Gadgets of Gadgets in Industrial Control Systems: Return Oriented Programming Attacks on PLCs

Adeen Ayub[†], Nauman Zubair[*], Hyunguk Yoo[*], Wooyeon Jo[†], Irfan Ahmed[†]
[†] Department of Computer Science, Virginia Commonwealth University, USA
[*] Department of Computer Science, University of New Orleans, USA
[†]{ayuba2, jow, iahmed3}@vcu.edu, [*]{nzubair@my.uno.edu, hyoo1@uno.edu}

***Abstract*—In industrial control systems (ICS), programmable logic controllers (PLCs) directly control and monitor physical processes in real-time such as nuclear plants, and power grid stations. Adversaries typically transfer malicious control logic to PLCs over the network to sabotage a physical process. These control logic attacks are well-understood containing machine instructions in network packets and are likely to be detected by network intrusion detection systems (IDS). On the other hand, return-oriented programming (ROP) reuses blocks (or gadgets) of existing code in computer memory to create and execute malicious code. It limits or eliminates the need to transfer machine instructions over the network, making it stealthier. Currently, ROP attacks on control logic has never been discussed in the literature to explore it as a practical ICS attack. This paper is the first attempt in this direction to explore challenges for a successful ROP attack on real-world PLCs, including maintaining a continuous (control logic) scan cycle through ROP gadgets, no user input (to cause a buffer overflow) to overwrite the stack for gadget installation, and limited ROP gadgets in a PLC memory to find blocks of instructions equivalent to the high-level constructs of PLC programming languages (such as instruction list, and ladder logic). We identify and utilize typical PLC design features (that we find exploitable) to overcome these challenges, which makes ROP attacks applicable to most PLCs e.g., no stack protection, and remote access to certain PLC memory regions via ICS protocols. We demonstrate two successful ROP attacks on the control logic programs of three fully-functional physical processes, i.e., a belt conveyor system, a four-floor elevator, and a compact traffic light system. The first ROP attack manipulates a PLC's current control logic and has two variants involving either a single or multiple gadgets; the second ROP attack constructs a control logic from scratch using gadgets in a PLC's memory. Our evaluation results show that the attacks can be performed using a set of small-sized gadgets with no significant effect on a PLC's scan time.**

***Index Terms*—ROP, PLCs, ICS attacks, control logic**

I. Introduction

Industrial control systems (ICS) automate critical operations in physical processes, e.g., power generation and distribution, gas pipelines, and water treatment plants [1], [2]. Several pieces of malware demonstrated destructive capabilities against ICS networks including CrashOverride [3], Havex [4], and HatMan [5], [6]. Recently, DHS CISA reported Russian state-sponsored cyber operations against Ukrainian Critical Infrastructure [7] causing denial-of-service and deployment of KillDisk and other destructive malware. Effective defensive strategies require a deeper understanding of cyber attacks [8], [9] and malware targeting ICS environments. This paper contributes to this direction and explores the feasibility and challenges of return-oriented programming (ROP) in ICS.

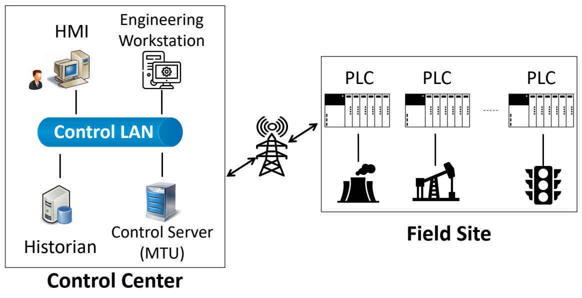


Fig. 1: Overview of an industrial control system environment

Figure 1 shows a typical ICS environment. It consists of 1) a control center and 2) a field site. The control center consists of ICS services such as a human-machine interface (HMI), an engineering workstation, a historian, and a master terminal unit (MTU). The field sites consist of the actual physical processes monitored and controlled via sensors, actuators, and programmable logic controllers (PLCs). PLCs are embedded devices equipped with control logic programs that define how a physical process is controlled. Attackers typically target a PLC's control logic to sabotage a physical process [10]–[22]. They typically involve transferring malicious control logic to a PLC over the network to either modify an original control logic or inject a new control logic into a PLC's memory. These attacks are well-understood containing substantial machine instructions in network packets that can be detected by network IDS [19], [23], [24].

On the other hand, ROP reuses existing code in a target computer system, limiting or eliminating the need to transfer machine instructions over the network, making it stealthier. Specifically, an ROP attack uses in-memory blocks of consecutive machine instructions called gadgets, each ending with a 'return' instruction. It executes the gadgets in a specific sequence to perform a malicious operation on a target application. Thus, it does not require injecting a new malicious code to launch an attack [25]–[27]. Currently, ROP attacks on a PLC's control logic have not been discussed in the literature, showing the research gap in exploring ROP attacks as a potential attack vector for ICS.

This paper is the first attempt to fill the gap and explore challenges for a successful ROP attack on real-world PLCs. We show that ROP can be used to attack a control logic in a

PLC, which hampers the functioning of the connected physical process. Our ROP attack consists of three main steps: 1) We use an ICS communication protocol to read the PLC's memory over the network and acquire its dumps; 2) we then utilize a disassembly tool to disassemble the dumps and find gadgets in the memory; 3) we finally update the stack with the gadgets we want to execute. To perform these steps, we identify several challenges including maintaining a continuous scan cycle through ROP gadgets, no user input (to cause a buffer overflow) to overwrite the stack for gadget installation, and limited ROP gadgets in a memory to find blocks of instructions equivalent to the high-level constructs of PLC programming languages such as instruction list, and ladder logic.

We overcome these challenges by utilizing typical PLC design features (that we identify as exploitable), making ROP attacks applicable to most PLCs, e.g., no stack protection, and ICS protocols reading/writing to certain PLC memory regions remotely. We demonstrate two successful ROP attacks (i.e., ROP1 and ROP2) on PLCs' control logic. ROP1 adds gadgets to current control logic programs to induce malicious control behavior. It has two variants; one involves only a single gadget, while the other utilizes multiple gadgets to manipulate actuators. ROP2, on the other hand, constructs a control logic from scratch using gadgets. We evaluate the attacks on the control logic programs of three fully-functional physical processes, i.e., a belt conveyor system, a four-floor elevator, and a traffic light system. Our evaluation results show that small-sized (11 bytes) code and gadgets are undetectable and can create malicious control logic without substantially affecting PLCs' scan time in μs.

The contribution of the paper is as follows:

- We show that ROP gadgets can create malicious control logic that can be installed and executed by exploiting typical PLC design features applicable to most PLCs.
- Our ROP attacks cover two practical scenarios, where they either add one or more gadgets to a PLC's original control logic to induce malicious functionality or use a gadget chain to create malicious logic from scratch.
- We achieve a continuous (control logic) scan cycle through ROP gadgets. It is contrary to typical Information Technology (IT) systems where an ROP gadget chain executes once to obtain a command shell.
- We perform our experiments on real-world settings involving Schneider Electric's M221 PLC and three fully-functional physical processes (belt conveyor system, four-floor elevator, traffic light system.).

The rest of the paper is organized as follows. Section III gives the motivation while Section II provides the background. Section IV presents our adversary model and our proposed ROP attacks. Section V presents the challenges in a successful ROP attack on a control logic and their solutions through exploitable PLC design features. Section VI presents the steps taken in the attack implementation. Section VII evaluates our attacks on the control logic programs of three different physical processes in terms of scan time, and number and size of gadgets. Section VIII discusses the related work, followed by mitigation strategies for the attacks in Section IX. Section X concludes the ROP study.

