Detecting Payload Attacks on Programmable Logic Controllers (PLCs)

Huan Yang*, Liang Cheng†, and Mooi Choo Chuah‡
Department of Computer Science and Engineering, Lehigh University, Bethlehem, Pennsylvania 18015
E-mail: *huy213@lehigh.edu, †cheng@cse.lehigh.edu, ‡chuah@cse.lehigh.edu

***Abstract*—Programmable logic controller (PLC) is a critical component of industrial control system (ICS). Providing hardware peripherals and firmware support for control program (i.e., a PLC's "payload") written in languages such as ladder logic, PLCs directly receive sensor readings and control ICS physical process. An attacker with access to PLC development software (e.g., by compromising an engineering workstation) can modify the payload program and cause severe physical damages to the ICS. To protect critical ICS infrastructure, we propose to model runtime behaviors of legitimate PLC payload program and use runtime behavior monitoring in PLC firmware to detect payload attacks. By monitoring the I/O access patterns, network access patterns, as well as payload program timing characteristics, our proposed firmware-level detection mechanism can detect abnormal runtime behaviors of malicious PLC payload. Using our proof-of-concept implementation, we evaluate the memory and execution time overhead of implementing our proposed method and find that it is feasible to incorporate our method into existing PLC firmware. In addition, our evaluation results show that a wide variety of payload attacks can be effectively detected by our proposed approach. The proposed firmware-level payload attack detection scheme complements existing bump-in-the-wire solutions (e.g., external temporal-logic-based model checker) in that it can detect payload attacks that violate real-time requirements of ICS operations and does not require any additional apparatus.**

I. Introduction

In industrial control systems (ICS), programmable logic controllers (PLC) play a critical role in process automation. As cyber attacks targeting ICS increase in sophistication, field devices, such as PLCs, are of particular concerns because they directly monitor and control physical processes. As shown in Figure 1, PLCs are typically deployed close to sensors and actuators, implementing local control actions (i.e., regulatory control). In addition of utilizing sensor data and controlling actuators locally, PLCs transmit real-time process data to operator workstations and execute their commands, facilitating the realization of supervisory control. Due to the unique and vital role of PLCs in critical ICS infrastructure [1], they are one of the major targets of cyber attacks. For example, the Stuxnet attack managed to silently sabotage centrifuges in a uranium-enrichment plant by reading and writing code blocks on PLCs from a compromised engineering workstation [2], [3]. By modifying a PLC's control program, severe damages (e.g., data loss, interruption of system operation, and destruction of ICS equipment) can be induced by attackers. In [4], it is shown that malicious code can easily be slipped into PLC control programs and evade the scrutiny of relay engineers from both academia and industry. Therefore, it is crucial to devise automated detection method against cyber attacks launched by modified PLC's control program.

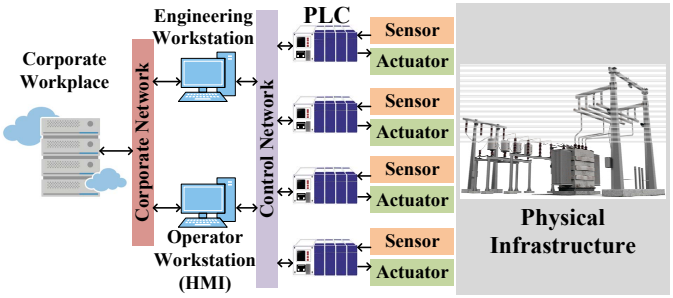

Fig. 1. Architecture of industrial control systems and the role of PLCs.

As PLCs are special-purpose computers interfacing with various sensors/actuators and providing firmware support to run control programs (also known as "payload" programs [5], [6]) that emulate the behaviors of an electric ladder diagram [7], [1], attacks on PLCs can be launched by modifying or overwriting the PLC payload program. Such attacks are known as *PLC payload attacks*. A PLC control program is typically written by a team of PLC engineers using the suite of programming languages specified in IEC 61131-3 [8]. Such a control program is regarded as the payload of a PLC's firmware, which controls access to hardware resources (e.g., inputs, outputs, and timers) and repeatedly loops through the payload instructions. An attacker with PLC access (e.g., by gaining control of an engineering workstation running PLC development and monitoring software) can download malicious payload and gain full control over its sensors and actuators. In the Stuxnet attack, a component of Stuxnet is capable of launching payload attacks on PLCs by first infecting an engineering workstation and then downloading malicious code blocks [3]. Payload attacks can also be carried out by an insider (e.g., a disgruntled employee) with the help of tools such as SABOT [5], which generates malicious payload based on adversary-provided specifications. Since legitimate payload relies on PLC programming instructions implemented by the firmware to carry out control and monitoring tasks, a malicious payload program can execute any combination of these instructions to sabotage the physical process.

In this paper, we introduce runtime behavior monitoring into PLC firmware to detect payload attacks and protect ICS from severe physical damages. Based on control system specification provided by control system engineers, we establish runtime behavior profile of normal/legitimate payload program in terms of I/O access patterns, network access patterns, as

well as payload program timing characteristics. When a newly updated payload program is downloaded into a PLC (either by an attacker or by a trusted control system engineer), its runtime behavior data is collected by the PLC firmware. When abnormal behaviors are observed by the firmware, execution of the payload program is terminated so that abnormal control signals will not be sent to actuators. The contributions of our work are as follows:

- We introduce runtime behavior monitoring into PLC firmware to enable automated detection of PLC payload attacks. In contrast to existing detection methods based on linear temporal logic, our proposed approach can identify attacks that violate real-time requirements of an ICS and does not require the introduction of bump-in-the-wire apparatus between engineering workstation and PLCs.
- We present a proof-of-concept implementation of the firmware-level payload attack detection scheme on ARM® Cortex®-M4F microcontrollers. Our evaluation results show that the proposed approach can detect a wide variety of payload attacks revealed by prior research [4] and reported cyber-security incidents.
- Furthermore, we evaluate the overhead of implementing the proposed detection method and find that it is feasible to incorporate our scheme on microcontrollers used by existing PLCs to detect payload attacks.

