Good Night, and Good Luck: a Control Logic Injection Attack on OpenPLC

Wael Alsabbagh, Chaerin Kim and Peter Langendörfer

May 9, 2023

Good Night, and Good Luck: A Control Logic Injection Attack on OpenPLC

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

Abstract—**Real hardware PLCs are quite pricey, and sometimes are unaffordable for scientists/engineers to build up small testbeds, and conduct their experiments or academic researches. For all that, the OpenPLC project introduces a reasonable alternative option and offers flexibility in programming codes, simulating physical processes and also the possibility of being utilized with low-cost devices e.g., Raspberry Pi and Arduino Uno. Unfortunately, the OpenPLC project was designed without any security in mind i.e., it lacks protection mechanisms such as encryption, authorization, anti-replay algorithms, etc. This allows attackers to fully access the OpenPLC and makes unauthorized changes e.g., start/stop the PLC, setting/updating passwords, removing/altering the user-program, and others. In this paper we conduct intensive investigations and disclose some vulnerabilities existing in the OpenPLC project, showing that an attacker without any prior knowledge neither to the user credentials, nor to the physical process; can access critical information and maliciously alter the user-program the OpenPLC executes. All our experiments were conducted on the latest version of the OpenPLC i.e., V3. Our experimental results proved that attackers can confuse the physical process controlled by the infected OpenPLC. Finally we suggest security recommendations for the OpenPLC founder and engineers to close the disclosed vulnerabilities and have more secure OpenPLC based environments.**

Index Terms— OpenPLC; Cyber Attacks; Cyber security; Control Logic Injection Attacks;

I. Introduction

OpenPLC is an open-source environment that supports software simulation and hardware implementation on devices such as Raspberry Pi, Arduino, ESP8266, etc. [1]. It has increasingly gained fame in the engineering, academic and education communities since it was first introduced by Thiago Alves in 2015. The OpenPLC project[1] is comprised of *Editor*[2], *Run Time*[3] and Human Machine Interface (HMI) *Builder* [4]. The development environment that is used to create programs is called *Editor*. This *Editor* supports program-development using several programming languages identified by IEC 61131-3 [2] e.g., Function Block Diagram (FBD), Ladder Diagram (LD), Structured Text (ST), Instruction List (IL), and Sequential Function Chart (SFC). Based on IEC 61131-3, the programs are saved as XML files. Once the program is created, the built-in module in *Editor* compiles all programs into an ST file. This ST file is then utilized in the OpenPLC *Run Time* for the execution of control logics. The communication protocols supported by OpenPLC are Modbus and Distributed Network Protocol (DNP3) using the default ports 502 and 20,000 respectively. Figure 1 depicts the architecture of the OpenPLC project. As any other real hardware PLC, the control logic is processed cyclically. Meaning that, inputs are first read in each execution cycle, and fed to the control logic that processes the user-program. Afterwards, outputs are updated accordingly and finally transferred to physical I/O using the hardware layer provided by the software. ICS operators can interact with the OpenPLC operations using the HMI Builder supported by the project. However, for more technical details about the hardware and software of the OpenPLC see [1].

Despite the fact that the OpenPLC project provides the users with divers of benefits and can be opened in almost all the web browsers, it was developed with no security in mind. The project lacks fundamental security means e.g., encryption, authorization, anti-replay mechanisms, etc. that are implemented in real hardware controllers e.g., SIMATIC S7 PLCs. In this paper, we disclose some vulnerabilities and security gaps in the design of this project. Based on our findings, we launch a severe control logic injection attack against an OpenPLC based environment that maliciously alters the currently running user-program and confuses the physical process controlled by the infected OpenPLC. This scenario is conducted without knowing neither the user credentials, nor the physical process the OpenPLC controls. Furthermore, our investigations show that attackers can retrieve all the previous programs that were uploaded to the OpenPLC in the past. In our experiments, we managed also to keep our injection hidden from the ICS operator, as well as, we proved that attackers can fake the information displayed for the operator, and trick him by showing what he is expecting to see. To validate our results, we conducted all our experiments on the OpenPLC V3 software as it is the last version developed by the project founder in August 2021. Finally, we provided the OpenPLC founder with some security recommendations to close those vulnerabilities.

Please note that our attack tool[5] as well as a proof-of-concept[6] of our attack scenario are publicly available.

[1]https://openplcproject.com/
[2]https://openplcproject.github.io/plcopen-editor/
[3]https://openplcproject.github.io/runtime/
[4]https://openplcproject.github.io/reference/scadabr/

[5]https://github.com/rnrn0909/A-Control-Logic-Injection-Attack-on-OpenPLC.git
[6]https://www.youtube.com/watch?v=rEBeV982gWQ