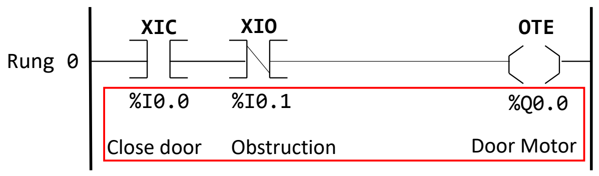
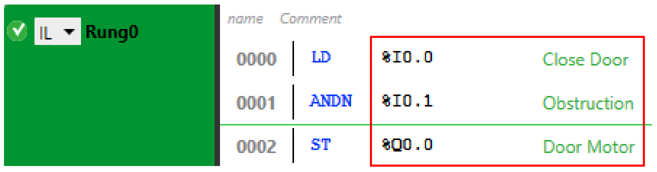


Fig. 2: Door Motor in Ladder Logic programming



```
0000 | LD    %I0.0    Close Door
0001 | ANDN  %I0.1    Obstruction
0002 | ST    %Q0.0    Door Motor
```

Fig. 3: Door Motor in Instruction List

II. Background

A. Programmable Logic Controllers

PLCs are embedded devices programmed to monitor and control physical processes automatically. They have dozens to thousands of digital/analog inputs and outputs wired with sensors and actuators that connect them to a physical process.

Control Logic is a program that determines how to control and monitor physical processes at the field site of ICS. These control logic programs are compiled and programmed using vendor-supplied engineering software running on an engineering workstation which is typically located at the control center of an ICS. IEC61131-3, a standard adopted by most of the PLC manufacturers, defines five programming languages to write a control logic: ladder logic (LD), instruction list (IL), functional block diagram (FBD), structured text (ST), and sequential function chart (SFC). The process of writing a control logic program from engineering software to a PLC is known as *downloading*, while reading from the PLC is known as *uploading*. A control logic program runs in a scan cycle once it is compiled and downloaded into a PLC. This scan cycle consists of three main steps:

1) *Read inputs*: The CPU reads inputs of sensors and other connected devices and updates an *input table* in memory.

2) *Run*: After the CPU has read or scanned the inputs, it runs the control logic on the *input table* and modifies an output image table based on the execution results.

3) *Control*: Controlling connected output devices (actuators, lights) by giving signals based on the *output table*.

Figure 2 shows *rung 0*'s LD of a control logic program that closes the door if "close door" is selected and no obstruction between the door is detected. The open contact represents the "close door" push button, while the closed contact represents an "obstruction" detector. Figure 3 shows *rung 0*'s IL of the PLC control logic program.

B. Return Oriented Programming

ROP was first introduced by Shacham [25] as a return-to-libc attack. This attack bypasses data execution prevention (DEP) that makes certain parts of the memory (particularly writable ones) non-executable. ROP uses executable instruction sequences (referred to as gadgets) already present in the memory to run malicious code; it doesn't need to transfer new executable code to a target system. Each gadget ends with a 'return' instruction. An attacker typically populates a process stack with the gadget addresses to execute malicious code.

III. Motivation and Problem Statement

An ICS malware targets a PLC's control logic to disrupt industrial processes. For instance, Stuxnet [10] infects the control logic of Siemens S7-300 PLCs to attack centrifuges of a nuclear plant; Triton/Trisis [28] disables Schneider Electric's Triconex Safety Instrumented Systems (SIS) by replacing the original ladder logic with an infected one. Further, control logic attacks have been studied to understand the manipulation of ICS protocols to inject malicious control logic into a PLC [16], [18], [29]–[31].

While these attacks successfully target a PLC's control logic, most of them have a considerable downside. They involve sending substantial malicious code to a PLC over the network to be noticed by IDS [19], [32], [33]. For instance, Stuxnet malware is half a megabyte in size [34]. Furthermore, a strict firewall can prevent reading/writing to the control logic of a PLC remotely. For instance, ICS protocols have an address field to access different memory regions of a PLC, including control logic. The firewall rules can block packets containing control logic addresses.

Under these usual circumstances, the attacker can still access the I/O data blocks of a PLC remotely using ICS protocols. HMI, historian, and other ICS services in the control center exchange I/O data with PLCs. IDS and firewalls do not block I/O data to maintain remote visibility of the underlying physical process. The attacker may use a PLC's data blocks to inject malicious control logic. Yoo *et al.* [16] has demonstrated a data execution attack, which involves transferring a malicious control logic to the data blocks of a PLC over the network and executing it by changing a pointer in a configuration block that points to the start of a PLC's control logic. While their attack subverts the firewall, an IDS can still notice their control logic code in transit [19].

ROP can potentially exploit a PLC's data blocks effectively since it traditionally does not require transferring executable code over the network and uses gadgets already in a target device's memory.

Problem Statement. Given an ICS environment running IDS to monitor any transfer of control logic, the attacker's goal is to utilize a PLC's data blocks remotely for ROP to execute malicious control logic.

IV. ROP Attacks on Control Logic

A. Adversary Model

Our adversary model assumes that an attacker accesses the control center network via a typical IT attack vector, such as an infected USB stick, similar to real-world ICS attacks, e.g., TRITON [28] and Ukraine Power grid attack [7]. The control center infiltration enables the attacker to communicate with a target PLC over the network through an ICS protocol to launch an ROP attack. During this process, the attacker uses existing instructions in a PLC's memory to avoid sending huge amounts of malicious code over the network that stands a chance of being detected by intrusion detection tools.

If the PLC has password authentication against unauthorized read/write messages, the attacker will utilize the attack methods documented by Ayub *et al.*'s study [35] to bypass PLC authentication.

B. Proposed ROP Attacks

We introduce two ROP attacks on a PLC's control logic to sabotage a running physical process by changing the process to an unexpected state. For instance, the attacker turns off the motor that moves a conveyor belt. The attacker typically achieves it by manipulating actuators through a compromised PLC. Section VI-B illustrates the ROP attacks in detail.

ROP 1. This attack integrates gadgets with a PLC's original control logic to induce malicious control behavior. We explore two variants; one uses a single gadget to manipulate the running control logic, and the other uses a gadget chain consisting of multiple gadgets.

Variant I: This ROP variant utilizes a gadget that allows manipulation of the value of the register associated with the output port of the PLC. Since only one gadget is used, it simplifies the attack execution and has less impact on the control logic scan cycle (depending on the gadget size). However, a single gadget gives less control over the values of each output port. In other words, the attacker can only set or unset some bits of the output register, limiting the capability to manipulate associated actuators.

Variant II: This ROP variant utilizes a set of gadgets to have greater control of the output ports of a PLC. It employs a gadget chain to set or unset any value of the output register and hence the corresponding output ports. Note that the first variant does not give an attacker much flexibility and control of the PLC's output ports, whereas, with the second variant, the attacker can control any output port and the connected actuators. However, more gadgets add complexity to the attack, and their combined code is likely to be more than a single gadget code of variant I, causing a longer scan cycle of the control logic program.

ROP 2. This ROP attack utilizes gadgets to construct a new control logic from scratch, allowing the attacker to manipulate a physical process to a large extent. The malicious control logic uses the data from the PLC's input devices, such as sensors, to manipulate a physical process to an unexpected state. For instance, while riding an elevator, a user requests a second floor, but the connected (compromised) PLC always skips the second floor and takes the elevator to a random floor.

The effectiveness of this attack depends on the availability of gadgets in a PLC memory, which may limit the attacker from creating a desired malicious control logic. Note that this ROP attack differs from ROP 1 attack, which retains the original control logic and adds a gadget or a sequence of gadgets to induce malicious functionality.

V. Challenges in ROP Attacks on PLC

We identify several challenges for a successful ROP attack on a PLC and devise techniques through PLC design features/choices that we find exploitable.

A. *Challenges*

We define the challenges into three groups related to malicious control logic, stack manipulation, and ROP attack execution. Figure 4 shows which part of the ROP attack each challenge corresponds to. C1 (in yellow) represents challenges related to malicious control logic, C2 (in orange) represents challenges related to stack manipulation, and finally, C3 (in green) represents challenges related to ROP attack execution. A gray area indicates that the stack is not directly accessible in a PLC via an ICS protocol. The four steps (a) - (d) on the right shows that ROP attack is continuously executed. The details are described in the proposed techniques.

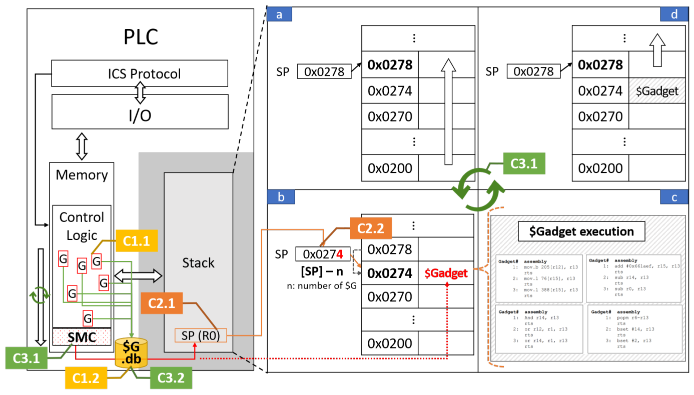


Fig. 4: Challenges in ROP Attacks on PLC
(a) Normal Operation Running
(b) SMC Executed to Run Gadget
(c) Gadget Execution
(d) Back to Normal Operation

1) *Challenge 1: Malicious Control Logic:* .

