with robust security solutions. These enhancements significantly bolster the overall security and trustworthiness of the OpenPLC project. Figure 19 provides a visual representation of the internal architecture of our new OpenPLC Aqua software.

As can be seen from the figure, this OpenPLC version incorporates five supplementary security features:

- Enhancing the Security of *Webserver*.
- AES-128 Encryption.
- Whitelisting Function.
- SSL/TSL communication channels.

In the following subsections, we elaborate each security feature in more detail.

A. ENHANCING THE SECURITY OF WEBSERVER

1) ROOT-USER TO ACCESS WEBSERVER

Considering the critical security vulnerabilities outlined in Section III, especially pertaining to the *Webserver*, OpenPLC Aqua has implemented substantial security measures. Access to the *Webserver* is currently limited solely to authorized users with root permissions, as depicted in Figure 20.

Consequently, only the root user has the capability to read and write to the *Webserver*. This implies that if an attacker somehow gains access to the *Webserver*, he would need to provide the password of the user account on Linux. Furthermore, because the script responsible for running the OpenPLC service is owned by the root user, only a user with root permissions can initiate the service. In other words, a user without root permissions, such as an attacker (using the username "*rnrn0909*," in figure 20) cannot start the service or access the *Webserver* without the Linux password, as illustrated in Figure 21. It is worth mentioning that the *Webserver* stores critical data associated with security features, including keys for AES encryption, certificates for SSL/TLS communication, and the *openplc.db* database, among others. By necessitating users to install the service with root permissions, OpenPLC Aqua can effectively limit access to the *Webserver*. This measure ensures that all critical data is securely stored, or at the very least, makes the task more challenging for potential attackers.

2) AUTOMATIC REMOVAL OF PROGRAM COPIES

We discovered that OpenPLC retains copies of all uploaded programs within the *Webserver*. These copies, which are in the form of ready-to-execute ST files, pose a potential risk. If unauthorized users manage to access the *Webserver*, they could retrieve these programs and exploit them to manipulate the OpenPLC *Runtime*, potentially replacing the currently running program with an older one. The issue with the OpenPLC software lies in the permanence of these copies; even when user deletes a program from the user-dashboard, the corresponding copies in the *Webserver* are not removed. To mitigate this concern, we have enhanced the OpenPLC Aqua security by incorporating a script that automatically removes any copy from the *Webserver* when the user deletes the associated program from the user-dashboard.

However, this solution has its limitations. The OpenPLC *Runtime* initiates ST program execution by referencing the name of the ST copy in the "*active-program*" index and accessing the corresponding ST copy file in the *Webserver*. Since the OpenPLC Aqua deletes copies alongside the removal of the original program from the user-dashboard list, the name of the copy persists in the "*active-program*" index. Consequently, the "*active-program*" points to a missing copy in the *Webserver*, resulting in errors during the OpenPLC *Runtime* execution. To address this challenge, OpenPLC Aqua is configured with a blank program (a "fake program") specifically designed to assists the OpenPLC *Runtime* in launching itself. It is important to note that this program is not intended for execution on hardware controllers. Instead, it continuously alerts the user to launch a proper program. Moreover, if any user attempts to start the blank program in the OpenPLC Aqua, the session will be immediately closed. This adds more security to the software as it prevents any attempt to abuse this blank program.

