Charlie, Charlie, Charlie on Industrial Control Systems: PLC Control Logic Attacks by Design, Not by Chance

Adeen Ayub
Virginia Commonwealth University
Richmond, VA
ayuba2@vcu.edu

Wooyeon Jo
Virginia Commonwealth University
Richmond, VA
jow@vcu.edu

Irfan Ahmed
Virginia Commonwealth University
Richmond, VA
iahmed3@vcu.edu

***Abstract*—Programmable logic controllers (PLCs) in industrial control systems (ICS) run a control logic program to monitor and control critical infrastructures in real-time, such as nuclear plants and power grids. Attackers target PLC control logic remotely to sabotage or disrupt physical processes. Network intrusion detection systems (IDS) are increasingly used to detect malicious control logic. This paper demonstrates that standard IDS features in a protocol message header and payload are not resilient for detecting (control logic) binary programs, such as entropy, n-gram, and decompilation. It identifies and utilizes a PLC design feature, redundant address pins (RAP), unexplored in the literature, to bypass IDS for injecting a small piece of programmable malicious code (PMC) into a PLC's control logic as an initial attack vector, allowing it to execute with every scan cycle. We propose three unique attack methods (GizmoSplit, BuffWarp, and EnigmaFlow) using PMC as a proof of concept that blends control logic with network traffic via payload encoding, small-size payloads, or sparse memory addressing. The GizmoSplit attack divides the control logic into small gadgets and writes them in random memory locations in a PLC; PMC modifies the stack with the location of the gadgets to execute them as return-oriented programming. The BuffWarp attack employs a small-size buffer where the attacker writes malicious code periodically to bypass stateful inspection at the payload level; PMC, in turn, keeps moving the buffer content to consecutive memory locations to execute. The EnigmaFlow attack encodes control logic and sends it to a PLC's typically unused memory region, which PMC decodes and executes. The evaluation results indicate that these attacks are stealthy and can subvert IDS utilizing standard message header and payload features. This work points to a research gap in intrusion detection that caters to control logic attacks exploiting PLC design features.**

***Index Terms*—industrial control systems, programmable logic controllers, ICS attacks, control logic, intrusion detection systems**

I. Introduction

Industrial control systems (ICS) control and monitor physical processes in critical infrastructures such as nuclear plants, power grids, and oil and gas pipelines. They consist of control centers and field sites (shown in Figure 1). The control center runs ICS services such as historian, human-machine interface (HMI), and engineering workstation. Physical processes are located at field sites that include programmable logic controllers (PLC), sensors, and actuators. PLCs are directly connected to physical processes and equipped with control logic programs that define how the physical processes are controlled and monitored. Attackers target PLCs' control logic to disrupt physical processes [1]–[5]. Intrusion detection systems (IDS) are increasingly used to detect any transfer of malicious control logic over the network [6]–[11]. They use standard features in message headers and payloads for control logic detection, such as entropy, n-gram, and decompilation.

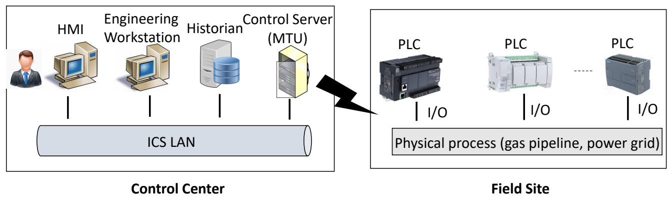


Fig. 1: Overview of an industrial control system

In this paper, we demonstrate that IDS based on these features are not resilient to detect control logic in ICS network traffic and are subject to subversion by exploiting PLC design features. Specifically, we identify and utilize *Redundant Address Pins (RAP)* in PLCs as an exploitable feature unexplored in the literature. Many PLCs use RAP to optimize the number of address pins to manage their addressable memory locations, such as Schneider Electric's M221 and M241 and Allen-Bradley's 1756-L61 ControlLogix. We empirically find that RAP leads to multiple address locations appearing to have the same content because they point to the same underlying physical memory addresses. Thus, attackers can use RAP to achieve address randomization in ICS protocols, which is useful to subvert stateful deep packet inspection (DPI) [12].

As an initial attack vector, we leverage RAP to remotely insert a small programmable malicious code (PMC) into a PLC's control logic, allowing it to execute with every scan cycle. It involves splitting the code into small (undetectable) parts and putting them into separate PLC memory spaces that seem unconnected. However, the result is a complete program neatly written in a single continuous address space.

PMC allows attackers to create stealthy control logic transfer methods that bypass IDS monitoring. We propose three unique attack methods, i.e., GizmoSplit, BuffWarp, and EnigmaFlow, as proof of concept that blends control logic with network traffic via payload encoding, small-size payloads, or sparse memory addressing.

The GizmoSplit (segmented control logic injection) attack segments a malicious control logic program into gadgets (instruction sequences ending with 'return') and places these gadgets randomly in memory. It then uses PMC to modify the stack with the addresses of these gadgets to run the control logic program in the correct sequence as return-oriented programming. Finally, the program's control flow is changed to run PMC instead of the original control logic.

The BuffWarp (buffered control logic injection) attack divides control logic into segments and repeatedly overwrites them in a designated buffer in the PLC. With each update, PMC copies the buffer contents to free memory space, incrementing the address by the buffer size. Once the entire control logic is written, the program's control flow shifts to the new region containing consecutive malicious control logic.

The EnigmaFlow (mutated control logic injection) attack mutates/encodes malicious control logic before writing it to free memory block in a PLC. PMC subsequently decodes and replaces it chunk by chunk. Finally, the attack redirects the program's control flow to the new malicious control logic instead of the original version.

We thoroughly evaluate the proposed attack methods to determine their efficacy in bypassing IDS. Our results show that GizmoSplit and BuffWarp can successfully bypass most of the detection features, such as n-gram, instructions, decompilation, and protocol header, while EnigmaFlow can evade detection features, such as decompilation and n-gram.

This paper makes the following contributions:

- We identify RAP as an exploitable feature in a typical PLC design. It can cause address randomization in ICS protocols to subvert stateful deep packet inspection while inserting small malicious code in a PLC's single continuous address space. The current ICS literature does not discuss RAP from a cybersecurity standpoint.
- We show that RAP with ICS protocols can append PMC in a PLC's control logic without detection. We further leverage PMC and propose three stealthy control logic injection attacks as a proof of concept.
- We successfully demonstrate the proposed attacks' impact on a laboratory-scale fully functional elevator.
- We evaluate the effectiveness of RAP and PMC-based attacks against standard intrusion detection features. We show that the attacks are stealthy and challenging for existing IDS, pointing out a research gap in intrusion detection for ICS.

The rest of this paper is organized as follows. Section II discusses existing control logic attacks. Section III gives the motivation behind our work. Section IV outlines the attack model and stages of our attacks followed by three proof-of-concept attacks in Section V. Section VI discusses the testbed and the attack implementation steps. Section VII and Section VIII present evaluation results and attack mitigation strategies, followed by the conclusion in Section IX.

II. Related work

This section discusses existing control logic injection attacks; none leverages RAP.

Yoo *et al.* [12] demonstrate two effective control logic injection techniques: data execution and fragmentation with noise padding. Data execution exploits PLCs' lack of data execution prevention (DEP), executing malicious logic from data blocks. Fragmentation with noise padding sends logic in small chunks with added noise to bypass Deep Packet Inspection (DPI).

Govil *et al.* [13] introduced "ladder logic bombs," which attackers can implant into a PLC's existing control logic. These bombs are challenging to manually detect by control engineers verifying PLC control logic. They can be activated by trigger signals to disrupt operations or persistently harm physical processes over time.

