Control Corruption without Firmware Infection: Stealthy Supply Chain Attacks via PLC Hardware Implants (MalTag)

Mingbo Zhang
Department of ECE
Rutgers University
mingbo.zhang@rutgers.edu

Saman Zonouz
Schools of SCP and ECE
Georgia Tech
saman.zonouz@gatech.edu

***Abstract*— Critical infrastructures, e.g., power grids, are vital to national security, and their failure would have a significant impact on people's daily lives on a large scale. They are often automated and computer-controlled, and are under emerging advanced persistent threat (APT) attacks. The programmable logic controllers (PLCs) are the neurons that control the physical system. In most APT attacks, usually, a stealthy backdoor is the core that allows the attacker to hide in the dark without being detected and launch remote malicious operations at a particular moment. However, to achieve further stealthiness and bypass existing software mitigations, it needs to evolve from high-level software into low-level hardware.**

This paper presents MALTAG, a small parasitical hardware implant that attaches to the PLC's circuit board. Using MALTAG, the attacker can control the PLC remotely by hijacking the various buses on the boards and modifying the digital signal. This attack can be deployed either during the supply chain or stealthily installed in remote plants. The hardware implant contains a cellular chip that provides a remote control channel to allow the attacker to organize a multi-point distributed attack by controlling several PLCs simultaneously on an interconnected physical plant. We have implemented and evaluated MALTAG on popular and widely deployed Allen Bradley PLCs. The results show that such a hardware backdoor does not change the firmware, thus no integrity violation. MALTAG also induces almost no overhead to the system, thus not affecting the runtime of the PLC. It can secretly change the PLC's outputs to actuators and/or inputs from sensors without leaving any trace. Furthermore, the attacker can even penetrate air-gapped networks communicating with MALTAG and conduct a simultaneous attack with multiple controlled nodes.

I. INTRODUCTION

Automation and computer-controlling of critical infrastructures, such as the power grids, brings security concerns. The recent Ukrainian power grid attack, and the massive blackout in south American countries [1] demonstrated that influence not only affects people's livelihoods but even international politics. The cyber attack on industrial control systems (ICS) can even cause physical damage to the infrastructure [2], which makes it harder to recover. Incidents such as the Stuxnet proved this point.

Consequently, ICS has received considerable attention due to security concerns. There are many ways to breach a computer system, and most of them are focus on software-based approaches such as vulnerability hunting and exploiting, cracking of authentication and protocols. Therefore, the attacker needs to break into the network and avoid existing access control and other mitigations. Moreover, most critical infrastructures use their air-gapped network, and it may take a state-sponsored team to accomplish such as mission [3].

Under such circumstances, we believe that cyber-attacks with the assistance of physical approaches are underestimated, especially before the emergence of the so-called supply-chain attacks. Among those world-class attacks, the advanced persistent threat (APT) is often mentioned. In essence, the APT attack is an extremely well-hidden trojan that can be deployed for many years without being detected. Thus, installing the trojan and remotely triggering it is the crucial point of a successful attack.

The ICS interconnects and controls the physical production assets. Compared to traditional IT infrastructures, the physical assets and the computer-based network's interconnections are unique ICS features. These interconnections are managed by embedded systems known as the programmable logic controller (PLC). Since the PLCs are the core controllers for ICS, In recent years, several worldwide incidents, such as Triton and Stuxnet [3] have targeted PLCs to attack the ICS and sabotage physical facilities. Several other attacks, such as BlackEnergy [4] targeted power grid critical infrastructures operated by ICS. Since the power grids are more distributed and connected, any distributed coordinated attack on such infrastructure will devastate results.

With the continuous emergence of such attacks, protection measures have also been strengthened. ICS security has been traditionally handled using network security practices such as access control. However, recent works have shown that such a traditional access control alone is not sufficient to prevent such attacks [5], [6]. An overall strategy is to use an isolated network from the Internet. However, this is not sufficient since several attacks penetrated the air-gapped network [7], [3], [8]. Most of these defense mechanisms are driven by the NERC-CIP standards. However, several attacks targeting power grids have shown these regulations are not sufficient for a novel

979-8-3503-6927-4/24/$31.00 ©2024 IEEE
DOI 10.1109/ICCPS61052.2024.00029

class of attacks [9].

The core of an APT attack is essentially a trojan backdoor that gains continued access to a computer network and remains undetected for an extended period. The most crucial feature of a trojan is its stealthiness. Stay undetected over extended periods within the device is an essential issue that sophisticated APT attacks should consider. Therefore, to achieve it, several works have utilized firmware modification techniques [9]. They inject malicious code into the target PLC, changing the working logic that runs in the device. However, such attacks are subjected to the firmware integrity verification [10] and update authentication [11] method. It would be much harder for the attacker to implant the malicious code once the firmware update is encrypted and digitally signed and the system applies the methods mentioned above.

Another critical point for the APT attack is choosing the trigger event. Usually, the PLC has a dedicated real-time microcontroller to control the physical world through its I/O pins. However, the microcontroller does not directly communicate with the host, the central control terminal (human-machine interface, HMI). Therefore, firmware modification attacks can perform a preset task individually, but it is difficult to react to PLC firmware updates or coordinate a distributed attack with other controlled nodes, especially among air-gapped networks.

Therefore, we propose an alternative approach to circumvent such existing software mitigations, MALTAG, a parasitical hardware implant inside a PLC attaching to its circuit board and remotely controlled through GSM network. With the recent emerging concept of supply chain attacks and real-world incidents [12], [13], such a hardware attack appears to be more practical. The hardware implant can be pre-installed during the PLC device's assembly line or even during the shipment [14]. Such implants can be stealthily installed on the site due to their vast distribution and low physical security throughout the power grid [15].

We design it to be flexible because the PLC will load operating logic only after being deployed, and it is plausible that the ICS updates PLC's operating logic frequently. The hardware implant is specialized for the device's circuit board, the microcontroller, and other chips. It controls the I/O through the digital signal and bus-level protocol hijacking, independent of the PLC's firmware.

Memory bus and interconnect protocols such as SPI, I2C are all potential targets. Low-speed protocols are prevalent due to their simplicity. For instance, joint test action group (JTAG) is an industry standard for verifying designs and testing integrated circuits (IC) after manufacturing. Extensive hardware features are also provided on the ARM microcontroller through this interface for system-level debugging and tracing. It can read/write registers of processor and memory during system runtime. We leverage this interface for I/O controlling purposes, and we can fetch the PLC's firmware and operating logic for further offline analysis. Furthermore, to communicate through a cellular GSM network, it is practical to control multiple nodes and organize a distributed attack simultaneously.

Contributions. This paper makes the following contributions: *i)* We present a novel attack class on industrial control systems: a parasitical hardware implant, which is completely invisible to the ICS control system. *ii)* We disassembled and reverse engineered the circuit boards of a widely deployed Allen Bradley 1769-L18ER-BBIB CompactLogix 5370 PLC. *iii)* We develop a prototype implementation of MALTAG, which is a small size device installed inside the Allen Bradley PLC. We write a JTAG driver that runs bare-metally on a microcontroller with minimal resource usage, and test and evaluate MALTAG with multiple controlled Allen Bradley PLCs.

The rest of this paper is organized as follows. Section II describes the objectives, adversary model and scope, challenges, and architecture of MALTAG. Section III provides the necessary background. Section IV describes how we reverse-engineered the Allen Bradley PLC and prototyped MALTAG with implementation details (Section V) and evaluation and mitigation (Section VI), respectively. Section VII provides a review of related work. Finally, Section VIII concludes the paper.

II. OVERVIEW

We present MALTAG, an APT using a hardware implant that an attacker uses to perform coordinated distributed attacks on critical infrastructures. To achieve this, we reversed engineered a PLC, developed a prototype of a hardware implant, and showed that such attacks could be performed even without state-sponsored attack groups. MALTAG hijacks the data lines on the PLC and modifies them based on the command received by the attacker. Since the attacker can send such signals remotely, they can control various PLCs at different locations simultaneously for a coordinated distributed attack on the entire physical process such as a power grid.