Challenge 1.1: Determining the start and end addresses of the control logic. Engineering software downloads a control logic on a PLC. The control logic location in the PLC memory is helpful for the attacker to determine gadget addresses in the control logic code and set up an initial attack vector for stack manipulation (discussed later in detail).

Challenge 1.2: Deciding which gadgets are useful. A malicious control logic produces an undesired state of a physical process. Given a set of available gadgets in a PLC memory, it is a challenge to create a gadget chain of malicious control logic that can cause an adverse effect on a physical process.

2) *Challenge 2: Stack Manipulation:* .

Challenge 2.1: Overwriting the stack without buffer overflow. Traditional ROP attacks exploit a buffer overflow vulnerability in a user application to inject a malicious payload consisting of gadget addresses and register values. But in PLCs, control logic programs run standalone on I/O data and do not give access to the stack through remote user input. Figure 5 shows a PLC's high-level memory layout. Since a user input to control logic occurs through ICS protocols to read/write I/O data, a buffer overflow vulnerability does not apply as the I/O memory block lies outside of the stack region.

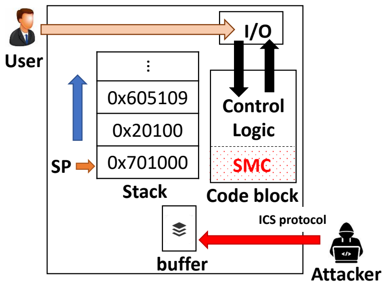


Fig. 5: High-level memory layout of a PLC

Challenge 2.2: Executing an ROP gadget chain while ensuring normal PLC operations. While updating the stack and the stack pointer to execute a gadget chain, it is challenging to ensure that the PLC is functioning normally. For instance, since the stack changes dynamically, the attacker cannot write a gadget chain in any part of the stack memory. Unlike PLC, in a traditional ROP attack, buffer overflow populates the stack automatically with a gadget chain.

3) *Challenge 3: ROP Attack Execution:* .

Challenge 3.1: Maintaining a continuous attack cycle through ROP gadgets. Control logic has a scan cycle to maneuver a physical process continuously using the process I/O data. The gadget chain has to run on each cycle to sustain the malicious state of the process. In other words, if the gadget chain runs only once, in the next cycle, the control logic (without the gadget chain) will undo the malicious change. For example, the benign control logic in a water treatment facility is supposed to release a chemical by opening a valve under a certain (water) condition. However, the gadget chain triggers and closes the valve. In the next cycle, the control logic will again attempt to open the valve, and this time without the gadget chain, the valve will remain open.

Challenge 3.2: Persistent control of a compromised PLC. The persistent control allows the attacker to update the gadget chain running on a compromised PLC. For instance, in a ransomware attack, the first gadget chain can show that a physical process is under the attacker's control (let's say by disabling a release valve in a gas pipeline); the attacker can later update the gadget chain to damage the process if a ransom amount is not received within a given time period.

B. *PLC Exploitable Features*

Predictable control logic location. A control logic address in a PLC memory is either fixed for a PLC model or can be found in a PLC's configuration block. For instance, Modicon M221 PLC (of Schneider Electric) has a configuration block that contains a pointer to the start of the code block (including the control logic), which can be read over the network using the UMAS protocol. MicroLogix 1100/1400 PLC (of Allen-Bradley) has a fixed control logic address that can be found by downloading a control logic to a PLC and then analyzing its network traffic.

Using the mapping between CPU registers and I/O ports for searching gadgets. PLCs use preset CPU registers for mapping to input and output ports. The attacker can use these registers to find relevant gadgets to read input port data and write to the output port data to control actuators.

For instance, a control logic program written for an elevator will have specific components (such as LED lights) connected to the output ports of a PLC, which turn on and off according to the program written. Disassembling the control logic program shows that each bit of the input and output register maps to a different input and output port, respectively.

Download and upload capability via ICS protocols. Control engineers use an ICS protocol to download/upload and maintain control logic remotely using PLC engineering programming software. An ICS protocol, if successfully reverse-engineered, can be used to read and write to a PLC's memory. Moreover, PLCs do not differentiate between a legitimate download request and one from an attacker. PLCs employ password-based authentication mechanisms to prevent attackers from writing malicious control logic. However, these mechanisms have been proven weak and are easily exploitable across most PLCs [35], [36].

No stack protection. The stack is a critical memory and is mostly protected by checking whether code is executable, valid, or contains prohibited functionality before being loaded onto the stack. However, PLC does not have proper stack protection, so if it has a way to access the stack area, many malicious capabilities can be secured.

The stack area in PLCs is inaccessible through user input, and PLC is no different. However, it is possible to indirectly get access to the stack area using the features of PLC that repeatedly executes control logic. PLCs typically use a stack pointer to maintain the position of the stack in the memory. After the first instruction in the stack is popped, the stack pointer increments and goes to the next instruction. PLCs do not enforce stack protection, allowing an attacker to modify its contents just by going to the address the stack pointer points to and then modifying that region with the attacker's gadget address. Also, the stack pointer register is fixed and allows all kinds of operations (such as logical and arithmetic) on it.

C. Proposed Techniques

We propose four techniques (T) to address the challenges.

T1: Finding Useful Gadgets (FUG). To solve challenge 1.2, we use the following technique to find useful gadgets in control logic programs. This technique finds registers that directly map to the input/output ports of a PLC. To accomplish this task, we first write different control logic programs, disassemble them, and notice a pattern from which we discover the register associated with input and the one related to output. In some attacks, however, we use other gadgets as well. Addresses of all gadgets determined to be useful are stored in the database as shown in Figure 4 and 6.

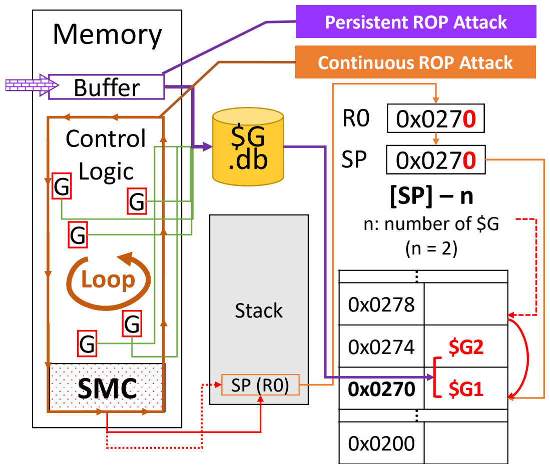


Fig. 6: Continuous and Persistent ROP Attack

T2: Stack Modification Code (SMC) Injection. Stack Modification Code (SMC) is a small-size machine code (only 11 bytes for M221 PLC and undetectable by IDS in our experiments), written in assembly language that is enough to modify a PLC's stack to run gadgets. In this technique, the attacker appends SMC to the control logic written on a PLC's memory (as shown in Figure 4- a to 4- d). The attacker first finds the end address of the control logic. This can be done by reading the start address of a control logic and then calculating its size or by reading the last 'return' instruction in the control logic area of a PLC's memory via an ICS protocol. Once the end address of the control logic is found, an attacker uses an ICS protocol to inject SMC in place of the last 'return' instruction of the control logic.

This technique ensures the attacker achieves a continuous (control logic) scan cycle through ROP gadgets. If SMC is properly inserted at the end of the control logic, SMC will be executed in every cycle of PLC operations as shown in Figure 6 as part of the control logic and is called a continuous ROP attack. Without this technique in place, an ROP attack would have been a one-time attack since once the stack pops a value, it needs to be pushed in again to run again. Moreover, our evaluation results show SMC is not detected by IDS.

