Portal: Enabling Accurate Siemens PLC Rehosting via Peripheral Proxying and Proactive Interrupt Synchronization

Haoran Li*†‡, Dakun Shen*†‡, Wenbo Shen‡, Zhen Zhu§
*The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China
†Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, China
‡Zhejiang University, China
§Zhejiang Lab, China
Email: lihaoran@bcds.org.cn; {dakun@, shenwenbo@}zju.edu.cn; zhuzhen@zhejianglab.org

Abstract—**Programmable Logic Controllers (PLCs) serve as the operational backbone of Industrial Control Systems (ICS), orchestrating critical physical processes across various industrial environments. Given their central role, the security posture of PLCs has a direct impact on the safety and reliability of industrial operations. However, the closed and proprietary nature of commercial PLCs continues to hinder security research, particularly in developing accurate and flexible emulation environments necessary for systematic vulnerability analysis.**

In this work, we present a specialized emulation framework tailored to Siemens S7 series PLCs, aiming to replicate low-level execution behaviors with high fidelity. Our design emphasizes faithful reproduction of peripheral I/O operations and interrupt-driven control flows, which are often overlooked in generic embedded system emulation. To bridge the gap between proprietary constraints and practical analysis needs, we implement a suite of mechanisms that extract and reinterpret vendor-specific runtime behavior without relying on access to internal documentation or source code.

To evaluate the robustness of our emulation, we conduct extensive analysis across 211 firmware versions, covering a wide range of device configurations and firmware updates. By integrating this emulation framework with a two-stage fuzzing methodology—combining hardware-assisted fuzzing and white-box fuzzing in a controlled virtual environment—we identify multiple previously undocumented security weaknesses, underscoring the utility of our platform in facilitating vulnerability discovery at scale. While we refrain from disclosing exploit specifics, our findings reveal systemic patterns of insecurity that merit further attention from both researchers and vendors.

Index Terms—**Programmable Logic Controller, Rehosting, Peripheral Redirection, Fuzzing**

I. Introduction

As the backbone of Industrial Control Systems (ICS), Programmable Logic Controllers (PLCs) are integral in managing the operational processes of various industrial applications. These specialized devices, designed to automate complex processes, are pivotal in industries ranging from manufacturing to energy management. However, a major concern arises from the internal security vulnerabilities of PLCs within these systems. These vulnerabilities, if exploited, can result in serious damage, not only affecting the operational integrity of industrial systems but also potentially causing significant economic losses and safety hazards [1]–[3].

Addressing the security of PLCs in ICS is crucial. Techniques like dynamic analysis (e.g., Fuzzing) are effective in uncovering potential security loopholes within PLC firmware. To perform dynamic analysis, it is necessary to rehost the operation of PLC firmware inside an emulator. This allows for monitoring the dynamic state of the PLC. However, fully virtualizing commercial PLC firmware is challenging. The primary reason is that the firmware is intricately linked to specific peripherals. Accurate rehosting requires the precise emulation of a wide range of hardware components. This is further complicated as most commercial PLCs are proprietary, with their firmware and hardware designs being confidential.

In response to these challenges, there are three strategies for rehosting PLC firmware. The first approach addresses the lack of peripheral models by re-implementing or integrating known libraries, thus avoiding hardware access [4]–[7]. The second strategy uses guided symbolic execution, treating peripheral registers as sources for symbolic inputs [8]–[12]. The third method forwards hardware accesses to a physical device during emulation, eliminating the need for hardware abstractions [13]–[16].

However, each of these approaches has its limitations. The first approach requires extensive manual effort to reverse engineer unique hardware components and replicate them in software. This is challenging due to the proprietary nature and diversity of hardware designs [4]. The second approach, while useful, cannot precisely emulate the I/O of peripherals. It generates data suitable for ongoing code execution but fails to provide the specific, detailed responses needed for proper PLC firmware operation. The limited applicability of the third approach to commercial PLCs is primarily due to their proprietary designs, which lack standard hardware debug interfaces and incorporate anti-debugging mechanisms. Additionally, the requirement to route peripheral access to

Corresponding author: Dakun Shen (dakun@zju.edu.cn).

physical devices results in notable performance degradation.

To bridge this gap, our research introduces *Portal*, a tool specifically designed for emulating Siemens PLCs. This decision acknowledges Siemens' prominent position in the PLC market, where it holds a substantial 31% share and S7 series becomes a top selection of security research [17], [18]. *Portal* increases emulation accuracy by rerouting peripheral operations from a virtual environment to actual PLC devices for execution, thereby effectively replicating the full range of operations from firmware boot-up to regular functions.

In our efforts to rehost Siemens PLCs, we faced significant technical challenges primarily due to the proprietary nature of these devices. Unlike other IoT devices, Siemens models lack standard debugging interfaces and are equipped with robust anti-debugging mechanisms that prevent the use of conventional hardware-in-the-loop methodologies such as the *Avatar* framework [14]. These security measures ensure that any attempt to modify firmware or apply breakpoints can trigger mechanisms that erase critical flash memory components thus bricking the PLC. To overcome these barriers, we embarked on a comprehensive reverse engineering process to dissect both the software and hardware components of Siemens PLCs. We developed a component within QEMU, termed the *Agent Client*, designed to facilitate the precise transfer and execution of operations from the virtual environment to the actual PLC hardware. On the PLC side, despite the absence of debugging interfaces, we embedded an *Agent Server* capable of running custom-injected code. This server receives operations from the *Agent Client*, executes them, and then returns the results. This process made sure that the operations intended for the virtual PLC were accurately carried out on the real device. To mitigate the performance degradation associated with peripheral routing, we further implement the virtualization of key resource-intensive peripherals within QEMU.

The second challenge involved effective interrupt management, a critical component for PLC functionality. Precise synchronization and handling of interrupts are essential for the operation of PLC systems. To address this, we utilized the *Agent Server*, previously integrated for peripheral access, to also identify and manage interrupts. Additionally, we introduced a proactive query mechanism within QEMU to regularly check for pending interrupt requests.

To enhance the effectiveness of our security analysis, we adopted a dual-phase fuzzing strategy, integrating real-world PLC fuzzing with detailed white-box examination within the *Portal* framework. This approach harnesses the throughput advantage of blackbox fuzzing performed directly on physical Siemens PLCs to identify potential vulnerabilities. Identified anomalies and crashes are then replicated and analyzed in *Portal*, providing a comprehensive understanding of their origins and effects. This method not only allows for rapid detection of system weaknesses but also supports thorough vulnerability analysis under the controlled conditions of our emulation environment.

In our experimental evaluation, *Portal* was tested across the Siemens S7-1200 and S7-200 SMART series PLCs. We analyzed 211 different firmware versions, encompassing all 19 models from the S7-1200 series (spanning versions from v4.2.3 to the latest v4.6.1) and 16 models from the S7-200 SMART series. Our method effectively identified and interacted with peripheral memory regions across all these versions. We achieved successful emulation in all 211 firmware versions tested, as demonstrated by the PLC's LED turning yellow. This indicates that the CPU enters `Stop` mode and is ready to establish a TCP connection.

To summarize, this work makes the following contributions:

- **Siemens PLC Rehosting Framework**: Our research represents the first instance of accurate, full-system emulation of Siemens PLCs, marking a significant advancement in the field of ICS security.
- **Integrated Peripheral and Interrupt Management**: Our approach combines the transfer and execution of peripheral memory operations from a virtual environment to a Siemens PLC, along with sophisticated interrupt management.
- **Comprehensive Evaluation**: Our system successfully rehosted all 211 available firmware versions from the Siemens S7-1200 and S7-200 SMART series. Additionally, our dual-phase fuzzing strategy enabled the discovery of previously unreported security flaws, illustrating the framework's effectiveness in uncovering latent vulnerabilities within industrial control firmware.

II. Background

A. Siemens PLCs

Programmable Logic Controllers (PLCs) are specialized industrial computers that play an essential role in automating machinery and controlling processes across various industries. Among the leading manufacturers, Siemens stands out with its extensive range of PLCs, notably the Siemens S7 series. These controllers are pivotal in operations that require high reliability and precise control.

A typical Siemens PLC integrates a CPU, memory modules, I/O interfaces, and communication modules like Universal Asynchronous Receiver/Transmitter (UART) for serial communication. These components are critical for the seamless exchange of data between the PLC and other devices such as sensors and other PLCs. The CPUs in these PLCs are often based on ARM architecture, tailored to execute control logic programs efficiently under a real-time operating system (RTOS). This setup is crucial for managing real-time operations where delay could result in significant operational disruptions. The memory architecture in Siemens PLCs is designed to accommodate both the operational firmware and user-defined control logic, thereby facilitating versatile and robust control capabilities. Communication and data transfer in Siemens PLCs are governed by protocols like PROFINET [19] and S7-commplus [20].

B. ARM Interrupts

Siemens S7 series PLCs rely on a 32-bit ARM Cortex-R4 processor. ARM processors manage hardware interruptions through Interrupt Requests (IRQs) [21]. An IRQ notifies the processor of the need to address an external event. Coordinating with the Vectored Interrupt Controller (VIC), the CPU registers signal types, interrupt priorities, and addresses for Interrupt Service Routines (ISRs) during the boot process.

Upon the activation of an IRQ, the CPU transitions into IRQ mode, where it retrieves the relevant ISR address from the VIC to handle the interruption. After executing the ISR, the CPU restores its prior state and exits IRQ mode to continue with other processes.