MALTAG can be implanted into the PLC in two ways, including installing it during the supply chain or after the PLC has been deployed. These two scenarios are depicted in Figure 1. At first, during the supply chain, such as the factory and shipment, numerous employees can access the PLC. Installing an extra piece of a circuit board is not a difficult task for professionals [16], [17]. Secondly, after the PLCs are being deployed, the power grid's vast distribution network lacks strong physical security [15]. An attacker can sneak into one of the substations and install the hardware backdoor. The difficulty between the targeted PLC and the attacker, in reality, can be just a few padlocks. We believe that a substation's breach can cause a chain reaction in a power grid, and it is a real threat [18], [19].

After the attacker controls enough PLCs, he can remotely initiate a distributed attack to cause more significant damage to the critical infrastructure using a cellular network. The advantage of this attack is that it does not rely on the existing ICS network, nor is it limited to the firmware running on PLCs so that it can evade most software-based mitigations. It only needs to know the specific PLC model of the target and modify the hardware implant accordingly.

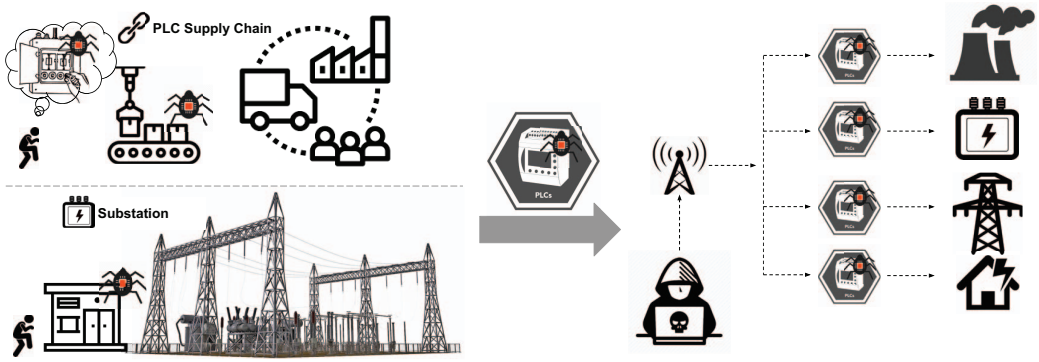

Fig. 1: In the PLC manufacturer factory or during the product shipment, numerous employees can access the PLCs. The hardware backdoor installation is accomplished without advanced software knowledge. The hardware backdoor can communicate with the attacker through the GSM network. Therefore it does not need to join the ethernet used by the ICS system.

A. Adversary Model

We assume that the attacker has physical access to the PLC once to implant the MALTAG. Many studies and reports have shown that those attacks are not superficial and have been done in the real world during manufacturing or shipment [16], [17], [14]. Since the attack is aware of the PLC model being targeted, we assume that the attacker knows the PLC's internal. The assumption could be made since the attacker can purchase a replica of PLC from any vendor. The printed circuit board (PCB) and the IC chips of the PLC should be well exposed. For example, the Chip-on-Board (COB) [20] packing brings extra challenges for the attacker. The black glob-top makes it challenging to identify the chip model and pins. Fortunately, high-end microcontroller products rarely use this packaging. It would be a great advantage if the attacker can use the JTAG interface of the microcontroller. In other words, the JTAG interface is not disabled by programming fusing bits at the factory. Nevertheless, the attacker can control the I/O or tamper with the firmware image when transferred through the bus without JTAG.

MALTAG is less invasive than firmware corruptions. There are no modifications to the operating system, the system software, or software mitigation solutions that run on a microcontroller. Hence, MALTAG is undetected by such software mitigation solutions. The PLC can have any memory protection mechanisms based on MMU [21] and MPU [22]. To remotely control the device, the attacker uses a GSM network or WiFi to communicate with the hardware backdoor. The PLC must not be in an electromagnetic isolation environment (Faraday cage) where the wireless signal can not be transmitted outside. The assumptions are reasonable since many communications in such substations happen through wireless communications used by remote terminal units (RTUs).

B. Challenges

The major challenge in attacking PLCs is lack of information about the device. Some vendors publish the microcontroller's datasheet, but some use proprietary design with highly customized instruction set architecture (ISA). The layout of the PCB board and the on-board pin definition are often not available publicly.

`Firmware.` In an embedded system, flash memory usually stores a file system and a real-time operating system (RTOS) such as VxWorks [23]. It is the so-called firmware. Specific to a real-time microcontroller, the firmware runs bare-metally on the microcontroller or with a lightweight RTOS such as FreeRTOS [24]. Allen Bradley 1769-L18ER-BBIB CompactLogix 5370 has minimal information about the firmware being used.

`Identifying Components on Board.` Due to the lack of information and many proprietary chips used by such vendors, it is difficult to identify the chips used on the PLC board. Some of the chips have BGA packaging [25] where the pins are buried underneath. So it is difficult to trace the circuit board and identify the connections and pins such as JTAG.

`Power Trace.` The main challenge of developing such an implant is power consumption. For reducing the footprint, a small chip had to be used that contains all the tools that could be used for hijacking, information retrieval, and modification of the signals. Not many tools are present for such a bare metal implant, and they have to be ported to work on a minimal computing base.

III. BACKGROUND

We provide background knowledge for the rest of the paper.

`ICS` is a distributed system used for industrial process control. It connects sensors and actuators that interact with the physical systems (e.g., power grid) with the cyber components such as networks and servers. In a factory, local operations are often controlled by PLCs that receive supervisory commands from a remote host. For example, a human operator monitors the system's state and sends out instructions through a human-machine interface (HMI). Most PLCs and HMI hosts are connected to the ICS via Ethernet.

`JTAG` is an IEEE1149.1 standard, which is used for testing printed circuit boards using boundary-scan. The JTAG interface uses very few pins (TDI, TDO, TMS, TCK, and

TRST) to connect to an on-chip test access port (TAP) that implements a stateful protocol, as shown in Figure 2. One or more devices can expose multiple TAPs in a daisy chain, also known as a scan chain. The host communicates with the TAPs by manipulating TMS and TDI in conjunction with TCK and reading results through TDO. The JTAG standard has four common registers: instruction register (IR) and data register (DR), IDCODE, and BYPASS. The IDCODE register contains data that uses a standardized format that includes a manufacturer code. The BYPASS register is a single-bit data register that allows this device to be bypassed (do nothing) while other devices in the scan chain are examined. The IR and DR register's size depends on the TAP implementation, and they are used to send in instruction and receive result data.

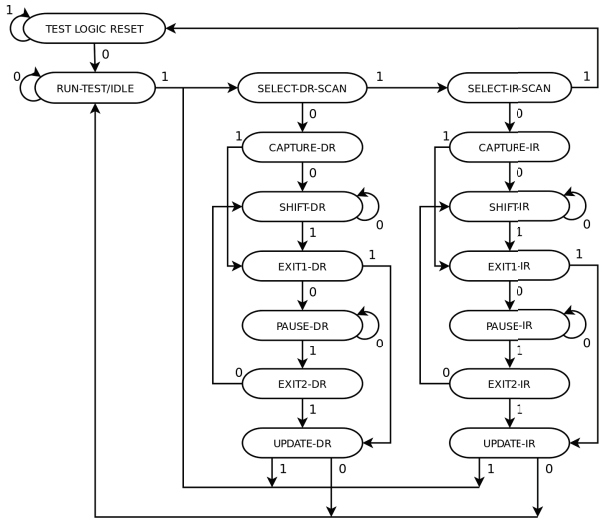

Fig. 2: JTAG TAP state machine.