Senthivel *et al.* [14] introduced three Denial of Engineering Operations (DEO) attack scenarios, where an attacker disrupts the downloading/uploading of PLC control logic. In DEO I, the attacker, as a man-in-the-middle, injects malicious logic during uploading, then reverts it to normal to fool engineering software. DEO II employs a similar strategy but uploads flawed logic to crash the software. DEO III doesn't need a man-in-the-middle; the attacker injects intricate, flawed logic that runs on the PLC but can't be decompiled by the software.

Zubair *et al.* [15] propose a novel attack, Denial of Decompilation (DoDe), which manipulates engineering software's decompilation process. The attack involves crafting malicious control logic mirroring legitimate software's choices to hinder digital forensics and incident response.

Alsabbagh *et al.* [16] introduce a control logic injection attack, which empowers external adversaries to inject their malicious code into vulnerable PLCs. This enables them to maintain their attack dormant within the compromised device and subsequently activate it at a later time, even without being connected to the target on the day of the attack.

Ayub *et al.* [17] show return-oriented programming attacks on PLCs, leveraging gadgets from the PLC memory to manipulate existing control logic and create new malicious control logic. Despite not transmitting malicious code to the PLC, these attacks face limitations due to the scarcity of gadgets in memory for constructing meaningful malicious control logic.

III. Motivation and Problem statement

Control logic attacks on PLCs replace legitimate control logic with malicious ones to disrupt the underlying physical process. Since they transfer control logic over the network, IDS can detect them. State-of-the-art stealthy attacks subvert deep packet inspection (DPI) of individual packets. For instance, fragmentation and noise padding attack [12] transfers small fragments (one byte) of control logic in each packet and adds noise data in the packet payload, making it challenging for DPI to detect. However, stateful inspection can still detect these attacks using standard IDS features such as n-gram [6].

Problem statement. Given an IDS that can detect a control logic from the network traffic, an attacker's goal is to craft a control logic injection method that can subvert individual and stateful DPI using standard IDS detection features.

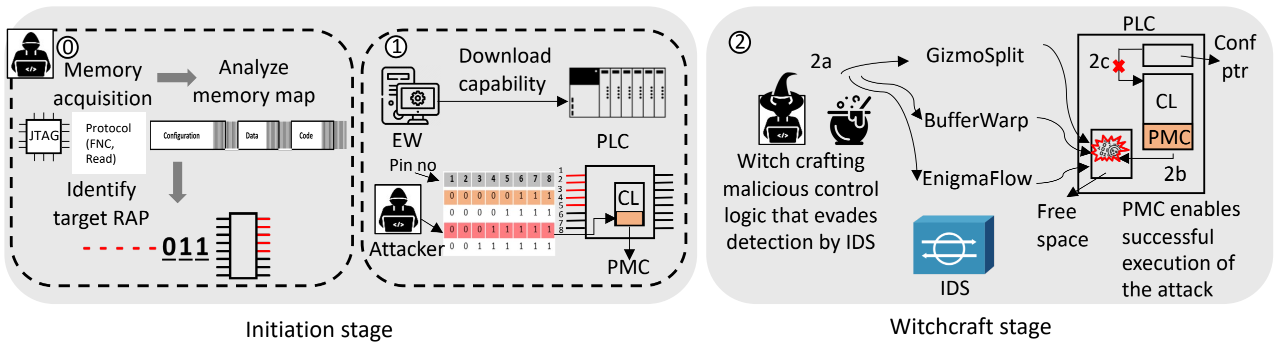


Fig. 2: Stealthy control logic injection attacks using redundant address pins (RAP) and programmable malicious code (PMC)

IV. Stealthy Control Logic Injection Attacks

Figure 2 provides an overview of our proposed stealthy control logic attack consisting of two main stages: 1) *Initiation stage* utilizes RAP (a common PLC design feature to optimize ICS protocol addressing) to install PMC in a target PLC's control logic secretly; 2) the *witchcraft stage* involves crafting and sending a new malicious control logic to the PLC that can evade IDS features for deep packet inspection. We propose three control logic crafting methods using PMC for this stage: GizmoSplit, BuffWarp, and EnigmaFlow (refer to Section V).

A. Attack Model

Assumptions. Our adversary model assumes that the attacker has access to the control center of an ICS. In practice, this access is achieved through typical attack methods in our IT world, such as introducing a malicious USB stick and social engineering as demonstrated by real-world ICS attacks like TRITON [5] and the Ukraine Power Grid attack [18]. This level of access enables the attacker to establish communication with the targeted PLC over the network using an ICS protocol.

We assume the attacker acquires the PLC device of the target make and model before launching the attack on a real ICS environment. It allows for developing and testing the exploit, including reverse-engineering relevant ICS protocols if their specifications are not publically available. They may employ tools like PREE for automated PLC protocol reverse engineering [19]. Furthermore, the attacker understands the memory map of the target PLC to identify code and data sections. This map can originate from datasheets/manuals or be generated using JTAG-based PLC memory acquisition [7] if needed details are lacking.

We also assume the adversary will not compromise the IDS monitoring the ICS network traffic between field devices and the control center.

Attacker's goal. The primary objective of the adversary is to covertly attack ICS physical processes, prioritizing stealth to evade detection by the network-based IDS.

B. Initiation Stage

Block 0 shows everything that an attacker does at her own end and not in the actual target environment. She can acquire the memory via protocol or JTAG and generate a memory map of the PLC to understand the different memory regions of the PLC. She then also identifies the different address pins that can be used to acquire the same memory region. Given an attacker's goal is to sabotage a physical process and yet not be detected by an IDS, she wants to send the control logic to the targeted PLC in a stealthy manner rather than using the traditional control logic injection techniques [5], [12], [20] that involve transmitting control logic through the network [5], [12], [20]. These attacks are easily detectable by IDS that are based on features such as entropy, n-gram, decompilation, source code, and machine instructions. One way an attacker can evade such detection features is to mutate or encrypt the control logic before sending it to the PLC. Such mutations, however, lead to a challenge, i.e., guaranteeing the effective activation of the attack at the device level, which in this case would be its decryption in order for it to run properly.

PLCs have certain design features that can be exploited by attackers to achieve their goals.

1) Download Capability: Block 1 shows PLCs have a download capability, which means writing a control logic program on it. They provide programming software for engineers to perform this task. PLCs operate in two modes: 1) Program mode, where the underlying physical process stops for new program downloads, and 2) Run mode, where the physical process starts to operate but the download operations are halted. While the engineering software does not allow download operation when the PLC is in run mode, write operations via the protocol are still allowed. This vulnerability allows an attacker to inject malicious control logic into the PLC without having to change the PLC's mode of operation.

2) Scan cycle: PLCs control physical processes by taking input from sensors, processing it via logic, and producing output for actuators. This occurs in a scan cycle. This feature, along with the download capability, allows an attacker to perform any operation using code injection at the device level (such as firmware object or stack manipulation, and volatile memory acquisition [8], [17]). In order to make sure the injected code runs successfully, it must be appended to the existing control logic so it runs as part of the control logic in every scan cycle. We call this appended code programmable malicious code (PMC).

3) Redundant Address Pins: While the attacker now has a way of successfully executing the mutated control logic using code injection at the device level, it leads to another challenge, i.e., sending the required code for executing the previously transmitted malicious control logic. Given the attacker's aim to remain undetected, her intention is to ensure this code goes unnoticed as well.