The Current Program Status Register [22] (CPSR) is critical in controlling IRQ management. It includes flags such as the `I` bit, which determines the processor's response to IRQs. Setting the `I` bit disables IRQs, ensuring that critical code executes without disruptions. This mechanism is crucial in maintaining operational integrity, particularly in complex systems where interrupt priorities vary.

III. Challenges and Our Techniques

This section discusses the key challenges and our solution in rehosting commercial PLCs.

A. Challenges

Our goal is to rehost Siemens PLCs by redirecting peripheral operations from a virtual environment to an actual device. This task involves navigating through several key challenges.

Peripheral Proxying. A key aspect of our rehosting process is ensuring the transfer and execution of operations related to peripheral memory from our virtual environment to a Siemens PLC. The efficiency and accuracy of rehosting depend significantly on how these peripheral interactions are handled. It is imperative for the rehosting to closely mirror the real device's behavior, ensuring every operation initiated virtually reflects precisely on the physical PLC. Achieving this is complicated due to limited public information about Siemens PLCs. The absence of detailed firmware and hardware specifications makes it a complex task to replicate these operations in a rehosting system.

Interrupt Synchronization. The second major challenge is the management of interrupts, a crucial aspect in systems running on RTOS. Interrupts play a key role in ensuring timely task completion and efficient thread management. Accurately emulating these interrupts is critical for mimicking the PLC's real-time operations in a virtual environment. This task, however, presents difficulties due to the complexity of the interrupt-driven architecture. The process demands meticulous synchronization and handling of these interrupts to maintain the system's operational integrity.

B. Virtual-to-Physical Peripheral Proxying

This section outlines our approach, focusing on establishing a communication protocol between the virtual and physical realms, and detailing the Peripheral Command Execution process.

```c
/* Create a Virtual Representation for HSC */
DeviceState *hsc0_create(hwaddr addr)
{
   DeviceState *dev = qdev_new(TYPE_HSC0);
   ...
   return dev;
}
/* Register read and write callbacks */
static const MemoryRegionOps hsc0_ops = {
   .read = hsc0_read,
   .write = hsc0_write,
   .endianness = DEVICE_NATIVE_ENDIAN,
};
/* Write callback function */
static void hsc0_write(void *opaque, hwaddr addr,
     uint64_t val64, unsigned int size){
   uint32_t res = tcp_send_to_plc(CMD_WRITE, addr
    , val64, size);
   ...
   return;
}
/* Read callback function */
static uint64_t hsc0_read(void *opaque, hwaddr
    addr, unsigned int size){
   uint32_t res = tcp_send_to_plc(CMD_READ, addr,
     0, size);
   return res;
}
```

Listing 1: HSC Representation Created by QOM

```
hsc0: hsc0@0xfffb9100 {
   ...
   reg = <0x0 0xfffb9100 0x0 0x80 0x0>;
};
```

Listing 2: HSC Representation Registered on Device Tree

1) Peripheral Command Bridging Between QEMU and PLCs: In our rehosting process, a crucial step is the transfer of peripheral memory operations from the virtual QEMU environment to a Siemens PLC device.

For this, we create virtual versions of each peripheral based on the peripheral memory map outlined in Section IV-C. These virtual versions are designed using QEMU's Object Model (QOM) [23] and device tree configuration files [24], allowing them to not just mirror memory address layouts but also control how peripherals respond to read and write requests. This setup does not just replicate peripheral behavior; it also lets us tweak the outcomes of these operations, offering us the flexibility to handle and manipulate data as needed during emulation.

Each time a memory operation hits one of these virtual peripherals, it activates the corresponding virtual representation. It handles write operations by dispatching instructions directly to the actual device. For read operations, it goes a step further – sending the request, awaiting the device's response, and then delivering the retrieved data back to the CPU. This interaction is managed by what we call the *Agent Client*, a component we implemented within QEMU that acts like a network client. To send the request, we established a TCP-based communication bridge between the virtual and real environments. This bridge operates on a simple yet effective protocol that structures every peripheral access into a comprehensive format.

On the side of the Siemens PLC, we utilize a serial port linked to an identified UART [25], [26] device to handle the sending and receiving of commands. To make this work, it is necessary to inject and run our custom code on the PLC device. The specifics of how we obtain this capability are outlined in following Section III-B2.

To demonstrate the functionality of our peripheral operation bridging method, Listing 1 provides an example of how a High-Speed Counter (HSC) [27] is virtually modeled in QEMU. This includes setting up the HSC, defining its characteristics, and implementing its read and write operations. Specifically, the functions on lines 14-23 are programmed to translate HSC-related actions into TCP commands for communication. Furthermore, Listing 2 details the registration of memory mapping for the HSC within a device tree configuration file, marking its start address and memory allocation.

2) Peripheral Command Execution on PLC: To manage peripheral operations redirected from a virtual environment to the Siemens PLC, we have embedded a specialized code module within the PLC, designated as the *Agent Server*. This server is engineered to process peripheral memory operation commands and subsequently convey the results to QEMU.

We have explored multiple strategies to inject code into the actual PLC. Initially, we anticipated locating a standard hardware debugging interface, such as SWD/JTAG [28]. The hardware reverse engineering conducted by Weber [29] successfully identified the JTAG pinout of the S7-1200. However, due to Siemens' intervention, we were unable to access complete details. Our second approach involved exploiting known vulnerabilities. Specifically, the bootloader exploit by [26] proved effective in this regard. By adapting these exploits for our PLC bootloader version, we gained the capability to inject custom code and hijack execution flow. Nonetheless, this technique is limited to specific PLCs with vulnerable bootloader. Therefore, we combined it with a third method: using an IC clip and a programmer to rewrite the SPI flash [30] containing the bootloader code. This allows us either to 'downgrade' the bootloader to a vulnerable version or to directly write the *Agent Server* into it. We also reverse-engineered the checksum algorithm and its CRC table, which is necessary to recalculate the checksum after any modifications to the bootloader.

The *Agent Server* is structured to manage an array of commands with a straightforward protocol, organizing each peripheral access into a format that includes the target address, the length of the operation, the data to be processed, and a flag indicating the type of operation. This format aligns with the ARM32 architecture, where memory access lengths vary from 1, 2, to 4 bytes, corresponding with specific instruction sets. After completing a read or write task, the *Agent Server* sends back the relevant data or a confirmation, thereby ensuring precise control over memory operations and aligning with our project's objectives.

For communication purposes, the *Agent Server* employs a serial port, utilizing a compatible UART device. We built a bidirectional data conduit, bridging the serial and TCP data flows. This setup allows QEMU to send peripheral access commands through TCP, which are then relayed by the serial USB device. Responses from the PLC are routed back to QEMU via the reverse communication path, ensuring a continuous and dynamic exchange of data. We encapsulate the serial communication with TCP to enhance flexibility, allowing both the bootloader exploits and QEMU to access the device in a standardized manner.

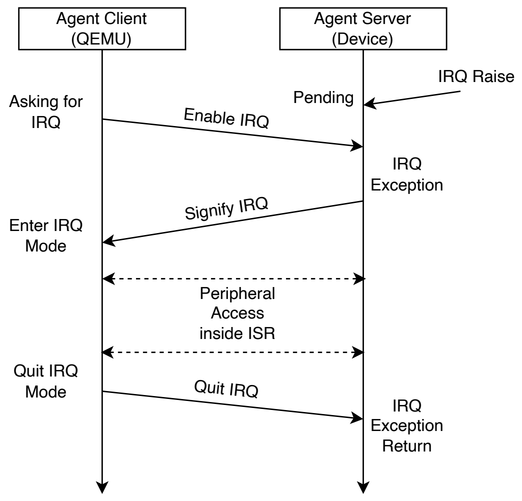

Fig. 1 : Interrupt Query Flow.

C. Proactive Interrupt Synchronization

In the realm of PLCs, especially those operating on RTOS, managing interrupts is a critical aspect. These systems adopt an interrupt-driven architecture to prevent CPU idling and ensure continuous task processing. This approach is particularly crucial in single-core, multi-threaded PLCs, where efficient task switching and system responsiveness hinge on accurately handling interrupts.

Previous research in this area has often approached interrupt handling in a somewhat generalized manner [8], [10], [12], typically generating random or periodic interrupts, detached from real-world scenarios. These approaches do not truly reflect the dynamic nature of interrupts generated in actual devices, often triggered by peripheral operations. In contrast, our method focuses on dealing with real interrupts as they occur in genuine devices. This approach ensures that our emulation not only replicates the timing of these interrupts but also their causes and effects, thus maintaining the integrity and flow of operations in line with the PLC's real-time functioning.

1) Interrupts Query: To effectively manage interrupts generated by PLCs, our approach begins with identifying the interrupt types during the firmware's boot-up phase. During this phase, the CPU configures the VIC by issuing specific peripheral commands, which register interrupt types, priorities, and addresses for the ISR. Since we redirect all peripheral interactions to the actual device, our *Agent Server* running on the PLC is equipped to detect these interrupts.

In our setup, the PLC's *Agent Server* acts as an intermediary, capturing all interrupts before they are processed. However, it is crucial to note that our *Agent Server*, primarily functioning as a server, does not initiate communication with QEMU on its own. Unprompted communication could disrupt the message order and lead to inconsistencies. Additionally, the handling of interrupts is closely tied to the state of QEMU's virtual CPU.