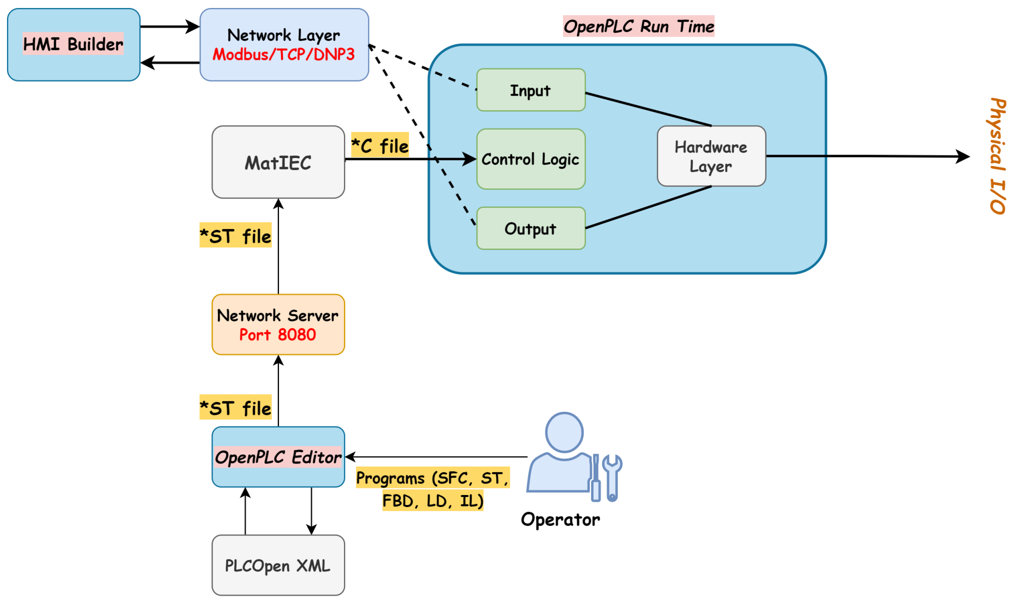


Fig. 1: The structure of the OpenPLC project

The rest of the paper is structured as follows. Section II discusses related works, while section III presents the attacker model assumed in this paper. In section IV, we conduct security investigations, and present our attack approach in Section V. Afterwards, we suggest some security recommendations in Section VI, and conclude the paper in section VII.

II. Related Work

Alves et al. in [1] presented a Modbus command injection attack against PLCs from different vendors (Schneider, Siemens, Omron and OpenPLC), and evaluated the behavior of each device to such attacks. Their attack approach aims at sending write messages to the Modbus holding register "0" on the target PLC as fast as possible. The purpose of those messages was to overwrite the internal count on the target PLC with the value "99". Another research group conducted a command injection attack against OpenPLC [3]. The authors exploited a security gap in the "Hardware Layer Code Box" in the OpenPLC *Run Time*, and managed successfully to execute an arbitrary code via this box. After overwriting a specific code in the vulnerable code box, he could establish a communication between the attacker machine and the OpenPLC. However, the founder of the OpenPLC project responded to this attack by closing the "Hardware Layer Code Box" and replacing it with a Python Sub Module (PSM) Code Box. A research group in [4] introduced two attack scenarios against the OpenPLC. In the first scenario, they placed the attacker machine in Man-in-the-Middle (MitM) position between the OpenPLC and the HMI *Builder*. Then, they launched 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. [5] conducted a stealthy FDI attack scenario against a virtual system using the OpenPLC and its HMI *Builder*. The authors created a database containing real Modbus request-response pairs (captured prior to the attack) between the OpenPLC and HMI stations. Their attack approach generated two independent communication flows: one between the PLC and the attacker, and the other between the attacker and the HMI. In the opposite to all the aforementioned works, our paper is the first work that hits the OpenPLC control logic, precisely it maliciously alters the running user-program causing abnormal behavior to the physical process controlled by the infected OpenPLC. Furthermore, it fakes information displayed to the operator, showing him always what he is expecting to see.

III. Attacker Model

Figure 2 illustrates the attacker model we assumed in this paper. As can be seen, the attacker is placed in the control center and has access to the same network where the OpenPLC is located. All the attack scenarios presented in this work are conducted with the help of *MITRE ATT@CK* knowledge base of adversary tactics and techniques [6]. They are discussed as follows under the given ICS context.

T1555 - Credentials from Password Stores. The attacker can send a crafted request to read the cache containing the last entered password.

T1040 - Network Sniffing. The attacker can sniff the network traffic at the time of authentication and uploading a project.

T1040 - Unauthorized Password Reset. The attacker sends a crafted request to reset the password.

T1110.002 - Password Cracking. The attacker exploits vulnerabilities to crack the password if he/she is able to sniff the network traffic.

T830 - Man in the Middle (MitM). The attacker sits between the machine running the engineering software and

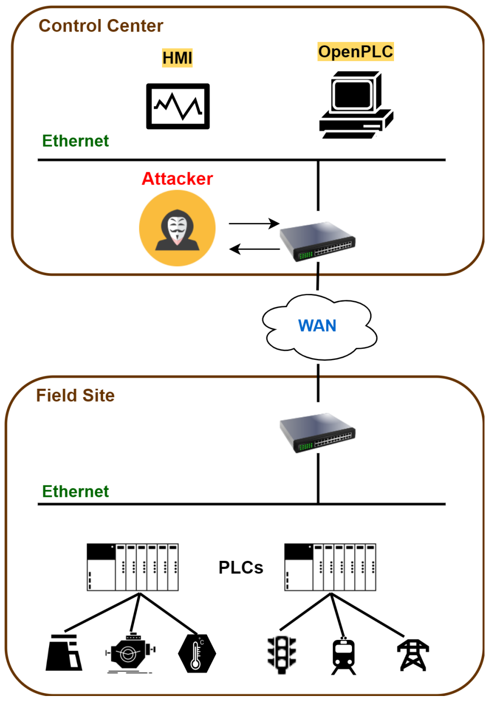


Fig. 2: Attacker model

the PLC by poisoning the ARP cache of the two machines to manipulate data.