Our analysis of PLCs' memories from various vendors revealed the presence of aliased memory regions. Usually, intended memory aliasing is caused when multiple virtual addresses are aligned to the same physical addresses for a certain purpose. However, in the context of PLCs, this aliasing arises due to a distinct phenomenon: the disregard of specific address bits. This unique form of aliasing, which we call "Redundant Address Pins" (RAP), is attributed to the optimization of the number of address pins used to manage small-sized memory in PLCs. This design feature, while might be efficient and easy, leads to multiple address locations appearing to have the same content, even though they all point to the same underlying physical memory. Such a phenomenon is observed in various PLCs, including Schneider Electric's M221 and M241, and Allen-Bradley's 1756-L61 [7].

In Figure 2, Block 1 illustrates how the attacker exploits RAP to inject PMC into the PLC. The key insight here is that because pins 1 - 5 are unspecified, addresses like "00000111," "00001111," "00011111," "00111111," "01111111," and "11111111" all point to the same memory region. In this case, it's the specific location targeted for PMC, which is "00000111." This method capitalizes on the fact that not all memory address spaces physically exist due to redundant address pins.

RAP lets the attacker insert code by splitting it into pieces and putting them into separate memory spaces that seem unconnected. However, the end result is a complete program neatly written in a single continuous address space. It's important to note that with RAP, the attacker can bypass stateful DPI that tracks memory addresses in ICS protocols. Consequently, the attacker can write to the code block ("00000111") using diverse address ranges, but network traffic indicates that different memory regions in PLCs have been updated.

The randomization of protocol addresses and the transfer of small pieces of code over the network can subvert stateful DPI, which is useful for launching an initial stealthy attack vector. We leverage RAP to append a small-size PMC to PLC's control logic. Appending to the control logic ensures that PMC is run as part of the existing control logic, acting as an enabler for the next stage of the attack [8], [17].

C. Witchcraft Stage

In this phase, the attacker, playing the role of a crafty witch, develops a clever method to send harmful control code to the PLC (as shown in Block B of Figure 2). She ensures that her method avoids detection by the IDS. Once the control code is sent, the attached PMC, added during the initiation stage, ensures the malicious control code runs smoothly. Lastly, she alters the program's flow from the original logic to the malicious one. PMC allows various ways to transmit the control code secretly and ensures they are all executed on the PLC. In the next section, we demonstrate three methods to create this control logic, i.e., 1) GizmoSplit, 2) BuffWarp, and 3) EnigmaFlow.

TABLE I: Standard network intrusion detection features

Study	**Common Detection Features**				
	F1 Prot. Header	F2 Entropy	F3 N-gram	F4 Decomp.	F5 Instr.
McPAD [21]		✓	✓		
POSEIDON [22]	✓		✓		
Anagram [23]			✓		
Shade [6]		✓	✓	✓	✓
Chang [24]					✓
ICSREF [25]	✓			✓	✓
PLC-READER [26]				✓	
SCRUTINIZER [27]				✓	✓
Zonouz [28]				✓	✓
Zhou [29]				✓	✓
Serhane [30]					✓
Valentine [31]				✓	✓
Klick [32]	✓				✓
Hui [33], [34]	✓			✓	
Schuett [35]	✓				✓
ACCM [36]		✓			
Balikcioglu [37]				✓	
Yang [38]					✓
McLaughlin [39]					✓

V. Witchcraft Stage Attacks

This section showcases three distinct attack techniques as proof of concept for the witchcraft stage to subvert standard intrusion detection features. However, more attacks are possible via PMC to subvert IDS features not discussed here.

A. Standard Intrusion Detection Features

We outline common detection features based on fundamental characteristics of detection techniques (Table I). Since our attacks occur over the network and aim to inject malicious control logic, these recommended features are primarily derived from network packet inspection and binary analysis.

Protocol Header (F1). The protocol header feature focuses on analyzing the header information in network packets. It's crucial for extracting payload data and conducting further detection. Network-based code injection attacks and detection often require analyzing protocol headers. [14], [20], [40]–[44].

Entropy (F2). Entropy measures data randomness, posing challenges for attackers due to the difficulty in mimicking genuine entropy patterns [21], [23], [45]. It plays a crucial role in detecting and countering code injection attacks by evaluating data randomness levels.

N-gram (F3). An N-gram is a contiguous sequence of 'n' items from a text and is used in binary-based anomaly detection [46]. Pairing N-grams with techniques like Bloom filters [6], [23], [47] or Self-Organizing Maps (SOM) [22], [48] enhances their efficacy. Bloom filters capture unique n-grams from write request message payloads, particularly those with control logic code. Two derived features from these bloom filters include the total count of unique n-grams (*#NGram*) and the longest consecutive n-gram sequence (*LNGram*).

Decompilation (F4). The decompilation feature focuses on the process of converting compiled machine code into a higher-level programming language. It counts the total number of bytes that can be successfully decompiled.

Instructions (F5). This feature comprises byte sequences that translate into machine instructions, such as opcodes, and, specifically in PLCs, both rungs and opcode [14]. In PLC control programs, ladder logic contains rungs, each representing a specific set of control instructions. Opcodes, on the other hand, signify low-level instructions executed by the PLC processor. By matching these opcodes to their respective instruction list (IL) instructions, we can grasp the control logic's behavior. This feature quantifies the total count of detected rungs and opcodes in the binary.

B. GizmoSplit Attack

A control logic binary code is made up of a series of instructions that must be executed sequentially for it to operate successfully. For this reason, it is always written consecutively in the PLC memory, whether by a legitimate user or an attacker [2], [12], [20]. Yoo et al. [12] demonstrate control logic injection attacks that add significant noise to the attack payload while keeping the size of the control logic significantly small to subvert DPI techniques. However, these attacks can be detected by stateful IDS, which create a virtual PLC memory using the address fields in the protocol header of write request messages, rather than relying solely on individual packet payloads [6]. GizmoSplit is resilient against such IDS.

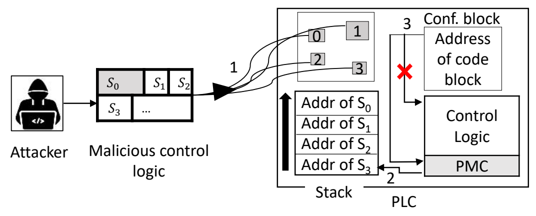


Fig. 3: GizmoSplit attack

Attack Flow. Figure 3 shows the complete flow of the attack. It splits a malicious control logic program into different segments(such as s_0, s_1), with each segment being a sequence of instructions that end with a 'return' instruction. Hence, we can say that each segment is a gadget [49]. The attack then places each gadget in a random memory location. And then finally, to execute each gadget, the attack appends PMC to the existing control logic using RAP, which is designed to activate various gadgets located at random memory locations in a specific sequential order, similar to return-oriented programming. This is achieved by modifying the stack with addresses corresponding to the control logic segments or gadgets positioned across different memory locations. The objective is to ensure the control logic code executes in the desired sequence.

Subsequently, the pointer responsible for indicating the start of the control logic is altered to point to the beginning of PMC instead. By doing so, the original control logic is effectively bypassed and replaced with the malicious version.

Detection features bypassed. The attack bypasses the protocol header feature (F1) since the address in the write request message cannot be used to create a virtual memory to detect control logic. It is to be noted that the segments can be of varying size but small enough so as not to be identified as code using the different features that are used in IDS. For instance, the size of a segment should be shorter than an instruction that can be identified as a rung(F5). The attack also bypasses higher-order n-gram. Moreover, since the attack splits the control logic into segments, the entropy (F2) of individual segments is very low.

C. BuffWarp Attack

Buffwarp attack involves the use of a small buffer located at a specific memory address to inject chunks of control logic repeatedly until the entire control logic code is written. Similar to GizmoSplit, this attack cleverly evades IDS systems that establish virtual memory by scrutinizing write requests from network packets [6]. However, the attack's repetitive overwriting of a specific region thwarts such IDS efforts.