To navigate these challenges, we devised a strategy where the *Agent Client* in QEMU actively inquires about pending interrupts from the *Agent Server*. We have crafted a specific command that prompts the *Agent Server* to momentarily enable interrupts, allowing all pending interrupts to be collated and forwarded to QEMU for processing. This sequence is depicted in Figure 1.

As illustrated, when interrupts arise in the device segment, they are initially intercepted by the *Agent Server*, placing them in a pending state. The *Agent Client* in the virtual environment sends commands to the *Agent Server* to check for and activate interrupt processing on the device side. Consequently, all pending interrupts are relayed from the device's *Agent Server* to QEMU's *Agent Client*. The CPU in QEMU then enters IRQ mode to handle the received interrupts.

2) Interrupt Query Timing: The timing of inquiries for interrupts from the *Agent Client* is critical for the efficiency of our emulation. Ideally, QEMU would check for interrupts after every instruction when they are enabled to ensure no interrupts are missed. However, this approach could lead to excessive bandwidth usage and slow down the emulation process significantly.

To optimize performance without compromising the accuracy of interrupt handling, we have established an approach where QEMU checks for interrupts in two specific scenarios:

1) **Change in CPSR Register**: QEMU checks for interrupts when there is a change in the CPSR register that enables interrupts. This typically happens after executing the CPSIE instruction. During the period when interrupts are disabled, some may be generated and await processing. As soon as interrupts are enabled, addressing these pending interrupts promptly ensures timely system responses.
2) **After Peripheral Writes**: If interrupts are enabled, QEMU checks for interrupts after any write operation to peripheral devices. This is crucial because such write operations often induce external events that might generate interrupts. It is a strategic point to inquire about any pending interrupts from the device.

In the first scenario, the CPSIE instruction serves as a critical transition, signaling that previously disabled interrupts can now be processed. This ensures that any interrupts that occur during the critical section are not overlooked. The second condition is essential for the PLC model in focus, as operations on peripheral devices can trigger immediate interrupts, making it an opportune moment for QEMU to inquire about any pending interrupts.

```
 1 /* Set variable as 3*/
 2 MOV R0, #3
 3 MOV R1, #0x10010000
 4 STR R0, [R1]
 5 /* Enable interrupts */
 6 CPSIE I
 7 /* Write to VIC register */
 8 MOV R0, #0xFFFFFCA0
 9 MOV R1, #0x80000000
10 STR R1, [R0]
11 /* If variable is not equal to 4, then fail */
12 MOV R0, #0x10010000
13 LDR R1, [R0]
14 CMP R1, #4
15 BNE adonis_fail
```

Listing 3: VIC-Triggered Thread Swap

In our experiments, we observed a notable instance that emphasized the need for timely interrupt handling, showcased in Listing 3. Initially, the PLC firmware assigns a value of 3 to a variable at line 2. Subsequently, a write operation to the VIC register triggers an expectation for this variable to update to 4, a transition that should occur before reaching line 16 of the code. This update is contingent on a thread switch induced by an interrupt.

If this interrupt is not processed in time, the firmware redirects to a failure routine, entering an endless loop. This situation highlights the effectiveness of our interrupt inquiry strategy post-write operations to peripherals. By implementing this strategy, QEMU ensures any immediate interrupts, particularly those critical for thread transitions, are handled without delay. The ISR, responsible for thread management, executes right after the write operation, enabling the secondary thread to modify the variable.

3) Interrupts Deceleration: To manage the efficiency of handling interrupts in our rehosted PLC environment, we recognized the need to address the surge of frequent and rapid periodic interrupts. This surge, primarily induced by the inherent delay in our method of rerouting peripheral operations to the actual device, can significantly slow down the main program's execution. To counter this, we implemented two tailored strategies aimed at moderating the pace of these fast periodic interrupts.

Firstly, we identified that a subset of these interrupts is tied to countdown timers within the PLC. Typically, these timers issue interrupts upon hitting zero, resetting themselves in the process. To temper this cycle, we utilized the flexibility offered by the QOM to modify the timer settings. Specifically, we scaled up the countdown duration by a factor of S each time the timer is configured. This tweak effectively stretches the time it takes for the timer to hit zero, thereby spacing out the interrupt occurrences and allowing the main program more uninterrupted runtime.

For other generic periodic interrupts not controlled by timers, we introduced a counting mechanism within the *Agent Server*. Here, a counter associated with each interrupt type increments with each occurrence of the interrupt. Instead of

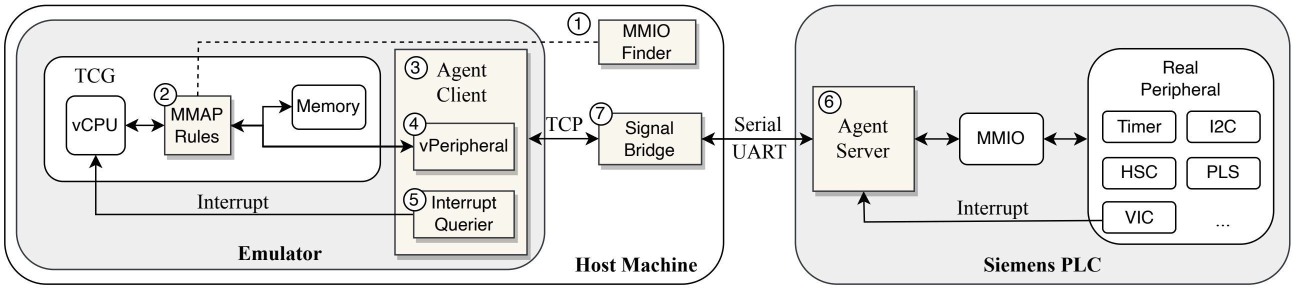

Fig. 2 : System Architecture. The MMIO Extraction tool ① analyzes firmware to define peripheral memory mapping rules ②, distinguishing between general and MMIO memory. The *Agent Client* in QEMU ③ includes vPeripheral ④ for virtual peripheral representation and an Interrupt Querier ⑤ for handling Siemens PLC interrupts. The Agent Server ⑥ on the PLC executes memory operations and manages interrupts, while the Signal Bridge ⑦ facilitates communication between the *Agent Client* and *Agent Server*, ensuring seamless emulation.

forwarding every single interrupt to QEMU, we only allow one through after every S counts. This approach, while allowing the PLC to continue its regular interrupt routine, significantly reduces the volume of interrupts communicated to QEMU.

4) Nested Interrupts: In managing nested interrupts within our system, it is essential to preserve the return address to ensure the system resumes correct execution post-interruption. Typically, when an interrupt occurs, the next instruction's address is stored in the link register `LR_IRQ`. However, the challenge in a nested interrupt scenario is that new interrupts can overwrite the `LR_IRQ`, risking the loss of the return point for the initial interrupt, which could compromise system stability.

To facilitate this process within our architecture, we have enhanced our agent with capabilities that not only permit the re-enabling of interrupts within the ISR but also secure the preservation of the current `LR_IRQ`. Before enabling new interrupts, the existing `LR_IRQ` is safely stored on the stack. This ensures that once a nested interrupt has been addressed and the ISR is set to return, the `LR_IRQ` can be retrieved from the stack, maintaining system stability and ensuring the correct continuation of operations post-interruption.

IV. System Design and Implementation

Portal[1] is a unique tool crafted for the rehosting of Siemens S7 series PLCs, emphasizing the precise handling of peripheral operations and effective management of interrupts within a RTOS. To validate our approach, we focused on Siemens S7-1200 and S7-200 SMART series PLCs, recognizing Siemens' status as a global leader in PLC technology with a significant 31% market share [17], [18].

A. System Overview

Our system architecture, as depicted in Figure 2, operates with the firmware code executed inside a modified emulator running on a traditional personal computer. In this setup, any I/O access is intercepted and forwarded to the physical Siemens PLC device, while signals and interrupts generated on the device are collected and injected back into the emulator.

The initial phase involves utilizing the MMIO Extraction tool (①) to analyze the firmware image, extracting rules for peripheral memory mapping (②). This delineates between general memory and MMIO registers, guiding the precise redirection of peripheral access operations to the PLC.

Within the QEMU emulator, the *Agent Client* (③) has two components: the vPeripheral and the Interrupt Querier. The vPeripheral (④) acts as a virtual representation of all peripheral devices extracted from the firmware. The Interrupt Querier (⑤) processes pending interrupt requests from the Siemens PLC, maintaining operational synchronicity.

The *Agent Server* (⑥), running on the Siemens PLC, receives, executes, and responds to peripheral memory operations. It also identifies and forwards any resulting interrupts to the emulator, ensuring a comprehensive emulation environment. Finally, the Signal Bridge (⑦) connects the *Agent Client* and *Agent Server*, enabling stable communication and ensuring the integrity of the emulation process.

B. Firmware and Bootloader Acquisition

Firmware acquisition is an initial step in re-hosting experiments, especially for devices like PLCs. Our research reveals that Siemens S7 PLCs commonly utilize the proprietary Adonis OS. To facilitate our study, we have pinpointed several methods to acquire the firmware of our target PLC.

One method involves downloading firmware files from the official Siemens website. These files are often compressed and can be made ready for use by applying the LZP3 [31], [32] decompression algorithm, yielding executable Adonis RTOS images.