The TAP implementation defines instructions and associated them with internal data registers. For instance, the host sends the IDCODE instruction through IR and subsequently gets the value of a 32-bit register (IDCODE) from TDO. The PLC we used in this paper uses an ARM core microcontroller, namely, Texas Instruments Stellaris LM3S2793 [26]. The debug functionality provided in LM3S2793 is as CoreSight components. It provides real-time access for the debugger without halting the processor to advanced microcontroller bus architecture (AMBA) [27] system memory, peripheral registers, and all debug configuration registers. The TAP controller is implemented using CoreSight technologies, and it is called debug access port (DAP) instead. Each DAP contains debug ports (DPs) and access ports (APs). The DP provides access to the DAP from an external debugger. Then the DAP uses the APs to access on-chip resources. Multiple APs such as AHP-AP, ABP-AP, and JTAG-AP respond to each type of bus and the devices that connect to it. For instance, the AHB-AP provides an AHB-Lite master for accessing the system AHB bus, which we use to access the RAM and MMIO.

IV. PLC Reverse Engineering

The major challenge in making a PLC-specific hardware backdoor is that the PLCs' standards are not that well-known like that of general computers. During the development of MalTag, a significant amount of time was spent on the reverse engineering of the PLC to understand the modules and ICs used. This information is critical for controlling the PLC by intercepting low-speed buses. We present the reverse-engineering of the firmware in this section.

a) ***Startup Process****:* The reset vector table is at a fixed address 0x00000000 after the system reset. The core starts to execute from memory 0x00000004, which is the reset vector. The reset vector resides in the flash memory instead of the ROM. The ROM boot loader is only executed in two scenarios: *i)* when the flash memory is empty. *ii)* when an application initiates a firmware update and calls the ROM boot loader to execute. For instance, if data at 0x00000004 is 0xFFFFFFFF, which indicates an empty flash, then the ROM is mapped to 0x00000000 to substitute the flash and execute instead. The data at 0x00000000 and 0x00000004 will be loaded into the stack pointer (SP) and the program counter (PC), respectively. In our case, the value of SP is 0x20000B48, and the value of PC is 0x000000E3. Notice that 0xE3 is an odd number. As we know, RISC processors such as ARM use fixed-length instructions. On ARM processors, setting the PC's least significant bit indicates that the following code will be executed as the two-byte Thumb instructions. Therefore, we dump the flash memory and disassemble it at address 0x000000E2 instead.

Right after the flash boot loader starts, it copies itself to SRAM, starting at 0x20000000. Although both SRAM and flash memory can be accessed in a single cycle, flash memory can do that as long as the code is linear and branches incur a one-cycle stall. The bootloader copies 0x00000000 - 0x00000A88 to 0x20000000 - 0x20000A88, and clear the data section (0x20000A88 - 0x20000F54). As mentioned earlier, on system reset, the vector table is at the fixed address 0x00000000. However, it can be relocated by writing the vector table offset register (VTOR:0xE000ED08). Changing the vector table indicates a complete change of the system's behavior because all the interrupt handlers are new, and they interpret how the system behaves regarding peripheral device's requests. The bootloader sets the vector table to 0x20000000 and jumps back to SRAM to continue execution.

b) ***Stellaris Peripheral Driver Library****:* The Drivelib [28] is a set of APIs utilized to control the on-chip peripheral devices. It is provided in ROM code and placed in a fixed location on Cortex-M3 SoCs. It helps identify the functions of firmware calls. Through the fixed location of APIs and the Drivelib datasheet, the function matches with names. It is a two-level table structure. The main table is at 0x1000010, and it contains the address of the second-level table for each type of peripherals. Figure 3 shows a typical code snippet that calls a ROM API. The second-level table is at 0x1000010 + 0x10 (`ROM_APITABLE[4]`), which is the the

```
MOVS            R1, #1
LDR.W           R0, =0x40059000 ; GPIO Port B
LDR.W           R2, =0x1000020
LDR             R2, [R2]
LDR             R2, [R2, #0x2C]
BLX             R2      ; ROM_GPIOPinRead
                        ; Reads the values present
                        ; of the specified pin(s).
```

Fig. 3: A code snippet that calls `ROM_GPIOPinRead()`. The parameters are passed through registers. In this case, the `ROM_GPIOPinRead()` has two parameters. R0 contains the GPIO port address, and R1 indicates the pins to operate.

GPIO table. And the API it calls is `ROM_GPIOPinRead()` (`ROM_GPIOTABLE[11]`). Because all the ROM API call sites have the same pattern, it makes firmware's reverse engineering work much easier. Calling the API is merely locating the function address from the two-level tables and jumps to it, and the parameters are passed in through registers. There is no privilege change when calling into the APIs, and the firmware code all runs in the system mode.

*c) **GPIO**:* The PLC interacts with the physical world through inputs and outputs (sensors and actuators). In the microcontroller LM3S2793, the GPIO module comprises nine physical GPIO blocks, and each corresponds to an individual GPIO port. Depending on the microcontroller's configuration, it supports up to 67 programmable input/output pins or several pins grouped to provide peripheral functions such as I2C. The GPIO ports can be accessed either through AHB or APB bus, which matters when we access the GPIOs through JTAG. We choose the AHB-AP. To change the inputs and outputs, we primarily operate on the GPIO Data (GPIODATA) register that modifies individual bits in GPIO ports. The way to control the GPIO data is not straightforward. Different microcontrollers adopt different operating methods.

In our case, the LM3S2793 microcontroller uses a more complicated model to conduct bit-wise operations. The GPIO Data register is memory-mapped. When read/write, bits[9:2] of the address are treated as a bitmask, as well as the bits[1:0] are always zero because the memory access should be at 4-byte alignment on ARM. Therefore, for each GPIO port, the memory-mapped range should be 1KB long, that is, from GPIODATA to GPIODATA + 0x3FC. A write can only change the data bit when the corresponded bitmask is set. Otherwise, the data bit is unchangeable. For example, GPIO Port A (AHB) is mapped at 0x40058000, and it controls 8 GPIO pins. We want to set bit2 to 0 and bit5 to 1. As shown in Figure 4, the bitmask is 0x90, and the data is 0xF0. Hence, the operation should be writing 0xF0 to address 0x40058090. The same applies to reading. Only the corresponding bit in the bitmask will be read. Otherwise, it reads zero. For example, to read the four high bits from GPIO Port A, the address with bitmask should be 0x40058000 + 0x3C0, and it reads 0xA0, as shown in Figure 5.

*d) **SPI Flash Memory**:* It is pretty noticeable that right next to the LM3S2793 microcontroller, there is an 8-pin flash chip, which is an Adesto 45DB021E 2-Mbit SPI flash memory. It connects with the PLC's synchronous serial

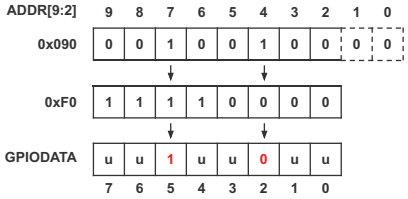

Fig. 4: Writing a byte of 0xF0 to address GPIODATA + 0x90. The bitmask only allows bit2 and bit5 to be modified. Therefore, only two bits are valid for the write operation. **u** indicates the new bit is ignored.

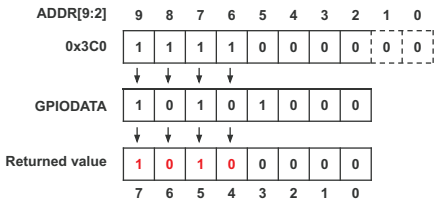

Fig. 5: The address GPIODATA+0x3C0 reads the high four bits of the GPIO port. The rest reads zero regardless of the actual value.