T0831 - Manipulating the Control. The attacker can change set points value, tags, or other parameters that will manipulate physical process control.

T8021 - Modify Controller Tasking. The attacker modifies the tasking of a PLC to allow for the execution of their own programs. This can follow to manipulate the execution flow and behavior of a PLC.

T0889 - Modify Program. The attacker alters or adds a program on a PLC to affect how it interacts with the physical process, peripheral devices and other hosts on the network.

T0843 - Program Upload. The attacker performs a program upload to transfer a user-program to a PLC.

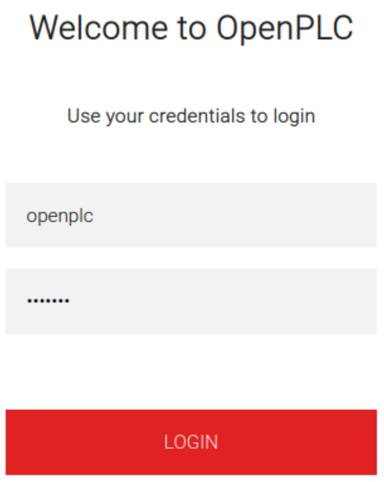
IV. Investigating the Security of OpenPLC

A. *Authentication Process*

The OpenPLC *Run Time* is a utility that allows the user-program to be compiled, uploaded and executed either virtually or in physical hardware devices. It is constantly running on webserver, precisely on port 8080, and can be opened in most of the web browsers. By opening a web browser and typing "localhost:8080", the OpenPLC *Run Time* runs, and a login web page is displayed to the operator who is required to enter both the user name and password see figure 3. The default username and password are "openplc". The operator can change his credentials right after the first login.

By using a Wireshark[7] tool and capturing the network traffic, precisely the HTTP packets exchanged between the

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



Welcome to OpenPLC

Use your credentials to login

openplc

•••••••

Fig. 3: Login web page to authenticate the operator

operator machine and OpenPLC during the authentication process, we found that the operator credentials are sent in plaintext without any encryption method implemented (see figure 6). For all that, getting the user name and password is feasible, and attackers can retrieve and even change the operator credentials without too much effort. To do so, they can perform a typical replay attack i.e., sending pre-recorded HTTP packets from old sessions e.g., during setting a new password, or changing an old password with a new one.

Our investigations to the OpenPLC showed also that critical information, including the operator credentials, are stored in a certain database called "*openplc.db*". Surprisingly, we found that this database is unlocked and can be read by attackers who can customize specific scripts for this purpose. Thus, we can conclude that all the information displayed on the OpenPLC for the operators are accessible and even vulnerable to manipulation scenarios as shown in Section V. What makes the situation worse is that the exposed database contains, among many identifiers (IDs) and values, a very interesting ID called *user_id*. This *user_id* is dedicated to provide information (including the operator credentials) about all the accounts registered in the OpenPLC. Each account is identified by a unique *user_id* and has always the value "10" if the operator uses the default account i.e., "openplc" as user name, and "openplc" as password. Our analyzes showed that, if an operator uses the default account for the first time and then changes the user name and password to another values on his will, the *user_id* value remains as "10". This value is only changed in case the operator removes the default account completely and registers another account. For all this, if an attacker reveals the *user_id* value(s), he will be able to have access to all the accounts registered in the OpenPLC. This scenario is quite severe since attackers can exploit the "*openplc.db*" database, and update the operator account(s) credentials to different ones on their will, so the operator will have no longer access to the OpenPLC.

B. Upload Process

Figure 4 depicts the fully upload-program process the OpenPLC executes, showing the packets order as sent. The operator sends first an "upload-program" packet informing the OpenPLC that he wants to upload a new program in ST format. Then, the OpenPLC responds with an "OK" packet having a status code "200". Afterwards, an "upload-program-action" packet is sent to the OpenPLC to generate a new *prog_id*, *ST file* name and a new upload date. Then the OpenPLC responds with an "OK" packet once all the aforementioned are stored in the database "openplc.db".

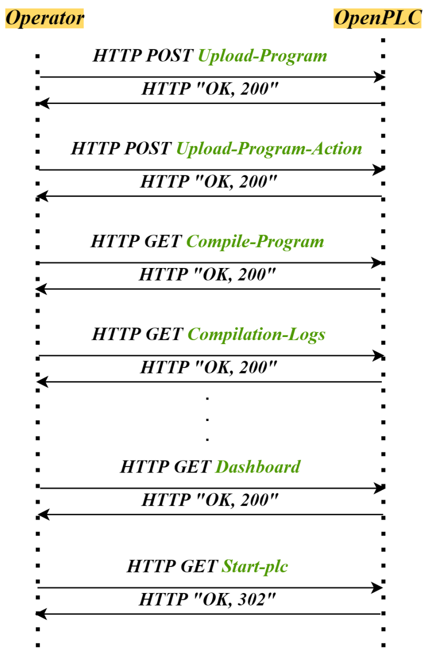


Fig. 4: Upload program sequence diagram

Thereafter, a "compile-program" packet is sent to the OpenPLC and the compilation process from "ST file" to "C file" starts at this point. During the compilation process, many "compilation-logs" packets are sent to the OpenPLC depending on the complexity of the compiled program. Once the compilation process is successfully accomplished, the OpenPLC sends the last "OK" packet ended with a "*compilation finished successfully*" message. In opposite to real PLCs, the OpenPLC does not turn on a *START* mode automatically when the upload process is done. Meaning that, it needs the operator himself to start the OpenPLC and the new program is then executed. If everything is alright and the OpenPLC is on *START* mode, it sends an "OK" packet containing a status code "302". If an error occurs during the upload-program process, the OpenPLC sends an error message with a status code "500", and terminates the upload process.