In our hardware reverse engineering efforts, we discovered that the PLC's firmware resides on a NAND flash [30], [33], [34] on the device's board. By desoldering and interfacing this chip with a chip programmer, we can access its contents. The firmware within is organized in a distinctive format, with each block of firmware content of length 0x800 is followed by a block of 0x80 control information, including data size and CRC integrity checks.

[1]The *Portal* framework is open-source and available at https://github.com/ICSSECLAB/PLCRehosting.

Furthermore, the bootloader provides another avenue for firmware extraction. It is possible to tap into the bootloader's operations to copy the firmware from the main memory (RAM) post-initialization. This captured data, transmitted through a UART serial interface, can be compiled into a binary image.

Lastly, our goal to emulate the PLC system in its entirety requires us to consider the bootloader's role in initializing hardware peripherals. Booting directly from the firmware on uninitialized hardware is impractical; hence, we start from the bootloader, allowing it to set up the hardware before handing over control to the firmware. We have identified that the bootloader code is stored on a separate NOR flash chip [35] on the board, from which we can extract the necessary data using a chip programmer, starting from address 0x0.

C. Peripheral Memory Map Extraction

In order to redirect peripheral operations from a virtual setup to actual devices, it is necessary to distinguish between general-purpose memory and memory-mapped peripheral devices in the Siemens firmware. If we route all memory access to the actual device, it would drastically decelerate our execution process. On the flip side, treating all memory as RAM would result in the peripherals ceasing to function effectively. Thus, we need to accurately distinguish between RAM and peripherals.

Our reverse engineering efforts have identified three main elements within the firmware that are critical to map the device's memory. We discovered a structure in the firmware named `secinfo`. Additionally, we found a memory map table, `mmtab`, associated with specific peripherals, and examined the configuration process of the Memory Protection Unit (MPU) termed `mpucfg`, which outlines the properties of different memory regions.

Leveraging these insights, we developed an automated tool to extract memory regions associated with peripherals. This tool processes a raw Siemens firmware image and generates a detailed table of memory regions linked to peripheral devices. The key data structure has been reversed and identified. The findings presented at BlackHat [36] also encompass these memory regions. However, the research did not yield a distinguish of MMIO peripherals and general purpose RAM, nor was there any automated extraction process presented.

D. Agent Client

On the emulator side, our architecture includes the *Agent Client*, composed of two integral elements: the Virtual Peripheral Representation and the Interrupts Querier.

1) Virtual Peripheral Representation: Utilizing the QEMU Object Model (QOM) framework, we have developed representations for each peripheral, specifying their memory addresses and defining their behavior upon read and write operations.

2) Interrupt Querier: Our interrupt handling mechanism is based on proactive querying, utilizing two key strategies for interrupt management.

Firstly, we monitor changes in the CPSR register. Whenever a modification activates interrupts (by clearing the `I` bit), we immediately send commands to the *Agent Server* to check for pending IRQs. This approach leverages a function within QEMU's source code, located at `target/arm/helper.c`, named `cpsr_write`. This function allows us to monitor the `I` bit after any write operation to the CPSR. Secondly, we have implemented a mechanism to request IRQs after specific write operations to peripherals. If the server indicates an interrupt occurrence, we directly set the CPU's exception vector [37] to `5` (IRQ Exception) and exit the main CPU loop to handle the exception immediately. Additionally, we have observed that just before an ISR returns, a write operation is typically executed to a specific VIC register to signal the completion of IRQ handling. Recognizing this pattern, we send commands to the *Agent Server* after any write to this particular VIC register, indicating that the IRQ has been processed on QEMU's side. This synchronization ensures that both the emulator and the *Agent Server* manage interrupts in a coordinated and orderly fashion.

E. Agent Server

On the device, we have implemented a crucial component called the *Agent Server*. This server includes a communication module specifically for handling the exchange of command packets. It utilizes the PL011 [38] UART device, found in both the S7-1200 and S7-200 SMART PLCs, to manage data transmission and reception. This module ensures the integrity of the data with a simple CRC check mechanism. The *Agent Server* is adept at receiving instructions from QEMU, processing these commands, and executing the associated operations. The results are then promptly sent back to QEMU.

Integral to the *Agent Server* is its interrupt handling capability. We have redefined the IRQ exception vector at address 0x18. This alteration ensures that any incoming interrupt redirects the program counter to our custom interrupt handler. Within this handler, we have embedded the functionality to manage peripheral access during the ISR in QEMU, allowing us to address the interrupt directly, forward it to QEMU for handling, or simply ignore it. Furthermore, to manage nested interrupts, our system has the capability to re-enable interrupts during the current ISR, capturing and addressing new interrupts as they occur. Importantly, we preserve the state of the link register, ensuring the continuity of nested interrupt handling process.

F. Enhancing Emulation Efficiency through Peripheral Virtualization

To mitigate the significant performance degradation associated with routing peripheral access to physical devices, our research targeted the virtualization of key resource-intensive peripherals within the QEMU environment. This strategy was particularly applied to the Siemens S7 series PLCs, where an extensive analysis of 211 firmware versions revealed that NAND flash, VIC, and Timer are the primary frequent accessed peripherals contributing to operational delays, as detailed in Section V-C1. Addressing these bottlenecks improves emulation performance by reducing sources of lag.

In the process of NAND Flash Virtualization, we identified the NAND flash storage device as the most frequently accessed peripheral during the boot-up process. It is crucial for storing firmware codes, configuration files, and project data. The virtualization involved extracting the flash content through desoldering and read raw content by programmer, followed by reverse engineering how the firmware interact with the NAND flash. We have fully reversed the NAND flash MMIO registers and their functions. By simulating NAND flash access as a binary file, we simplified the data retrieval and writing operations within the emulator.

The operation of the Siemens S7 PLC as a multi-threaded system on a single-core processor necessitated a robust mechanism for thread switching. Our analysis indicated frequent interactions with ITIMER10 and portions of the VIC, which are essential for managing thread switching. By reversing and virtualizing the timer and implementing specific VIC-related interrupts within QEMU, we facilitated more efficient thread management. In this way, the burden of top-3 most frequent accessed peripherals is optimized, which brings a significant speed improvement. It is a trade-off that investing more time in reverse engineering and model the peripherals, system performance can be improved, as it reduces the time required to access peripherals.

Besides, we reversed and implemented the DMA data synchronization between QEMU and real device, where DMA is used by network module. This way, the system is interactive through ethernet port.

G. DMA Data Synchronization

In embedded systems like PLCs, efficient data transfer is crucial, often achieved through Direct Memory Access (DMA) used by Media Access Controllers (MACs) or Network Interface Cards (NICs). DMA streamlines the data flow between the MAC's buffers and the system memory, reducing CPU load and enhancing data throughput, which is vital for managing network traffic effectively.

Through reverse engineering, we have concluded that for network operations in our PLC system, DMA is integral to the functioning of the MAC, facilitating both packet transmission and reception. The process begins with the operating system preparing and storing a packet in a designated memory space. The OS then informs the MAC about the packet's location and size by writing to specific MMIO registers, triggering the MAC to transmit the packet over the network.

In our emulation setup, all memory content resides within QEMU. This poses a challenge when the emulated MAC attempts to access the memory for packet transmission, as it may encounter uninitialized data. To ensure data integrity, we synchronize the packet memory within QEMU with the device's memory each time the MAC's packet address and length registers are written to. This synchronization is managed by our agent system, ensuring correct data transfer from QEMU's internal RAM to the device's memory.

Similarly, for packet reception, the MAC stores incoming packets in predefined memory spaces, signaling the OS with an interrupt. However, since these packets are initially placed in the PLC's memory, we synchronize this memory from the device to QEMU based on the MAC's specifications. This reverse synchronization allows QEMU to correctly process the incoming packets.

Our reverse engineering has identified all registers involved in packet address and length management for both transmission and reception. The MAC is equipped with 32 transmission registers and 16 reception buffers, indicating its capability to handle multiple packets simultaneously. For instance, the Siemens 6ES7-215/217 [39], [40] series PLCs, featuring dual Ethernet ports, also have their MAC registers mapped in this manner. This detailed mapping and synchronization facilitate accurate network communication within our emulated PLC, which is essential for functional and interactive emulation.

V. Evaluation

This section presents a broad evaluation of our rehosting process for Siemens S7 series PLCs.

A. Experiment Setup

To emulate Siemens S7 series PLCs accurately, which rely on the 32-bit ARM Cortex-R4 processor and operate in big-endian mode, we have tailored a QEMU-based emulation environment. We chose `qemu-system-aarch64` for its compatibility with both ARM32 and ARM64 instructions and configured it to support big-endian operations.

Although the standard version of QEMU does not natively support the Cortex-R4, we utilized a version from Xilinx [41], specifically adapted for their boards. This version is targeted at the Cortex-R5 processor and effectively meets our requirements for emulating Siemens PLCs.

Our setup involves a PC host running QEMU and connected to actual Siemens PLCs. The host PC is equipped with a 1.4 GHz quad-core Intel Core i5 processor and 16GB RAM.

For connectivity with the S7-1200 series PLCs, we have identified the PLC's UART pin layout and used a Transistor-Transistor Logic to USB converter for communication, as depicted in Figure 3. In contrast, the S7-200 SMART series PLCs come with a built-in COM Port that supports the RS-485 protocol, allowing direct connection through an RS-485 to USB converter. This setup also includes convenient connections for flash chip programmers, enabling straightforward bootloader modifications when necessary.