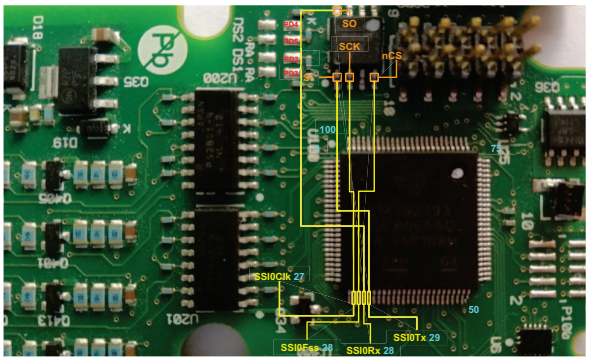

Fig. 6: Through wiring tracing, we found that the AT45DB21E SPI Flash Memory connects to LM3S2793's SSI0 interface. The eight solder joints on the left may be used to install four LEDs. They are controlled by GPIO port D.

interface (SSI0), as shown in Figure 6. During the boot process, the pins in GPIO port A are assigned for the SSI0 master device. The firmware first reads one byte from the flash chip (offset 0x2000), which looks like a status byte. If it equals 0x55 or 0xAA, the PLC will be reset. If not, the firmware checks the integrity of the address 0x4000 to 0x1FFFC, the compiled ladder logic. The algorithm is a simple checksum. Accumulate each byte in this address range, and the result should be equal to the byte in address 0x1FFFF. The status byte at 0x2000 indicates the status of the PLC last time it was running. The value 0x55 indicates that the system has encountered a severe failure. If the value is 0xAA, it means that the ladder logic binary has been broken, and the firmware will copy the code from 0x6100 in the SPI flash. This is a

```
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
ROM_GPIOPinTypeI2C(0x40059000, 0xC);
clock = ROM_sysCtlClockGet();
ROM_I2CMasterInitExpClk(0x40020000, clock, TRUE);
ROM_I2CMasterSlaveAddrSet(I2C0, 0x21, FALSE);
ROM_I2CMasterDataPut(I2C0, 0x6);
ROM_I2CMasterControl(I2C0, I2C_MASTER_CMD_BURST_SEND_START);
while (ROM_I2CMasterBusy(I2C0)) {
};
ROM_I2CMasterDataPut(I2C0, 0x0);
ROM_I2CMasterControl(I2C0, I2C_MASTER_CMD_BURST_SEND_COUNT);
while (ROM_I2CMasterBusy(I2C0)) {
};
ROM_I2CMasterDataPut(I2C0, 0x0);
ROM_I2CMasterControl(I2C0, I2C_MASTER_CMD_BURST_SEND_FINISH);
while (ROM_I2CMasterBusy(I2C0)) {
};
```

Fig. 7: The pseudo-code to initialize the I2C IO expander is reversed-engineered from the firmware. For presentation, we use some macro definitions as parameters. These macros' actual value is not difficult to find in the header files released by the vendor. The code is provided to help understand how to control the LED lights on the PLC through the I2C bus.

backup code and also a place where malicious code could be stored. SPI flash memory To read the AT45DB21E flash chip's content, we ported its driver to the Teensy 3.2 board.

e) ***PLC's Front Panel LED****:* There are four rows of LED lights on the PLC's front panel. Each LED shows individual input and output pins' status, which is an intuitive way for the administrator to check whether the device works properly. The microcontroller controls these LEDs through the I2C protocol [29]. There are two 24-pin PD9535 chips (Remote 16-bit I2C/SMBus, Low-Power I/O Expander [30]) on module **E**. It provides general-purpose I/O expansion for microcontroller via the I2C bus. The two pins in GPIO port B(PB2 and PB3) on the microcontroller are used as the SCL and SDA signal lines for the I2C master device. These two signal lines also pass through the connector between module **B** and **C**, as shown in Figure 18.

Each I2C slave device on the same bus has a unique 7-bit address. In our case, the addresses of the two PD9593 chips are 0x20 and 0x21. The master device successfully sends the address byte, followed by a byte to select the internal register. Figure 7 shows the pseudo-code to initialize the I2C IO expander using the ROM API. It first enables the I2C master device and sets the clock frequency to be the same as the microcontroller. After that, it calls `ROM_I2CMasterSlaveAddrSet()` to select the target device's address, whether to send or receive data. Then call `ROM_I2CMasterDataPut()` to set the data to be sent next. Only after calling `ROM_I2CMasterControl()`, the data will be sent out. In our example, two bytes with content zero are sent to Configuration Port0 and Port1. This operation sets a total of 16 pins as output mode. After initialization, Figure 8 shows how to control the LEDs. For example, we send two bytes with content 0xFF to the Output Port0 and Port1, lighting up 16 LED lights controlled by this device.

V. Stealthy Hardware Implant

We describe the design and implementation of the hardware implant, and how it is used to control the PLC remotely.

```
ROM_I2CMasterSlaveAddrSet(I2C0, 0x21, FALSE);
ROM_I2CMasterDataPut(I2C0, 0x2);
ROM_I2CMasterControl(I2C0, I2C_MASTER_CMD_BURST_SEND_START);
while (ROM_I2CMasterBusy(I2C0)) {
};
ROM_I2CMasterDataPut(I2C0, 0xFF);
ROM_I2CMasterControl(I2C0, I2C_MASTER_CMD_BURST_SEND_CONT);
while (ROM_I2CMasterBusy(I2C0)) {
};
ROM_I2CMasterDataPut(I2C0, 0xFF);
ROM_I2CMasterControl(I2C0, I2C_MASTER_CMD_BURST_SEND_FINISH);
while (ROM_I2CMasterBusy(I2C0)) {
};
```

Fig. 8: Each pin in the Output register controls an LED light separately, so two 0xFF bytes can control 16 LEDs. If we only want to control a few of the LED lights in one register, we can call `ROM_I2CMasterControl()` with parameter `I2C_MASTER_CMD_SINGLE_SEND`.

A. Design

Typically, the PLC has a dedicated real-time microcontroller that handles IO, in this case, module **C**. It runs minimal code, mainly the compiled ladder logic. To receive ladder logic update, the communication module **B** communicates with the HMI and then updates the real-time microcontroller through the CAN bus. Therefore, to remotely control the PLC's I/O, the attacker needs to control the communication module **C**.

To successfully have an implant with few physical characters such as low power consumption, small enough to fit inside the PLC case, support all the internal protocols that we are planning to control, and have a persistent connection to the attacker without detection, we utilized a Teensy board. The board is small enough to fit inside the PLC, consumes low power, and supports most of the required protocols. The power module (**A**) provides stable 3 volts for other boards. Our implant can either get power directly from it or the JTAG pad. ICS networks are often protected by network-based intrusion detection systems, i.e., anomaly detectors. Even if an attacker manages to penetrate the ICS network and control it, they could be flagged by these anomaly detectors. Therefore, to avoid all these hazel, we choose to use a separate network using GSM. To achieve this, we used SIM800C, a Quad-Band GSM/GPRS module. It has strong extension capability with interfaces including UART, USB2.0, and GPIO. Figure 9 shows that the SIM800C module connects with the Teensy board through a serial port. First, the Teensy board initializes the cellular module using AT command, connecting to the network. Once a text message is received, the Teensy board reads it and looks for control commands. In such a case, the command will be parsed as IO operations that eventually turn into particular memory read/write on PLC's GPIO port.

B. Implementation

a) ***Control Output****:* After knowing how to control GPIO, we can directly control the output of the PLC. There are 16 inputs and 16 outputs on the IO connector. Through reverse engineering, we know the GPIO port corresponding to the IO. That are, GPIO port E (0x4005C000) GPIO port F (0x4005D000) for inputs, and GPIO port G (0x4005E000), GPIO port H (0x4005F000) for outputs. Intuitively, each GPIO

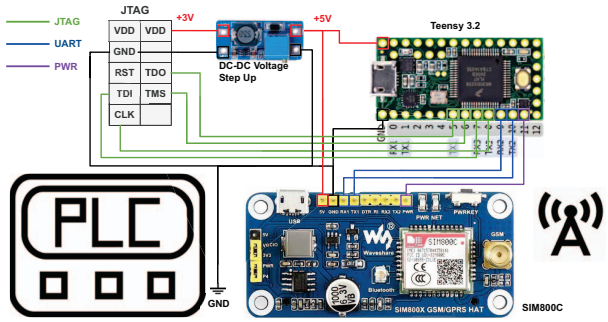

Fig. 9: The power can be drawn from the JTAG pad or directly from the border connector p607. Since the JTAG pad provides the 3 volts source, we also need a DC-DC voltage step-up module that boosts from 3 volts to 5 volts. One GPIO pin from Teensy connects the PWR pin of the SIM800C board. It is used to deliver the power and reset sequence. After SIM800C initialized, the Teensy module sends AT commands through the serial port.

bit corresponds to a pin on the connector. One indicates the high voltage, which is the field power voltage; zero dictates the low voltage (8 volts).