Our investigations on the upload-program process showed that once the upload is done successfully, the OpenPLC generates a copy of the uploaded program and stores it automatically in a specific folder in the OpenPLC project called *webserver*. This *webserver* is merely a folder containing copies of all the ST files that were uploaded to the OpenPLC, and each copy is identified by a unique *prog_id*. Furthermore, those copies are irremovable and remain in the *webserver* forever. This is confirmed by attempting manually to remove the copies from the *webserver*. Even if the operator removes programs from the OpenPLC dashboard, their corresponding copies will not be removed from the *webserver* folder. From the security point of view, this is a critical security gap as adversaries with appropriate attacking tools can access and read all the programs uploaded to the OpenPLC, including the currently running one. Knowing that, those copies are indeed ready-to-execute ST files, so attackers need only to maliciously modify a copy, and re-upload it again to the OpenPLC using his patching tool, or replacing the currently running program with any other copy of an old program stored in the *webserver*. This attack scenario is extremely effective and not revealed by traditional control logic detection methods e.g., [8]–[11].

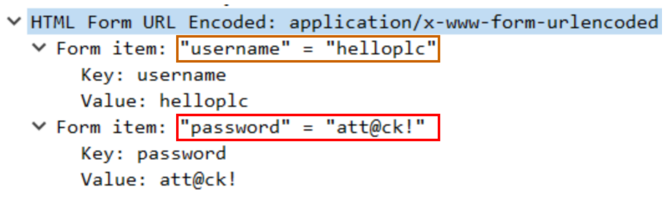
V. Attack Description

Figure 5 depicts a high-level overview of our control logic injection attack. It is comprised of two main phases. 1) Breaking the Authentication, and 2) Patching the OpenPLC. In the following, we elaborate each phase in more detail.

A. Breaking the Authentication

The OpenPLC *Run Time* is protected by requiring user name and password from the operator. Therefore, the attacker needs first to steal the correct credentials to be able to execute any further operation in the OpenPLC, and makes malicious changes afterwards. This can be achieved in two ways. Either by conducting a typical replay attack, or by retrieving the credentials from the *openplc.db* database.

*1) **Replay Attack**:*
Since the OpenPLC does not implement any encryption method on the packets transferred over the HTTP protocol, an attacker can easily capture the user credentials i.e., user name and password exchanged during an authentication session. To this end, we establish our Wireshark tool and capture the "login" packet sent from the operator to the OpenPLC as shown in figure 6.



```
v HTML Form URL Encoded: application/x-www-form-urlencoded
  v Form item: "username" = "helloplc"
      Key: username
      Value: helloplc
  v Form item: "password" = "att@ck!"
      Key: password
      Value: att@ck!
```

Fig. 6: Login packet - user credentials are sent in plaintext

In our given example, the operator credentials are "*helloplc*" as a username, and "*att@ck!*" as a password. After that, an attacker can re-use the credentials to authenticate himself with the OpenPLC and conduct further attacks.

This scenario has limitations. It is only valid if the operator provides his credentials to the OpenPLC while the attacker has already access and listens to the network. Meaning that, if the

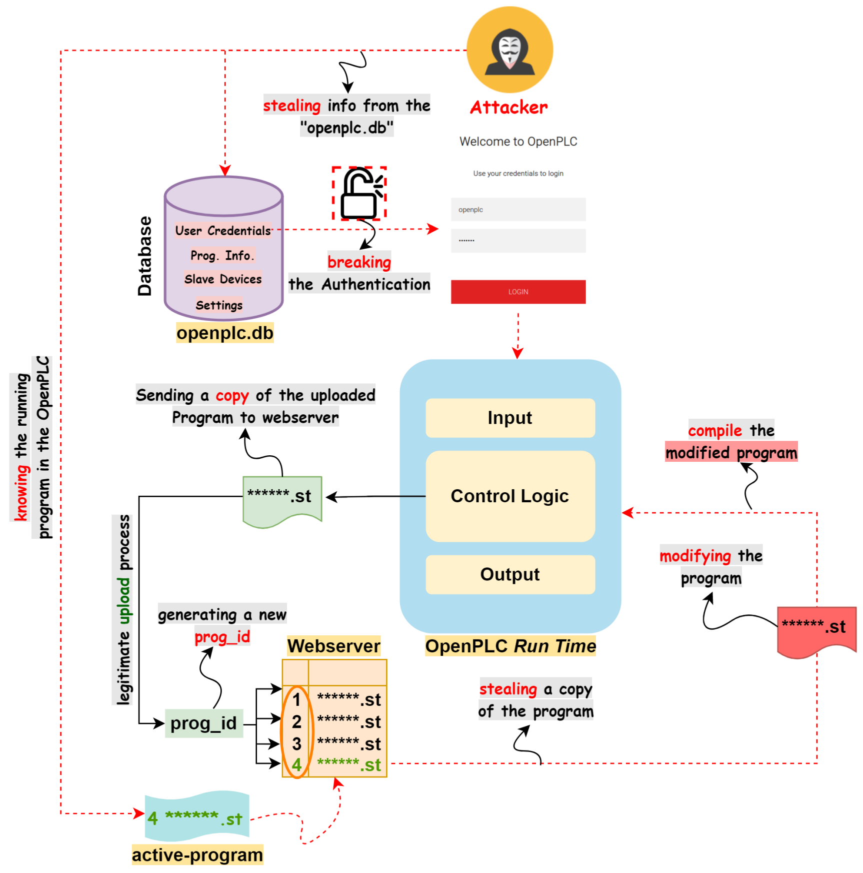
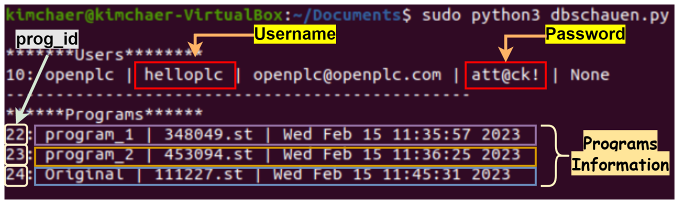