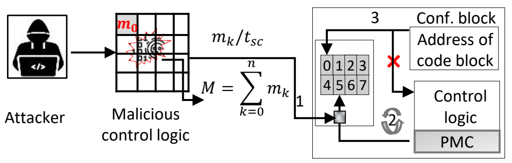


Fig. 4: BuffWarp attack

Attack Flow. Figure 4 shows the flow of the BuffWarp attack. It initiates by obtaining a copy of the malicious control logic 'M', which is divided into segments of size N. The attacker then transmits these control logic segments to the buffer, overwriting its contents. Each segment(m_k) should be sent per scan time (t_{sc}) of the control logic of the PLC. Each transmission includes a valid bit that indicates whether the data in the buffer is valid. The inclusion of sending a valid bit with every transmission, along with making sure that each segment is sent per scan time(time taken by a PLC to complete one cycle of processing its program logic) is necessary for PMC's success since it ensures that the contents of the buffer are copied only when it is updated.

In this attack, PMC continually monitors the buffer's valid bit to determine if the buffer has been updated. When the bit is valid, PMC copies the buffer contents to another memory location. If the bit is invalid, no copying occurs. As each segment is written in one scan time, the process guarantees that no packets are missed. The copied buffer contents are then sequentially written to a new memory location to ensure the complete transfer of the control logic.

Finally, the attacker alters the control flow to execute the malicious control logic from the new memory location.

Detection features bypassed. Similar to GizmoSplit, BuffWarp is also resilient against IDS that use protocol header information (F1) to create a virtual memory because of having the same address in all the write request messages. Moreover, since the attack is divided into small chunks, the entropy (F2) of each chunk would be quite low. The attack also bypasses higher-order n-gram. Each chunk is also small enough not to be successfully decompiled to a valid instruction (F4) or identified as a rung or opcode (F5) from the database.

D. EnigmaFlow Attack

In this attack, an attacker tries to conceal malicious control logic from an IDS by encoding it to make it difficult to understand. By encoding the control logic, an IDS is unable to recognize individual instructions as control logic code with instructions forming rungs and opcodes. Even if an encoded instruction is identified as a rung, the probability of the entire encoded control logic being detected is low.

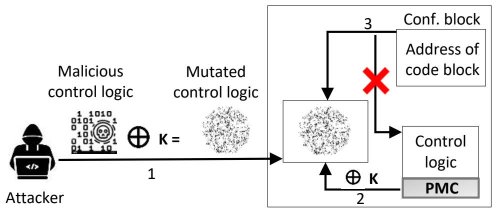


Fig. 5: EnigmaFlow attack

Attack Flow. Figure 5 depicts the sequential steps of the attack. Initially, the control logic is encoded using a secret key, following which the encoded program is transmitted to the PLC. The encoding technique used depends on the attacker's objectives for stealthiness. Alternatively, a more complex encryption algorithm can enhance evasion from an IDS, though it may impact real-time control. The appended PMC extracts the encoded control logic from memory and decodes it using the same key. The decoding is done chunk by chunk, overwriting the encoded code with the decoded one.

Once the decoding process is complete, EnigmaFlow changes the pointer that points to the start of the PLC's code block. Instead of pointing to the original code, the pointer now points to the memory location where the decoded malicious code is written. As a result, when the PLC executes its control logic, it will unintentionally run the modified, malicious code instead of the original control logic.

Detection features bypassed. N-gram (F3) is circumvented due to the difficulty in locating identical data sequences between the mutated control logic and the control logic in the trained attack dataset. Similarly, the mutated control logic cannot be successfully decompiled to valid instructions (F4), and hence, it is difficult to detect rung and opcode (F5).

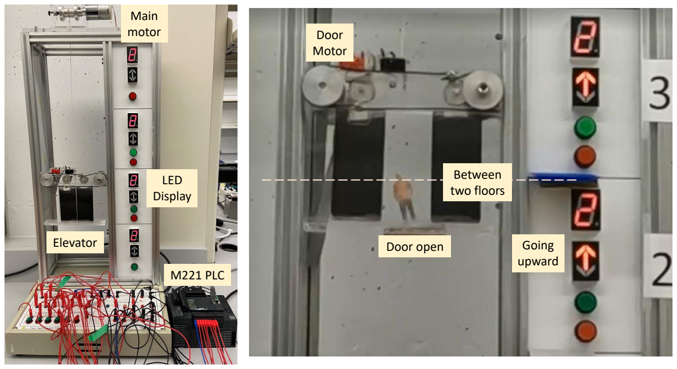


Fig. 6: (a) Front view of the elevator model (b) Malfunctioning elevator under attack

VI. Implementation

A. Testbed

We executed the attacks on Schneider Electric's Modicon M221 PLC (firmware v1.5.1.0), equipped with a 32-bit Renesas RX630 microcontroller and connected to a laboratory scale fully functional elevator (shown in Figure 6(a)). The four-floor elevator model allows users to input their desired floors from both inside and outside the elevator. In response, the elevator moves to the desired floor while the LED lights show the floor it is on. The elevator door stays closed by default, opening only upon reaching the desired floor, and closing a few seconds later. Each input/output port of the PLC is connected to some elevator component. The engineering software for crafting malicious control logic is SoMachine Basic (version 1.5).We developed attack scripts in Python 2.7 and utilized RX Renesas assembly language for PMC. We employed Unified Messaging Application Services (UMAS) over Modbus protocol to generate the attack scripts.

B. General implementation steps

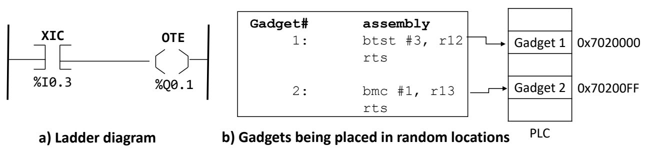
The attacker carries out the same steps to run the attacks.

Step 1: Getting compiled control logic code. To execute a successful attack, an adversary prepares malicious control logic using engineering software. Once the logic is ready, they capture network packets during the download process to an M221 PLC. By analyzing these packets, the attacker can extract the compiled logic code from payload data within write request messages using address fields.

Step 2: Analyzing the memory of a PLC. After capturing and extracting the compiled logic code, the attacker analyzes the M221 PLC's memory. This analysis reveals available free space and identifies regions that share the same content when a specific pinpoints to a location.

Step 3: Transferring PMC using RAP. The attacker requests to read the PLC's configuration block, determining the control logic's address and size. This information is crucial for appending PMC. PMC is then appended to the control logic using RAP. For example, the M221 PLC has RAP set in the 5(4-8) bits of the total $32(2^5)$ regions, similar to as shown in Figure 2. Due to this RAP configuration, in the M221's total 32-bit (4-byte) address space memory, the following addresses point to the same physical memory space: `0x701e000`, `0x709e000`, `0x711e000`, `0x719e000`, and so on. If PMC has an 11-byte size and is inserted into this space one byte at a time, more than $5.15 * 10^{15}$ methods (calculated by ${}_{32}P_{11}$) can be used to insert it into a physically contiguous address space without using adjacent addresses. The motivation for appending PMC to existing control logic is to ensure its successful execution as part of the control logic during a scan cycle.