T3: Stack Modification. In this technique, we use the stack pointer in the PLC to go to the stack to modify its contents to solve the problem of overwriting the stack with our gadget addresses. To ensure that the modification does not affect the normal functionality of PLCs (as the modification can overwrite the stack contents that are still to be executed), we decrement the stack pointer's value by one step (per gadget) and then modify the stack location it points to.

Figure 4(a) shows the state of the stack and the stack pointer value in a normal setting. Figure 4(b) shows the state after SMC runs as part of the control logic. Decrementing by one step ensures that only the return address that has already been returned (after successful execution of the normal instruction) gets overwritten. After the gadgets are executed, the pointer increments again and executes the next instruction (as seen in Figure 4(d)). If we modify the current stack pointer value, the instruction at the return address still to be run would be overwritten, which can make the PLC malfunction. Using this stack modification technique, it is possible to ensure that PLC functioning is not affected.

T4: Persistent ROP Attack. As shown in Figure 5 (in purple), an attacker uses an ICS protocol to write a payload onto a PLC's memory region that is free to use. It can be in the data block or another free region. We call this space a 'buffer' loaded with gadget addresses that are later loaded onto the stack with the attacker's SMC. This technique makes sure the attacker has persistent control over the kinds of gadgets he wants in the stack. An attacker needs to load this buffer with his new payload (containing the gadget addresses) every time he wishes to execute his ROP attack.

Figure 6 shows a complete ROP attack. An attacker pre-loads the buffer with a payload which is loaded onto the stack with SMC by first decrementing the value of the stack pointer per gadget and then loading the gadget addresses in the location where the stack pointer points. We assume the address is 4 bytes long, so each decrement results in a decrement of 4 bytes. Finally, since the SMC is part of the control logic, the gadget chain keeps running with the control logic in every scan cycle. Injecting SMC into the control logic ensures the attacker's payload is pushed back to the stack every time the control logic is run.

VI. Implementation

We use Schneider Electric's Modicon M221 as a target PLC for the implementation and demonstration of our attacks. This PLC serves as a typical example of a traditional PLC. It has both digital and analog input and output ports and supports communication interfaces such as USB and Ethernet. It communicates with vendor-supplied engineering software, which is used to write a control logic program on it. Like most of the other PLCs, the communication between the engineering software and the PLC takes place in the form of request and response packets with the PLC acting as a server and the engineering software as a client. The communication protocol is, however, proprietary and needs to be reverse-engineered for successful communication other than the vendor-supplied engineering software. The PLC's memory can be acquired through the network protocol and then dumped for further analysis. M221 PLC uses RX Renesas architecture. The RX CPU has sixteen general purpose registers (R0 - R15) with R0 being the stack pointer, nine control registers (ISP, USP, etc), and one accumulator used for digital signal processor (DSP) instructions. The proposed techniques in Section V-C can be applied to most PLCs. For instance, all PLCs have input/output ports required for using the FUG technique to find useful gadgets. Additionally, since PLCs are embedded devices, a stack (and hence, gadgets) is fundamental to them [37]–[40]. Based on the hardware architecture, each PLC may have a certain designated register as the stack pointer. Similarly, all PLCs have a download capability and a communication protocol that can be used to append SMC to the control logic.

A. Steps Taken in Implementing ROP Attacks

Below, we present implementation steps that are derived from the above-stated techniques.

1) STEP I: PLC's memory acquisition: First, we acquire the entire memory of the PLC via the ICS communication protocol. M221 uses a proprietary protocol Unified Messaging Application Services (UMAS) embedded in Modbus so it needs to be reverse-engineered in order to initiate communication. We utilized the information from [18], [16] and [35] who have partially reverse-engineered the protocol to develop a virtual client that initiates a session with the PLC and sends requests to read the desired memory region.

A PLC's memory can be divided into protocol-mapped and non-protocol-mapped memory space. While the protocol-mapped memory space can be acquired via ICS protocol, the non-protocol-mapped memory cannot be. Modicon M221 PLC's memory map shows memory regions such as External RAM, Peripheral I/O, On-chip ROM, and FCU-RAM [41]–[44]. Among these memory areas, only 'External RAM' is a protocol-mapped memory space, and all others are non-protocol-mapped memory spaces. In order to access the non-protocol-mapped memory, we utilize PLC Memory Extractor (PEM) [30] that appends a duplicator code to the control logic in order to copy the non-protocol-mapped memory space to protocol-mapped memory space. This way we are able to acquire the entire memory through the ICS network protocol. Note that the memory can also be acquired via JTAG [45], [46] but that requires having physical access to the PLC which the attacker does not always have in the real world.

2) STEP II: Memory Dump Analysis: Next, we utilize a disassembly toolchain to disassemble the binary files of the memory dumps acquired in the previous step. In this step, we analyze the disassembled file and derive important information. For instance, we figured '0x02' translates to 'rts' instruction in M221 PLC.

In this step, we also find gadgets in the memory dump, FUG, and then generate a gadget chain that we want to run as part of our attack as explained below.

Finding all gadgets in the PLC's memory and generating a gadget database. All gadgets must be indexed to do FUG. To find all gadgets, we use the same approach followed by [25] except to find the opcode `'0x02'` instead of `'0xc3'` to detect the return instruction. We not only look for gadgets that exist as a result of the code-generation choices of the compiler but the ones that exist otherwise as well. For instance, the disassembly of the on-chip ROM region in M221 PLC shows the machine code `"e5 12 08 02"` which translates to **"mov.l 32[r1], 8[r2]"** as a result of the compiler code-generation choice. However, the last two bytes of this machine code `"08 02"` can also form the instruction sequence, **"bra.s 0x8 rts"** which we can use as a gadget.

Algorithm 1 shows how we find gadgets in the memory dump. First, we look for `'0x02'` byte in the binary file. Then at this byte, we step back and look for the maximum number of bytes that can make a valid instruction. There can be multiple

Algorithm 1 Finding valid ROP gadgets in memory

```
$maxIntSize \leftarrow$ 0x08
$gadgets \leftarrow$ []
for each offset of 0x02 do
    $failCount \leftarrow 0$
    $startAddr \leftarrow$ the offset -1
    $stopAddr \leftarrow$ the offset +1
    while $failCount \leq maxIntSize$ & $startAddr \geq 0$ do
        $res \leftarrow$ the result of the following command:
        rx-elf-objdump D b binary m rx filename
        –start-address =start_addr –stop-address =stop_addr
        $startAddr \leftarrow startAddr - 1$
        if res ends with 'rts' and does not contain the string "unknown" then
            $gadgets \leftarrow$ res
            $failCount \leftarrow 0$
        else:
            $failCount \leftarrow$ failCount +1
        end if
    end while
end for
```

possible instructions that can exist before a return such as `'Add'`, `'Sub'`, `'Jump'`, etc. We record all the possible instruction sequences and keep repeating the process of finding instruction sequences until no more instruction sequence is found. This way we get a combination of gadgets from just one return instruction.

FUG from the gadget database. Once we have all the gadgets available, we use the FUG technique to find gadgets that can help us accomplish our tasks. We specifically look for gadgets that are associated with the input/output register among others. In the case of M221, we discover that register R12 maps to the input port while R13 maps to the output ports of the PLC.

Generating a gadget chain. Finally, using the gadgets that we have, we generate a gadget chain for the attacks.

3) STEP III: Executing the ROP chain: In order to execute the ROP chain successfully, we use SMC injection and the Stack Modification technique mentioned in Section V-C.

SMC injection. We use information derived from existing work [16], [18] to determine the start and end address of control logic. Both the start address of the control logic and its size can be found in the configuration block of the PLC. The size along with the start address can be used to determine the end address of the control logic where SMC can be inserted. Note that SMC is inserted in the place of the last 'return' instruction of the control logic. Also, we show in our evaluation results, that SMC injection is not detectable by IDS.

Stack Modification. As mentioned before, a stack pointer increments after each value gets popped from the stack and then points to the value that needs to be popped next. If we just modify the current value in the stack, it can hinder the normal functioning of the PLC because the actual instruction that was supposed to run as a result of the pop operation may not run at all. So in order to populate the stack with our gadget addresses, we decrement the stack pointer's value and then update the address it points to in the stack to the location of the gadget that we want to be executed.