Fig. 5: High-level overview of our attack approach

attacker starts listening to the network after the "login" packet is sent, he would not be able to capture the user credentials over any other HTTP packets. So he needs to wait until the operator authenticates himself again.

2) ***Exploiting the openplc.db database***:

The OpenPLC uses a database to store critical information e.g., uploaded programs, settings, slave devices and user credentials. This database is vulnerable and accessible as explained in Section III. Therefore, we can overcome the challenge of the former scenario V-A1 and extract the operator credentials without the need to wait until a "login" packet is captured. To this end, we wrote a python script that reads the *openplc.db*, and managed successfully to retrieve, among other information, the user credentials, see figure 7.



```
kimchaer@kimchaer-VirtualBox:~/Documents$ sudo python3 dbschauen.py
*******Users********
10: openplc | helloplc | openplc@openplc.com | att@ck! | None
---------------------------------------------------------
******Programs******
22: program_1 | 348049.st | Wed Feb 15 11:35:57 2023
23: program_2 | 453094.st | Wed Feb 15 11:36:25 2023
24: Original | 111227.st | Wed Feb 15 11:45:31 2023
```

Fig. 7: Reading information from the "openplc.db" Database

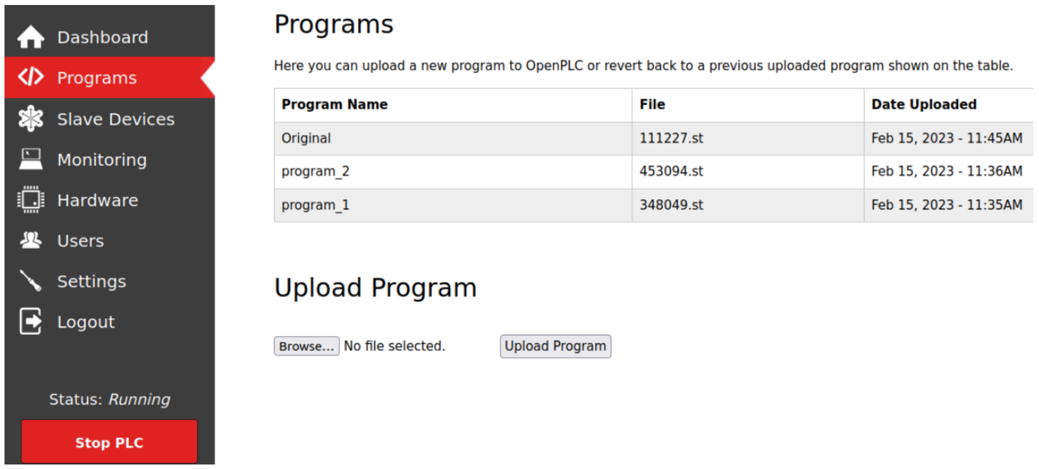
From this point on, the attacker can authenticate himself and conduct further malicious operation by sending first a crafted HTTP "login" packet containing the correct user name and password. Furthermore, the attacker can also prevent the legitimate operator from accessing the OpenPLC by manipulating the user information stored in the *openplc.db*. Thus, when the operator provides his credentials to access the OpenPLC, the later will compare the given credentials with the ones stored in the *openplc.db*, and there will be no match. This makes the OpenPLC not authenticate the legitimate operator causing a denial of access situation.

B. *Patching the OpenPLC*

To update the program running in the OpenPLC, the attacker needs first to find out which program the OpenPLC is executing currently. Then, he steals the program, modifies it, and finally forces the OpenPLC to execute the malicious program. In the following, we illustrate each step in detail.

1) ***Stealing the Currently Running Program***:

Accessing the *openplc.db* database helps the attacker to get critical information about all the programs uploaded to the OpenPLC (see figure 7). However, the attacker still needs to know which program exactly the OpenPLC is currently executing. Our investigations showed that the attacker can reveal the name of the running program by capturing specific packets exchanged between the operator and the OpenPLC, precisely the "compile-program" packet and "200 OK" responds to certain operations as shown in figure 8.



Programs

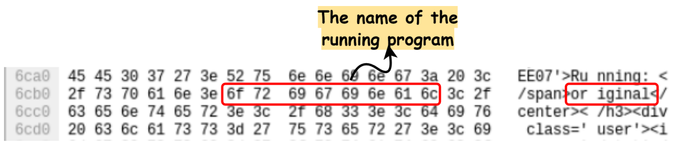
Here you can upload a new program to OpenPLC or revert back to a previous uploaded program shown on the table.