Step 4: Preparing the method of transferring the malicious control logic to the PLC. With the necessary information about the control logic's address and size, the attacker can now transfer the malicious control logic to the target PLC. There are three primary methods for doing this:



```
Gadget#      assembly
   1:        btst #3, r12
             rts

   2:        bmc #1, r13
             rts
```

Fig. 7: Control logic program (that turns on output 1 if input 3 is on) and its gadgets

1- Segmenting the control logic and ending each segment with a “return" instruction, turning each segment into a gadget. These segments are then written to random locations in the PLC’s memory that are free to use. Note that these random locations should lie outside of the code block, which always falls between `0x701e000` and `0x701fed4`. Figure 7 shows a simple control logic program (figure 7(a) that is segmented into multiple gadgets (figure 7(b). The control logic program turns on a light if a particular switch is triggered. Gadget 1 is written at memory address `0x7020000`, and gadget 2 at `0x70200FF`. These gadgets are then run with the help of PMC, which is explained in Step 5.

2- Similar to the first technique, the attacker divides the control logic into N segments and writes them to a buffer at a specified memory address (`0x7020000`). This buffer is overwritten with each subsequent write request. For our experiments, we used a 4-byte buffer size. Additionally, with each write request, the attacker writes the value `0x01` to another memory location (`0x701FFEE`) to signal that the buffer was overwritten. This step is crucial for PMC’s success (explained in Step 5) in copying the buffer contents of the malicious control logic to a designated area. This ensures that with each subsequent write request, the contents are not overwritten in this new area where the control logic is consecutively written.

3- Mutating the entire control logic by encoding it with a one-byte key (`0x02`) and writing it consecutively to free space outside the code block. This encoded control logic, stored at `0x7020000`, is later decoded and replaced with the decoded version using PMC (explained in Step 5).

Step 5: Running the PMC. PMC ensures the activation of each attack technique for it to run successfully. If the first technique was used, PMC modifies the stack with gadget addresses in the sequence forming the prepared malicious code in step 1. Figure 8 shows the PMC in the RX assembly language responsible for stack modification. In order to modify the contents of the stack, we utilize the stack pointer, which is the R0 register in the case of RX renesas architecture. However, it is to be noted that modifying the contents that the current stack pointer value points to may make the PLC malfunction. Hence, in order to keep the PLC functioning as normal, we decrement the value of the stack pointer and update the memory address it points to with the gadget addresses of the control logic segments written in random memory locations in Step 4.

For the BuffWarp attack, PMC continually checks whether the buffer is overwritten. If it was, the contents are copied to another memory location where the control logic is eventually written in sequence.

```
line     assembly                     line      assembly
  1:     sub #8, r0                     4:      mov.l #0x070200FF, r3
  2:     mov.l #0x07020000, r2          5:      mov.l r3, [r0]
  3:     mov.l r2, [r0]                 6:      rts
```

Fig. 8: PMC for Stack Modification in GizmoSplit attack

```
line       assembly                  line       assembly
   1:      push.l r1                   11:      mov.l r5, [r8]
   2:      push.l r8                   12:      add #4, r8
   3:      push.l r5                   13:      add #1, r1
   4:      push.l r10                  14:      cmp r10, r1
   5:      mov.l #0x7030000, r8        15:      ble.b _top
   6:      mov.l #0x4000, r10          16:      pop r10
   7:      mov.l #0, r1                17:      pop r8
   8:      _top                        18:      pop r1
   9:      mov.l [r8], r5              19:      rts
  10:      xor #2, r5
```

Fig. 9: PMC decoder for EnigmaFlow attack

For the EnigmaFlow attack, the attacker decodes the previously encoded entire control logic using an assembly code decoder shown in Figure 9. Line 5 loads the address (`0x7030000`) where the encoded malicious control logic is stored in register R8, along with the control logic’s size, into register R10. The loop from lines 8 to 15 executes the decoding process. Each 4-byte block of the control logic, starting from the address in register R8, is decoded using an XOR operation with the key value (`0x02`). After each XOR operation, the address register is incremented by 4 (registers can hold 4-byte values) to move to the next block of encoded data. This loop continues until the entire control logic is decoded.

Step 6: Changing the control flow. Finally, the attacker changes the pointer in the configuration block to point to the new control logic. In the case of BuffWarp and EnigmaFlow, the attacker changes the pointer to point to the new memory location where the new control logic is stored. However, in the case of GizmoSplit, since the segments are located at random locations, the attacker changes the pointer to point to the start of the PMC which modifies the stack with addresses of the attacker-specified gadgets, allowing the malicious control logic to execute exclusively. Note that due to the alteration in the control flow, PMC will no longer execute for BuffWarp and EnigmaFlow attacks. However, this change does not pose a problem as PMC has already activated the attacks.

VII. Evaluation

This section presents 1) the evaluation metric and results of witchcraft stage attacks and their resilience against IDS, 2) the effectiveness of RAP during the initial attack stage, and 3) the impact of the attacks on an elevator, representative of a real-world physical process.

A. Experimental Settings

We utilized 49 different control logic programs for evaluation targeting various physical processes such as traffic lights, belt conveyor systems, and elevators; 22 for training IDS to detect control logic payload in ICS network traffic, and 27 for evaluating control logic attacks. Each attack utilizes a distinct method of sending the control logic over the network, resulting in individual network traffic datasets.

B. Evaluation Metric

This section introduces three key metrics for evaluating witchcraft-stage attacks. We first explore the difference between stateful and individual DPI. Following that, we discuss the employed detection system. Lastly, we introduce a control logic injection attack from existing literature for comparative analysis with our proposed attacks.

1) Individual DPI vs. Stateful DPI: We assess each attack using both individual and stateful DPI. In individual DPI, each packet is individually tested for control logic code occurrence in the payload. Although effective against certain control logic attacks [2], [14], DPI techniques fall short against attacks that transmit small-sized control logic blended with noise to seamlessly integrate with normal traffic [12].

To address this challenge, we leverage the *Shade* approach in Yoo *et al.*'s [6] study. Shade integrates DPI with traditional stateful inspection to detect control logic attacks. Specifically, it observes the protocol header in each packet, containing a write request message. Shade utilizes addresses in the protocol header and the content in the write request message payload to construct a shadow memory of an underlying PLC. This shadow memory is then examined to identify the occurrence of any control logic code.

2) Subversion of Standard IDS Features: We evaluate our attacks on a network intrusion detection system that uses classic machine learning algorithms. Employing a two-stage strategy with Gaussian Naive Bayes (GNB) and Support Vector Machine (SVM) [50]–[52], our approach involves an initial one-dimensional GNB classifier evaluating each feature individually to identify the most discriminative ones for model creation. The discriminative features play a crucial role in enhancing the model's ability to distinguish between different classes in the binary classification scenario. Subsequently, SVM is employed to build and assess the detection performance of models using these selected features. The goal of this detection system is to identify if control logic is being transferred over the network, not to distinguish between malicious and benign logic. In the ICS domain, where control logic updates are infrequent, any attempt at control logic transfer should be treated as a potential malicious event. Hence, the training datasets in the SVM model consist of control logic programs that are not necessarily malicious.

The detection system incorporates five detection features (listed in Table I). Features F2-F4, such as entropy, n-gram, decompilation, and instructions, are utilized for model training, while Feature F1, the protocol header, serves solely for stateful inspection and does not contribute to model training. We use it to employ Shade [6] to form shadow memory for stateful DPI. Throughout the evaluation, we consistently use this feature to construct a shadow memory (SM) of the PLC for comparison between individual and stateful DPI. Note that these features are used in various detection systems (refer to Table I), and our approach involves recreating a detection system that incorporates these features to evaluate our attacks' resilience. We measure the attack's resilience by calculating a detection rate (DR) for each feature. It is calculated as;

$$DR = \frac{\sum(\text{Total count of detected features in each packet})}{\text{Number of packets}}$$

In stateful DPI, the total count of a feature in the SM region is calculated instead of in individual packets.