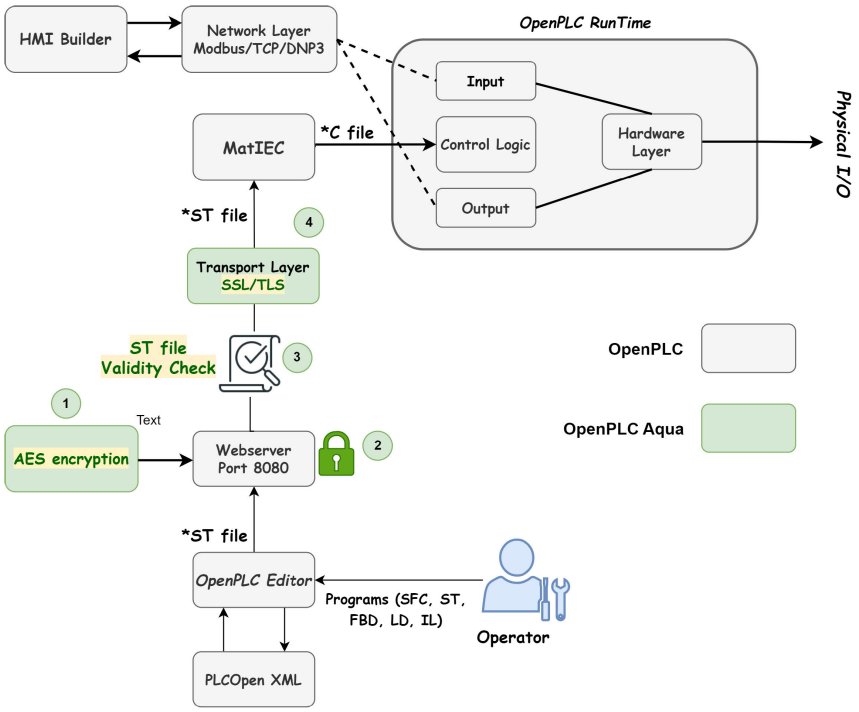

FIGURE 19. Architecture of OpenPLC Aqua: parts highlighted in green color indicate the security features added in OpenPLC Aqua.

```
rnrn0909@rnrn0909-VirtualBox:~/Desktop/OpenPLC-Aqua$ ls -l
total 264
-rw-rw-r-- 1 rnrn0909 rnrn0909    314 Sep 27 16:11 appveyor.yml
-rwxrwxr-x 1 rnrn0909 rnrn0909  12459 Sep 27 16:11 background_installer.sh
-rw-rw-r-- 1 rnrn0909 rnrn0909    979 Sep 27 16:11 Dockerfile
drwxrwxr-x 3 rnrn0909 rnrn0909   4096 Sep 27 16:11 documentation
-rw-rw-r-- 1 rnrn0909 rnrn0909 106373 Sep 27 16:11 doxygen.conf
-rw-rw-r-- 1 rnrn0909 rnrn0909   4175 Sep 27 16:11 installed.db
-rw-r--r-- 1 root     root      97081 Jan  5 20:14 install_log.txt
-rwxrwxr-x 1 rnrn0909 rnrn0909     71 Sep 27 16:11 install.sh
-rw-rw-r-- 1 rnrn0909 rnrn0909    881 Sep 27 16:11 README.md
-rw-rw-r-- 1 rnrn0909 rnrn0909     62 Sep 27 16:11 requirements.txt
-rwxr-xr-x 1 root     root         48 Jan  5 20:14 start_openplc.sh
drwxrwxr-x 8 rnrn0909 rnrn0909   4096 Sep 27 16:11 utils
drw-r----- 7 rnrn0909 rnrn0909   4096 Jan  5 20:14 webserver
```

FIGURE 20. Screenshot of a linux terminal indicating that OpenPLC Aqua is restricted to running only with root permissions.

B. AES-128 ENCRYPTION

Our implementation of encryption in the OpenPLC Aqua is designed to secure user credentials e.g., username and password. We achieve this through customize AES-128 encryption algorithm, depicted in figure 22. The reasons behind adopting AES-128 encryption in OpenPLC Aqua over AES-256 (which provides a higher level of security) are as follows:

- **Balancing Security and Performance:** AES-128 offers a good balance between security and computational efficiency.
- **Resource Constraints:** In the context of ICS and cyber-physical systems, where real-time response is critical, resource constraints may influence the choice of encryption strength. The use of AES-128 is a conscious decision to minimize computational load, especially in environments with limited processing power.
- **Compatibility:** Some legacy industrial components may have limitations in processing AES-256 efficiently. By selecting AES-128, we aim to ensure broader compatibility across a wide-range of industrial environments.

To secure plaintext messages (username and password), we employed a process that divides them into 128- bits chunks. In cases where the original message size is not a multiple of 128 bits, we pad the last block with random bits to create a complete 128 bit block. The Cipher Block Chaining (CBC) is used then to handle these 128-bit blocks. At this stage, we initiate the process with a random Initialization Vector (*IV*) to serve as the starting pseudo-block. In our

```
rnrn0909@rnrn0909-VirtualBox:~/Desktop/OpenPLC-Aqua$ sudo ls -l ./webserver
[sudo] password for rnrn0909:
total 9136
-rwxrwxr-x 1 rnrn0909 rnrn0909      17 Jan  5 20:14 active_program
-rw-r--r-- 1 root     root        2130 Jan  5 20:14 cert.pem
-rwxrwxr-x 1 rnrn0909 rnrn0909    6747 Sep 27 16:11 check_openplc_db.py
drwxrwxr-x 5 rnrn0909 rnrn0909    4096 Jan  5 20:14 core
-rw-rw-r-- 1 rnrn0909 rnrn0909    2006 Sep 27 16:11 dnp3.cfg
-rw-rw-r-- 1 rnrn0909 rnrn0909    1244 Sep 27 16:11 fileComparison.py
-rwxr-xr-x 1 root     root     8878288 Jan  5 20:06 iec2c
-rw-r--r-- 1 root     root          16 Jan  5 20:11 iv.bin
-rw-r--r-- 1 root     root          16 Jan  5 20:11 key.bin
-rw-rw-r-- 1 rnrn0909 rnrn0909     796 Sep 27 16:11 key_create.py
-rw-r--r-- 1 root     root        1620 Jan  5 20:11 key_create.pyc
-rw------- 1 root     root        3272 Jan  5 20:11 key.pem
drwxrwxr-x 2 rnrn0909 rnrn0909    4096 Sep 27 16:11 lib
-rw-rw-r-- 1 rnrn0909 rnrn0909    6033 Sep 27 16:11 monitoring.py
-rw-r----- 1 root     root       49152 Jan  5 20:11 openplc.db
-rw-rw-r-- 1 rnrn0909 rnrn0909    9302 Sep 27 16:11 openplc.py
-rw-rw-r-- 1 rnrn0909 rnrn0909   98744 Sep 27 16:11 pages.py
-rw-r----- 1 rnrn0909 rnrn0909     113 Sep 27 16:11 registeredIP.json
drwxrwxr-x 2 rnrn0909 rnrn0909    4096 Sep 27 16:11 scripts
drwxrwxr-x 3 rnrn0909 rnrn0909    4096 Sep 27 16:11 static
drw-r----- 2 rnrn0909 rnrn0909    4096 Jan  5 20:11 st_files
-rwxr-xr-x 1 root     root       46472 Jan  5 20:06 st_optimizer
-rwxrwxr-x 1 rnrn0909 rnrn0909  172500 Sep 27 16:11 webserver.py
```

FIGURE 21. Screenshot of a Linux Terminal indicating that only root-user can access the Webserver: SSL/TLS Certificates are highlighted in red box; AES key and IV are highlighted in blue box; openplc.db database is highlighted in yellow box.