Program Name	File	Date Uploaded
Original	111227.st	Feb 15, 2023 - 11:45AM
program_2	453094.st	Feb 15, 2023 - 11:36AM
program_1	348049.st	Feb 15, 2023 - 11:35AM

Upload Program

No file selected.

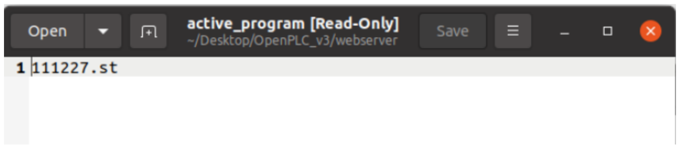
Fig. 10: OpenPLC Run Time - Programs List



```
6ca0  45 45 30 37 27 3e 52 75  6e 6e 69 6e 67 3a 20 3c   EE07'>Ru nning: <
6cb0  2f 73 70 61 6e 3e 6f 72  69 67 69 6e 61 6c 3c 2f   /span>or iginal</
6cc0  63 65 6e 74 65 72 3e 3c  2f 68 33 3e 3c 64 69 76   center>< /h3><div
6cd0  20 63 6c 61 73 73 3d 27  75 73 65 72 27 3e 3c 69    class=' user'><i
```

Fig. 8: "OK" packet contains the running program name

By conducting further analyzes, we found that the OpenPLC uses an index called "active-program" to indicate the currently running program. This index has only a single value which is the ST file name see figure 9. Therefore, an attacker can read the value of this index to know the ST file name of the program running on the OpenPLC.



```
1 111227.st
```

Fig. 9: Reading the content of the active-program index

With the help of the *openplc.db* information, we can get the *prog_id*, and thus recall the corresponding copy of this program that is stored in the *webserver* folder, knowing that the attacker can recall any copy from the *webserver* with only knowing the ST file name and its corresponding *prog_id*. In our example, the actual running program has the value "24" as a *prog_id*, and "111227" as an ST file name (see figure 7).

2) ***Manipulating the User-Program:***

The attacker has the original program the OpenPLC runs in an ST format. This ST file is executable only on OpenPLC *Run Time*. Meaning that, the attacker can not retrieve the high-source code written in one of the IEC 61131-3 programming languages. Therefore, he needs to modify the ST file in its currently format either manually, or automatically as a part of our attack by applying a based-rules modification approach [7]. The modification includes overwriting the original program by inserting/removing instructions, modifying set-points, replacing operators in equations and changing inputs/outputs statuses.

3) ***Updating the OpenPLC Program:***

After we modify the user-program successfully, the next step is to upload and execute the malicious program in the OpenPLC. This can be done by conducting a completely new upload-program process as depicted in figure 4. But using the upload-process "as-is" will reveal our injection. The OpenPLC will generate a new copy of the uploaded malicious program with a new *prog_id* in the *webserver* folder. Furthermore, the operator will easily notice that there is a new program added to the "Programs List" with a new "ST file" name and date as shown in figure 10. This holds true even if we re-upload the same program to the OpenPLC i.e., each time there is an upload-program process performed, the OpenPLC adds a new program to the "Programs List" with different information. For all this, patching our modified program using an upload-program process is not an appropriate method since we aim at making our attack as stealthy as possible.

Our attack approach is designed to steal a copy of the actual program running in the OpenPLC (see figure 5). In fact, this stolen program was already uploaded to the OpenPLC by the legitimate operator. Meaning that, it has already a certain *prog_id*, an ST file name, and a date of upload. For all this, we just need to compile our modified program to get its "C file" version, and then force the OpenPLC to turn on *START* mode, executing the attacker program as depicted in figure 11.

This scenario is feasible due to two facts. First, the OpenPLC generates a new *prog_id*, ST file name and date of upload at the very beginning of the upload process and before the compilation process starts, precisely over "upload-program" and "upload-program-action" packets. Therefore, an attacker can skip those packets since the program that he modified was already registered in the OpenPLC and has a *prog_id*. Secondly, compiling the program is processed anyway without inspecting the arrival of prior packets to the OpenPLC. For all this, compiling the modified program is sufficient to update the OpenPLC program.

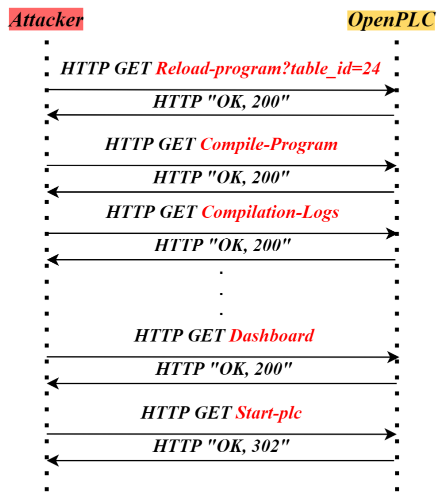
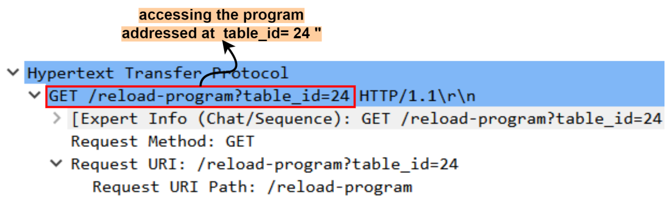