Figure 4(b) shows executing the ROP chain while ensuring PLC functionality. In this case, the number of gadgets is 1, so the value of the original stack pointer (SP), in this case, the R0 register, is decremented by one address value (from 278 to 274). In the case of running a set of gadgets, the decrease in SP may increase depending on the value of n.

```
line          assembly
  1:          sub #4, r0
  2:          mov.l #0x07020000, r2
  3:          mov.l r2, [r0]
  4:          rts
```

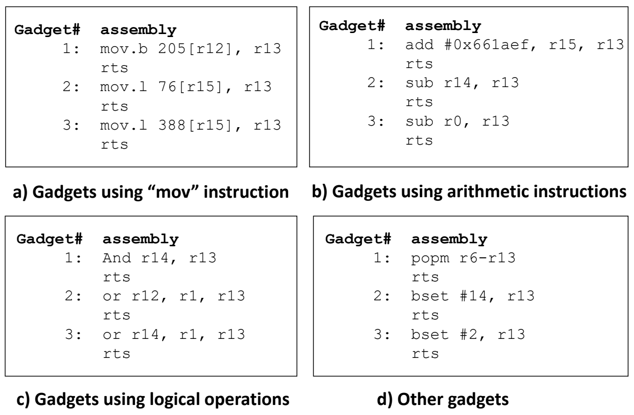
Fig. 7: Stack modification code (SMC)

To change the value of the stack pointer, we utilize the RX Renesas dataset to find the register that acts like a stack pointer. Figure 7 shows SMC. In order to make sure the PLC does not lose its basic functionality, before changing the address that the current stack pointer points to, we decrement the stack pointer (line 1) and then we update the address it points to (lines 2 - 3). This address corresponds to the location of our selected gadgets. Moving the pointer one step back prevents overwriting the contents of the stack that have yet to be popped and executed. Instead, we overwrite the return address that has already been returned after running. Altering the stack pointer to point to a new location containing the gadget addresses, instead of loading the addresses onto the existing stack, can adversely affect the normal functioning of the PLC. This is because it completely changes both the stack location and its contents, making it impossible to restore the original stack contents. Note that changing the stack pointer to a new location with the gadget addresses instead of loading them onto the existing stack can affect the normal functioning of the PLC since it changes the stack location and its contents entirely and there is no way of restoring the original stack contents.

B. Proposed ROP attack implementation

1) ROP 1: ROP 1 is further divided into two approaches. The first approach uses just one gadget present in memory to attack the control logic while the second approach consists of an ROP chain that has more than one gadget.

ROP Variant I. From our analyses and experiments, we figured that register R12 maps to the PLC's input ports while register R13 always maps to the PLC's output ports. We figured out this information after writing different control logic programs that deal with different output ports and then reading and disassembling the compiled binary from the PLC's memory, followed by a differential analysis of the disassembled control logic programs. So in this kind of attack, we extract the gadgets that are associated with register R13 from the gadget database and then select the relevant gadget that we want to execute. For instance, the following instruction; **`bset #2, r13`** sets bit number 2 of R13, which in turn turns on %Q0.2 output port. In order to execute our selected gadget, we inject a code in place of the last 'return' instruction of the control logic that basically updates the stack with our selected gadget address. Once the gadget address gets written inside the stack, the gadget gets executed, and the 'return' instruction takes the



```
Gadget#  assembly
     1:  mov.b 205[r12], r13
         rts
     2:  mov.l 76[r15], r13
         rts
     3:  mov.l 388[r15], r13
         rts
```

a) Gadgets using "mov" instruction

```
Gadget#  assembly
     1:  add #0x661aef, r15, r13
         rts
     2:  sub r14, r13
         rts
     3:  sub r0, r13
         rts
```

b) Gadgets using arithmetic instructions

```
Gadget#  assembly
     1:  And r14, r13
         rts
     2:  or r12, r1, r13
         rts
     3:  or r14, r1, r13
         rts
```

c) Gadgets using logical operations

```
Gadget#  assembly
     1:  popm r6-r13
         rts
     2:  bset #14, r13
         rts
     3:  bset #2, r13
         rts
```

d) Other gadgets

Fig. 8: Gadgets that modify R13 register

control back to the stack. Figure 8 shows some of the gadgets from the memory of a PLC that can be used for this attack. While these gadgets give us the ability to manipulate output values, we are still limited to only being able to use a few output ports. We also do not have much control over the value we want to keep in each of the output ports. So if an attacker wants to set or unset a chosen output port, she will have to utilize the second approach.

ROP Variant II. In our first approach, we enhance the probability of finding relevant gadgets capable of manipulating the output ports of a PLC. Additionally, we employ a second approach in which we carefully select gadgets that grant us a greater degree of control over the physical process. This is because our selection of gadgets helps us to manipulate any of the output bits of the PLC. This gives us more flexibility to set our choice of value into the selected output port. For instance, we can give output `n` the value "1" or "0", where `n` is any number from 1 to the number of output ports of a PLC. A typical example of a gadget chain for this attack is shown in Figure 9 for M221 PLC. All of the gadgets in this chain were found in the on-chip ROM region of this PLC. The first gadget that executes is **popm r14-r15** which pops `0xFFFFFFFF` and `0x00000000` into `r14` and `r15`, respectively. The second gadget, **or r14, r1, r13**, does a logical `OR` operation between r14, r1, and r13 and keeps the result in r13. So now r13 has the value `0xFFFFFFFF`. The next gadget, **popm r14-r15**, again pops `0x00001001` and `0x00000000` and places the value into `r14` and `r15`, respectively. Finally, the last gadget, **and r14,r13**, performs a logical `AND` operation between `r14` and `r13`. So now `r13` has the value `0x00001001`, which would turn on output bits 1 and 4.

2) *ROP 2:* For the second kind of attack, we construct a control logic from scratch using gadgets in a PLC's memory. Due to the limitations in the variety of gadgets available in memory, the complexity of control logic that can be developed is constrained. For a brief example, consider a control logic that turns on output %Q0.1 if input %I0.3 is on. Figure 10(a) shows the ladder diagram for this control logic while Figure 10(b) shows its disassembled version in RX architecture. The control logic has two instructions (excluding the 'return' instruction). From our analysis, we found these instructions

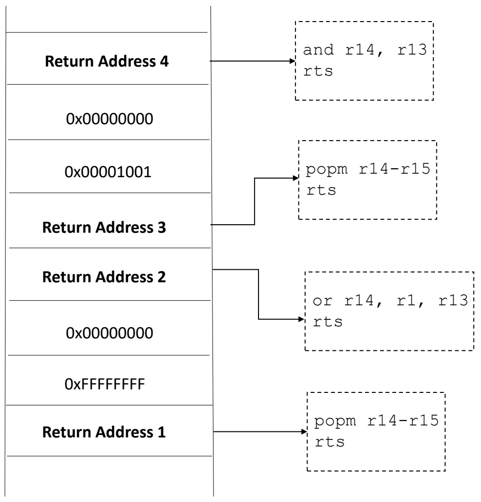
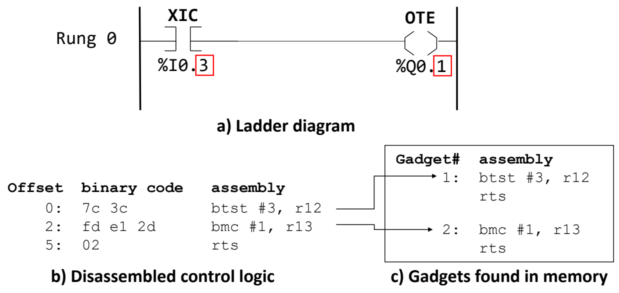


Fig. 9: Gadget chain that sets output bits 1 and 4



```
Offset  binary code   assembly
    0:  7c 3c         btst #3, r12
    2:  fd e1 2d      bmc #1, r13
    5:  02            rts
```

b) Disassembled control logic

```
Gadget#  assembly
     1:  btst #3, r12
         rts
     2:  bmc #1, r13
         rts
```

c) Gadgets found in memory

Fig. 10: Control logic that turns on output 1 if input 3 is on

in the memory as individual gadgets (see Figure 10(c). This construction is, however, limited and may not allow an attacker to construct any kind of control logic.

VII. ROP Case studies on physical processes

We evaluate our attacks on three different physical processes i.e., elevator, belt conveyor system, and traffic light system.