In our experiments, we define the attempted rehosting scope as the execution span starting from the bootloader through to the successful initialization of the network stack and a valid response to an external ICMP request. All evaluation metrics presented in Table I and II, such as interrupt handling and peripheral access, are measured within this scope.

B. Dataset

We have gathered a broad range of firmware for the S7-1200 and S7-200 SMART series from the official Siemens website,

Peripheral	Size (KB)	Read Count	Write Count	Avg. TCP Time (ms)	Avg. Serial Time(ms)	Exec. Time (ms)
MAP_NAND_FLASH*	65,536	$46,380 \rightarrow 0$	$41,170 \rightarrow 0$	$2.03 \rightarrow 0$	$9.229 \rightarrow 0$	$0.00092 \rightarrow 0$
MAP3_VIC*	0.488	$22,123 \rightarrow 0$	$22,400 \rightarrow 0$	$2.05 \rightarrow 0$	$10.011 \rightarrow 0$	$0.00068 \rightarrow 0$
MAP3_ITIMER10*	0.016	$7,789 \rightarrow 0$	$19,444 \rightarrow 0$	$1.99 \rightarrow 0$	$10.435 \rightarrow 0$	$0.00077 \rightarrow 0$
MAP3_BOOL_HELPER	16	6,009	6,012	2.13	9.852	0.00068
MAP3_TIMERS	0.340	4,334	276	2.08	10.816	0.00077
MAP_MAC_MEM	1.145	2,196	1,266	1.98	11.332	0.00082
MAP3_I2C2	0.105	277	187	1.99	10.118	0.00077
MAP3_MAC	0.160	292	97	2.04	10.323	0.00068
MAP3_INPUTS	1.0	110	72	2.08	9.741	0.00078
MAP3_I2C0	0.105	84	48	1.98	10.108	0.00068

TABLE I: Top 10 Peripherals Operations in rehosting S7-1200 firmware v4.6.1

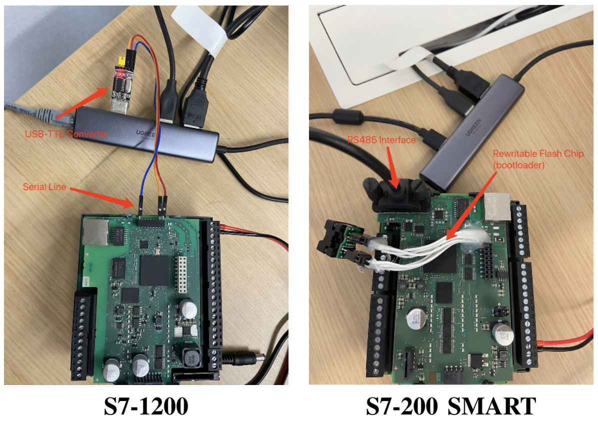

Fig. 3 Experiment Devices

encompassing releases from 2018 to the present. In total, we tested 211 firmware versions. After LZP3 decompression, the average firmware size is approximately 22MB for the S7-1200 and around 5MB for the S7-200 SMART series.

C. *Evaluating Peripheral Proxying Performance*

1) Peripheral Command Bridging and Execution Efficiency: Our assessment of peripheral command bridging and execution involved a thorough examination across 211 firmware versions. The results, particularly for the ten most interacted peripherals in the S7-1200's latest firmware version (v4.6.1), are encapsulated in Table I. This table provides a comprehensive view of the access frequency and timing efficiency for each peripheral during the firmware's boot-up phase.

The peripheral proxying approach involves two critical transmission stages: TCP and serial. The average time for TCP transmission per command was found to be around 2 milliseconds, while the serial transmission took approximately 10 milliseconds. The execution time for each command on the real device was observed to be about 0.0008 milliseconds. The analysis of these timings indicated that the transmission process, especially the serial phase, contributes significantly to the overall duration of firmware rehosting. This is because the UART protocol employed for communication with the PLC is constrained by limited bandwidth as the baud rate is set to 38,400.

2) Enhancing the Performance of Peripheral Routing: In accordance with the optimization techniques detailed in Section IV-F, our focus on virtualizing crucial, resource-demanding peripherals within QEMU was pivotal for improving peripheral routing performance. Our experiments revealed the NAND Flash as a particularly high-frequency access peripheral, crucial for loading the system configuration files necessary for the firmware's initial boot-up process. Likewise, the VIC and ITIMER10 were identified for their extensive access due to their roles in facilitating frequent thread switching. Following the virtualization of these peripherals, we observed an immediate reduction in their read and write counts to zero, as demonstrated in Table I, indicating that their processing had been seamlessly integrated within QEMU. This precise virtualization approach markedly decreased the firmware rehosting time from an average of 20 minutes to just under 3 minutes, signifying a performance boost of approximately 85%.

D. *Assessing Interrupt Synchronization Efficacy*

Our analysis of the interrupt synchronization process across 211 firmware versions focused on various dimensions of interrupt handling during the rehosting of these firmware versions. Our findings revealed an evolving pattern in the frequency of interrupts across firmware versions. In early S7-1200 versions, the average count was around 5637 interrupts, which escalated to a peak of approximately 8190 in later versions, before reducing to about 4114. This pattern suggests a refinement in Siemens firmware over time, likely indicating optimization efforts by the manufacturer. Similarly, nested interrupts exhibited a corresponding increase and subsequent decrease, paralleling the overall interrupt trend.

A crucial aspect of our method involved strategically bypassing a significant number of interrupts. This decision was based on the recognition that many interrupts in the rehosting process were repetitive and non-essential. Our experiments confirmed the effectiveness of this approach, as it contributed to a more efficient rehosting process by reducing unnecessary interrupt handling.

Furthermore, our method's ability to adjust timers was a key factor in managing interrupt frequencies effectively. By scaling up the duration of countdown timers, we effectively reduced the frequency of interrupts, allowing the main program to run with fewer interruptions. This approach, detailed in Section III-C3, not only enhanced the emulation performance but also maintained the operational integrity of the rehosted firmware.

PLC Series	Firmware Version	Download Images	Instructions Count	Peripheral Access Count	Interrupts Count	TCP Respond Time (ms)	Success Rate of Interaction
Siemens S7-200	v2.4.1	8	120,035	$88,334 \rightarrow 11,454$	1,433	2,165	100%
	v2.5.0	8	114,054	$84,669 \rightarrow 10,829$	1,187	1,933	100%
	v2.5.1	8	114,068	$86,442 \rightarrow 12,498$	1,339	2,802	100%
	v2.7.0	8	126,459	$92,758 \rightarrow 12,012$	1,313	1,696	100%
	v2.8.0	8	131,023	$97,981 \rightarrow 12,619$	1,754	2,189	100%
Siemens S7-1200	v4.2.3	19	364,204	$268,469 \rightarrow 35,196$	5,637	17,076	100%
	v4.3.1	19	365,983	$238,915 \rightarrow 30,748$	8,185	17,225	100%
	v4.4.0	19	403,858	$234,549 \rightarrow 30,421$	8,060	27,441	100%
	v4.4.1	19	404,100	$231,953 \rightarrow 29,876$	8,190	23,093	100%
	v4.5.0	19	425,693	$179,762 \rightarrow 23,171$	4,017	22,990	100%
	v4.5.1	19	425,651	$178,403 \rightarrow 22,960$	4,086	27,857	100%
	v4.5.2	19	425,620	$178,504 \rightarrow 23,205$	4,050	20,975	100%
	v4.6.0	19	425,427	$184,184 \rightarrow 23,878$	4,159	24,117	100%
	v4.6.1	19	425,475	$183,035 \rightarrow 23,729$	4,114	26,665	100%

TABLE II: Firmware Rehosting Results

E. Validating Siemens PLC Rehosting Success

To assess the effectiveness of our rehosting approach, we conducted evaluations on a total of 211 firmware images from the Siemens S7 series.

In Siemens Adonis RTOS, the `.th_initial` segment corresponds to the OS initialization process. Through reverse engineering, we determined this process to be the primary RTOS initialization routine. During this phase, an LED blinks indicating boot progression. Upon completion, the LED switches to steady yellow signaling entry into CPU STOP mode – the state where the CPU becomes ready to receive instructions from TIA Portal for executing ladder logic.

We therefore designate this transition as the successful rehosting benchmark. Successful rehosting is confirmed upon meeting all following criteria: 1) Complete execution of the `.th_initial` initialization segment. 2) Physical device LED status stabilizes to yellow. 3) Device broadcasts identity via Link Layer Discovery Protocol (LLDP) packets. 4) Device responds to external ICMP echo requests.

These four criteria also validate the successful operation of IRQ and DMA synchronization. Specifically, the complete execution of the `.th_initial` thread, as opposed to branching to the `adonis_fail` error handler, indicates that IRQ synchronization is functioning correctly. Similarly, the device's ability to successfully respond to network packets signifies the proper operation of DMA synchronization.

The results, detailed in Table II, showcase the capability of our system across different firmware versions.