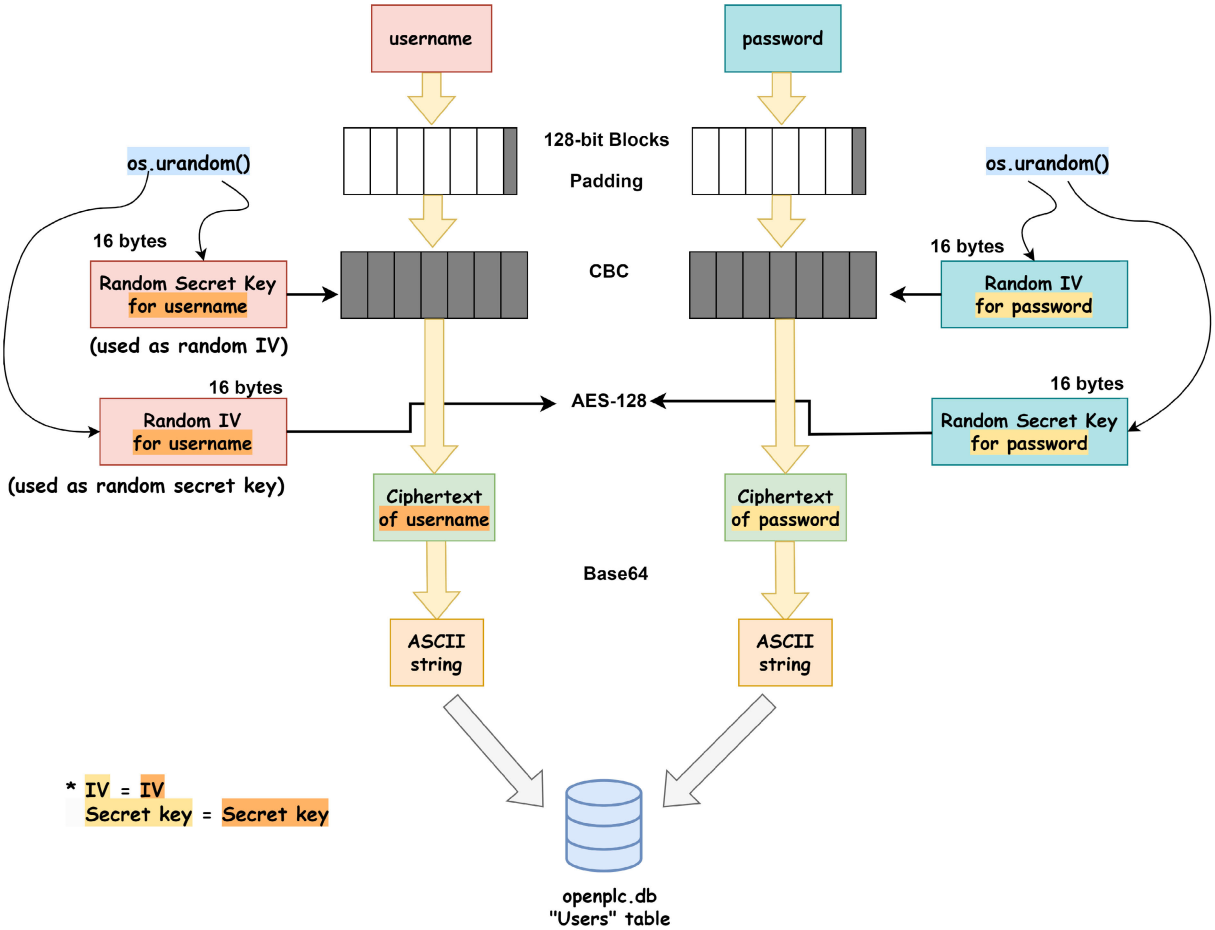

FIGURE 22. AES-128 Encryption Algorithm used in OpenPLC Aqua.

approach, we use a function called os.urandom from the python library to generate two random 16 bytes values.

One of these values becomes a secret key (K), while the other serves as the IV. Both K and IV are stored in the *Webserver*. Please note that, once the OpenPLC Aqua is installed, the access to the *Webserver* is restricted to only users with root permissions as illustrated in Section V-A1. However, when encrypting the password, we employ the IV to

encrypt the initial block, while *K* is used to encrypt the entire blocks. Conversely, for the username, we reverse this process. *K* is utilized as an *IV* to encrypt the initial block, while the *IV* functions as a *K* to encrypt the entire blocks. Once we have calculated the ciphertexts for both username and password, we encode them using the Base64 encoding algorithm. The results of the encoding process are finally presented in *ASCII* format.

C. WHITELISTING FUNCTION

In the OpenPLC Aqua, we have implemented a novel function that verifies both the content of ST files and the associated Internet Protocol (IP) addresses whenever a new ST file is uploaded. This security measure effectively thwarts any attempts by suspicious users to maliciously upload ST files, as shown in figure 23. At the beginning, when a trusted user creates a new account through the user dashboard, their IP address and username are automatically added to a whitelist. Assuming that only the default account is registered on the user dashboard, if the user of the default account tries to upload a different program than what the OpenPLC Runtime is currently executing, the OpenPLC Aqua initiates a two-step process.