Experimental Settings. The scripts (to acquire memory, to find the control logic's start and end address, and to find gadgets from the acquired memory) are written in Python. SMC is written in RX architecture assembly language which we convert to machine code using GCC for Renesas 8.3.0.202202-GNURX Linux Toolchain [47]. The firmware version of Modicon M221 is v1.6.0.1. We use Windows 10 virtual machine (VM) to run EcoStruxure Machine Expert - Basic version 1.2 and Ubuntu 16.04 (VM) to run our scripts. The PLC and both our VMs are connected via Ethernet to our internal network.

A. Case Study 1: Elevator

Testbed Setup. Elevators transport people and/or freight from one floor or level to another. This saves time and energy and especially benefits people with moving disabilities. Figure 11 shows a four-floor elevator model used for our experiments. The model itself has four floors. A user can select a floor from both inside and outside the elevator as input to the PLC. In response, the elevator moves to the desired floor while the LED lights show the floor it is on. Each input/output port of the PLC is connected to some elevator component. In the

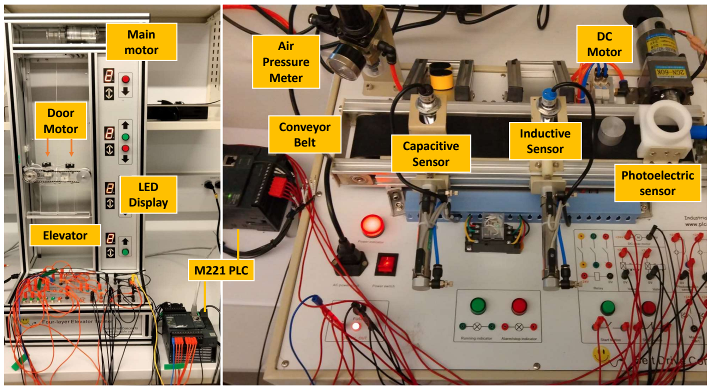


Fig. 11: (L) Front view of the elevator model
(R) Top view of the conveyor belt model

case of our lab model, %Q0.0, %Q0.4, %Q0.5 are connected to one of the LED lights, %Q0.1 drags the elevator downward, %Q0.2 drags it upward, %Q0.3 is connected to the motor that opens the door, while %Q0.6 is connected to door closing.

Experiments. We test the first variant of ROP 1 on the elevator with four floors. We use the gadget `bset #2, r13` to set output bit 2. We append the code to modify the stack pointer value to the end of the written control logic. While the control logic is being executed, the control goes to the location of our gadget, which turns on %Q0.2. As mentioned, this output port is connected to the motor that drags the elevator upward. With our attack, since %Q0.2 is always set, the elevator keeps going upward. Once it reaches the limit (last floor), the motor keeps working, causing the elevator to exceed its intended stopping point, which could result in the elevator falling due to gravity and the force caused by the rotating motor. Note that this can also lead to loss of lives if implemented in the real world.

Evaluation Results. To evaluate how ROP attacks affect a PLC's runtime, we measure the PLC's scan time. We also measure the program size, number, and size of gadgets as shown in Table I.

a) ROP Attack Detection: We use Shade [19] to evaluate if SMC can be detected in network traffic. Shade is a stateful open-source IDS, specifically designed to detect control logic in network packets.

The SMC contains 0 rungs and at the same time has a very short length(11 bytes) with a high entropy (2.85) compared to the attack packets. Since attack packets are often correlated in proportion to the number of rungs and entropy, the correlation of SMC is the opposite of the attack group. Hence, it is difficult to detect as long as the IDS learns features that are frequently used when classifying codes such as rung and entropy. As a result, the extracted features of SMC looked naive compared to other control logic that causes malicious behavior.

b) Scan time and program size: In a clean state, the PLC's scan time controlling the elevator process is 120-122 μs. When we append the stack modification code (SMC), the scan time is not significantly affected. This is because the size of SMC is only 11 bytes which, when added to the 659 bytes of the original control logic code, results in a total of 670 bytes, and this increase does not have a substantial impact on the scan time.

TABLE I: Performance evaluation of clean and malicious state (using the first variant of ROP 1) on elevator

	Scan time (μs)	Program size	Number of gadgets	Size of gadgets
Clean state	120 - 122	659 bytes	-	-
Malicious state	121 - 122	670 (include 11 bytes of SMC)	1	3 bytes

TABLE II: Performance evaluation of the clean and malicious state of conveyor belt

	Scan time (μs)	Program size	Number of gadgets	Size of gadgets
Clean state	125 - 127	38 bytes	-	-
Malicious state	126 - 127	49 (include 11 bytes of SMC)	4	13 bytes

c) Number and size of gadgets: For ROP 1, we use just one gadget that is 3 bytes in size.

B. Case Study 2: Belt Conveyor System

Testbed Setup. Conveyor Belts are used in industrial settings to move and sort supplies in order to save time and energy. Figure 11 shows the top view of the model used for our experiments. It has different sensors such as Inductive, Capacitive, and Photoelectric which are used to sort different objects, and has a DC Motor that provides power. This conveyor belt is connected to M221 PLC via input and output ports. For instance, %Q0.0 is connected to the DC gear voltage motor that runs the belt, %Q0.1 is connected to valve 2 (the inductive sensor which sorts metal objects), %Q0.2 is connected to valve 1 (the capacitive sensor which sorts plastic objects).

Experiments. We test the second variant of ROP 1 on the conveyor belt the result of which is that valve 1 stays ahead while the belt is running. Our gadget chain is similar to the one shown in Figure 9 except that the value that we keep in register R14 is `0x00000101` (note the value in the chain that comes right after "Return address 3"). This attack disrupts the conveyor belt in a way that metal objects are not allowed to pass through since valve 1 is pushed forward.

Evaluation Results. We use an IDS to detect the ROP attacks and further evaluate performance in terms of scan time and gadget size as shown in Table II.

a) ROP Attack Detection: Similar to case 1, we use Shade [19] to evaluate the detection of SMC code when it is transferred. Since the only change in SMC is the address of the gadget, it is not classified as an attack either.

b) Scan time and program size: In a clean state, the PLC's scan time in controlling the elevator process is 125-127 μs. When we append the stack modification code (SMC), the scan time is not affected much.

c) Number and size of gadgets: We test the conveyor belt using a gadget chain that comprises eight addresses in the stack with four of them being gadgets and the rest being values that we pop from the stack. Three gadgets are 3 bytes and one is 4 bytes, for a total of 13 bytes.

C. Case Study 3: Compact Traffic Light System

Testbed Setup. Our traffic light system consists of red, green, orange, and blue lights and an emergency toggle switch. When the toggle switch is on, it disables the system, and the blue light starts blinking.

Experiments. For testing ROP2 with the gadgets in Figure 10, we construct and run a control logic that activates %Q0.1 when %I0.1 is triggered.

Evaluation Results. We use Shade [19] to detect the ROP attacks and further evaluate performance in gadget size. Since ROP 2 does not use the original control logic, the evaluation of the impact of scan time and program size on the original control logic is irrelevant and omitted.

a) ROP Attack Detection: Similar to cases 1 and 2, Shade [19] could not detect SMC.

b) Number and size of gadgets: We use two gadgets to execute the control logic, with both gadgets being 3 bytes long.

VIII. Related Work

While the existing literature presents a wide range of ICS attacks [13], [14], [17], [20], [35], [36], [48]–[61], this section first covers control logic attacks on PLCs and then briefly discusses ROP attacks. However, none of them combine the two to show control logic attacks on PLCs via ROP.

Control Logic Attacks on PLCs. Senthival *et al.* [29] present denial of engineering operations attacks in which the attacker downloads an infected ladder logic to a PLC (either remotely or manipulation of legitimate network traffic) which when uploaded crashes the engineering software.

Kalle *et al.* [18] present a CLIK attack that steals the control logic from a PLC after compromising its security measures. Then, it decompiles the stolen binary of the control logic to inject malicious logic, followed by transferring the infected binary back to the PLC and hiding the infection from the engineering software by employing a virtual PLC.

Similar to CLIK, McLaughlin *et al.* presented SABOT [15] a tool that uploads a targeted PLC's control logic byte code and decompiles it to find a mapping between the devices connected to the PLC and variables within the control logic. This mapping can then be changed arbitrarily and the control logic downloaded to the PLC to cause damage to the plant.