II. Related Work

A. Programmable Logic Controller (PLC) and Payload Program Execution Model

A programmable logic controller (PLC) is a special-purpose computer designed to replace relay panels and control a physical process [7]. Figure 2 presents the general hardware and software architecture of PLCs. There are several important characteristics that distinguish PLCs from personal computers [9]: PLCs are designed to operate in harsh industrial environments and are programmed in relay ladder logic or other PLC programming languages [8]. In addition, a PLC executes a simple payload program in a sequential fashion. Once deployed in an ICS, a PLC continuously collects readings from sensors connected to its inputs, runs the PLC payload program, and generates outputs that control the physical process. As shown in Fig. 1, PLC control program can be developed on engineering workstations using programming software that supports ladder logic or other PLC programming languages and downloaded to target PLC for execution. Operator of an ICS may monitor and control the physical process via a human-machine interface (HMI), which communicates with PLCs to receive real-time process data and issue control commands.

To control and monitor physical process, a PLC's firmware implements input and output image tables as well as a program scan cycle [7], [9]. A program scan cycle consists of input scan, program scan, output scan, and housekeeping phases, which are shown in Fig. 3. After system start-up, a PLC repeatedly walks through the four phases of the program scan cycle as follows: First, in the input scan phase, the

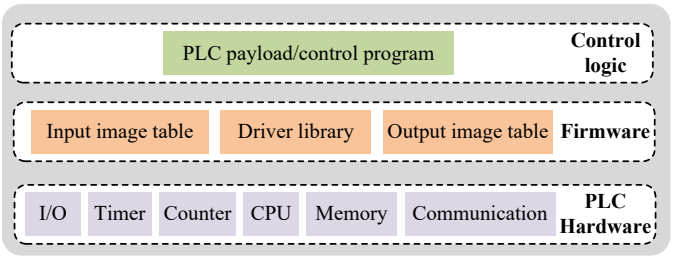

Fig. 2. General PLC hardware and software architecture.

PLC firmware samples the I/O pin values and writes them into the input image table. Then, in the program scan phase, instructions in the payload program are executed one by one using values stored in the input image table. Output values are generated during this phase and written into the output image table. Next, in the output scan phase, values in the output image table are transferred to the external output terminals, making control actions specified in the payload program take effect. Finally, in the housekeeping phase, internal checks on memory and system operation are performed. Additionally, communication requests originated from other hosts (e.g., the HMI) or generated by the payload program itself are also serviced before the next program scan cycle starts.

B. PLC Ladder Logic

Many widely-used PLC programming languages are standardized in IEC 61131-3 [8] and ladder logic is the most commonly used one [9] since it is straightforward to control system engineers who prefer to define control actions in terms of relay contacts and coils. Instructions specified by ladder logic have their own symbolic representation. A PLC payload program written in ladder logic has one or more ladder-formatted schematic diagrams. Within each diagram, ladder logic instructions are organized into rungs. Each rung may contain multiple ladder logic instructions, which are evaluated from left to right. Instructions on the left of a rung test input conditions or outputs generated by other rungs, and instructions on the right generate rung outputs. Multiple input condition checks can be placed in tandem, and the input logic evaluates to true if and only if all input conditions are true. Parallel branches can be used on a rung to accommodate more than one input condition combinations. The rung logic is evaluated to true as long as one of the branches forms a true logic path. When multiple output branches are present on a rung, a true logic path controls multiple outputs.

Fig. 4 shows a sample subroutine of a ladder logic program consisting of three rungs. The "XIC" instruction on the first rung examines if an input is true. If so, the instruction evaluates to true. The "OTE" instruction energizes a specified output bit. Input condition of the first rung first checks if input bit I:0/4 or

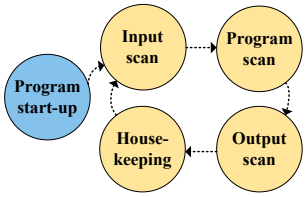

Fig. 3. PLC payload program execution model.

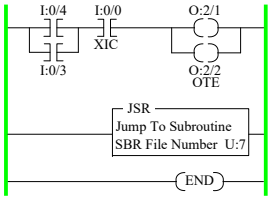

Fig. 4. A sample ladder logic program with three rungs.

I:0/3 is true and then checks if I:0/0 bit is true. The output of this rung controls both output bits, i.e., O:2/1 and O:2/2. The second rung's input condition is always true, so the subroutine in file U:7 is executed. Note that the subroutine is essentially another ladder logic diagram. When the subroutine returns, the second rung completes and the third rung is evaluated, which signals the end of the payload program. Note that hierarchical addressing is used in ladder logic program to specify the data type, slot number, and bit position of PLC data and peripherals [9]. For example, I:0/4 is the fifth bit of binary input slot 0 (with the first bit being I:0/0). For analog I/Os, the hierarchical address is slightly different. For example, O:2.0 is an analog output on the output module installed on slot 2, and the output value is written to the first (zero-indexed) word of its allocated memory.

Ladder logic provides a wide range of instructions for PLC engineers to specify control actions. Bit instructions examine status of individual input/internal bit or control a single output bit. Word instructions, such as mathematical operations, data transfer, and logical operations, operate on data words or registers. Program control instructions, such as subroutine invocation and return, control the execution flow of the payload program. For control program of large and complex ICS, subroutines are frequently used to better organize the instructions and enhancement maintainability. In addition, communication instructions allow a PLC to communicate with other hosts via a particular ICS network protocol. From the perspective of PLC control program development, a malicious payload is essentially a combination of legitimate PLC programming instructions causing disastrous impacts on an ICS. In this paper, we focus on detecting payload attacks implemented via ladder logic, but the proposed techniques are applicable to attacks written in other languages [8] as well because different PLC programming languages can be used to implement the same control system specifications [9].