First, it generates hashes for both the old and new programs and compares them. If the hashed do not match, it proceeds to check the IP address of the user attempting the upload. If the user's IP address is not found in the whitelist, the upload process is promptly rejected, the session is closed, and the suspicious user is automatically logged out. Conversely, if the user's IP address is on the whitelist, the OpenPLC *Runtime* authorizes the upload, making the new program ready for execution.

This approach furnishes a robust security advantage to our software. For instance, even if the attacker somehow gains access to the OpenPLC *Runtime* using legitimate user-credentials and accounts, he would remain unable to upload a malicious program, bolstering the overall security of our system. However, to add additional security features to this approach, we integrated our OpenPLC Aqua software with three security measures that ensure the integrity of scripts from malicious manipulations as follows:

- **File Integrity Checks:** OpenPLC Aqua incorporates file integrity checks to detect any unauthorized modification to the scripts. This involves cryptographic hashing as well as checksum verification to ensure that only authorized users can modify the OpenPLC Aqua scripts.
- **Encrypted Storage:** We implemented in the OpenPLC Aqua an encryption mechanism (AES-128) to secure the storage of OpenPLC scripts, ensuring that even if an attacker gains access to the file-system, the content remains confidential.
- **Revision Control:** OpenPLC uses a version control system e.g., ongoing tracking to monitor changes to scripts over time, allowing easy rollback in the event of unauthorized modifications.

D. SSL/TLS COMMUNICATION CHANNEL

Our research on the last versions of OpenPLC has revealed a vulnerability to replay attacks. In Section IV, we demonstrated that adversaries equipped with appropriate attacking tools can reproduce certain traffic captures (HTTP packets) from previous communication sessions over the internet. This allows them to gain access to OpenPLC based systems and make malicious changes.

SSL/TLS, a secure communication protocol, offers a robust solution for establishing secure channels over the internet. It creates an encrypted connection between a client (user) and a server (OpenPLC), ensuring that the transmitted data remains confidential and resistant to interception or exposure by unauthorized parties, a weakness present in the current OpenPLC software. Therefore, in the development of OpenPLC Aqua, we have implemented the SSL/TLS handshake approach, as illustrated in figure 24, to enhance confidentiality, integrity, and end-point authentication.

Our handshake approach works as follow:

- The client (user) initiates the handshake by sending a ''Hello'' message to the server (OpenPLC). This message serves as a request to establish a secure communication channel between the two parties. In this message, the client also includes its cipher suites and the compatible SSL/TLS version.
- The OpenPLC responds to the client's ''Hello'' message by sending back a random value generated by the server, the chosen cipher suite, and the SSL certificate.
- The client, upon receiving the server's response, verifies the authenticity of the SSL certificate by checking it against a trusted authority. Once authenticated, the client extracts the server's public key from the certificate.
- With the server's public key in hand, the client generates an encrypted pre-primary key. This pre-primary key is encrypted using the extracted server public key.
- The OpenPLC receives the encrypted pre-primary key from the client and proceeds to decrypt it. This decryption step validates the correctness of the extracted key.
- After a successful verification, both the client and server independently generate a shared session key using the random values exchanged earlier and the pre-primary key. This session key is used for encrypting all data transmitted between the client and server.
- From this point forward, all data exchanged between the client and server is secured through encryption and decryption processes facilitated by the shared session key.

VI. EVALUATION AND DISCUSSION

A. ASSESSING OPENPLC AQUA AGAINST ATTACKS

In order to evaluate the performance of our newly developed OpenPLC Aqua software in comparison to existing versions