Yoo *et al.* [16] present two control logic injection attacks such as 1) data execution and 2) fragmentation and noise padding. In the data execution attack, the attacker transfers a malicious control logic to the data blocks of the PLC and changes the control flow to execute the logic located in the data blocks. Fragmentation and noise padding attacks add a huge amount of noise to the write request packet containing the control logic in order to subvert deep packet inspection.

ROP Attacks on IT applications. Existing literature shows a lot of effort into exploring ROP attacks on IT applications [25], [26], [62], [63]. Here we discuss Weidler *et al's.* [27] work since it is closest to ours, i.e., ROP on microcontrollers. Their work presents a gadget set that is capable of erasing flash memory and then re-programming this region and a Turing complete gadget set that was capable of performing arbitrary computation. Our work is different in a way that instead of changing the firmware or the normal functionality of a PLC, we target the control logic and make it malicious in order to sabotage the connected physical process. Moreover, ROP on PLCs runs in a continuous scan cycle instead of being a one-time attack (unlike traditional ROP).

IX. ROP Attack Mitigation in PLC

The potential mitigation against the ROP attacks on PLCs includes the following:

Control Logic Integrity Checking. PLCs allow remote access to control logic for maintenance through ICS protocols. ICS malware typically exploits this legitimate functionality to inject malicious control logic. A device-level solution should monitor the control flow integrity of a PLC's control logic and raise an alert in case of a discrepancy. However, the solution must allow control engineers to update the logic remotely.

Message Authentication in ICS Protocols. ICS protocols require message authentication to allow PLCs to ignore control logic messages from an unauthenticated source. For instance, the DNP3 protocol uses a message authentication code for message integrity and authentication [64]. However, most ICS protocols do not have message authentication.

Address Space Layout Randomization of PLC Firmware and Control Logic. Address space layout randomization (ASLR) alters the location of the objects in the memory (e.g., executable programs and process stack) whenever a system reboots [65], [66]. It adds complexity to an ROP attack by restricting the attacker from determining the exact locations of the gadgets ahead of time using a test environment before the attack. PLCs should employ ASLR to randomize the memory layout of firmware, control logic, and other data structures, raising the bar for the ROP attack requiring finding gadget locations while launching the attack.

X. Conclusion

We studied ROP attacks on PLCs' control logic programs. We discovered several challenges to considering ROP as a practical attack vector for ICS environments, including overwriting the stack in a PLC memory with a gadget chain, executing the ROP chain, and maintaining the normal PLC operations during the attack. To overcome these challenges and make ROP attacks applicable to most PLCs, we identified and utilized typical PLC design features that are exploitable, e.g., no stack protection, predictable control logic location, and remote access to certain PLC memory regions via ICS protocols. The attack evaluations on three physical processes showed successful ROP attacks without affecting a PLC's scan time. The attacks were also not detected by Shade, an open-source IDS for control logic detection in network traffic.

Acknowledgment

This material is based upon work supported by the U.S. Department of Homeland Security under Grant Award Number 17STCIN00001-05-00.

Disclaimer. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

References

[1] I. Ahmed, S. Obermeier, M. Naedele, and G. G. R. III, "Scada systems: Challenges for forensic investigators," *Computer*, vol. 45, pp. 44–51, 2012.

[2] I. Ahmed, S. Obermeier, S. Sudhakaran, and V. Roussev, "Programmable logic controller forensics," *IEEE Security Privacy*, vol. 15, no. 6, pp. 18–24, November 2017.

[3] "CrashOverride Malware," https://www.cisa.gov/uscert/ncas/alerts/TA17-163A, 2022, [Online; accessed 29-June-2022].

[4] "Havex Malware," https://www.cisa.gov/uscert/ics/advisories/ICSA-14-178-01, 2022, [Online; accessed 29-June-2022].

[5] "HatMan Malware," https://www.cisa.gov/uscert/ics/MAR-17-352-01-HatMan-Safety-System-Targeted-Malware-Update-B, 2022, [Online; accessed 29-June-2022].

[6] J. Slowik, "Evolution of ics attacks and the prospects for future disruptive events," https://www.dragos.com/wp-content/uploads/Evolution-of-ICS-Attacks-and-the-Prospects-for-Future-Disruptive-Events-Joseph-Slowik-1.pdf, Dragos, Tech. Rep., 2022.

[7] Cybersecurity and I. S. A. (CISA), "Understanding and mitigating russian state-sponsored cyber threats to u.s. critical infrastructure," https://www.cisa.gov/uscert/ncas/alerts/aa22-011a, 2022, [Online; accessed 29-June-2022].

[8] M. H. Rais, M. Ahsan, and I. Ahmed, "Fromepp: Digital forensic readiness framework for material extrusion based 3d printing process," *Forensic Science International: Digital Investigation*, vol. 44, p. 301510, 2023.

[9] M. H. Rais, Y. Li, and I. Ahmed, "Dynamic-thermal and localized filament-kinetic attacks on fused filament fabrication based 3d printing process," *Additive Manufacturing*, vol. 46, p. 102200, 2021.

[10] N. Falliere, L. O. Murchu, and E. Chien, "W32.stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011.

[11] I. Ahmed, V. Roussev, W. Johnson, S. Senthivel, and S. Sudhakaran, "A SCADA System Testbed for Cybersecurity and Forensic Research and Pedagogy," in *Proceedings of the 2nd Annual Industrial Control System Security Workshop (ICSS)*, 2016.

[12] S. Senthivel, I. Ahmed, and V. Roussev, "Scada network forensics of the pccc protocol," *Digital Investigation*, vol. 22, pp. S57 – S65, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287617301998

[13] L. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. A. Mohammed, and S. A. Zonouz, "Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit." in *NDSS*, 2017.

[14] N. Govil, A. Agrawal, and N. O. Tippenhauer, "On ladder logic bombs in industrial control systems," in *Computer Security*. Springer, 2017, pp. 110–126.

[15] S. McLaughlin and P. McDaniel, "Sabot: specification-based payload generation for programmable logic controllers," *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[16] H. Yoo and I. Ahmed, "Control Logic Injection Attacks on Industrial Control Systems," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2019, pp. 33–48.

[17] A. Abbasi and M. Hashemi, "Ghost in the plc designing an undetectable programmable logic controller rootkit via pin control attack," *Black Hat Europe*, vol. 2016, pp. 1–35, 2016.

[18] S. Kalle, N. Ameen, H. Yoo, and I. Ahmed, "CLIK on PLCs! Attacking control logic with decompilation and virtual PLC," in *Binary Analysis Research (BAR) Workshop, Network and Distributed System Security Symposium (NDSS)*, 2019.

[19] H. Yoo, S. Kalle, J. Smith, and I. Ahmed, "Overshadow Plc to Detect Remote Control-Logic Injection Attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2019, pp. 109–132.

[20] S. McLaughlin and S. Zonouz, "Controller-aware false data injection against programmable logic controllers," in *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2014, pp. 848–853.

[21] M. Rais, M. Ahsan, V. Sharma, R. Barua, R. Prins, and I. Ahmed, "Low-magnitude infill structure manipulation attacks on fff-based 3d printersâ," in *Proceedings of the â16th IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection (March 2022). https://scholar. google. com/citations*, 2022.

[22] M. H. Rais, Y. Li, and I. Ahmed, "Spatiotemporal g-code modeling for secure fdm-based 3d printing," in *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, 2021, pp. 177–186.

[23] M. E. Locasto, K. Wang, A. D. Keromytis, and S. J. Stolfo, "Flips: Hybrid adaptive intrusion prevention," in *Recent Advances in Intrusion Detection*, A. Valdes and D. Zamboni, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 82–101.

[24] A. Shenfield, D. Day, and A. Ayesh, "Intelligent intrusion detection systems using artificial neural networks," *Ict Express*, vol. 4, no. 2, pp. 95–99, 2018.

[25] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 552–561.

[26] N. Carlini and D. Wagner, "{ROP} is still dangerous: Breaking modern defenses," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 385–399.

[27] N. R. Weidler, D. Brown, S. A. Mitchell, J. Anderson, J. R. Williams, A. Costley, C. Kunz, C. Wilkinson, R. Wehbe, and R. Gerdes, "Return-oriented programming on a resource constrained device," *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 244–256, 2019.