C. Firmware vs. Payload Attacks

As revealed by Fig. 2, both the PLC firmware and its payload program can become the target of cyber attacks. An attacker can reverse-engineer and modify the firmware on a PLC to launch firmware attacks. In this case, even though a legitimate payload program is downloaded to the PLC, its execution will still be monitored and/or intercepted by the modified firmware. In [10], a rootkit is developed on the CODESYS PLC runtime to intercept I/O operations of the payload program. When the payload wants to read or write a certain I/O pin, interrupt handler installed by the attacker is called first, within which the attacker can reconfigure the I/O pins or modify values to be read/written. In [6], a more advanced rootkit is developed for an Allen Bradley CompactLogix PLC firmware. In addition to intercepting PLC inputs and outputs at the firmware, it incorporates physical-process awareness and always presents modified sensor measurements, hoaxing ICS operator in front of the HMI to think that the system runs normally.

Firmware attacks typically requires detailed knowledge on target PLC's hardware components and reverse-engineering of its firmware because PLCs are closed-source embedded devices [11]. An attacker needs to install the rootkit on PLCs either via the built-in remote firmware update mechanism or by loading it via JTAG interface [6]. For firmware update process protected by cryptographic means (e.g., certificate in the X.509 standard), it is hard to install a modified version of the firmware on the PLC. Alternatively, an attacker can load modified PLC firmware via JTAG interface. However, such an approach will require physical access to the PLC and possibly disassembling it.

PLC payload attacks, on the other hand, are much easier to launch. An insider with proper privileges can easily download (e.g., a disgruntled control system engineer) a malicious payload program. As shown in Fig. 1, such an insider may download a malicious payload program via the engineering workstation to one or multiple PLCs. Integrity checks on PLC payload program cannot effectively prevent such attackers from downloading malicious payload because warnings on payload program changes can always be overridden once proper privileges are acquired (e.g., a password allowing engineers to repeatedly download revised payload program for development and debugging purposes). Alternatively, sophisticated cyber attacks, such as Stuxnet [2], [3], may include payload attack as a component to induce physical damages on ICS. Partial knowledge on the physical process can be sufficient to create a malicious payload using automated tools such as SABOT [5]. In [4], a small-scale challenge shows that malicious code snippets are likely to evade the scrutiny of code reviewers. Therefore, it is necessary to develop automated payload attack detection mechanisms to protect physical infrastructure from PLC payload attacks.

D. Payload Attack Detection

As payload attacks can easily be launched by insiders or from compromised engineering workstations, several techniques that detect payload attacks have been proposed. In [12], a bump-in-the-wire device, called PLC guard, is introduced to intercept the communication between an engineering workstation and a PLC, allowing engineers to review the code and compare it against previous versions. Features of the PLC guard include various levels of graphical abstraction and summarization, which makes it easier to detect malicious code snippets. In [13], an external runtime monitoring device (e.g., a computer or an Arduino microcontroller board) sits alongside the PLC, monitors its runtime behaviors (e.g., inputs, outputs, timers, counters), and verifies them against ICS specifications converted from a trusted version of the PLC payload program and written in interval temporal logic. It is shown that functional properties of payload program can be verified against ICS specifications, but the types of payload attacks that can be detected by this approach remain to be explored.

In [14], [15], a trusted safety verifier is introduced as a bump-in-the-wire device that automatically analyzes payload

program to be downloaded onto a PLC and verifies whether critical safety properties are met using linear temporal logic. However, linear temporal logic implicitly assumes that states of the systems are observed at the end of a set of time intervals. In the case of PLC payload program, snapshot of system states is taken at the end of each program scan cycle. As a result, real-time properties that does not span multiple program scan cycles cannot be checked by the trusted safety verifier. For example, a legitimate payload program is required to energize its output immediately when a certain input pin is energized. An attacker can inject malicious code and prolong the program scan cycle to cause real-time property violation while evading code analytics based on linear temporal logic. In [16], the timer on-delay (TON) ladder logic instruction is modeled using linear temporal logic. The TON instruction starts a timer when its input condition evaluates to true and energizes its output (i.e., the "Done" bit) when the timer reaches the preset value. It is shown in [16] that TON behavior can be approximated with the combination of liveness and fairness properties: Either TON instruction is not used or TON output bit will eventually be energized. However, linear temporal logic cannot verify whether the TON output bit is energized at the exact program scan cycle designated by control system engineers. Therefore, such an approximation does not capture critical real-time requirements of ICS.

In this paper, we introduce runtime behavior modeling and monitoring of PLC payload in PLC firmware. Our proposed approach complements existing detection techniques and can detect violations of ICS real-time properties. In addition, our proposed approach does not require the introduction of any external apparatus that may introduce new vulnerabilities into ICS.

E. Runtime Behavior Monitoring for Anomaly Detection

The idea of detecting abnormal program behaviors by monitoring its execution at runtime has been applied to an rich array of computer systems. Runtime behavior monitoring techniques on operating systems such as Windows, Linux, and Android are reviewed in [17], [18]. However, these techniques cannot be directly applied to PLCs since PLCs are closed-source systems [11] running specialized firmware and payload programs. System calls utilized by existing techniques are not available in PLC systems. In [19], a runtime anomaly detector hardware design is proposed for embedded systems, which eliminates performance overheads incurred by software-based runtime monitoring methods. In [20], a timing-based PLC program anomaly detector is designed. An external data collector is deployed to collect program execution time measurements and detect unauthorized modifications to the PLC system. In [21], runtime behaviors are monitored via dedicated hardware performance counters, which are not widely available in microcontrollers utilized by PLCs. To detect payload attacks in existing ICS, runtime behavior monitoring technique must utilize only the resources available on microcontrollers used in existing PLCs and does not require external apparatus (e.g., data collector proposed in [20]).