To conduct a stealthy attack, we want to change the output secretly. For example, we want to keep the LEDs in their original state and the host (HMI) not to find any abnormalities. The PLC periodically scans the inputs, runs ladder logic, and updates outputs. It is trivial to find the ladder logic binary by following the timer handler routine. There are a few local variables that control how the ladder logic behaves. For example, one local variable determines if any logic state has changed. If so, the corresponded GPIO pin will be updated according to the ladder logic. We modify this local variable so that everything looks up to date. In the meantime, we can change the outputs without triggering any alarms.

b) ***Read/Write Memory****:* We introduced the JTAG protocol in section III. MALTAG sends instructions to the IR and reads the returned result according to the JTAG state machine. The JTAG debug port has several interfacing registers. Through DPACC and APACC registers, the debugger can access resources provided by other access ports (AP). An access port provides the interface between the debug port interface and one or more debug components present within the system. There are two kinds of access ports: memory access ports (MEM-AP) and JTAG access ports (JTAG-AP), where MEM-AP is designed for connects to memory bus system with address and data controls. Since MALTAG wants to access memory and GPIO, we need to access either AHB-AP or the MEM-AP, which handled the differences between the underlying bus.

According to ARM Debug Interface Architecture Specification v5.2, every bank has four registers, and the two bits A[3:2] are used to select the register from the bank. The DP only has one bank, and the MEM-AP has 16 banks. To select the AP's bank, we also need to write the bank address into the DP's SELECT register. To read and write memory, we need to use the internal registers provided by the MEM-AP. Specifically, we need to use CSW, TAR, DRW, and others. Since these registers are in the MEM-AP's bank 0, we must first use DPACC to select it. Take writing memory as an example. Next, we need to write the destination address to TAR and then write the value to DRW. Figure 10 shows a pseudo-code for writing memory.

```
TAP_Idle();
// shift to DPACC
TAP_ShiftIR(DPACC);
// select DP.CTRL/STAT, scan in data
TAP_ShiftDR(DP.CTRL/STAT, CSYSPWRUPREQ | CDBGPWRUPREQ);
// scan out status
TAP_ShiftDR(status);
// select DP.SELECT, set AP bank0
TAP_ShiftDR(DP.SELECT, AP_BANK0);

// shift to APACC
TAP_ShiftIR(APACC);
// select AP.CSW, set write size
TAP_ShiftDR(AP.CSW, SIZE_WORD);
// select AP.TAR, set destination address
TAP_ShiftDR(AP.TAR, address);
// select AP.DRW, write data
TAP_ShiftDR(AP.DRW, value);
```

Fig. 10: Pseudo-code for memory writing through MEM-AP. Notice, CSW, TAR and DRW are all in the bank zero. However, the bank is specified in the DP.SELECT instead of a register in AP.

c) ***Send/Receive SMS Message.****:* SIM800C provides a serial port as the interface to receive AT commands. When MALTAG starts, we use the AT+CMGF command to set the GSM chip in SMS Text Mode. Then we use the AT+CNMI command to set how to notify when new messages come. After that, the Teensy board keeps checking the serial port for new messages every second. If there is one, it reads the content and parses if it is a pre-defined attack command. To send out a message, we use the AT+CMGS command to set the destination phone number and then send the text message to the serial port. This enables real-time communication between the infected distributed PLCs and the adversaries remotely.

d) ***Intercepting SPI Data.****:* We can directly control the power switch chips to change the output or modify the transmission data by intercepting the bus on the circuit board, mainly low-speed buses such as SPI. For example, if we change a pin from zero to one, we need to connect it to the high voltage (3.3v) power supply to override the original signal. Similarly, we ground it to force it a zero. However, these operations need to consider the IC's interface. The pins of the SPI flash chip use a push-pull configuration rather than an open drain. The output pins can actively create their own logical high and low states instead of relying on pull-up resistors to generate a default state. However, in this way, we must add an appropriate current-limiting resistor when forcing the signal to the ground or power supply to avoid short circuits. For the SPI protocol, we modify the MOSI data line according to the SCLK clock signal, and we use FPGA to implement this circuit. In each clock cycle of SPI, we first read the data transmitted on the MOSI and store it into a shift register. When finding the target pattern, we modify the following data. To

meet the setup time required for the SPI data line, we choose to make the circuits work on the clock's falling edge, given that the SPI works on the rising edge.

VI. Evaluation

We evaluated several aspects of MalTag. Since it is a hardware backdoor, we need to measure its physical dimensions and appearance. We also evaluated the effects of MalTag on the PLC, including the impacts on the performance, memory storage, and power consumption. As mentioned earlier, the PLC model we used is the Allen Bradley 1769-L18ER-BBIB CompactLogix 5370, which is equipped with a TI Stellaris LM3S2793 microcontroller. The microcontroller is based on ARM Cortex-M3 architecture and operates at 80MHz. It has 128KB single-cycle flash memory and 64KB single-cycle SRAM. The PLC has 16 inputs and 16 outputs, which eventually corresponds to the microcontroller's GPIO port.

Performance. Since this backdoor is implemented on the hardware level, it almost cost zero performance overhead. For instance, if we choose to change I/O through override signals transmitting in the low-speed bus, pull-up or pull-down the voltage level, there will be no overhead added to the microcontroller. On the other hand, if we use the JTAG interface instead, it may cause a slight performance overhead depending on the microarchitecture. Due to various implementations, JTAG debugging capabilities can be from non- to negligibly-intrusive. The conventional JTAG debug is invasive, which halt the processor using breakpoints and watchpoints. It also needs to halt the processor before it can modify any register.

Nevertheless, the debug functionality provided in LM3S2793 is as CoreSight components. It provides real-time access for the debugger without halting the processor to AMBA system memory, peripheral registers, and all debug configuration registers. Therefore MalTag can modify the GPIO through the AHB bus without any software overhead. Additionally, the external or timer interrupt may be delayed for a few clock cycles when encountering JTAG related operations. However, MalTag has no operation when in standby mode, and controls on I/O only take a few memory reads/writes. Furthermore, we regulate our attacking operations at a low pace not to jam the system. The resulting performance overhead is generally negligible, as shown in Figure 11.

Memory Consumption. MalTag neither occupies flash memory to reside on the system nor takes SRAM space during the runtime. Even when it is controlling the PLC's I/O, the operations are performed on memory-mapped I/O for GPIO ports. Therefore, the memory consumption overhead for MalTag on the PLC's main processor is zero.

Power Consumption. Because of the extra circuitry we bring to the system, the power consumption of the PLC increases. Although the two embedded devices consume very little power compared to the field power that the PLC provides, this is an anomaly we brought into the system. We measure it with the PLC that is connected to the field power but carries no

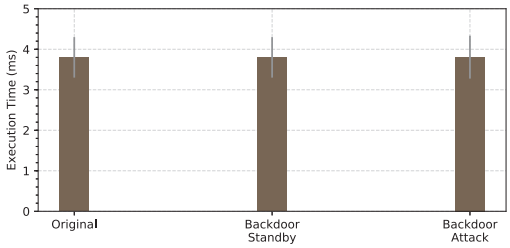

Fig. 11: To measure execution time, we use a counter in the ladder logic and an output signal to indicate the begin/end time of the test. During the standby test, MalTag has attached the PLC, but no command is sent. We alter the PLC's output every 500ms to imitate a malicious operation for the attack test.

applicants. We connect a resistor in series to the power supply line of the PLC and use an oscilloscope to measure the voltage across the resistor. Figure 12 shows that several slight current peaks occur when the cellular chip is searching and connecting to the GSM network and when MalTag receives the SMS message and sending JTAG commands to the PLC.