S7-1200 Series: The dataset for the S7-1200 series includes 9 distinct firmware versions, spanning from v4.2.3 to v4.6.1. There are 19 different hardware-specific firmware images for each version. This range represents the series' evolution from 2018 to the present. The number of executed distinct instructions in each firmware varied between approximately 360,000 to 420,000. Before optimization, the count of peripheral operations ranged from 180,000 to 260,000. After optimization, these figures dropped to between 20,000 and 30,000, marking a approximately 90% improvement. Besides, the number of interrupts is in the vicinity of 4,000 to 5,000. Our experiments successfully rehosted all of these firmware versions. The criterion for successful rehosting was the ability to boot the firmware from the booting phase without errors and reach the 'CPU Stop' phase, indicating readiness for external commands and the establishment of a TCP connection with the host machine. The time taken to establish TCP communication varied between 17 to 26 seconds.

S7-200 SMART Series: The collection for the S7-200 SMART series covered 16 different models from the SR and ST series. We did not include the CR series due to the absence of corresponding real devices. There are 40 firmware images in total, ranging from v2.4.1 to v2.8.0. Each rehosting experiment in this series generally executed around 12 million distinct instructions. After optimization, there are approximately 12,000 peripheral accesses and about 1,400 interrupts. The TCP communication establishment time for this series was consistently around 2.2 seconds.

F. Fuzzing and Vulnerability Analysis

In our research with the *Portal* framework, we encountered performance constraints due to the necessity of redirecting peripheral access to the actual PLC. This redirection inevitably slows down the process, making it inefficient for high-throughput coverage based fuzzing. To address this, we devised a method involving rapid black-box fuzzing directly on the PLC, followed by a detailed white-box reproduction of any crashes within *Portal*. This approach combines the speed of black-box fuzzing on actual hardware with the detailed analysis capabilities of fully transparent *Portal* virtual environment, enabling effective debugging, vulnerability analysis, exploitation, and mitigation.

1) Black-box Fuzzing on the Real PLC: The input for black-box fuzzing on the real PLC comes via its Ethernet port. The PLC receives data from the TIA PORTAL V17 [42] control machine, which uses the encrypted S7-commplus TLS protocol. To fuzz this encrypted data, we reversed the TIA PORTAL to understand and decrypt the protocol. We then extracted plaintext packets from the memory before they were encrypted by the TLS layer. These plaintext packets, corresponding to various operations executed through TIA PORTAL, served as seeds for our fuzzing inputs. To inject these seeds back into the PLC, we developed a proxy [43] that adds TPKT [44] and COTP [45] headers beneath the TLS layer, enabling the replay of collected plaintext S7 packets. Using the S7-commplus Wireshark dissector [46], we parsed the protocol and understood the role of each field. Our S7CommPlus fuzzing employs protocol-aware mutation by preserving critical headers and session IDs while strategically targeting vulnerable fields like function codes, PLC memory addresses, and industrial data types. We combine byte-level flipping with smart dictionary attacks using known PLC command patterns and edge-case values to probe security boundaries while maintaining valid packet structure.

To detect crashes, we injected software breakpoints into the real PLC and patched the firmware's integrity checks. This setup allowed us to detect when the PLC entered a failure state, log the relevant packets, and immediately reboot the PLC, all without triggering Siemens' anti-debugging mechanisms.

2) Vulnerability Analysis on Portal: Following a crash on the real PLC, we replicated the input within our *Portal* framework to reproduce the crash under controlled conditions. *Portal*'s perfect emulation of PLC operations allows for an exact replication and detailed analysis of the crash.

To demonstrate *Portal*'s capability for firmware analysis, we applied it to dissect the parsing logic of network protocols within the firmware, using ARP [47] as a case study. After initializing *Portal* and observing the status LEDs on the device, we sent an ARP packet to the PLC's Ethernet port. The DMA synchronization mechanism ensured that packets received and stored in the physical PLC RAM were also available in the virtualized environment. We then traced data flow within this memory segment to identify how specific data were processed by the firmware. This approach enabled us to locate and analyze the firmware function that parses ARP packets, improving our understanding of the firmware's operation.

Building upon *Portal*, we have developed a rapid black box protocol fuzzing system following by a white box root cause analysis system to assess the robustness of the S7-CommPlus protocol. Through our testing, two distinct vulnerabilities were identified: specifically crafted payloads sent to TCP port 102 can trigger denial-of-service conditions in target PLCs. Under such attacks, TIA Portal V17 becomes unable to establish new S7 connections with the affected PLC. The attacks circumvent all configured security measures, including a Level 4 password protection enabled on the CPU and necessitate a full PLC reboot to restore functionality.

Vulnerability 1: The latest implementation of the S7-CommPlus protocol utilizes TLS to secure communication channels. During our investigation, we discovered a vulnerability: if TLS Alert messages are transmitted concurrently with TLS Application Data, the target PLC enters a persistent fault state that prevents any new S7 sessions from being established thereafter. Although the TIA Portal retains the ability to scan and detect the compromised PLC and initiates TLS handshakes, all subsequent connection attempts are abruptly terminated by TIA Portal after the exchange of merely two packets. In this attack scenario, when the TIA Portal initiates the S7comm-plus handshake, it sends a request to the `CreateObject` function at sequence number 2 to create a server session. However, because the PLC's S7comm-plus stack has already crashed, it replies with a `ServiceMultiESNotSupported` error code. After two such failed retries, the TIA Portal aborts the connection attempt.

Vulnerability 2: Persistent exploitation of vulnerability 1 leads to a cascading failure. When attacks leveraging vulnerability 1 are repeated approximately 45 times or more, the PLC escalates to a severe denial-of-service state characterized by the complete collapse of TLS connectivity. In this degraded state, the PLC instantly resets any TCP connection upon receiving an `initSSL` packet from TIA Portal, rendering the establishment of new TLS connections fundamentally impossible.

We empirically validated these vulnerabilities on a SIMATIC S7-1200 CPU running firmware version 4.6.1. Furthermore, using *Portal*, we successfully replicated the attacks in an emulated white-box environment. Following the emulated RTOS boot, we delivered malicious packets via proxy, monitoring the PLC's internal state at the same time. Subsequent dynamic analysis with a QEMU-based GDB server pinpointed the root causes of the two issues: Vulnerability 1 was traced to the PLC's S7comm-plus protocol stack, while Vulnerability 2 was found within the TLS protocol stack's implementation. For security reasons, fully detailed disclosure will occur once these vulnerabilities have been formally acknowledged.

In the realm of Industrial Control Systems, the impact of a Denial of Service attack exceeds that in traditional IT environments. This is because the consequences are not confined to data or service unavailability, but can directly propagate to the physical world, potentially causing equipment damage, production halts, and even severe personnel safety incidents.

VI. Discussion

A. Scalability and Adaptability

To extend our framework to any other platform, we need to fulfill two aspects: hardware prerequisites and software configurations.

While the current implementation of *Portal* is tailored for the Siemens PLCs, our core architectural innovations are platform-agnostic. The success of rehosting hinges on two prerequisites: 1) achieving arbitrary code execution on the target hardware and 2) establishing robust communication between the virtual logic and physical I/O. In our work, we leverage the specific means to fulfill these requirements on Siemens platforms. In our experiment, code execution can be achieved by either reprogramming the bootloader's flash chip or exploiting the vulnerability as showcased by Abbasi et al. [36]. Communication is established using a serial line on the S7-1200 and an RS485 adapter for the S7-200 SMART. It provides the environment for code execution and the necessary real-time primitives, which we abstract for our interrupt handling, DMA synchronization, and timer techniques.

This dependency on Adonis RTOS is why migration between Adonis-based systems like the S7-1200 and S7-200 SMART is straightforward. Consequently, extending our method to any other PLC ecosystem, whether from Siemens or other manufacturers, primarily involves finding alternative ways to implement the required software configurations, including MMIO, interrupt and DMA parameters. Our core real-time management logic, which constitutes 80% of the framework, can be ported with minimal redesign. This positions our approach as a validated and extensible framework for emulating a wide range of commercial PLCs.

B. Future Work

As we continue to advance our research on PLC security, two main areas of future work have been identified to further enhance the capabilities and performance of our fuzzing and emulation methodologies.

To elevate the performance of our fuzzing efforts, we can employ binary instrumentation on the PLC firmware to gain insight into the device's execution flow. Specifically, this method involves injecting lightweight code at the entry point of each basic block to monitor its invocation. By tracking which basic blocks are executed during fuzzing operations, we can record fine-grained coverage information. This approach enables path-aware fuzzing, allowing us to understand which execution paths are being traversed. With this knowledge, we can intelligently guide the fuzzer to mutate inputs and explore under-tested code areas more thoroughly, thereby increasing the efficiency and comprehensiveness of our security testing.

Building on our work with the *Portal* framework, we will pioneer a new peripheral emulation technique by automatically learning Finite-State Machine (FSM) models from real-world interaction data. This state-centric methodology advances beyond the black-box pattern recognition used in prior work like Pretender [48] by constructing an interpretable model of a peripheral's internal operational logic. By training an AI on our extensive dataset of access records, we can infer the FSM that governs the hardware's behavior. Integrating this state-aware model into our virtualization platform will enable highly precise emulation of complex, multi-step peripheral interactions. The FSM should also be able to raise interrupts. This advancement will significantly improve the fidelity of our virtual PLC, reduce reliance on physical testbeds, and streamline the configuration of robust security testing environments.