TABLE I. CONTROL SYSTEM SPECIFICATIONS VS. LEGITIMATE PLC CONTROL LOGIC

Control System Specification	Legitimate Control System Logic
Digital I/O pins, values & functionality	Control logic of binary inputs and outputs
Analog I/O pins, value ranges, & functionality	Sensor output and actuator input ranges, control logic of analog I/Os
Legitimate sequences and timing relationships of I/O operations	Control logic of I/Os, possibly controlled by counters and timers
Network data packet and timing relationships	Data from network for local control tasks or data required by remote hosts (e.g., HMI or other networked PLCs), and real-time requirements for these network events
Network commands and timing relationships	Control tasks mandated by operator workstation and their real-time requirements

III. SYSTEM OVERVIEW

A. Adversary Model

A malicious payload may be directly downloaded by an insider with PLC programming privilege. For instance, the insider can be a PLC programmer responsible for deploying tested PLC payload program. However, he/she downloads a different payload, which may be written anew or modified from the tested version. Since such an attacker has proper privilege to program PLCs, integrity checks on PLC payload program can be overridden and will not prevent malicious payload from being downloaded. For an external attacker, security flaw of other ICS components may be exploited to gain access to an engineering workstation, which allows he/she to download malicious payload. For example, in the Stuxnet attack [2], many potential attack vectors, including the PLC programming environment, are exploited to eventually compromise a PLC-connected engineering workstation.

We assume that the attacker is not capable of changing the PLC firmware, which requires either attacking the cryptographically protected firmware image or loading modified firmware directly via JTAG interface. Therefore, firmware-level detection mechanism proposed in this paper is not tampered by the attacker. The goal of a payload attack is not limited to blocking legitimate outputs, causing system interruption, and destruction of system equipment. Sophisticated attacks such as the PLC blaster worm [22] which replicates itself to other PLCs can also be launched. However, such attacks download a payload program that are significantly different from the legitimate version in terms of program size and functionality, which can be identified by human operator monitoring the control system. In this paper, we consider stealthy payload attacks that are modified from legitimate payload programs. Such attacks preserve certain legitimate payload properties (e.g., always sending sensor readings requested by HMI) while carrying out malicious tasks.

B. PLC Program Development Process and Control System Specifications

To develop PLC payload program for an ICS, the following process is typically adopted by PLC engineers:

1) *Specification Formulation.* Control tasks to be carried out by a PLC are identified and input/output signals required by these tasks are defined. The logical sequence of operations for the PLC are specified,

e.g., in the form of sequence table, flow chart, or relay schematic [9].

2) *PLC Program Development.* At this step, PLC program is developed based on the formulated specifications. Although an engineering team usually has its own set of guidelines and best practices on program organization and documentation, the generated PLC payload always aims to accurately implement the specifications. At this stage, an attacker (e.g., a disgruntled control system engineer) may collect legitimate payload program and modify it to generate malicious payload.
3) *Testing.* Before deploying the PLC program, PLC engineers need to test the program via simulation or under some test environment. Safety properties (e.g., a circuit breaker must trip if a fault is detected) can be provided by system operators and/or identified during specification formulation. In addition, different combinations of input values are fed to the PLC to ensure that correct responses are taken under different system operation scenarios. Although the test cases may not be exhaustive (e.g., it is hard to implement all test cases when analog inputs are used), important system properties, such as safety and real-time requirements, should always be validated.
4) *Maintenance.* After an initial version of the PLC control program is deployed, the ICS may go through hardware upgrades and design improvements. Accordingly, the specifications should be updated and the PLC program should be revised. After necessary testing, the new payload is downloaded to the PLC.

In this paper, we assume that *control system specifications*, such as the number of I/Os, functionality of each I/O pin, and possible ranges of I/O values, are available. Such specifications are usually provided by the control system engineering team that develops the legitimate payload program. Table I summarizes the control system specifications required by our detection mechanism and the corresponding legitimate control system actions. For instance, when designing the legitimate payload, a digital output pin may be used to control a circuit breaker to trip. The engineering team knows whether a "0" or a "1" corresponds to the "trip" signal, so it is straightforward to generate control system specifications describing the functionality of this output pin. To implement control operation sequences (e.g., tripping a circuit breaker and then re-closing it), timers and counters are generally used. When the legitimate payload program is created, timers and counter must be properly configured to control the temporal behaviors of the payload program. These configurations can then be converted into timing relationships among I/O and network events.

C. Payload Attack Detection at PLC Firmware

Using control system specifications, runtime behavior model of legitimate PLC payload program is established and stored in the PLC firmware. The timing relationships between inputs and outputs, the number of network packets generated after different control actions, as well as timing relationships between I/O and network events, are modeled. By modifying the PLC firmware, runtime behaviors of the payload program

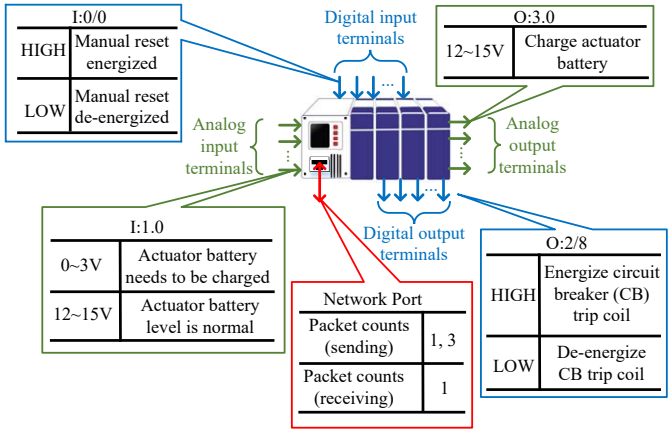

Fig. 5. PLC wiring diagram with sample control system specifications for I/O and network events. Note that wiring of I/O terminals is simplified (digital ground terminal as well as terminal pairs for each analog I/O are not shown).

(e.g., I/O and network access patterns) are time-stamped and compared against the established runtime behavior model. In addition, a backup version of the output image table is separately stored by the firmware at the beginning of each program scan cycle. If a certain abnormal runtime behavior is detected, the backup output image table is loaded to overwrite the output generated by the payload. As a result, any output related to the detected abnormal runtime behavior will not affect the physical system. For PLC payload sending/receiving network packets, network requests are also blocked when a runtime behavior anomaly is detected by the firmware.