SMS Message. One concern we have is that since the antenna is also encapsulated inside the PLC's case, the signal strength may not be good enough for real-time communication with the remote adversaries. We test it for receiving various lengths of SMS messages as shown in Figure 13. We use a phone to send the command messages to MalTag, and send each command 20 times to calculate the average. The cellular networks we used are T-Mobile and Google Fi. Different lengths of information have different purposes. For starting a denial-of-service attack or a pre-defined function, one byte is enough. To change specific I/O, we need a few bytes to describe its address or index. We validate real-time reception of hundreds of bytes in runtime. It can be used to update the firmware on the PLC's flash remotely.

Mitigations. Reducing the size of the hardware implant is necessary but not the main factor in disguising. A customized PCB board allows all the required chips to be installed together so that there is no Dupont jumper wire, which makes the hardware implant very suspicious. The SIM800C chip already is a system in a package (SiP), and the JTAG driver can run on a minimal core such as Cortex-M0. Combining these two can make the backdoor simpler, and we can find such WiFi chips (e.g., ESP32 [31]) as of the time of writing this paper.

To mitigate a hardware backdoor attack such as MalTag, we first measure the power usage to find the overhead caused by the extra circuitry as evaluated in section VI. The result shows a few current peaks when MalTag connects to the GSM network and sends JTAG commands to PLC. However, its power consumed is not much, a few milliamperes. Moreover, the PLC's power consumption is also constantly fluctuating, and setting a threshold with little redundancy will affect the system's reliability. Additionally, the implant could be battery-powered. To avoid malicious use of the JTAG interface, the manufacturer can blow the corresponding

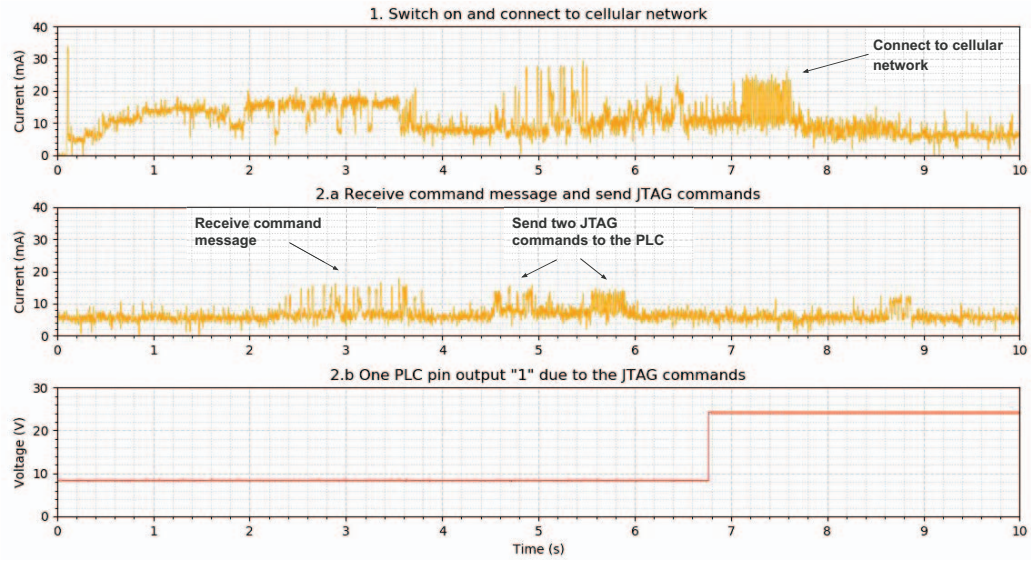

Fig. 12: The hardware implant does not consume much power, and the power consumption will only increase slightly when starting and executing the attack command. Sub-figure 1 shows the power consumption during the startup. Sub-figure 2.a shows the power consumption during a demo attack. 2.b indicates an output pin of the attacked PLC.

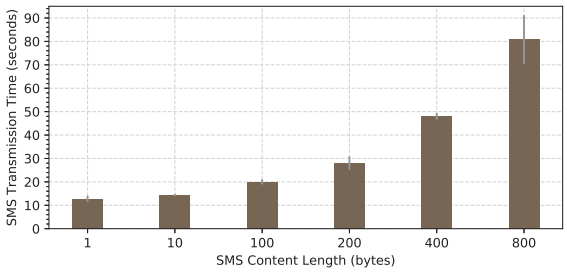

Fig. 13: The GSM network may be congested when there are many users around. The large piece of message, such as 800 bytes, will be segmented into several packets, and the transmission time varies, but still, the received order and the message content are correct.

physical JTAG fuse at the factory [32], [33]. Blowing a fuse will completely disable the JTAG port and is not reversible.

However, by tapping the open wire on the boards, we can directly control each peripheral device because they are connected to the microcontroller through various buses. We think that some physical protection can cause trouble to the attacker. For example, the chip-on-poard (COB) [20] packaging with a black blob for low-cost IC items makes it more challenging to identify the chip underneath. Nevertheless, we believe it is not a good practice to cover the whole board, and heat dissipation is critical for large ICs. One possible direction for preventing bus signal hijacking is to send packet-based encrypted data, which requires the microcontroller and peripherals to exchange encryption keys and maintain a connection state. However, it may not be practical for low-speed devices and low-end microcontrollers.

VII. Related Work

Firmware modification attacks constitute significant attacks targeting embedded systems, industrial control systems, and IoT devices [34], [35]. Harvey [9] is a physical-aware stealthy rootkit against a cyber-physical power grid control system. IronSpider [6] performs similar attacks through web channels.

For defense, there are many proposed solutions to protect the firmware from being modified. ConFirm [36] is a low-cost technique to detect malicious modifications in the firmware of embedded control systems. It measures the number of low-level hardware events that occur during the execution of the firmware. Lee et al. [10] presented a technique for binding software to hardware instances that use the devices' hardware security properties. The proposed technique assures manufacturers that only they can perform their hardware and software binding and create their products.

Remote software attestation [37] is also a defense against firmware modification. SWATT [38] verifies embedded devices' memory contents and establishes the absence of malicious changes to the memory contents without using extra security hardware features. It uses a challenge-response protocol between the verifier and the embedded device. The verifier sends a challenge to the embedded device. The embedded device computes a response to this challenge in a pre-defined protocol between the verifier and the device. The device can only give the correct answer if the memory content is intact. Otherwise, the attacker has to know the verifier's secret algorithm to break the verification. Similarly, SBAP [39] also provides a software-only solution to verify the firmware integrity but with the help of an existing peripheral device.

Other methods, such as firmware binary obfuscation [40], make it very challenging for firmware modification attacks. It requires comprehensively analyzing each device to find a suitable place to inject malicious code.

VIII. Conclusions

We presented MALTAG, a parasitical hardware implant that directly controls the PLC through wire and bus signal hijacking. MALTAG does not modify the firmware nor relies on PLC's network communication. It can manage/damage the ICS system's physical assets by controlling PLC's IO. At the same time, it provides a faked expected view of the system to circumvent detections. Our prototype is small in size, and the experiments demonstrate the MALTAG's feasibility.

Acknowledgement. We would like to thank the National Science Foundation (NSF) for their support of this research.

References

[1] Hassan Haes Alhelou, Mohamad Esmail Hamedani-Golshan, Takawira Cuthbert Njenda, and Pierluigi Siano. A Survey on Power System Blackout and Cascading Events: Research Motivations and Challenges. *Energies*, 12(4):682, 2019.

[2] Mark Zeller. Myth or Reality—Does the Aurora Vulnerability Pose a Risk to My Generator? In *2011 64th Annual Conference for Protective Relay Engineers*, pages 130–136. IEEE, 2011.

[3] Ralph Langner. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security Privacy*, 9(3):49–51, 2011.

[4] Saleh Soltan, Mihalis Yannakakis, and Gil Zussman. Power Grid State Estimation Following a Joint Cyber and Physical Attack. In *IEEE Transactions on Control of Network Systems*, volume 5, pages 499–512. IEEE, 2016.

[5] Sriharsha Etigowni, Dave Tian, Grant Hernandez, Saman Zonouz, and Kevin Butler. Cpac: Securing critical infrastructure with cyber-physical access control. In *ACSAC*, pages 139–152, 2016.