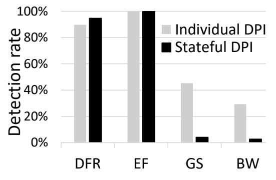
3) Benchmarking with Related Work: We assess the effectiveness of the attacks by comparing their outcomes with a referenced attack in the existing literature [12]. The designated attack, named DEFRAN (Data Execution, Fragmentation, and Noise padding), combines two established techniques. In data execution attack, the attacker downloads the control logic code into the data section of a PLC. This strategic move allows the attacker to bypass restrictions that typically prohibit writing directly to the code section of a PLC. The second technique, Fragmentation and Noise padding, adopts a distinctive strategy. It systematically fragments the control logic into a compact size, introduces noise, and iteratively writes the entire code in incremental steps. The fragmentation process aims to minimize the size of the control logic, making it challenging to detect through mechanisms such as deep packet inspection. DEFRAN is designed to bypass signature-based IDS as well as DPI techniques. Comparatively, we utilize DEFRAN to demonstrate that our attacks are stealthier, evading additional detection systems beyond DEFRAN's capabilities [6].

C. Witchcraft-stage Evaluation

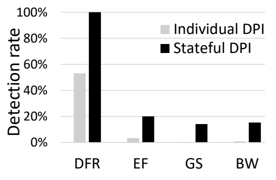
In this section, we present the evaluation results for the witchcraft-stage attacks utilizing the previously outlined metrics. First, we examine the bypassing of each detection feature, followed by an aggregation of all features for an overall assessment as shown in Figure 10. It's important to note that the protocol header, utilized in stateful DPI for all features, is not evaluated separately.

Entropy. Figure 10(a) presents the entropy-based detection rates for DEFRAN and the proposed attacks. The presented detection rates in all figures represent the average detection rate across the 27 control logic programs used for testing. Notably, EnigmaFlow and DEFRAN consistently demonstrate high detection rates in both individual and stateful DPI. This consistency stems from EnigmaFlow's obfuscation techniques (with no data loss), preserving entropy values similar to those in the DEFRAN attack. It's crucial to emphasize that the detection rate is calculated based on the resemblance of entropy values to those found in payloads containing control logic code. However, GizmoSplit and BuffWarp manifest lower detection rates due to payload segmentation, which alters total entropy. Segmenting the payload essentially increases overall entropy, aligning with entropy's nature to escalate with data complexity. Consequently, distinct entropy values in attacks like GizmoSplit and BuffWarp diminish the detection rates.

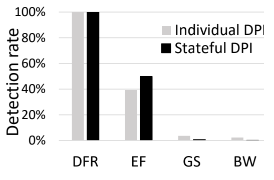
It is evident that, in both of these attacks, the detection rate under stateful DPI is lower than that of individual DPI. This occurs because both GizmoSplit and BuffWarp are designed to trick detection systems that create an SM. Consequently, each write request (with a control logic segment) is treated



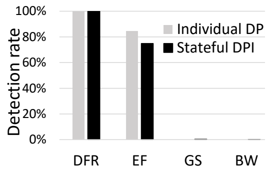
(a) Entropy



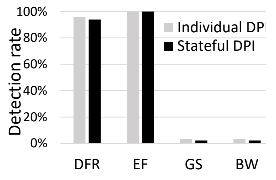
(b) N-gram



(c) Decompilation



(d) Rung



(e) Opcode

Fig. 10: Attack detection rate of features using both individual and stateful deep packet inspection (DPI) for DEFRAN (DFR), EnigmaFlow (EF), GizmoSplit (GS), and BuffWarp (BW)

as a single control logic injection, resulting in substantially different entropy values compared to those observed for the entire control logic in a typical detection system.

N-gram. Figure 10(b) depicts the outcomes of n-gram analysis. Notably, EnigmaFlow, GizmoSplit, and BuffWarp demonstrate near-complete evasion of detection in individual DPI. Even with stateful DPI, their detection rates remain significantly lower compared to the DEFRAN attack. It is noteworthy that EnigmaFlow's detection rate was slightly higher than that of GizmoSplit and BuffWarp. This discrepancy arises because both GizmoSplit and BuffWarp segment the control logic, making it more challenging for the detection system to identify continuous sequences of bytes corresponding to control logic code within the payload. In the case of EnigmaFlow, however, given its mutation method using a simple XOR with a one-byte value, the detection rate was comparatively higher. It is crucial to recognize that the effectiveness of EnigmaFlow may vary based on the specific mutation technique employed.

Decompilation. Figure 10(c) presents decompilation-based evaluation results. DEFRAN achieves the highest detection rate, followed by partially detected EnigmaFlow due to its simple XOR mutation. GizmoSplit and BuffWarp show comparable results, with GizmoSplit's detection rate slightly surpassing BuffWarp. This difference is attributed to GizmoSplit's segments, each functioning as an executable instruction, resembling a gadget and allowing for successful decompilation.

Instructions. Figure 10(d) and Figure 10(e) show the evaluation results using instructions (such as rung and opcode) as a feature. In the case of Rung, both GizmoSplit and BuffWarp show very low detection rates under both individual and stateful DPI. This is attributed to the fact that the smallest size of bytes translatable to a rung is typically greater than the segment size chosen for these attacks. Additionally, GizmoSplit's random addresses and BuffWarp's fixed buffer for placing control logic segments make it challenging to create a shadow memory using the protocol header from network traffic packets. EnigmaFlow demonstrates a lower detection rate compared to DEFRAN, even under stateful DPI, due to the data mutation preventing precise feature matching. However, in the case of opcodes, EnigmaFlow exhibits a higher detection rate compared to DEFRAN. The mutation process may generate byte sequences that correspond to opcodes in the database. For instance, XORing with the key value, `0x02`, transforms `0x00` to `0x02` which corresponds to a 'return' instruction, resulting in an increase in the number of detected instructions.

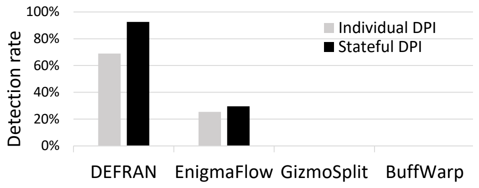


Fig. 11: Attack detection rate using aggregated features

Evaluation with all features combined. Figure 11 illustrates the detection rates of attacks using all features combined. Notably, the DEFRAN attack exhibits a 23.63% increase in detection rate under stateful DPI. Among the proposed techniques, EnigmaFlow shows a slight improvement under stateful DPI, while GizmoSplit and BuffWarp, designed to subvert stateful DPI, demonstrate negligible detection rates for both DPI techniques. The inclusion of all features in a detection system is crucial to avoid false positives specific to particular features. Our attacks successfully evade detection, maintaining rates below 30% using both DPI techniques.

D. Initial Attack Stage: RAP and PMC Evaluation

We assess RAP's effectiveness, for PMC injection, in evading IDS using n-gram, instructions, and decompilation as detection features. To minimize detection, we divide PMC into byte-sized segments and then each segment is inserted via the RAP approach, appearing in random locations while actually being inserted at consecutive ones.

RAP randomness. Our examination of the M221 PLC revealed that the aliased region can be accessed from 32 different addresses (equivalent to 2^5, indicating five RAPs). This configuration allows injections into contiguous physical memory addresses through 32 seemingly non-contiguous addresses.

Detection of PMC payload. Table II presents the detection results with and without RAP being employed for PMC's injection. Without RAP, PMC could be detected using features like n-gram, instructions, and decompilation. However, with RAP in use, PMC remained undetected in n-gram and decompilation analyses. In the case of the instructions feature, only one instruction (0x02, the 'return' instruction) was detected. since it is one byte long and is necessarily part of all programs including PMC. However, the detection of a single instruction with just one byte in the entire code is often dismissed as a false positive [53], [54].