IV. System Design

A. PLC Payload Runtime Behavior Model

Given the control system specifications, it is possible to create a *runtime behavior model* for legitimate PLC payload. Suppose that we need to create control system specifications for the PLC shown in Fig. 5. In this figure, sample specifications for I/O terminals and the network port is provided. We note that timing relationships are not shown in Fig. 5. The information categorized in Table I allows us to create the runtime behavior model as follows: First, the number of (analog and digital) I/Os and their feasible values are determined. For instance, for digital input I:0/0 in Fig. 5, its legitimate values are "1" and "0". For analog input I:1.0 (note that the notation for analog I/Os is different from that for digital I/Os as mentioned in Sec. II-B), the legitimate value ranges are 0∼3V and 12∼15V. In the PLC firmware, such information can be stored as a table (see Fig. 6 for an example), with each row storing the legitimate values/ranges of a particular pin. We call this table the *I/O event table*.

Next, the number of network packets received or sent by the legitimate payload are extracted from the specifications. Since PLC payload program is designed to control physical process, network packets are typically associated to specific I/O conditions. For instance, when an alarm signal is energized to sound a horn, the same alarm signal is usually transmitted via a network packet to the HMI at the same time. When a process data request from the HMI is received, the PLC generates process data response(s) to transmit the requested

I/O Event Table

I/O Event	Legitimate Values/Ranges
I:0/0	1 (HIGH), 0 (LOW)
⋮	⋮
I:1.0	0:3 (0~3V), 12:15 (12~15V)
⋮	⋮
O:2/8	1 (HIGH), 0 (LOW)
⋮	⋮
O:3.0	12:15 (12~15V)
⋮	⋮

Network Event Table

Network Event	Legitimate Packet Counts
receiving	1
sending	1, 3

Timing Behavior Matrix

	I:0/0:1	I:0/0:0	⋮	I:1.0:0:3	I:1.0:12:15	⋮	O:2/8:1	O:2/8:0	⋮	O:3.0:12:15	⋮	Recv:1	Send:1	Send:3
I:0/0:1													2000	
I:0/0:0														
⋮														
I:1.0:0:3				300										
I:1.0:12:15														
⋮														
O:2/8:1													500	
O:2/8:0													500	
⋮														
O:3.0:12:15														
⋮														
Recv:1													1500	1500
Send:1														
Send:3														

Fig. 6. A sample runtime behavior model established based on control system specifications in Fig.5. The model consists of two tables and a sparse matrix.

data. In the PLC firmware, network event information can be stored as a table with two rows (see Fig. 6 for an example). The first row lists the numbers of network packets that can be received, and the second row lists those that can be sent. We call this table the *network event table*. Using the I/O and network event tables, we are able to model the legitimate runtime behaviors of I/Os and network port(s) at any particular time instant.

Then, timing relationships between inputs, outputs, and network accesses are established. To store these relationships, a sparse matrix is created in the PLC firmware (see Fig. 6 for an example). We call this sparse matrix the *timing behavior matrix*. Both the rows and the columns of the matrix are indexed by legitimate I/O and network operations. For instance, the I:0/0:1 event in the matrix in Fig. 6 represents the I/O event where digital input pin I:0/0 is set to HIGH. Each column of the matrix represent a particular payload program action, whereas the rows with non-zero values represent its preconditions. For instance, the matrix in Fig. 6 indicates that there are four preconditions under which a network packet will be generated and sent by a legitimate PLC payload. Note that the non-zero value in the matrix represent the maximum time (in microseconds) within which a column event will occur.

Once all information provided in the control system specifications is converted into a runtime behavior model, three tables are stored into the PLC firmware (i.e., the I/O event table, the network event table, and the timing behavior matrix). These tables will only be updated if changes to the control system specifications are made (e.g., additions of new sensors/actuators). When a PLC payload is downloaded to a PLC, the PLC firmware assumes that its runtime behaviors match the ones specified in the supplied control system specifications. Any deviation from the encoded runtime behavior model will be regarded as an anomaly.

B. Payload Attack Detection at PLC Firmware

Our detection scheme introduces runtime behavior monitoring into the PLC firmware and compares the runtime behaviors of the currently deployed payload against the runtime behavior model established from control system specifications. To implement the proposed detection scheme, the following modifications to the PLC firmware are incorporated:

1) Logging Access to Input and Output Images: As introduced in Sec. II-A, input image is updated before each run of the payload program, and output image is updated after each run. In existing PLC firmware, I/O reads move values from the input/output image to a designated memory location. When an output pin is written, value stored in a memory location is moved to the output image table. To receive/send a packet, receive/transmit queue is either explicitly (via ladder logic instruction) or implicitly (at the end of the housekeeping phase) queried. To monitor the I/O and network access patterns, we modify the implementation of PLC firmware to log the system time-stamp of these operations. This can be achieved by setting up the memory protection unit (MPU) to enter interrupt when the user program accesses the input/output images or the network queues. In existing PLC firmware, a separate system timer is typically supported. This timer provides the time-stamps for the I/O and network events to be monitored. If I/O images are accessed, the interrupt handler decodes the I/O pin address and log the time-stamp of the operation. Suppose that the same input pin is accessed multiple times during a single program scan cycle, only the time-stamp of the first read operation is logged. For an output pin, both the first read and the last write operations are time-stamped. For access to network queues, the number of packets received/sent is logged and time-stamped. Time-stamps of I/O and network operations are stored in a separate table (known as the *runtime time-stamp table*) in the PLC firmware. Each entry of the table corresponds to a particular I/O event (e.g., a legitimate I/O value is observed) or network event (e.g., a legitimate number of packets are sent).

In our current implementation, the maximum number of time-stamps logged by the runtime time-stamp table is 10 for each I/O event. If more than 10 time-stamps are collected, newly generated time-stamps will be discarded. We log the time-stamp for the first I/O read operation and last output operation within each program scan cycle because control system specifications typically use the observation of an I/O value on the physical process as precondition. Take the output pin O:2/8 in Fig. 5 as an example. Even if the payload program operates on O:2/8 multiple times during a program scan cycle, it is the last value written into the output image that will actually take effect. For each legitimate network event, our current implementation logs a maximum of 20 time-stamps. Newly collected time-stamps will be discarded if there are already 20 time-stamps pending in the table.