[6] Ryan Pickren, Tohid Shekari, , Saman Zonouz, and Raheem Beyah. Compromising industrial processes using web-based programmable logic controller malware. *Network and Distributed System Security (NDSS) Symposium*, 2024.

[7] Anton Cherepanov and Robert Lipovsky. Blackenergy - what we really know about the notorious cyber attacks. *Virus Bulletin October*, 2016.

[8] Alessandro Di Pinto, Younes Dragoni, and Andrea Carcano. Triton: The first ics cyber attack on safety instrument systems. In *Black Hat*, pages 1–26, 2018.

[9] Luis Garcia, Ferdinand Brasser, Mehmet Hazar Cintuglu, Ahmad-Reza Sadeghi, Osama A Mohammed, and Saman A Zonouz. Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit. In *NDSS*, 2017.

[10] Robert P Lee, Konstantinos Markantonakis, and Raja Naeem Akram. Binding Hardware and Software to Prevent Firmware Modification and Device Counterfeiting. In *Proceedings of the 2nd ACM international workshop on cyber-physical system security*, pages 70–81, 2016.

[11] Brendan Moran, Milosch Meriac, Hannes Tschofenig, and David Brown. A Firmware Update Architecture for Internet of Things Devices. In *Internet Engineering Task Force, Internet-Draft*, 2019.

[12] Constantine Doumanidis, Yongyu Xie, Prashant HN Rajput, Ryan Pickren, Burak Sahin, Saman Zonouz, and Michail Maniatakos. Dissecting the industrial control systems software supply chain. *IEEE Security & Privacy*, 2023.

[13] Oxford Analytica. SolarWinds Hack will Alter US Cyber Strategy. In *Emerald Expert Briefings*, 2021.

[14] Jordan Robertson and Michael Riley. The Big Hack: How China Used a Tiny Chip to Infiltrate US Companies. *Bloomberg Businessweek*, 4, 2018.

[15] Michael Mabee. Loopholes in grid physical security identified.

[16] Jacob Harrison, Navid Asadizanjani, and Mark Tehranipoor. On Malicious Implants in PCBs Throughout the Supply Chain. *Integration*, 2021.

[17] O'Donohue. Sarah. Special delivery: Allow two to three weeks for shipping and nsa bugging. *BU Pub. Int. LJ*, 24:81, 2015.

[18] Behr Peter. Substation Attack is New Evidence of Grid Vulnerabilit, 2016.

[19] Lei Chen, Dong Yue, Chunxia Dou, Jianbo Chen, and Zihao Cheng. Study on Attack Paths of Cyber Attack in Cyber-physical Power Systems. *IET Generation, Transmission & Distribution*, 14(12):2352–2360, 2020.

[20] John H Lau. *Chip on Board: Technology for Multichip Modules*. Springer Science & Business Media, 1994.

[21] Mohamed Shalan and Vincent J Mooney. A Dynamic Memory Management Unit for Embedded Real-time System-on-a-Chip. In *Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 180–186, 2000.

[22] Chung Hwan Kim, Taegyu Kim, Hongjun Choi, Zhongshu Gu, Byoungyoung Lee, Xiangyu Zhang, and Dongyan Xu. Securing Real-Time Microcontroller Systems through Customized Memory View Switching. In *NDSS*, 2018.

[23] Henry Neugass, G Espin, Hidefume Nunoe, Ralph Thomas, and David Wilner. VxWorks: An Interactive Development Environment and Real-time Kernel for Gmicro. In *IEEE TRON Project Symposium*, pages 196–207, 1991.

[24] Richard Barry et al. FreeRTOS. *Internet, Oct*, 2008.

[25] R Joshi, H Granada, and C Tangpuz. MOSFET BGA package. In *IEEE Electronic Components and Technology Conference*, 2000.

[26] Texas Instruments. Stellaris lm3s2793 microcontroller data sheet, 2014.

[27] David Flynn. AMBA: Enabling Reusable On-chip Designs. *IEEE Micro*, 1997.

[28] Texas Instruments. LM3S2793 ROM user guide, 2011.

[29] Philips Semiconductors. The I2C-bus Specification. *Philips Semiconductors*, 9397(750):00954, 2000.

[30] TEXAX INSTRUMENTS. PD9535 Datasheet, 2007.

[31] V Pravalika and Rajendra Prasad. Internet of Things Based Home Monitoring and Device Control using ESP32. *International Journal of Recent Technology and Engineering*, 8(1S4):58–62, 2019.

[32] Kurt Rosenfeld and Ramesh Karri. Attacks and Defenses for JTAG. *IEEE Design & Test of Computers*, 27(1):36–47, 2010.

[33] Ronald F Buskey and Barbara B Frosik. Protected JTAG. In *2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, pages 8–pp. IEEE, 2006.

[34] Samuel Bennett Moore, William Bradley Glisson, and Mark Yampolskiy. Implications of Malicious 3D Printer Firmware. 2017.

[35] Ang Cui, Michael Costello, and Salvatore Stolfo. When Firmware Modifications Attack: A Case Study of Embedded Exploitation. 2013.

[36] Xueyang Wang, Charalambos Konstantinou, Michail Maniatakos, and Ramesh Karri. Confirm: Detecting Firmware Modifications in Embedded Systems using Hardware Performance Counters. In *Proceedings of the IEEE/ACM international conference on computer-aided design*, pages 544–551. IEEE Press, 2015.

[37] Yanlin Li, Jonathan M McCune, and Adrian Perrig. VIPER: verifying the integrity of PERipherals' firmware. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 3–16. ACM, 2011.

[38] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. SWATT: Software-based Attestation for Embedded Devices. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 272–282. IEEE, 2004.

[39] Yanlin Li, Jonathan M McCune, and Adrian Perrig. SBAP: Software-based Attestation for Peripherals. In *International Conference on Trust and Trustworthy Computing*, pages 16–29. Springer, 2010.

[40] Benjamin Cyr, Jubayer Mahmod, and Ujjwal Guin. Low-cost and Secure Firmware Obfuscation Method for Protecting Electronic Systems from Cloning. *IEEE Internet of Things Journal*, 6(2):3700–3711, 2019.

APPENDIX

A. Dimensions and appearance

We use commercial off-the-shelf modules to build the prototype, namely the Teensy 3.2 development board and Waveshare SIM800C HAT. They are not the smallest in size. For example, the SIM800C in SiP packaging with a minimal PCB board is much smaller. However, the SiP needs a voltage of 3.7 volts which a Lithium-Ion battery usually provides. Cellular GSM consumes an impulse type transmitting power; even though the average current is low, but the instantaneous current can reach more than 1.5A, so the board's external power supply is necessary.

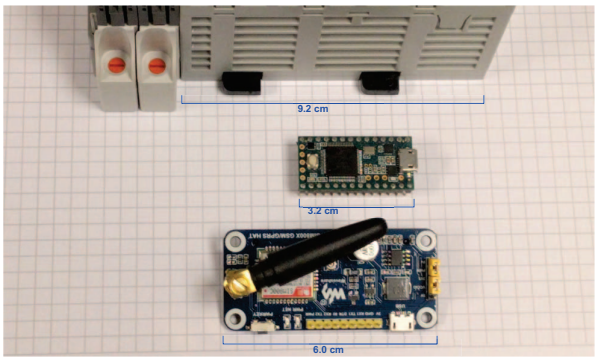

Fig. 14: The Allen Bradley 1769-L18ER-BBIB CompactLogix 5370 PLC is in a 9.2cm x 13cm rectangle shape with sufficient space to contain the two boards and extra wires and connectors. The SIM800C HAT takes a full-size sim card, and the antenna takes much space.

Figure 14 shows the physical size of the two boards compared to the Allen Bradley PLC. Notice that the two main chips on each board are relatively small. Some WiFi chips also provide a microcontroller for general-purpose tasks, so the two chips can combine into one reducing the hardware implant size. An effective way to visually hide the hardware implant is to use a customized PCB with the same color and connector style attaching to the PLC's board. Some examples in general-purpose desktop computers are the separate intelligent platform management interface (IPMI) and keyboard, video, and mouse (KVM) modules that are usually mounted on the server's motherboards.