E. Evaluating Attacks' Effectiveness on an Elevator

Our attacks compromise a fully functional four-floor elevator in the testbed successfully. Figure 12 illustrates malicious

TABLE II: PMC injection made stealthier using RAP

	Without RAP			With RAP		
	N-gram	Instructions	Decompilation	N-gram	Instructions	Decompilation
GizmoSplit PMC	●	●	●	○	◐	○
BuffWarp PMC	●	●	●	○	◐	○
EnigmaFlow PMC	●	●	●	○	◐	○

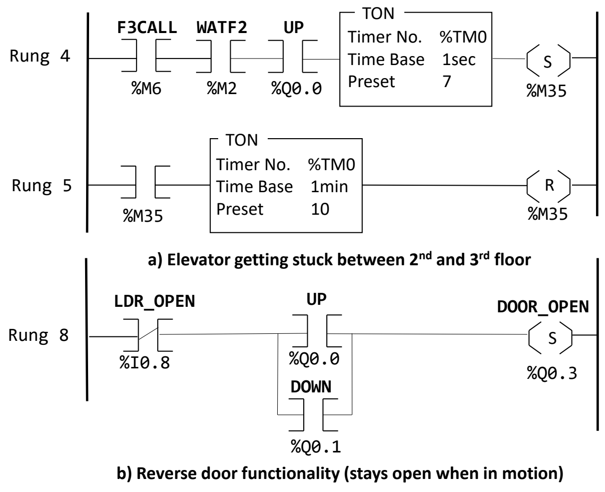


Fig. 12: Malicious ladder logic snippets for elevator

ladder logic snippets from a compromised elevator program with four essential instructions: XIC (e.g., F3CALL, WATF2, UP, DOWN), TON (%TM2, %TM3), XIO (LDR_OPEN), and OTE (%M35, %Q0.3). The XIC instruction examines a bit, evaluating as true if the bit value is one and vice versa, while XIO evaluates as true if the bit value is zero. When Floor 3 is pressed (F3CALL), the elevator is at Floor 2 (WATF2), and it's moving upward (UP); it continues for 7 seconds based on timer %TM2. Afterward, memory bit %M35 is set, initiating a 10-minute delay with timer %TM3. This delay effectively halts the elevator between the two floors until %M35 is reset, allowing the elevator to resume movement. Additionally, the program includes a reverse door functionality. When the elevator is moving upward (UP) or downward (DOWN), output bit %Q0.3 is set, causing the door to open. We successfully implanted the control logic into the PLC using all three attack methods. Figure 12(b) illustrates the elevator malfunctioning under attack.

TABLE III: Impact of PMC appended to an elevator control logic. The original control logic size is 659 bytes, and its baseline scan-cycle time is 120-127μs.

Attack Method	PMC Size (bytes)	Control Logic Scan Time (μs)	Delayed Time (μs)
GizmoSplit	11	120-127	>1
BuffWarp	29	140-150	>23
EnigmaFlow	43	150-170	>43

We evaluate the impact of injecting PMC on real-time control of the elevator, measured by the PLC's scan time. After transmitting witchcraft-stage attacks to the PLC, PMC activated as an initial vector. This may introduce a slight delay, potentially increasing the scan cycle time. The extent of this delay depends on PMC's size. Table III shows scan times after appending PMC to the 659 byte-sized control logic. Notably, GizmoSplit maintains the same scan time as control logic without PMC (120-127μs), indicating no significant delay due to its small PMC size. BuffWarp, with a fixed size of 29 bytes, adds a processing step, resulting in a scan cycle of 140-150μs. EnigmaFlow, featuring a larger size of 43 bytes and complex operations, exhibits the highest scan time, ranging from 150-170μs.

VIII. Mitigation

Addressing our proposed attacks may appear feasible by enhancing learning mechanisms related to packet timing, header, and payload in network traffic. However, this approach has its own challenges. For example, BuffWarp operates predictably with a consistent address during the scan cycle, making it challenging to differentiate between malicious and benign activities. Additionally, predicting payload encoding in EnigmaFlow is uncertain and introduces substantial unpredictability, making reliance on payload learning a precarious strategy.

Given our attack strategies exploit PLC memory architecture vulnerabilities, traditional methods like DPI are obsolete. To detect such attacks, the following approaches should be considered if traditional security measures (such as demilitarized zones) are not feasible.

Memory-associated Monitoring System. Since all inputs that can impact the PLC control logic, including vulnerabilities like RAP, are written through the network, scrutinizing memory alterations with precision could illuminate suspicious write requests. Coupling this with insights about the sender of these requests might give away any illicit activities. Such a memory-focused monitoring system can be designed to detect inconsistencies or anomalies in the memory write patterns using fingerprinting, and digital signature.

Code verification methods. Like formal techniques, may be considered for mitigation [28], [55]. However, they might not be suitable here as the injected code falls outside the verification scope due to its indirect nature, rendering these methods ineffective against such attacks.

Control-flow integrity (CFI). prevents control-flow hijacking attacks using predefined control-flow graphs. While CFI is common in general computers, its use in real-time PLCs can cause significant performance issues for industrial control systems (ICS). To address this, specialized CFI solutions like the one in [56] are designed for ICS. They minimize performance overhead while ensuring effective protection against control-flow hijacking attacks.

IX. Conclusion

In this paper, we discussed vulnerabilities in PLC design features and explored two features i.e., control logic download and redundant address pins (RAP), to show that malicious actors can exploit to stealthily inject a small piece of code (PMC) into a PLC device. We demonstrated that PMC can be attached to a PLC's control logic code to run as part of the program scan cycle allowing the actors to further compromise the PLC to attack an underlying physical process. We proposed three attack techniques such as GizmoSplit, EnigmaFlow, and BuffWarp, as proof of concept that can bypass common IDS features to further download and run a complete malicious control logic. The attacks blended their payload with the network traffic via payload encoding or small-sized payloads and were challenging to detect by traditional IDS.

References

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

[2] R. Langner, "Stuxnet: Dissecting a Cyberwarfare Weapon," *IEEE Security Privacy*, vol. 9, no. 3, pp. 49–51, May 2011.

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

[4] R. Dudley and D. Golden, "The colonial pipeline ransomware hackers had a secret weapon: self-promoting cybersecurity firms," 2021.

[5] 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].

[6] H. Yoo, S. Kalle, J. Smith, and I. Ahmed, "Overshadow plc to detect remote control-logic injection attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19–20, 2019, Proceedings 16*. Springer, 2019, pp. 109–132.

[7] 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.

[8] 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.

[9] M. Sun, Y. Lai, Y. Wang, J. Liu, B. Mao, and H. Gu, "Intrusion detection system based on in-depth understandings of industrial control logic," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 3, pp. 2295–2306, 2022.

[10] M. Hailesellasie and S. R. Hasan, "Intrusion detection in plc-based industrial control systems using formal verification approach in conjunction with graphs," *Journal of Hardware and Systems Security*, vol. 2, pp. 1–14, 2018.

[11] L. Garcia, S. Zonouz, D. Wei, and L. P. De Aguiar, "Detecting plc control corruption via on-device runtime verification," in *2016 Resilience Week (RWS)*. IEEE, 2016, pp. 67–72.

[12] 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.

[13] N. Govil, A. Agrawal, and N. O. Tippenhauer, "On ladder logic bombs in industrial control systems," in *Computer Security: ESORICS 2017 International Workshops, CyberICPS 2017 and SECPRE 2017, Oslo, Norway, September 14-15, 2017, Revised Selected Papers 3*. Springer, 2018, pp. 110–126.