2) Validating Runtime Behaviors: When time-stamping I/O and network events, any event that is not included in the I/O and network event tables is regarded as an abnormal runtime event. In addition, a separate sparse matrix (known as the *runtime sparse matrix*) is created and maintained in the PLC firmware to keep track of the timing relationships at runtime. The sparse matrix is also updated in the MPU interrupt handler. Runtime behaviors specified in the timing behavior matrix are validated in the output scan phase before the values in the output image are transferred to external output terminals. If any of the preconditions specified by the runtime behavior model are met, the timing relationships are checked. If an event occurs but none of its preconditions are active, a runtime behavior anomaly is detected. Take the timing behavior matrix in Fig. 6 as an example. Suppose that during a program scan cycle, we observe two occurrences of the event

"Send:1". For the first time-stamp of "Send:1", we check the all the time-stamps for its preconditions. If any of the timing relationships is met, the corresponding entry in the runtime sparse matrix is cleared. In the runtime time-stamp table, the oldest time-stamp for the corresponding precondition event is removed. If a violation of the timing relationship is detected, a runtime behavior anomaly is found and the execution of the payload program should be terminated. Then, for the second time-stamp of "Send:1", previously cleared precondition fields are set if the corresponding entries in runtime time-stamp table have pending time-stamps. The timing relationships for "Send:1" are then validated again.

3) Backing Up the Output Image: At the beginning of each program scan cycle (i.e., in the input scan phase), a backup version of the output image table is separately stored by the PLC firmware. Values in this backup image are simply the output of the preceding program scan cycle. If runtime behavior anomaly is detected at the current program scan cycle, the backup image is used to overwrite the output image generated by the payload program. In this way, output values corresponding to illegitimate payload program behaviors are blocked.

4) Canceling Network Send/Receive Requests: There are two scenarios where network send/receive requests generated by ladder logic instructions are processed: Network send/receive requests generated by a payload program are always processed in the housekeeping phase. To block these packets, we modify the firmware so that all pending network requests are cleared in the output scan phase if runtime behavior anomaly is detected. Alternatively, a subset of network-related ladder logic instructions can request the PLC firmware to service pending network tasks immediately. To prevent such network access, the implementation of MPU interrupt handler is further modified to check the preconditions of requested network operations. Suppose that a network-related ladder logic instruction is executed, after the network requests are generated (e.g., four packets will be retrieved from the receive queue), the firmware first enters the MPU interrupt handler and checks the preconditions of the requested network event. If any of the preconditions is met yet the corresponding timing relationship is violated, the network requests will not be executed because a runtime behavior anomaly is detected.

It should be noted that our proposed detection scheme can easily be customized to notify ICS operators of the detection of PLC payload attacks. Suppose an on-site operator is to be notified, an extra output pin can be energized to set up an alarm during the output scan phase when runtime behaviors are examined. It is also possible to send out an alarm message to a remote HMI during this phase after the runtime behavior validation is done.

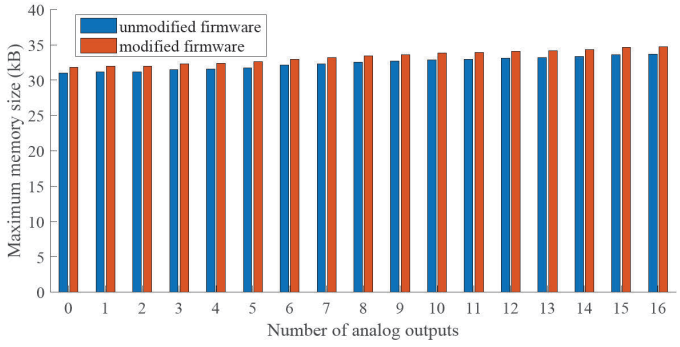

Fig. 7. Maximum memory utilization of unmodified and modified PLC firmware running PLC payload programs with different numbers of utilized analog outputs.

V. Evaluation

We implement the proposed payload attack detection method on Texas Instruments TM4C12x ARM® Cortex®-M4F core-based microcontrollers. Payload attacks are written in ladder logic, which are converted into machine code and loaded onto the PLC prototype. Hardware resources of the chosen microcontroller series are the currently active equivalents to the microcontrollers used by existing PLCs [6]. Memory protection unit (MPU) and system timer are available to implement our proposed detection scheme. Runtime behavior data collected by the PLC firmware is read from an Universal Asynchronous Receiver/Transmitter (UART) module connected to a PC. We first evaluate the overhead of implementing the proposed detection mechanism and then its detection performance.

A. Memory Overhead

Memory overhead of implementing the proposed detection method comes from both the firmware and payload levels. In the PLC firmware, runtime behavior model converted from control system specifications needs to be stored. Extra tables and sparse matrix are required to time-stamp and keep track of the runtime behaviors of the currently deployed payload. The sizes of these matrices and tables will grow as the number of I/O and network events specified in the control system specifications grows. In addition, interrupt handler for the MPU as well as initialization code for the system timer and MPU need to be added to the PLC firmware. In our prototype, these firmware modifications translate to about 200 lines of assembly code (compared to the unmodified PLC firmware with about 6000 lines of assembly code).

To evaluate whether the memory overhead of our proposed detection mechanism is acceptable, we create payload programs utilizing different numbers of I/Os and generating different numbers of network packets. Note that each of these payload programs generates two types of network events (i.e.,

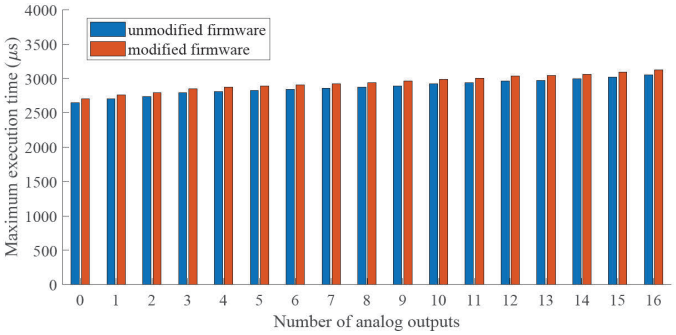

Fig. 8. Maximum execution time of PLC programs with different numbers of utilized analog outputs. All payload programs are executed on both unmodified and modified PLC firmware.