VII. Related Work

PLC Security. Recent investigations into Siemens PLC systems have uncovered several vulnerabilities. Flaws in the S7CommPlus-V3 protocol could allow unauthorized control of S7-1500 systems, potentially leading to 'Stuxnet-like' attacks via rogue clients [49]. Moreover, hidden features at UART special access in the S7-1200 series have surfaced, granting attackers take control of these devices once they can physically access PLCs [6], [36]. It also offers valuable insights into the proprietary Siemens ADONIS operating system used in Siemens PLCs, along with techniques for reverse-engineering it. Significant vulnerabilities disclosed by Claroty (CVE-2020-15782) enable unauthorized remote attackers to manipulate protected memory areas [50]. Based on that, Team82 also developed a new, innovative method to extract heavily guarded, hardcoded, global private cryptographic keys embedded within the Siemens SIMATIC S7-1200/1500 PLC and TIA Portal product lines [51]. Issues within the ET200SP open controller's boot process could allow attackers to decrypt, boot their operating systems and access vital system files [52]. In 2023 blackhat [17], the Siemens S7-1500 Software Controller PLC has been reverse-engineered, revealing that despite the obfuscation of keys and modification of cryptographic primitives, the security level remains unchanged. Their analysis highlights the widespread yet deprecated communication protocol, and they release tools to assist in the reverse-engineering of this firmware, aiming to facilitate further research and reproducibility in the field. Recent study [53] examines the implementation of hardware Root-of-Trust (RoT) in embedded systems, highlighting vulnerabilities in the discrete RoT components of Siemens S7-1500 series PLCs. Their findings reveal that flawed RoT assumptions can allow malicious actors to spoof credentials and gain privileged system control without destructive actions.

These insights have informed the development of *Portal*. It serves as a virtualization platform tailored for Siemens PLCs, transforming these systems from black-box or grey-box entities into a white-box format. This transformation enables a more transparent examination of the PLCs' operations, setting the stage for more detailed and effective security assessments.

Firmware Rehosting. The challenge of rehosting PLC firmware primarily stems from its deep integration with specific hardware peripherals. Successful rehosting necessitates an accurate simulation of a broad array of hardware components. One prevalent strategy in rehosting firmware addresses the absence of peripheral models by reconstructing and integrating with known libraries, thus bypassing direct hardware interaction [4]–[7]. However, this method still demands considerable manual effort and in-depth knowledge of the target system to develop these abstract models. The development of hardware abstraction layer implementations for Adonis RTOS faces fundamental constraints due to three critical gaps: the absence of debugging symbols, undocumented peripheral specifications, and lack of industry-standard emulation for proprietary devices. These limitations manifest concretely in our attempts to interface with Siemens PLC hardware. Crucially, simulation environments like QEMU lack models for these specialized components; while QEMU supports standard ARM Cortex peripherals like generic timers and UARTs, it contains no implementation of Siemens-specific hardware such as the memory-protection unit in S7-1200 CPUs or the real-time industrial Ethernet controllers in S7-1500 systems.

Another common approach employs dynamic symbolic execution, creating suitable inputs or outputs for specific peripherals to maintain firmware execution continuity [8]–[12]. Yet, this method falls short in emulating the precise I/O behavior of peripherals. While it generates data conducive to ongoing code execution, it often fails to deliver the exact, nuanced responses required for authentic PLC firmware operation. For instance, consider the HSC [27] in PLCs, a critical component for high-speed pulse inputs used in tasks like precise motor position control. The output of HSC—whether it is `a`, `b`, or `c`—might allow the firmware to continue running, but often, only one of these outputs, say `b`, leads to the correct execution pathway.

A third approach to firmware rehosting is the redirection of peripheral operations to the actual hardware devices [13]–[16], [54]. This method was pioneered by the *Avatar* framework [14], which orchestrates the execution between an emulator and real hardware to enable dynamic analysis of embedded devices. Despite its pioneering status, the direct application of the *Avatar* framework to Siemens PLC rehosting is met with notable challenges. Our approach enhances interrupt handling compared to Avatar, particularly for Siemens PLC emulation. Whereas Avatar's interrupt forward mechanism is not directly applicable to Siemens PLCs, our solution provides support through robust interrupt querying, precise timing control, and fully functional nested interrupts. Furthermore, we achieve a critical breakthrough in DMA support. This essential functionality is fully implemented in *Portal*, enabling the high-fidelity interactive emulation that Avatar cannot deliver. Prospect [13] focuses more on unix environments and user space applications. Inception [16] relies on source code, which Siemens's proprietary firmware environment does not provide. Furthermore, Pretender [48] is unable to deliver the precise hardware value emulation required for accurate and interactive Siemens PLC operation, making it an inadequate solution. Sizzler [54] is a fuzzing framework for PLC ladder diagrams, focusing on user-level application code rather than the underlying firmware. The framework bypasses direct analysis of proprietary vendor firmware by converting ladder diagrams into C code and executing them on emulated MCUs. Furthermore, Sizzler's use of HIL framework is confined to simulating high-level network protocol communications like Modbus/TCP; it is not used for accessing low-level hardware peripherals and interrupts. Therefore, none of these prior solutions can be effectively experimented with or applied to Siemens PLC target.

VIII. Conclusion

This research introduces a framework designed for emulating Siemens PLCs, particularly focusing on the Siemens S7 series. The framework incorporates Peripheral Proxying and Proactive Interrupt Synchronization to address the complexities of emulation. It has been validated through the successful emulation of more than 200 firmware versions, illustrating its strong capacity to manage peripheral operations and interrupts effectively. This demonstrates the framework's practical applicability and robustness in replicating the nuanced functionalities of Siemens PLCs. Furthermore, the incorporation of a dual-phase fuzzing approach—combining real-world PLC testing with in-depth white-box analysis within the emulation environment—has enhanced the framework's ability to detect and analyze security vulnerabilities.

IX. Acknowledgement

We would like to thank anonymous shepherd and reviewers for their helpful comments and feedback. This work was supported by the National Key R&D Program of China under Grants No. 2024YFE0203800.

References

[1] Z. Wang, Y. Zhang, Y. Chen, H. Liu, B. Wang, and C. Wang, "A Survey on Programmable Logic Controller Vulnerabilities, Attacks, Detections, and Forensics," *Processes*, vol. 11, p. 918, 2023.

[2] W. Alsabbagh and P. Langendörfer, "Security of Programmable Logic Controllers and Related Systems: Today and Tomorrow," *IEEE Open Journal of the Industrial Electronics Society*, vol. 4, pp. 659–693, 2023.

[3] R. Sun, A. Mera, L. Lu, and D. Choffnes, "SoK: Attacks on Industrial Control Logic and Formal Verification-Based Defenses," in *2021 IEEE European Symposium on Security and Privacy*, 2021, pp. 385–402.

[4] A. A. Clements, E. Gustafson, T. Scharnowski, P. Grosen, D. Fritz, C. Kruegel, G. Vigna, S. Bagchi, and M. Payer, "HALucinator: firmware re-hosting through abstraction layer emulation," in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20. USA: USENIX Association, 2020.

[5] W. Li, L. Guan, J. Lin, J. Shi, and F. Li, "From Library Portability to Para-rehosting: Natively Executing Microcontroller Software on Commodity Hardware," in *Proceedings of the Network and Distributed System Security Symposium (NDSS 2021)*. Online: NDSS, Jan. 2021.

[6] P. Kovač, A. Pantina, S. Groš, and D. Sumina, "Development of Programmable Logic Controller Emulator With QEMU," in *Proceedings of the 20th International Conference on Smart Technologies (EUROCON 2023)*. TORINO, ITALY: IEEE, Aug. 2023, pp. 770–775.

[7] B. Feng, A. Mera, and L. Lu, "P2IM: Scalable and Hardware-independent Firmware Testing via Automatic Peripheral Interface Modeling," in *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*. Online: USENIX Association, Aug. 2020, pp. 1237–1254. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/feng

[8] W. Zhou, L. Guan, P. Liu, and Y. Zhang, "Automatic Firmware Emulation through Invalidity-guided Knowledge Inference," in *Proceedings of the 30th USENIX Security Symposium (USENIX 2021)*. Online: Usenix, Aug. 2021, pp. 226–236.

[9] C. Cao, L. Guan, J. Ming, and P. Liu, "Device-agnostic Firmware Execution is Possible: A Concolic Execution Approach for Peripheral Emulation," in *Proceedings of the 36th Annual Computer Security Applications Conference*, ser. ACSAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 746–759. [Online]. Available: https://doi.org/10.1145/3427228.3427280

[10] T. Scharnowski, N. Bars, M. Schloegel, E. Gustafson, M. Muench, G. Vigna, C. Kruegel, T. Holz, and A. Abbasi, "Fuzzware: Using Precise MMIO Modeling for Effective Firmware Fuzzing," in *Proceedings of the 31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 1239–1256. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/scharnowski

[11] L. Seidel, D. Maier, and M. Muench, "Forming Faster Firmware Fuzzers," in *Proceedings of the 32st USENIX Security Symposium (USENIX 2023)*. ANAHEIM, CA, USA: Usenix, Aug. 2023, pp. 2903–2920.

[12] E. Johnson, M. Bland, Y. Zhu, J. Mason, S. Checkoway, S. Savage, and K. Levchenko, "Jetset: Targeted Firmware Rehosting for Embedded Systems," in *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*. Online: USENIX Association, Aug. 2021, pp. 321–338. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/johnson

[13] M. Kammerstetter, C. Platzer, and W. Kastner, "Prospect: peripheral proxying supported embedded code testing," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 329–340. [Online]. Available: https://doi.org/10.1145/2590296.2590301

[14] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares," in *Proceedings of the Network and Distributed System Security Symposium (NDSS 2014)*. New York, NY, USA: ACM, Feb. 2014.