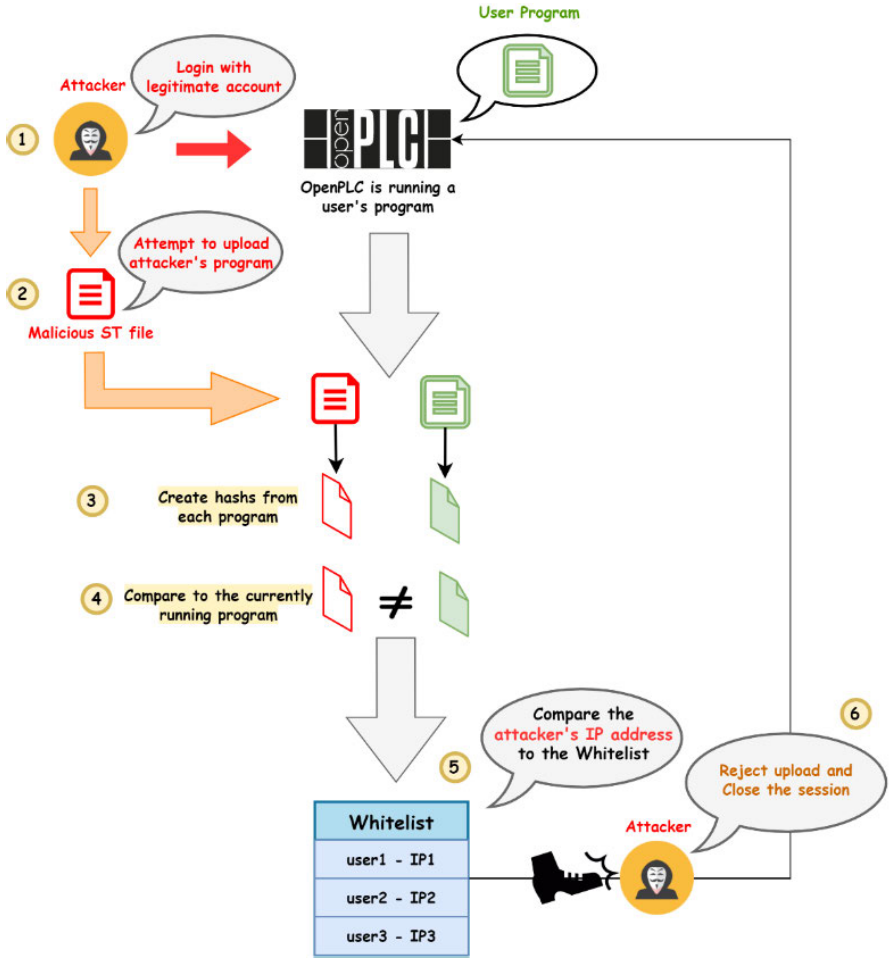

FIGURE 23. Whitelisting approach in OpenPLC Aqua.

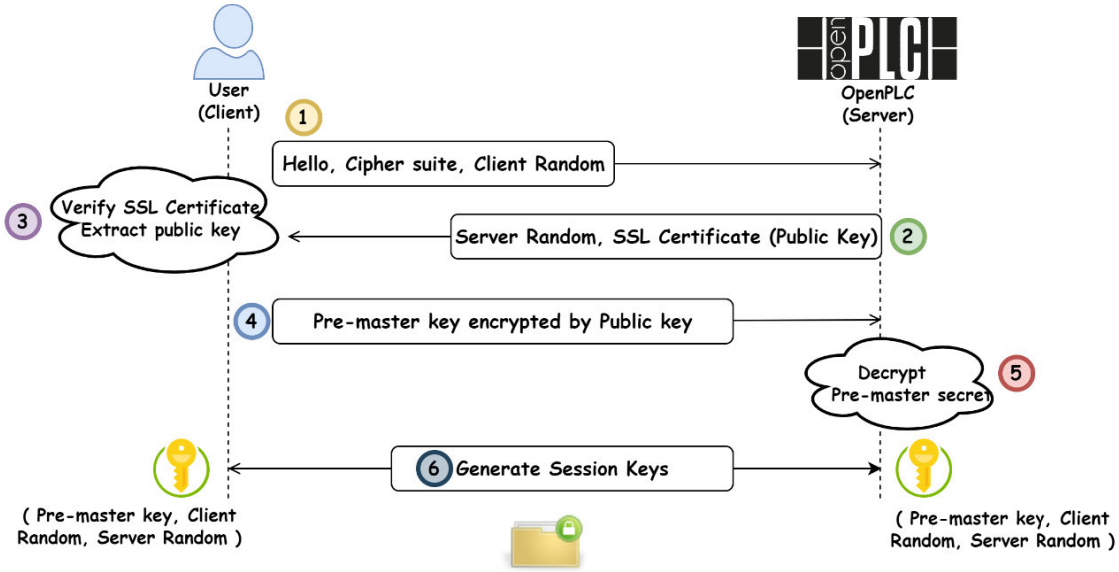

FIGURE 24. SSL/TLS communication process approach.

of OpenPLC software, we executed five distinct attack scenarios on four different OpenPLC software variants: OpenPLCV3, OpenPLC Neo, OpenPLC61850, AESI-PLC and OpenPLC Aqua. All our results are listed in table 2.

TABLE 2. Success of various attacks against different OpenPLC software versions.

Attack Scenario	V3	Neo	61850	AESI-PLC	Aqua
Authentication Attack	✓	X	✓	X	X
MitM Attack	✓	X	✓	X	X
Injection Attack	✓	✓	✓	✓	X
Replay Attack	✓	X	✓	X	X
Access Attack	✓	✓	✓	X	X

In our experimental investigations, it becomes evident that the OpenPLC Aqua exhibits a significantly higher level of security and resilience when compared to its older OpenPLC versions. Our empirical findings demonstrate that unauthorized users are unable to glean any insights into registered user accounts, user credentials, control logic programs, or any other sensitive data. This heightened security is achieved by implementing strict access controls on both the *Webserver* and the *openplc.db* database within OpenPLC Aqua. To gain access to these components, an external attacker would need to acquire root permissions. Assuming, for instance, that an adversary somehow managed to gain access to the database, he would only obtain encrypted versions of usernames and passwords. Hence, decrypting these ciphertexts would necessitate further reverse engineering efforts, with the additional challenge of deciphering the data without knowledge of the encryption key. This added layer of security ensures that even if access is breached, the attacker remains unable to decipher the stored information.

Furthermore, the introduction of the whitelisting function in our enhanced software serves as an effective deterrent against unauthorized users, attempting to upload malicious programs. We put this to the test by attempting to upload an ST program from an attacker's machine, resulting in OpenPLC Aqua rejecting the upload request and subsequently logging out the attacker from the OpenPLC *Runtime*. In stark contrast to earlier OpenPLC versions that relied on the HTTP protocol for data transmission between the OpenPLC *Runtime* and user machines, OpenPLC Aqua employs secure channels based on SSL/TLS. This strategic choice guarantees both the integrity and confidentiality of data exchanged between the various stations, providing an additional layer of protection against potential threats.

B. COST OF ENHANCING SECURITY

In the following, we provide a comprehensive analysis of the costs associated with integrating our four proposed security measures in the new OpenPLC Aqua software: AES-128 Encryption Process, Accessible restricted *Webserver*, Whitelisting Function, and SSL/TLS communication channels. The evaluation focuses on three critical aspects: computational overhead, real-time Responsiveness, and Scalability. To assess the impacts, we conducted performance tests comparing the software processing speed and efficiency before (OpenPLC) and after (OpenPLC Aqua) the integration of the four security measures. Our evaluations are elaborated in the following subsection in detail.