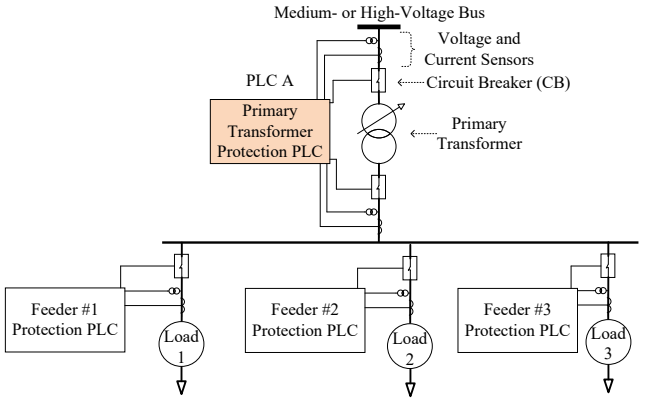

Fig. 9. Sample power substation protection system implemented by multiple PLCs. Note that our PLC prototype only emulates PLC A.

sending two packets or receiving one packet within each program scan cycle) and utilizes 16 digital I/Os. The number of analog outputs utilized by these payload programs varies from 0 to 16. Each analog output has two legitimate value ranges. The timing relationships in the control system specifications all describe preconditions for analog outputs. These payload programs are then loaded onto our PLC prototype twice: First, unmodified PLC firmware is used to execute the payload programs and the maximum sizes of the PLC firmware in the RAM are logged. Then, PLC firmware with our payload detection mechanism is used and the maximum firmware sizes are also recorded. Fig. 7 shows the memory overhead of implementing our PLC payload attack detection method in our PLC prototype. For a PLC system with 16 analog outputs, the memory overhead (compared to unmodified PLC firmware) is about 1 kB, which translates to a 3% increase in memory size. This memory overhead is acceptable for existing PLC systems on the market, which typically have more than 32 kB of memory [9].

B. Execution Time Overhead

PLC payload program needs to satisfy execution time requirements in order to control physical process correctly. If a program scan cycle takes too long to complete, the PLC will not be able to track the changes of the physical process and generate control outputs timely. Since our payload detection mechanism incorporates runtime behavior monitoring and validations in the PLC firmware, it is necessary the ensure that execution time of the program scan cycle does not significantly increase.

TABLE II. ATTACK INSTANCES IMPLEMENTED ON PLC PROTOTYPE

Attack Instance Group	Description
Illegitimate analog inputs (Group 1, 5 instances)	Scaling factors of analog input modules are modified by attacker to generate out-of-range input values.
Illegitimate network events (Group 2, 5 instances)	When trip coils are energized, the attack payload sends process data to multiple pre-specified destinations. When process data request is received, a packet containing intentionally modified process data is sent.
Illegitimate I/O event timing (Group 3, 5 instances)	Trip coils are not energized within 1000 μs when a voltage/current fault is detected.
Illegitimate network event timing (Group 4, 5 instances)	Packet containing up-to-date process data is not sent within 500 μs after process data request is received.

TABLE III. ATTACK INSTANCES AND DETECTION RESULTS

Group/ID	1	2	3	4	5	6	7	8	9	10
1/1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1/2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1/3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1/4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1/5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2/1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2/2	✓	✓	×	×	✓	✓	×	✓	×	✓
2/3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2/4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2/5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3/1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3/2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3/3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3/4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3/5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4/1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4/2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4/3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4/4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4/5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

To evaluate the execution time overhead of the proposed detection mechanism, we measure the execution time of the payload program instances created in Sec. V-A. Each payload program are executed for 1,000 program scan cycles on both unmodified and modified PLC firmware. Note that we added six extra assembly instructions in the PLC firmware to set up an extra output pin of the prototype PLC: At the beginning of each program scan cycle, this pin is set to HIGH. At the end of each program scan cycle, this pin is set to LOW. Fig. 8 shows the maximum execution time of the payload program instances. The average increase in maximum execution time is about 65 μs, which is far above the typical execution time of PLC payload programs (e.g., 1$\sim$10 ms [9]).

C. Detection Performance

To evaluate the detection performance of our proposed method, our PLC prototype emulates PLC A shown in Fig. 9. To implement the protection tasks assumed by PLC A, four analog inputs and two digital outputs are utilized. Our control system specifications require that both circuit breakers are tripped within 1000 μs once a voltage/current fault is detected on either side of the transformer. In addition, when process data request (sent by a PC emulating an HMI) is received, a packet containing up-to-date current and voltage readings must be sent within 500 μs. We create 20 different payload attack instances, which can be categorized into the four groups and are described in Table II. Each payload attack instance is executed for 10 times (each run consisting of 1,000 program scan cycles). Table III shows the detection results when running the payload attacks on modified PLC firmware. 19 out of the 20 payload attack instances can always be detected during our evaluation, which shows that our proposed detection mechanism can help prevent PLC payload attacks without introducing external apparatus.

One of the attack instances (Group 2, Instance 2) cannot always be detected. This attack instance either generates illegitimate outputs or transmits modified process data as network packets. When it sends network packets, it simply modifies the process data values stored in memory before they are encapsulated. The preconditions of this network events are still

met and the timing relationships are not violated. Although this attack instance can sometimes evade our detection, it can be easily identified by existing detection methods against false data injection attacks [23].

VI. Discussion

In this paper, we propose incorporating runtime behavior monitoring and establishing runtime behavior models from control system specifications to detect PLC payload attacks. Although our evaluations show that it is feasible to implement our proposed method in existing PLC firmware and achieve good detection performance, we note that further enhancements to the proposed method are possible. For instance, it is possible to encode correlations between I/O events at certain time instants during the program scan cycle (e.g., by identifying legitimate I/O combinations in the runtime behavior model). However, such an enhancement will require overly detailed control system specifications. Control system engineers may not be aware of all the legitimate I/O combinations when creating the PLC payload program. Furthermore, the memory and execution time overhead of such an enhancement will also increase. Therefore, it remains to be further evaluated whether other runtime behavior specifications should be included in our model.