Figure 15 shows that, after wiring two boards and wrapping them with tape to prevent a short circuit, it is small enough to fit inside the PLC's plastic case. We connect the JTAG pads with GPIO pins from the Teensy board. One of the PLC we use for this experiment turns out to have a 10-pin socket soldered on the pad, but all other PLCs we own do not have such a setting. The hardware implant is not noticeable from the PLC's outside appearance; it only takes up small space with no parts exposed externally. Low power consumption eliminates the heat dissipation problem for practical deployment of the implant for long-term persistent threats.

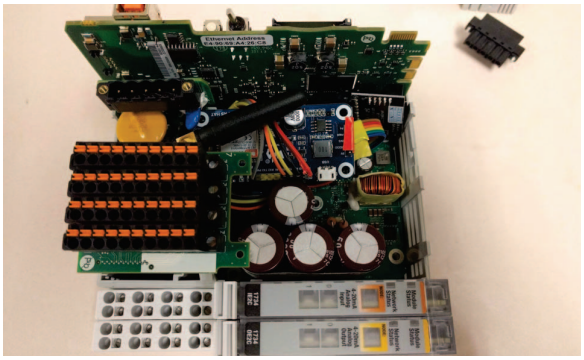

Fig. 15: The MALTAG connects to the PLC's microcontroller board's JTAG pad through a 10-pin socket and a long cable. The GSM antenna resides inside the PLC.

B. Reverse Engineering Hardware

In this section, we will describe the reverse engineering of the PLC's hardware. We will describe the different boards in PLC, on-board connectors, and how the pins and ICs were identified, and finally about the microcontroller present in the PLC used to control the IOs of PLC.

*a) **Backplanes**:* The disassembling of PLC from its shell lead to discovering several PCB module boards. These boards are commonly known as backplanes. Figure 16 shows all the five backplanes present. For convenience, we named them from **A**-**E**. We name the board with the power supply as `power supply module` (**A**), the board with Ethernet socket and USB port as the `communication module` (**B**), the board that has a microcontroller and controls the 16 digital DC input pins and 16 digital DC output pins as the `real-time module` **C**, the board that is used as a connector for the digital IOs as `digital connector module` (**D**), and finally the board that connects the LEDs as `LED module` (**E**). We focus more on (**C**) the real-time module since this is responsible for reading input from the sensors and controlling the actuators. All these backplane modules were identified based on the components (capacitors, coils, and the ICs) present on the boards and tracing them to the peripherals.

The `communication module` (**B**) includes a CPU, DRAM, and other peripherals such as Ethernet, SD card, and USB. It communicates with the host (HMI) to receive firmware and ladder logic updates. It also hosts a web server to display status. However, this module does not directly interact with IO. Therefore, we focus on the `real-time module` (**C**) that runs ladder logic and directly controls IO. The `real-time module` uses a commercial microcontroller, Texas Instruments Stellaris LM3S2793 SoC.

The digital IO sockets on the `digital connector module` (**D**) are connected to the `real-time module` (**B**). The IO goes through the power switches followed by optically coupled isolator chips and eventually connects to the microcontroller, as shown in Figure 17. There are 32 LEDs

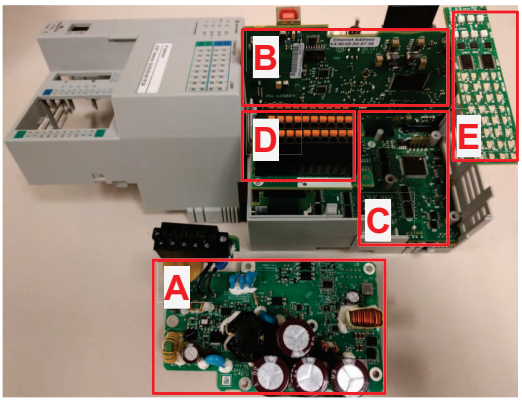

Fig. 16: Opened Allen-Bradley 1769-L18ER-BB1B/B CompactLogix 5370 PLC. A: Power supply module B: Communication module C: Real-time module D: Digital Connector Module - DC Digital Outputs (16) & DC Digital Inputs Connector (16) E: LED module

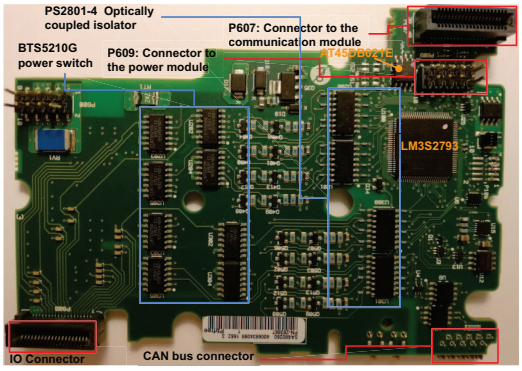

Fig. 17: The real-time module reads the input signals, runs the ladder logic, and then drives the output signal according to the result. Therefore this module needs to be connected to all other modules. There are power regulators and IO chips on this module in addition to the microcontroller and the flash memory. The optically coupled isolators prevent high voltages from affecting the microcontroller when receiving the signal, and the power switches provide the electrical connection between the output pin and the voltage source.

corresponds to each IO socket. The real-time module controls the LED module through the I2C protocol.

b) ***Onboard Connectors****:* Once the backplanes are identified, to understand the functionality of the `real-time module`, we look deep into the components present. First, we assess the functionality of each component on board of PLC based on the on-board ICs. Then, we use a multimeter to conduct the connectivity test for wire tracing. The purpose of this is to find the interconnections between chips and between boards. The conductivity test leads to the identification of microcontroller pins that are connected to other ICs on the board. We found different pins such as JTAG, SPI, I2C, and CAN through this method.

There are several connectors on the real-time module **C**, as shown in Figure 17. P607 connects to the communication module **B** and P609 connects to the power module **A**. Usually, we think that the module **A** is just a power module responsible for providing three volts to others. However, this is a good place for purposefully damage the PLC by a short circuit, where the power flow is large.

By conducting the connectivity tests with a multimeter, we perceive some pins' function in P607 as shown in Figure 18. The I2C bus and CAN bus passes through this connector. In this way, both the communication module and the real-time module are connected to the CAN bus connector in the bottom right corner in Figure 17. These two modules can not only control the external CAN bus device, but they can also communicate via the CAN bus between these modules. However, the communication module has a higher priority. Setting pin29 on the P607 connector can block the transmission of the signal on pin21, the CAN bus's RxD signal line. On the microcontroller side, the master device CAN0 uses the PA6 and PA7 pins.

c) ***Microcontroller****:* The TI Stellaris LM3S2793 SoC has an ARM Cortex-M3 processor core that operates at 80 MHz. It contains 64 KB SRAM and 128 KB flash. And

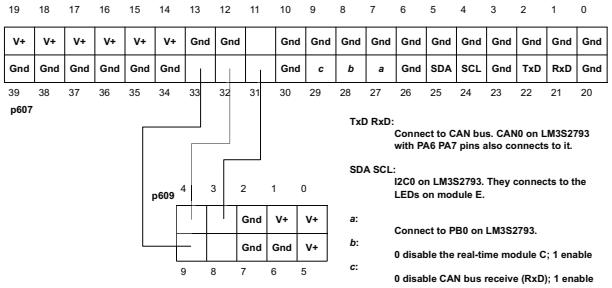

Fig. 18: The connector between the communication board (module **B**) and real-time board (module **C**). There are many pins, but the majority of them are power and ground. The microcontroller controls the LED module through the P607 connector. The communication module can block the operation of the entire real-time module through pin28. However, there are still a few pins that we have not figured out their functions. We consider it as one of our further works.

the internal ROM is preprogrammed with Stellaris Peripheral Driver Library (DriverLib) to drive the on-chip peripheral devices. We dump the firmware from the microcontroller and flash chip for reverse-engineering analysis.