1) COMPUTATIONAL OVERHEAD

The integration of our security measures in OpenPLC Aqua software introduced computational overhead due to the additional processing required for encryption and decryption processes. However, the average increase in processing time for these operations in OpenPLC Aqua was minor. Thus, we can conclude that despite the overhead, the OpenPLC Aqua maintained acceptable performance levels for typical industrial control applications.

In addition, OpenPLC Aqua consumes a few kilobytes (KB), less than 50 KB, to be executed since it includes symmetric keys, certificate, a few lines of code to implement security functions, etc. Considering the environment where it will be used, we believe it is critical to reduce overhead, as much as we can, to keep its performance. Therefore, we utilized AES-128, and reversed the uses of K an IV to encrypt both usernames and passwords which eventually introduces encrypted user credential with less computational overhead comparing to the overhead caused by using an AES-256 encryption process.

2) REAL-TIME RESPONSIVENESS

Real-time responsiveness was assessed through simulation scenarios mimicking real-world industrial control operations. Negligible delays were observed in non-intensive cryptographic operations. During uploading the program, slight delays of 21.4 milliseconds were noticed, which may be acceptable for many industrial applications.

3) SCALABILITY

We tested our OpenPLC Aqua software performance under varying workloads. The new software demonstrated satisfactory scalability even with the inclusion of the security measures. We noticed that the OpenPLC Aqua maintained stability and performance as the number of connected devices increased. This proves that the OpenPLC Aqua works effectively in various industrial applications and can handle the increasing number of connected devices i.e., it is reasonable for medium-scale automation systems.

In conclusion, the implementation of our security measures in the OpenPLC Aqua software incurs manageable costs in terms of computational overhead, real-time responsiveness, and scalability. However, the new software introduced a reasonable secure and efficient software-based PLC to various industrial applications.

VII. CONCLUSION AND FUTURE WORKS

A. CONCLUSION

This article highlights significant security vulnerabilities in the existing OpenPLC software, which its founder overlooked during its initial release. Through our extensive investigations, we have demonstrated that attackers can compromise the authentication mechanism of the OpenPLC *Runtime*, and replace the user program with a malicious

one. Our attack approach is entirely stealthy, leaving no trace of abnormal activity within the OpenPLC *Runtime*, *Webserver*, or *openplc.db* database. In order to enhance the security of the OpenPLC project, we have developed OpenPLC Aqua, a software solution that addresses as many disclosed vulnerabilities, in previous OpenPLC versions, as possible by introducing four security features. OpenPLC Aqua has undergone rigorous testing against various cyberattacks, affirming its superior resilience and security compared to the vulnerabilities identified in earlier software versions.

B. FUTURE WORKS

The OpenPLC project still offers significant opportunities for improvement to align with the stringent requirements of ICS and automation systems, particularly in terms of availability, security, and real-time performance. In the following, we delineate four key directions for future research initiatives.

1) MACHINE LEARNING-BASED ANOMALY DETECTION

Further works should focus on exploring the possibility of integrating machine learning techniques for anomaly detection in OpenPLC environment. The founder of OpenPLC introduced to use Intrusion Prevention System (IPS) with unsupervised machine learning techniques on his Secure OpenPLC. His work succeeded in detecting and blocking Injection attack and Denial of Service (DoS) attack. Boateng and Bruce [29] utilized OpenPLC to train their anomaly detection model and tested various anomaly scenarios on OpenPLC with their detection model. OpenPLC can play the role to expand the research for PLC with advanced machine learning skills. The researches in the future can develop diverse models based on machine learning techniques that can respond to more various attacks proactively, detecting novel attacks or previously unseen pattern. As machine learning techniques evolves fast, the research with new techniques can become more active by using OpenPLC.

2) TECHNOLOGICAL ADVANCEMENTS

The research and industrial communities still lack experimental studies in the direction of introducing technological advancements in the realm of OpenPLC software. Such studies encompass developments in the integration the adoption of secure-by-design principles, quantum-resistant cryptographic algorithms and protocol to ensure the long-term security in future OpenPLC implementations.

3) COLLABORATIVE EFFORTS AND STANDARDIZATION

We highly recommend collaborative efforts within the research and industrial communities to establish standardized security practices for OpenPLC based systems. This involves encouraging information sharing, promoting test practices, and fostering a community-driven approach to addressing security challenges collectively.

4) HUMAN-CENTRIC SECURITY IN OPENPLC ENVIRONMENTS

More examination for the human factors in OpenPLC security, considering the impact of user behavior and decision-making on system vulnerabilities. For instance, investigate the design and implementation of user-aware security mechanisms, such as adaptive access controls, user training programs, and user-behavior analytics, to mitigate the risk of human-induced security incidents in OpenPLC based automation systems.

VIII. DEMO AND OPEN-SOURCE CODES

You can find the demonstration of our attack scenario in [30], and the OpenPLC Aqua software test in [31]. Our attack code is accessible on GitHub via [32], and for the open-source version of our developed OpenPLC qua software, please refer to [33].

REFERENCES

[1] W. Alsabbagh and P. Langendöerfer, "A new injection threat on S7-1500 PLCs–disrupting the physical process offline," *IEEE Open J. Ind. Electron. Soc.*, vol. 3, pp. 146–162, 2022, doi: 10.1109/OJIES.2022.3151528.