Our current implementation focuses on payload attack detection instead of mitigation. Although output and network packets related to abnormal control logic are blocked, the operations of the ICS may still be affected. As our future work, we will devise better mitigation strategies for ICS with different mitigation resources.

VII. Conclusion

In this paper, we propose the detection of PLC payload attacks via runtime behavior monitoring in PLC firmware. Through modeling and monitoring the runtime behaviors, our proposed firmware enhancements can detect abnormal runtime behaviors of malicious payload. Using our proof-of-concept PLC prototype, we show that the proposed approach can identify a wide variety of PLC payload attacks revealed by prior research. In addition, our evaluations show that the execution time and memory overhead of the proposed detection mechanism are acceptable for existing PLC firmware. Our proposed approach complements existing bump-in-the-wire solutions in that it can detect payload attacks that violate real-time requirements of ICS operations.

Acknowledgment

This work is supported by the U.S. Department of Energy (DoE) under Award Number DE-OE0000779.

References

[1] E. R. Alphonsus and M. O. Abdullah, "A Review on the Applications of Programmable Logic Controllers (PLCs)," *Renewable and Sustainable Energy Reviews*, vol. 60, no. Supplement C, pp. 1185–1205, July 2016.

[2] D. Kushner, "The Real Story of Stuxnet," *IEEE Spectrum*, vol. 50, no. 3, pp. 48–53, March 2013.

[3] N. Falliere, L. O. Murchu, and E. Chien, "W32. Stuxnet Dossier," *White Paper, Symantec Corp., Security Response*, vol. 5, no. 6, 2011.

[4] N. Govil, A. Agrawal, and N. O. Tippenhauer, "On Ladder Logic Bombs in Industrial Control Systems," *arXiv:1702.05241 [cs.CR]*, February 2017.

[5] S. McLaughlin and P. McDaniel, "SABOT: Specification-Based Payload Generation for Programmable Logic Controllers," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, 2012, pp. 439–449.

[6] L. Garcia and S. A. Zonouz, "Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit," in *Proceedings of the 2017 Network and Distributed System Security Symposium (NDSS '17)*, 2017.

[7] A. Rullán, "Programmable Logic Controllers versus Personal Computers for Process Control," *Computers & Industrial Engineering*, vol. 33, no. 1, pp. 421–424, October 1997.

[8] "Programmable Controllers - Part 3: Programming Languages," International Electrotechnical Commission (IEC), International Standard, February 2013.

[9] F. Petruzella, *Programmable Logic Controllers*, 5th ed. New York, NY, USA: McGraw-Hill Education, 2017.

[10] A. Abbasi and M. Hashemi, "Ghost in the PLC: Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack," in *Black Hat Europe '16*, November 2016, pp. 1–35.

[11] L. Cojocar, K. Razavi, and H. Bos, "Off-the-Shelf Embedded Devices as Platforms for Security Research," in *Proceedings of the 10th European Workshop on Systems Security (EuroSec'17)*, April 2017, pp. 1:1–1:6.

[12] J. O. Malchow, D. Marzin, J. Klick, R. Kovacs, and V. Roth, "PLC Guard: A Practical Defense against Attacks on Cyber-Physical Systems," in *2015 IEEE Conference on Communications and Network Security (CNS)*, September 2015, pp. 326–334.

[13] H. Janicke, A. Nicholson, S. Webber, and A. Cau, "Runtime-Monitoring for Industrial Control Systems," *Electronics*, vol. 4, no. 4, pp. 995–1017, December 2015.

[14] S. E. McLaughlin, S. A. Zonouz, D. J. Pohly, and P. D. McDaniel, "A Trusted Safety Verifier for Process Controller Code," in *Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[15] S. Zonouz, J. Rrushi, and S. McLaughlin, "Detecting Industrial Control Malware Using Automated PLC Code Analytics," *IEEE Security & Privacy*, vol. 12, no. 6, pp. 40–47, November 2014.

[16] O. Rossi and P. Schnoebelen, "Formal Modeling of Timed Function Blocks for the Automatic Verification of Ladder Diagram Programs," in *Proceedings of the 4th International Conference on Automation of Mixed Processes - Hybrid Dynamic Systems (ADPM2000)*, 2000, pp. 177–182.

[17] N. Delgado, A. Q. Gates, and S. Roach, "A Taxonomy and Catalog of Tuntime Software-Fault Monitoring Tools," *IEEE Transactions on Software Engineering*, vol. 30, no. 12, pp. 859–872, December 2004.

[18] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A Survey on Malware Detection Using Data Mining Techniques," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 41:1–41:40, October 2017.

[19] S. Lu, M. Seo, and R. Lysecky, "Timing-Based Anomaly Detection in Embedded Systems," in *The 20th Asia and South Pacific Design Automation Conference*, January 2015, pp. 809–814.

[20] S. Dunlap, J. Butts, J. Lopez, M. Rice, and B. Mullins, "Using Timing-Based Side Channels for Anomaly Detection in Industrial Control Systems," *International Journal of Critical Infrastructure Protection*, vol. 15, no. Supplement C, pp. 12–26, 2016.

[21] X. Wang, C. Konstantinou, M. Maniatakos, R. Karri, S. Lee, P. Robison, P. Stergiou, and S. Kim, "Malicious Firmware Detection with Hardware Performance Counters," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 160–173, July 2016.

[22] R. Spenneberg, M. Brüggemann, and H. Schwartke, "PLC-Blaster: A Worm Living Solely in the PLC," in *Black Hat Asia '16*, 2016.

[23] R. Deng, G. Xiao, R. Lu, H. Liang, and A. V. Vasilakos, "False Data Injection on State Estimation in Power Systems – Attacks, Impacts, and Defense: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 411–423, April 2017.