[15] S. M. S. Talebi, H. Tavakoli, H. Zhang, Z. Zhang, A. A. Sani, and Z. Qian, "Charm: Facilitating Dynamic Analysis of Device Drivers of Mobile Systems," in *Proceedings of the 27th USENIX Security Symposium (USENIX 2018)*. New York, NY, USA: Usenix, Aug. 2018.

[16] N. Corteggiani, G. Camurati, and A. Francillon, "Inception: System-Wide Security Testing of Real-World Embedded Systems Software," in *Proceedings of the 27th USENIX Security Symposium (USENIX 2018)*. New York, NY, USA: Usenix, Aug. 2018.

[17] C. Finck and T. Dohrmann, "A Decade After Stuxnet: How Siemens S7 is Still an Attacker's Heaven," in *Proceedings of the Blackhat 2023 Technical Briefing / whitepaper*. San Francisco, CA, USA: Blackhat, Dec. 2023.

[18] E. López-Morales, U. Planta, C. Rubio-Medrano, A. Abbasi, and A. A. Cardenas, "SoK: Security of Programmable Logic Controllers," 2024.

[19] P. N. e.V., "PROFINET," https://www.profinet.com/, 2024, [Online; accessed 30-Jan-2024].

[20] WIRESHARK, "S7 Communication (S7comm)," Jan 2024. [Online]. Available: https://wiki.wireshark.org/S7comm

[21] Arm Limited, "ARM Cortex-R4 Interrupts," Jan 2009. [Online]. Available: https://developer.arm.com/documentation/ddi0363/e/programmer-s-model/exceptions/interrupts

[22] ——, "ARM Cortex-R4 Program status registers," Jan 2009. [Online]. Available: https://developer.arm.com/documentation/ddi0363/e/programmer-s-model/program-status-registers

[23] T. Q. P. Developers, "The QEMU Object Model (QOM)," https://qemu-project.gitlab.io/qemu/devel/qom.html, 2024, [Online; accessed 18-Jan-2024].

[24] X. Inc., "Device Trees," https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/862421121/Device+Trees, 2024, [Online; accessed 18-Jan-2024].

[25] Wikipedia, "Universal asynchronous receiver-transmitter," Jan 2023. [Online]. Available: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

[26] A. Abbasi, "Siemens S7 PLCs Bootloader Arbitrary Code Execution Utility," https://github.com/RUB-SysSec/SiemensS7-Bootloader, 2024, [Online; accessed 18-Jan-2024].

[27] Siemens, "S7-1200: Application Examples for High-Speed Counters (HSC)," Jan 2016. [Online]. Available: https://support.industry.siemens.com/cs/document/109742346/s7-1200-application-examples-for-high-speed-counters-(hsc)?dti=0&lc=en-WW

[28] M. H. Rais, R. A. Awad, J. Lopez, and I. Ahmed, "JTAG-based PLC memory acquisition framework for industrial control systems," *Forensic Science International: Digital Investigation*, vol. 37, p. 301196, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281721001049

[29] T. Weber, "Reverse Engineering Architecture And Pinout of Custom Asics," https://sec-consult.com/blog/detail/reverse-engineering-architecture-pinout-plc/, 2024, [Online; accessed 18-Jan-2024].

[30] Wikipedia, "Flash Memory," Jan 2024. [Online]. Available: https://en.wikipedia.org/wiki/Flash_memory

[31] C. Bloom, "LZP: A new data compression algorithm," in *Proceedings of the 6th Data Compression Conference (DCC '96), Snowbird, Utah, USA, March 31 - April 3, 1996*, J. A. Storer and M. Cohn, Eds. Snowbird, Utah: IEEE Computer Society, 1996, p. 425. [Online]. Available: https://doi.org/10.1109/DCC.1996.488353

[32] jibeee, "S7 firmware unpck algorithm," https://github.com/jibeee/s7unpack, 2024, [Online; accessed 24-Jan-2024].

[33] Micron, "MT29F1G08ABBFAH4-ITE NAND Flash chip," Jan 2021. [Online]. Available: https://www.micron.com/products/nand-flash/slc-nand/part-catalog/mt29f1g08abbfah4-ite

[34] Y. Wu, G. Skipper, and A. Cui, "Cryo-Mechanical RAM Content Extraction Against Modern Embedded Systems," in *17th IEEE Workshop on Offensive Technologies (WOOT '23), co-located to IEEE Symposium on Security and Privacy*, 2023.

[35] ISSI, "Data Sheet: 4M/2M/1M/512K/256KBIT 3V QUAD SERIAL FLASH MEMORY WITH MULTI-I/O SPI," Jan 2021. [Online]. Available: https://www.issi.com/WW/pdf/25LQ025B-512B-010B-020B-040B.pdf

[36] A. Abbasi, T. Scharnowski, and T. Holz, "Doors of Durin: The Veiled Gate to Siemens S7 Silicon," in *Proceedings of the Blackhat 2019 Technical Briefing / whitepaper*. San Francisco, CA, USA: Blackhat, Aug. 2019.

[37] Arm Limited, "ARM Cortex-R4 Exception Vecotr," Jan 2009. [Online]. Available: https://developer.arm.com/documentation/ddi0363/e/programmer-s-model/exceptions/exception-vectors

[38] ——, "PrimeCell UART (PL011) Technical Reference Manual," https://developer.arm.com/documentation/ddi0183/latest/, 2024, [Online; accessed 18-Jan-2024].

[39] Siemens, "Industry Mall: 6ES7215-1AG40-0XB0," Jan 2024. [Online]. Available: https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7215-1AG40-0XB0

[40] ——, "Industry Mall: 6ES7217-1AG40-0XB0," Jan 2024. [Online]. Available: https://mall.industry.siemens.com/mall/en/ww/Catalog/Product/?mlfb=6ES7217-1AG40-0XB0

[41] Xilinx, "xilinx/qemu," https://github.com/Xilinx/qemu, 2024, [Online; accessed 24-Jan-2024].

[42] Siemens, "Totally Integrated Automation Portal – Always ready for tomorrow," https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html, 2024, [Online; accessed 9-Jan-2024].

[43] G. Jian, "Fuzzing and Breaking Security Functions of SIMATIC PLCs," https://i.blackhat.com/EU-22/Thursday-Briefings/EU-22-Jian-Fuzzing-and-Breaking-Security-Functions-of-SIMATIC-PLCs.pdf, 2024, [Online; accessed 9-Jan-2024].

[44] Wireshark, "ISO transport services on top of the TCP (TPKT)," Jan 2020. [Online]. Available: https://wiki.wireshark.org/TPKT

[45] ——, "Connection Oriented Transport Protocol (COTP, ISO 8073)," Jan 2020. [Online]. Available: https://wiki.wireshark.org/COTP

[46] T. Wiens, "S7comm Wireshark dissector plugin," https://sourceforge.net/projects/s7commwireshark/, 2024, [Online; accessed 9-Jan-2024].

[47] Wireshark, "Address Resolution Protocol (ARP)," Jan 2020. [Online]. Available: https://wiki.wireshark.org/AddressResolutionProtocol

[48] E. Gustafson, M. Muench, C. Spensky, N. Redini, A. Machiry, Y. Fratantonio, D. Balzarotti, A. Francillon, Y. R. Choe, C. Kruegel, and G. Vigna, "Toward the Analysis of Embedded Firmware through Automated Re-hosting," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. Chaoyang District, Beijing: USENIX Association, Sep. 2019, pp. 135–150. [Online]. Available: https://www.usenix.org/conference/raid2019/presentation/gustafson

[49] E. Biham, S. Bitan, A. Carmel, A. Dankner, U. Malin, and A. Wool, "Rogue7: Rogue Engineering-Station attacks on S7 Simatic PLCs," in *Proceedings of the Blackhat 2019 Technical Briefing / whitepaper*. San Francisco, CA, USA: Blackhat, Aug. 2019.

[50] T. Keren, "The Race to Native Code Execution in PLCs," https://claroty.com/team82/research/the-race-to-native-code-execution-in-plcs, 2021, [Online; accessed 22-Jan-2024].

[51] Team82, "The Race to Native Code Execution in PLCs: Using RCE to Uncover Siemens SIMATIC S7-1200/1500 Hardcoded Cryptographic Keys," Jan 2022. [Online]. Available: https://claroty.com/team82/research/the-race-to-native-code-execution-in-plcs-using-rce-to-uncover-siemens-simatic-s7-1200-1500-hardcoded-cryptographic-keys

[52] S. Bitan and A. Dankner, "sOfT7: Revealing the Secrets of the Siemens S7 PLCs," in *Proceedings of the Blackhat 2022 Technical Briefing / whitepaper*. San Francisco, CA, USA: Blackhat, Aug. 2022.

[53] Y. Wu, G. Skipper, and A. Cui, "Uprooting Trust: Learnings from an Unpatchable Hardware Root-of-Trust Vulnerability in Siemens S7-1500 PLCs," in *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 05 2023, pp. 179–190.

[54] K. Feng, M. M. Cook, and A. K. Marnerides, "Sizzler: Sequential Fuzzing in Ladder Diagrams for Vulnerability Detection and Discovery in Programmable Logic Controllers," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 1660–1671, 2024.