[28] B. Johnson, D. Caban, M. Krotofil, D. Scali, N. Brubaker, and C. Glyer, "Attackers deploy new ics attack framework triton and cause operational disruption to critical infrastructure," https://www.mandiant.com/resources/blog/attackers-deploy-new-ics-attack-framework-triton, 2021, [Online; accessed 26-Dec-2022].

[29] S. Senthivel, S. Dhungana, H. Yoo, I. Ahmed, and V. Roussev, "Denial of engineering operations attacks in industrial control systems," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '18. New York, NY, USA: ACM, 2018, pp. 319–329.

[30] N. Zubair, A. Ayub, H. Yoo, and I. Ahmed, "Pem: Remote forensic acquisition of plc memory in industrial control systems," *Forensic Science International: Digital Investigation*, vol. 40, p. 301336, 2022.

[31] S. Qasim, A. Ayub, J. Johnson, and I. Ahmed, "Attacking the IEC-61131 Logic Engine in Programmable Logic Controllers in Industrial Control Systems," in *Critical Infrastructure Protection XV*, J. Staggs and S. Shenoi, Eds. Cham: Springer International Publishing, 2021.

[32] I. A. Khan, D. Pi, Z. U. Khan, Y. Hussain, and A. Nawaz, "Hml-ids: A hybrid-multilevel anomaly prediction approach for intrusion detection in scada systems," *IEEE Access*, vol. 7, pp. 89 507–89 521, 2019.

[33] F. S. Toker, K. Ovaz Akpinar, and . ZELK, "Mitre ics attack simulation and detection on ethercat based drinking water system," in *2021 9th International Symposium on Digital Forensics and Security (ISDFS)*, 2021, pp. 1–6.

[34] "Blockbuster Worm Aimed for Infrastructure, But No Proof Iran Nukes Were Target," https://www.wired.com/2010/09/stuxnet-2/, 2010, [Online; accessed 16-Jan-2023].

[35] A. Ayub, H. Yoo, and I. Ahmed, "Empirical study of plc authentication protocols in industrial control systems," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 383–397.

[36] H. Wardak, S. Zhioua, and A. Almulhem, "Plc access control: a security analysis," in *Industrial Control Systems Security (WCICSS), 2016 World Congress on IEEE, pp. 16*, 2016.

[37] B. Middha, M. Simpson, and R. Barua, "Mtss: Multitask stack sharing for embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 4, pp. 1–37, 2008.

[38] "The Concept of Stack and Its Usage in Microprocessors," https://open4tech.com/the-concept-of-stack-and-its-usage-in-microprocessors/, 2023, [Online; accessed 12-January-2023].

[39] J. Zhou, Y. Du, Z. Shen, L. Ma, J. Criswell, and R. J. Walls, "Silhouette: Efficient protected shadow stacks for embedded systems," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1219–1236.

[40] T. Santini, L. Carro, F. R. Wagner, and P. Rech, "Reliability analysis of operating systems and software stack for embedded systems," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2225–2232, 2016.

[41] "RX Renesas Manual," https://www.manualslib.com/manual/1797019/Renesas-Rx-Series.html, 2019, [Online; accessed 09-May-2021].

[42] "M221 Hardware guide," https://download.schneider-electric.com/files?p_enDocType=User+guide&p_File_Name=EIO0000001384.06.pdf&p_Doc_Ref=EIO0000001384, 2017.

[43] "M221 programming guide," https://download.schneider-electric.com/files?p_Doc_Ref=EIO0000001360, 2017, [Online; accessed 10-Oct-2020].

[44] R. A. Awad, M. H. Rais, M. Rogers, I. Ahmed, and V. Paquit, "Towards generic memory forensic framework for programmable logic controllers," *Forensic Science International: Digital Investigation*, vol. 44, p. 301513, 2023, selected papers of the Tenth Annual DFRWS EU Conference. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281723000148

[45] M. H. Rais, R. A. Awad, J. Lopez Jr, and I. Ahmed, "Jtag-based plc memory acquisition framework for industrial control systems," *Forensic Science International: Digital Investigation*, vol. 37, p. 301196, 2021.

[46] ——, "Memory forensic analysis of a programmable logic controller in industrial control systems," *Forensic Science International: Digital Investigation*, vol. 40, p. 301339, 2022.

[47] "Renesas GNU Tools," https://gcc-renesas.com/, 2021, [Online; accessed 09-May-2021].

[48] M. Krotofil, K. Kursawe, and D. Gollmann, *Securing Industrial Control Systems*. Cham: Springer International Publishing, 2019, pp. 3–27.

[49] A. Erba, R. Taormina, S. Galelli, M. Pogliani, M. Carminati, S. Zanero, and N. O. Tippenhauer, "Constrained Concealment Attacks against Reconstruction-Based Anomaly Detectors in Industrial Control Systems," in *Annual Computer Security Applications Conference*, ser. ACSAC '20, New York, NY, USA, 2020, p. 480495.

[50] E. López-Morales, C. Rubio-Medrano, A. Doupé, Y. Shoshitaishvili, R. Wang, T. Bao, and G.-J. Ahn, "HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20, New York, NY, USA, 2020, p. 279291.

[51] S. E. McLaughlin, "On Dynamic Malware Payloads Aimed at Programmable Logic Controllers." in *HotSec*, 2011.

[52] B. Lim, D. Chen, Y. An, Z. Kalbarczyk, and R. Iyer, "Attack induced common-mode failures on PLC-based safety system in a nuclear power plant: practical experience report," in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2017, pp. 205–210.

[53] D. Beresford, "Exploiting siemens simatic s7 plcs," *Black Hat USA*, vol. 16, no. 2, pp. 723–733, 2011.

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

[55] R. Spenneberg, M. Brüggemann, and H. Schwartke, "Plc-blaster: A worm living solely in the plc," *Black Hat Asia*, vol. 16, pp. 1–16, 2016.

[56] G. P. H. Sandaruwan, P. S. Ranaweera, and V. A. Oleshchuk, "PLC security and critical infrastructure protection," in *Proc. 2013 IEEE 8th Int. Conf. on Industrial and Information Systems, pp. 81-85*, 2013.

[57] R. Grandgenett, W. Mahoney, and R. Gandhi, "Authentication bypass and remote escalated I/O command attacks," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, 2015, pp. 1–7.

[58] S. A. Qasim, J. Lopez, and I. Ahmed, "Automated Reconstruction of Control Logic for Programmable Logic Controller Forensics," in *Information Security*, Z. Lin, C. Papamanthou, and M. Polychronakis, Eds. Cham: Springer International Publishing, 2019, pp. 402–422.

[59] M. H. Rais, Y. Li, and I. Ahmed, "Spatiotemporal G-code Modeling for Secure FDM-based 3D Printing," in *Proceedings of the ACM/IEEE twelfth International Conference on Cyber-Physical Systems*, ser. ICCPS '21. New York, NY, USA: Association for Computing Machinery, 2021.

[60] S. A. Qasim, J. M. Smith, and I. Ahmed, "Control logic forensics framework using built-in decompiler of engineering software in industrial control systems," *Forensic Science International: Digital Investigation*, vol. 33, p. 301013, 2020.

[61] N. Zubair, A. Ayub, H. Yoo, and I. Ahmed, "Control logic obfuscation attack in industrial control systems," in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2022, pp. 227–232.

[62] G.-A. Jaloyan, K. Markantonakis, R. N. Akram, D. Robin, K. Mayes, and D. Naccache, "Return-oriented programming on risc-v," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 471–480.

[63] A. Homescu, M. Stewart, P. Larsen, S. Brunthaler, and M. Franz, "Microgadgets: Size does matter in turing-complete return-oriented programming." *WOOT*, vol. 12, pp. 64–76, 2012.

[64] "Overview of DNP3 Security Version 6," https://www.dnp.org/LinkClick.aspx?fileticket=hyvYMYugaQI%3D&tabid=66&portalid=0&mid=447&forcedownload=true, 2022, [Online; accessed 29-June-2022].

[65] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in *Proceedings of the 11th ACM conference on Computer and communications security*, 2004, pp. 298–307.

[66] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi, "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 574–588.