Fig. 11: Patching the OpenPLC sequence diagram

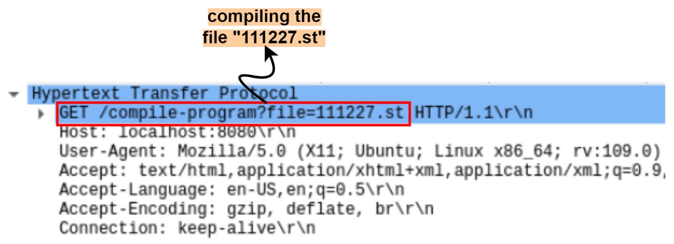
Our attack tool sends first a "reload-program" packet as shown in figure 12. This packet allows us to have access to the "*Programs Information*" on the OpenPLC, where we can perform different operations e.g., launch, update, or remove the currently running program from the OpenPLC. In our example, we have three programs: "program_1", "program_2" and "Original" using the *prog_id* values 22, 23 and 24 respectively (see figure 7 and 10). To craft our "reload-packet", we just need to use the *prog_id* (called also *table_id*) of the currently running program which is in our example "24".



```
Hypertext Transfer Protocol
  GET /reload-program?table_id=24 HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /reload-program?table_id=24
    Request Method: GET
    Request URI: /reload-program?table_id=24
      Request URI Path: /reload-program
```

Fig. 12: Reload program packet

After selecting the program that the attacker wants to change, next step is to compile the malicious program from its "ST file" format to "C file". This is done by sending a "compile-program" packet provided with the original ST file name e.g., 111227.st in our example, see figure 13.

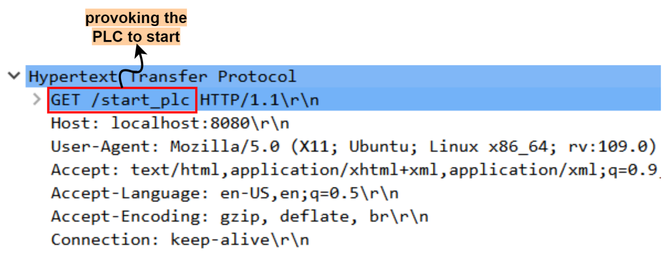


```
Hypertext Transfer Protocol
  GET /compile-program?file=111227.st HTTP/1.1\r\n
  Host: localhost:8080\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0)
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate, br\r\n
  Connection: keep-alive\r\n
```

Fig. 13: Compile program packet

The compilation process starts by recalling the copy of the "111227.st" file from the *webserver*. This copy was already modified by the attacker, and the OpenPLC compiles it to an executable "C file". After the compilation process is successfully done, the OpenPLC sends the last "OK" packet containing a "compilation finished successfully" message.

The OpenPLC *Run Time* does not run automatically after the compilation process is finished. Therefore, the attacker needs to provoke the PLC to start operating after injection his malicious program. Figure 14 shows a "start_plc" packet that attacker sends in order to run the OpenPLC and executes his malicious program.



```
Hypertext Transfer Protocol
  GET /start_plc HTTP/1.1\r\n
  Host: localhost:8080\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0)
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate, br\r\n
  Connection: keep-alive\r\n
```

Fig. 14: Start PLC packet

VI. Vulnerabilities & Security Recommendations

Our experimental results showed that the OpenPLC project is widely vulnerable and exposed to serious manipulations if an attacker manages successfully to access the control center. In the following, we summarize the disclosed vulnerability in this work and suggest some security recommendations for the founder to close those security gaps.

A. *Replay Attack is feasible*

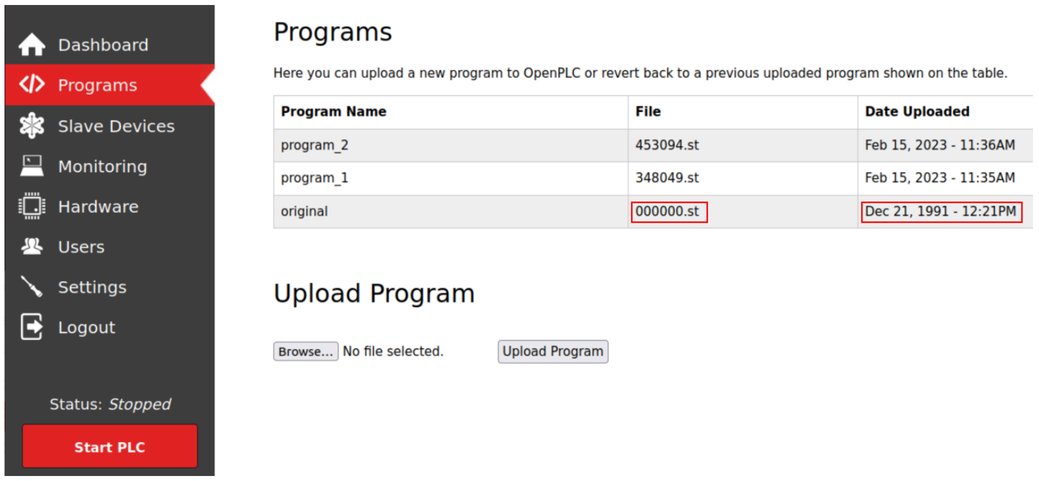
Since the OpenPLC does not implement any integrity check or anti-replay mechanisms to protect its control logic from unauthorized manipulation, attackers are capable of reproducing pre-recorded packets from old sessions to patch the OpenPLC with their malicious programs. For this, we highly recommend to improve the project by introducing encryption algorithms e.g., Advanced Encryption Standard (AES), Rivest Shamir Adleman (RSA), Triple Data Encryption Standard (DES), etc.