[2] W. Alsabbagh and P. Langendörfer, "Security of programmable logic controllers and related systems: Today and tomorrow," *IEEE Open J. Ind. Electron. Soc.*, vol. 4, pp. 659–693, 2023, doi: 10.1109/ojies.2023.3335976.

[3] W. Alsabbagh and P. Langendörfer, "A stealth program injection attack against S7-300 PLCs," in *Proc. 22nd IEEE Int. Conf. Ind. Technol. (ICIT)*, vol. 1, Mar. 2021, pp. 986–993, doi: 10.1109/ICIT46573.2021.9453483.

[4] W. Alsabbagh, C. Kim, and P. Langendörfer, "Good night, and good luck: A control logic injection attack on OpenPLC," in *Proc. IECON 49th Annu. Conf. IEEE Ind. Electron. Soc.*, Singapore, Oct. 2023, pp. 1–8, doi: 10.1109/iecon51785.2023.10312570.

[5] S. Annaswamy. *Soft PLCs: The Industrial Innovator's Dilemma*. Accessed: Jan. 3, 2024. [Online]. Available: https://iot-analytics.com/soft-plc-industrial-innovators-dilemma/

[6] E. V. Easwaran, R. Kushalkar, N. Tigadi, K. M. Moudgalya, A. Chipkar, A. Zoitl, M. Akshai, and T. Alves, "Programmable logic controller: Open source hardware and software for massive training," in *Proc. IECON 44th Annu. Conf. IEEE Ind. Electron. Soc.*, Washington, DC, USA, Oct. 2018, pp. 2422–2427, doi: 10.1109/IECON.2018.8592772.

[7] H. P. Guntaka, R. Kushalkar, N. Venkat, A. Chipkar, V. Easwaran, and K. Moudgalya, "Modular hardware, open source software and training material for PLC training," in *Proc. 10th Int. Conf. Control, Mechatronics Autom. (ICCMA)*, Belval, Luxembourg, Nov. 2022, pp. 237–242, doi: 10.1109/ICCMA56665.2022.10011589.

[8] T. R. Alves, M. Buratto, F. M. de Souza, and T. V. Rodrigues, "OpenPLC: An open source alternative to automation," in *Proc. IEEE Global Humanitarian Technol. Conf. (GHTC)*, San Jose, CA, USA, Oct. 2014, pp. 585–589, doi: 10.1109/GHTC.2014.6970342.

[9] T. Alves and T. Morris, "OpenPLC: An IEC 61,131–3 compliant open source industrial controller for cyber security research," *Comput. Secur.*, vol. 78, pp. 364–379, Sep. 2018.

[10] K.-H. John and M. Tiegelkamp, "The programming languages of IEC 611313," in *IEC 611313: Programming Industrial Automation Systems: Concepts and Programming Languages Requirements for Programming Systems Decision-Making Aids*. Berlin, Germany: Springer, 2010, pp. 99–205.

[11] CVE-2021-31630. *National Vulnerability Database*. Accessed: Aug. 24, 2023. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-31630

[12] M. M. Roomi, W. S. Ong, S. M. S. Hussain, and D. Mashima, "IEC 61850 compatible OpenPLC for cyber attack case studies on smart substation systems," *IEEE Access*, vol. 10, pp. 9164–9173, 2022, doi: 10.1109/ACCESS.2022.3144027.

[13] W. Alsabbagh, S. Amogbonjaye, D. Urrego, and P. Langendörfer, "A stealthy false command injection attack on modbus based SCADA systems," in *Proc. IEEE 20th Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, Jan. 2023, pp. 1–9, doi: 10.1109/CCNC51644.2023.10059804.

[14] W. Alsabbagh, C. Kim, and P. Langendrfer, "No attacks are available: Securing the openplc and related systems," in *INFORMATIK 2023 Designing Futures: Zukünfte Gestalten*. Bonn, Germany: Gesellschaft Für Informatik, Sep. 2023, pp. 2085–2096, doi: 10.18420/inf2023_206.

[15] T. Alves, T. Morris, and S.-M. Yoo, "Securing SCADA applications using OpenPLC with end-to-end encryption," in *Proc. 3rd Annu. Ind. Control Syst. Secur. Workshop*, Dec. 2017, pp. 1–6, doi: 10.1145/3174776.3174777.

[16] L. Xinxin, "AESI-PLC: Architecture for enhancing the security and integrity of PLC RUNTIME," *Res. Square*, Oct. 2023, doi: 10.21203/rs.3.rs-3500032/v1.

[17] C. Zheng, X. Wang, X. Luo, C. Fang, and J. He, "An OpenPLC-based active real-time anomaly detection framework for industrial control systems," in *Proc. China Autom. Congr. (CAC)*, Xiamen, China, Nov. 2022, pp. 5899–5904, doi: 10.1109/CAC57257.2022.10055121.

[18] D. Formby, M. Rad, and R. Beyah, "Lowering the barriers to industrial control system security with GRFICS," in *Proc. USENIX Workshop Adv. Secur. Educ. (ASE)*. Baltimore, MD, USA: USENIX Association, Aug. 2018. [Online]. Available: https://www.usenix.org/conference/ase18/presentation/formby

[19] M. M. Roomi, W. S. Ong, D. Mashima, and S. S. M. Hussain, "OpenPLC61850: An IEC 61850 MMS compatible open source PLC for smart grid research," *SoftwareX*, vol. 17, Jan. 2022, Art. no. 100917, doi: 10.1016/j.softx.2021.100917.