[14] 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*, 2018, pp. 319–329.

[15] 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.

[16] W. Alsabbagh and P. Langendörfer, "Patch now and attack later—exploiting s7 plcs by time-of-day block," IEEE, pp. 144–151, 2021.

[17] A. Ayub, N. Zubair, H. Yoo, W. Jo, and I. Ahmed, "Gadgets of gadgets in industrial control systems: Return oriented programming attacks on plcs," in *Proceedings of the 16th IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2023.

[18] 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].

[19] S. A. Qasim, W. Jo, and I. Ahmed, "Pree: Heuristic builder for reverse engineering of network protocols in industrial control systems," *Forensic Science International: Digital Investigation*, 2023.

[20] 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.

[21] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, "Mcpad: A multiple classifier system for accurate payload-based anomaly detection," *Computer networks*, vol. 53, no. 6, pp. 864–881, 2009.

[22] D. Bolzoni, S. Etalle, and P. Hartel, "Poseidon: a 2-tier anomaly-based network intrusion detection system," in *Fourth IEEE International Workshop on Information Assurance (IWIA'06)*. IEEE, 2006, pp. 10–pp.

[23] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Recent Advances in Intrusion Detection: 9th International Symposium, RAID 2006 Hamburg, Germany, September 20-22, 2006 Proceedings 9*. Springer, 2006, pp. 226–248.

[24] T. Chang, Q. Wei, Y. Geng, and H. Zhang, "Constructing plc binary program model for detection purposes," in *Journal of Physics: Conference Series*, vol. 1087, no. 2. IOP Publishing, 2018, p. 022022.

[25] A. Keliris and M. Maniatakos, "Icsref: A framework for automated reverse engineering of industrial control systems binaries," *arXiv preprint arXiv:1812.03478*, 2018.

[26] Y. Geng, Y. Chen, R. Ma, Q. Wei, J. Pan, J. Wang, P. Cheng, and Q. Wang, "Defending cyber-physical systems through reverse engineering based memory sanity check," *IEEE Internet of Things Journal*, 2022.

[27] O. Mirzaei, R. Vasilenko, E. Kirda, L. Lu, and A. Kharraz, "Scrutinizer: Detecting code reuse in malware via decompilation and machine learning," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 18th International Conference, DIMVA 2021, Virtual Event, July 14–16, 2021, Proceedings 18*. Springer, 2021, pp. 130–150.

[28] S. Zonouz, J. Rrushi, and S. McLaughlin, "Detecting industrial control malware using automated plc code analytics," *IEEE Security & Privacy*, vol. 12, no. 6, pp. 40–47, 2014.

[29] M. Zhou, F. He, M. Gu, and X. Song, "Translation-based model checking for plc programs," in *2009 33rd Annual IEEE International Computer Software and Applications Conference*, vol. 1. IEEE, 2009, pp. 553–562.

[30] A. Serhane, M. Raad, R. Raad, and W. Susilo, "Plc code-level vulnerabilities," in *2018 International Conference on Computer and Applications (ICCA)*. IEEE, 2018, pp. 348–352.

[31] S. E. Valentine, "Plc code vulnerabilities through scada systems," Ph.D. dissertation, University of South Carolina, 2013.

[32] J. Klick, S. Lau, D. Marzin, J.-O. Malchow, and V. Roth, "Internet-facing plcs as a network backdoor," in *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2015, pp. 524–532.

[33] H. Hui, K. McLaughlin, and S. Sezer, "Vulnerability analysis of s7 plcs: Manipulating the security mechanism," *International Journal of Critical Infrastructure Protection*, vol. 35, p. 100470, 2021.

[34] H. Hui and K. McLaughlin, "Investigating current plc security issues regarding siemens s7 communications and tia portal," in *5th International Symposium for ICS & SCADA Cyber Security Research 2018 5*, 2018, pp. 67–73.

[35] C. Schuett, J. Butts, and S. Dunlap, "An evaluation of modification attacks on programmable logic controllers," *International Journal of Critical Infrastructure Protection*, vol. 7, pp. 61–68, 2014.

[36] C.-H. Tsang and S. Kwong, "Multi-agent intrusion detection system in industrial network using ant colony clustering approach and unsupervised feature extraction," in *2005 IEEE international conference on industrial technology*. IEEE, 2005, pp. 51–56.

[37] P. G. Balikcioglu, M. Sirlanci, O. A. Kucuk, B. Ulukapi, R. K. Turkmen, and C. Acarturk, "Malicious code detection in android: the role of sequence characteristics and disassembling methods," *International Journal of Information Security*, vol. 22, no. 1, pp. 107–118, 2023.

[38] K. Yang, H. Wang, and L. Sun, "An effective intrusion-resilient mechanism for programmable logic controllers against data tampering attacks," *Computers in Industry*, vol. 138, p. 103613, 2022.

[39] S. E. McLaughlin, S. A. Zonouz, D. J. Pohly, and P. D. McDaniel, "A trusted safety verifier for process controller code." in *NDSS*, vol. 14, 2014.

[40] P. Fogla, M. I. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee, "Polymorphic blending attacks." in *USENIX security symposium*, 2006, pp. 241–256.

[41] 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.

[42] W. Alsabbagh and P. Langendörfer, "A stealth program injection attack against s7-300 plcs," IEEE, pp. 986–993, 2021.

[43] ——, “A control injection attack against s7 plcs-manipulating the decompiled code,” IEEE, pp. 1–8, 2021.

[44] 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.

[45] K. Wang and S. J. Stolfo, “Anomalous payload-based network intrusion detection,” in *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings 7*. Springer, 2004, pp. 203–222.

[46] A. Stolcke, “Entropy-based pruning of backoff language models,” *arXiv preprint cs/0006025*, 2000.

[47] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[48] T. Kohonen, “self-organizing maps, volume 30 of springer series in information sciences. spring-verlag,” *Berlin,*, 1997.

[49] 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.

[50] M. Cinelli, Y. Sun, K. Best, J. M. Heather, S. Reich-Zeliger, E. Shifrut, N. Friedman, J. Shawe-Taylor, and B. Chain, “Feature selection using a one dimensional naïve bayes’ classifier increases the accuracy of support vector machine classification of cdr3 repertoires,” *Bioinformatics*, vol. 33, no. 7, pp. 951–955, 2017.

[51] W. Feng, J. Sun, L. Zhang, C. Cao, and Q. Yang, “A support vector machine based naive bayes algorithm for spam filtering,” in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2016, pp. 1–8.

[52] A. Bustamam, A. Bachtiar, and D. Sarwinda, “Selecting features subsets based on support vector machine-recursive features elimination and one dimensional-naïve bayes classifier using support vector machines for classification of prostate and breast cancer,” *Procedia Computer Science*, vol. 157, pp. 450–458, 2019.

[53] D. Breitgand, M. Goldstein, E. Henis, and O. Shehory, “Efficient control of false negative and false positive errors with separate adaptive thresholds,” *IEEE Transactions on Network and Service Management*, vol. 8, no. 2, pp. 128–140, 2011.

[54] G. P. Spathoulas and S. K. Katsikas, “Reducing false positives in intrusion detection systems,” *Computers & Security*, vol. 29, no. 1, pp. 35–44, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404809000844

[55] E. Kuzmin, V. A. Sokolov, and D. Ryabukhin, “Construction and verification of plc-programs by ltl-specification,” *Automatic Control and Computer Sciences*, vol. 49, pp. 453–465, 2015.

[56] A. Abbasi, T. Holz, E. Zambon, and S. Etalle, “Ecfi: Asynchronous control flow integrity for programmable logic controllers,” pp. 437–448, 2017.