B. *Bypassing the Authentication is feasible*

Our investigations showed that the operator credentials are sent clear over HTTP "login" packets. Therefore, an attacker can retrieve the required credentials by capturing the "login" packets. Thus, we recommend to enhance the OpenPLC project by encrypting the password using one of the common encryption algorithms for this purpose e.g., XOR, Secure Hash Algorithm (SHA), etc.

C. *The openplc.db database is vulnerable*

The attack approach presented in this paper was successful due to the fact that the *openplc.db* is accessible and attackers can read/write from/to this database. We showed that even if the attacker missed the authentication session between the



Programs

Here you can upload a new program to OpenPLC or revert back to a previous uploaded program shown on the table.

Program Name	File	Date Uploaded
program_2	453094.st	Feb 15, 2023 - 11:36AM
program_1	348049.st	Feb 15, 2023 - 11:35AM
original	000000.st	Dec 21, 1991 - 12:21PM

Upload Program

Fig. 15: Programs List - attacker fakes the information of listed programs

legitimate operator and OpenPLC, he still can retrieve the credentials by reading the content of the *openplc.db*. Based on our findings, we recommend the founder to limit the access to the *openplc.db* to none at all. This database has critical information that helped us to read and manipulate many values e.g., *prog_id*, ST file names, upload dates, user credentials, etc. To highlight the sensitivity of the information stored in the *openplc.db*, we manipulated values to trick the operator by showing him false information related to the uploaded programs see figure 15. We managed successfully to fake the information displayed on the "Programs List" showing the operator wrong ST file name as well as upload date. This is a quite severe vulnerability and an immediate action is required.

D. The webserver is not Read/Write protected

Our experiments proved that seizing the *webserver* folder by attackers could bring serious risks to the OpenPLC. This folder contains ready-to-execute programs and is also accessible by anyone, including attackers with unauthorized access to the OpenPLC. Furthermore, the programs stored in the *webserver* are irremovable, and stay in the folder forever even if the operator removes programs from the OpenPLC i.e., from the "Programs List". The main purpose of the *webserver* folder is to recall copies and execute them in case the operator launches or updates an already uploaded program to the OpenPLC. But this also opens the door for attackers to exploit the *webserver*, and then steal and modify the copies stored there. For all this, we highly recommend the founder to protect the *webserver* with some read-write protections e.g., password, or implementing strict rules that prevent any kind of unauthorized accesses. We believe that such methods will prevent our attack scenario or similar control logic injection attacks.

VII. Conclusion

This work shows that the OpenPLC project is vulnerable and has various security issues the founder needs to consider in the near future. We conducted intensive investigations and proved that attackers can break the authentication of the OpenPLC, and change the user-program running with a malicious one. Our attack approach is completely stealthy and the operator will not record any abnormal changes neither on the OpenPLC interface nor in the *webserver* or *openplc.db* database. We finally suggest some security recommendations to the founder, engineers, and security community hoping they can close all the vulnerabilities reported in this paper to secure the OpenPLC based environments.

References

[1] T. Alves and T. Morris, "OpenPLC: An IEC 61131–3 compliant open source industrial controller for cyber security research," Comput. Secur., vol. 78, pp. 364–379, Sep. 2018.
[2] M. Tiegelkamp, K. John, "IEC 61131-3: Programming industrial automation systems," Springer, 1995.
[3] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-31630.
[4] 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," in IEEE Access, vol. 10, pp. 9164-9173, 2022, doi: 10.1109/ACCESS.2022.3144027.
[5] W. Alsabbagh, S. Amogbonjaye, D. Urrego and P. Langendörfer, "A Stealthy False Command Injection Attack on Modbus based SCADA Systems," 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2023, pp. 1-9, doi: 10.1109/CCNC51644.2023.10059804.
[6] "MITRE ATTCK," https://attack.mitre.org/, 2020.
[7] W. Alsabbagh, "Investigating Security Issues in Programmable Logic Controllers and related Protocols," Dissertation submitted to the faculty MINT - Mathematik, Informatik, Physik, Elektro und Informationstechnik of the Brandenburg University of Technology Cottbus-Senftenberg.
[8] S. McLaughlin, S. Zonouz, D. Pohly and P. McDaniel, "Trusted Safety Verifier for Process Controller Code," In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2014.
[9] S. Zonouz, J. Rrushi and S. McLaughlin, "Detecting Industrial Control Malware Using Automated PLC Code Analytics," IEEE Secur.Priv. 2014, 12, 40–47.
[10] T. Chang, Q. Wei, W. Liu and Y. Geng, "Detecting plc Program Malicious Behaviors Based on State Verification," Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2018; Volume 11067, pp. 241–255.
[11] Y. Xie, R. Chang and L. Jiang, "A malware detection method using satisfactorily modulo theory model checking for the programmable logic controller system," Concurr. Comput. Pract. Exp. 2022, 34, e5724.