[20] S. M. S. Hussain, T. S. Ustun, and A. Kalam, "A review of IEC 62351 security mechanisms for IEC 61850 message exchanges," *IEEE Trans. Ind. Informat.*, vol. 16, no. 9, pp. 5643–5654, Sep. 2020, doi: 10.1109/TII.2019.2956734.

[21] *MITRE ATTCK*. Accessed: Jul. 11, 2023. [Online]. Available: https://attack.mitre.org/2020

[22] T. Li, Y. Wang, C. Zou, Y. Tian, L. Zhou, and Y. Zhu, "Research on DoS attack detection method of modbus TCP in OpenPLC," *J. Comput. Commun.*, vol. 9, no. 7, pp. 73–90, 2021, doi: 10.4236/jcc.2021.97007.

[23] S. McLaughlin, S. Zonouz, D. Pohly, and P. McDaniel, "A trusted safety verifier for process controller code," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2014, pp. 23–26, doi: 10.14722/ndss.2014.23043.

[24] S. Fujita, K. Hata, A. Mochizuki, K. Sawada, S. Shin, and S. Hosokawa, "OpenPLC based control system testbed for PLC whitelisting system," *Artif. Life Robot.*, vol. 26, no. 1, pp. 149–154, Feb. 2021, doi: 10.1007/s10015-020-00635-1.

[25] 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. 2014, doi: 10.1109/MSP.2014.113.

[26] T. Chang, Q. Wei, W. Liu, and Y. Geng, "Detecting PLC program malicious behaviors based on state verification," in *Cloud Computing and Security* (Lecture Notes in Computer Science), vol. 11067, X. Sun, Z. Pan, and E. Bertino, Eds. Cham, Switzerland: Springer, 2018, doi: 10.1007/978-3-030-00018-9_22.

[27] Y. Xie, R. Chang, and L. Jiang, "A malware detection method using satisfiability modulo theory model checking for the programmable logic controller system," *Concurrency Comput., Pract. Exper.*, vol. 34, no. 16, Jul. 2022, Art. no. e5724, doi: 10.1002/cpe.5724.

[28] W. Alsabbagh, "Investigating security issues in programmable logic controllers and related protocols," Ph.D. dissertation, Dept. Wireless Syst., Faculty MINT Mathematik, Informatik, Physik, Elektro und Informationstechnik of the Brandenburg Univ. Technol. Cottbus-Senftenberg, Cottbus, Germany, 2023.

[29] E. A. Boateng and J. W. Bruce, "Unsupervised machine learning techniques for detecting PLC process control anomalies," *J. Cybersecur. Priv.*, vol. 2, Sep. 2022, Art. no. 220244, doi: 10.3390/jcp2020012.

[30] *Good Night, Good Luck*. Accessed: Jul. 4, 2023. [Online]. Available: https://www.youtube.com/watch?v=rEBeV982gWQ

[31] *No Attacks Are Available—OpenPLC Aqua*. Accessed: Aug. 23, 2023. [Online]. Available: https://www.youtube.com/watch?v=knVTQfUdNfU

[32] *A-Control-Logic-Injection-Attack-on-OpenPLC*. Accessed: Oct. 11, 2023. [Online]. Available: https://github.com/rnrn0909/A-Control-Logic-Injection-Attack-on-OpenPLC

[33] *OpenPLC-Aqua*. Accessed: Oct. 23, 2023. [Online]. Available: https://github.com/rnrn0909/OpenPLC-Aqua

WAEL ALSABBAGH (Member, IEEE) received the M.Sc. degree in automatic control and computer engineering from Al-Baath University, Homs, Syria, in 2015, and the Ph.D. degree in computer science from the Technical University of Cottbus, Cottbus, Germany, in 2023. Since 2018, he has been a Scientist with IHP—Leibniz-Institut für Innovative Mikroelektronik, Frankfurt (Oder), Germany. His research interests include cyber-attacks and security, mitigation methods of the attacks targeting industrial control systems (ICS), and supervisory control and data acquisition (SCADA). He serves as a Technical Papers Reviewer for many conferences and journals, including IEEE Access, IEEE Internet of Things, and *Computers & Security*.

CHAERIN KIM received the B.S. degree in information security from Konyang University, Nonsan-si, Republic of Korea, in 2018. She is currently pursuing the M.S. degree in cyber security with the Brandenburg University of Technology Cottbus–Senftenberg, Cottbus, Germany. Since 2022, she has been a Research Assistant with IHP—Leibniz-Institut für Innovative Mikroelektronik, Frankfurt (Oder), Germany. Her research interests include cyber-attacks and security solutions related to programmable logic controllers and industrial control systems.

PETER LANGENDÖRFER received the Diploma and Ph.D. degrees in computer science. Since 2000, he has been with IHP—Leibniz-Institut für Innovative Mikroelektronik, Frankfurt (Oder), Germany. In IHP—Leibniz-Institut für Innovative Mikroelektronik, he is leading the Wireless Systems Department. From 2012 to 2020, he was the Chair of Security in Pervasive Systems with the Brandenburg University of Technology Cottbus–Senftenberg. Since 2020, he has been the Chair of Wireless Systems with the Brandenburg University of Technical Cottbus–Senftenberg. He has published more than 150 refereed technical articles and 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 the Guest Editor of many renowned journals, such as *Wireless Communications and Mobile Computing* (Wiley) and *ACM Transactions on Internet Technology*.