Chapter 4

ATTACKING THE IEC 61131 LOGIC ENGINE IN PROGRAMMABLE LOGIC CONTROLLERS

Syed Ali Qasim, Adeen Ayub, Jordan Johnson and Irfan Ahmed

Abstract Programmable logic controllers monitor and control physical processes in critical infrastructure assets, including nuclear power plants, gas pipelines and water treatment plants. They are equipped with control logic written in IEC 61131 languages such as ladder diagrams and structured text that define how the physical processes are monitored and controlled. Cyber attacks that seek to sabotage physical processes typically target the control logic of programmable logic controllers.

Most of the attacks described in the literature inject malicious control logic into programmable logic controllers. This chapter presents a new type of attack that targets the control logic engine that is responsible for executing the control logic. It demonstrates that a control logic engine can be disabled by exploiting inherent features such as the program mode and starting/stopping the engine. Case studies involving control logic engine attacks on real programmable logic controllers are presented. The case studies present internal details of the logic engine attacks to enable industry and the research community to understand the control logic engine attack vector. Additionally, control engine attacks on power substation, conveyor belt and elevator testbeds are presented to demonstrate their impacts on physical systems.

Keywords: Programmable logic controllers, IEC 61131 logic engine, attacks

1. Introduction

Industrial control systems monitor and control infrastructure assets such as electric power grids, nuclear plants, water treatment facilities and gas pipelines [2, 3, 14]. An industrial control system environment comprises a control center and field sites. The control center houses human-machine interfaces (HMIs) and engineering workstations, and

Published by Springer Nature Switzerland AG 2022
J. Staggs and S. Shenoi (Eds.): Critical Infrastructure Protection XV, IFIP AICT 636, pp. 73–95, 2022.
https://doi.org/10.1007/978-3-030-93511-5_4

the field sites house the actual industrial (physical) processes that are monitored and controlled via sensors, actuators and programmable logic controllers (PLCs). Programmable logic controllers are embedded devices that monitor and control industrial processes [5]. They incorporate control logic programs written in IEC 61131 languages such as ladder diagrams, structured text and instruction lists that define how physical processes are to be monitored and controlled.

The control logic of a programmable logic controller is typically targeted by a cyber attack to sabotage a physical process [6]. Most control logic attacks described in the literature inject malicious control logic into programmable logic controllers over networks [4, 7, 13, 18, 19, 21, 25, 27, 28]. For example, the Stuxnet malware targeted Iran's uranium-235 processing facility by injecting control logic into Siemens Step 7 engineering software and S7-300 programmable logic controllers [9].

This chapter presents a new type of control logic attack that targets the control logic engine of a programmable logic controller, which executes the control logic. It demonstrates that the control logic engine can be disabled by exploiting inherent features such as the program mode and starting/stopping the engine. Two case studies drawing on attacks listed in the MITRE ATT&CK knowledge base [16], such as man-in-the-middle, unauthorized command message, loss of availability, denial of control and manipulation of control, are presented.

The first case study involves the Schweitzer Engineering Laboratory SEL-3505 Real-Time Automation Controller (RTAC) that is equipped with security features such as device access control and traffic encryption. The second case study involves three traditional programmable logic controllers with no security features, Schneider Electric Modicon M221, Allen-Bradley MicroLogix 1100 and 1400 programmable logic controllers.

The two case studies present internal details of the logic engine attacks, including proprietary programmable logic controller communications protocols to enable industry and the research community to understand the control logic engine attack vectors. Additionally, control engine attacks on power substation, conveyor belt and elevator testbeds are presented to demonstrate their impacts on physical systems.

2. Background and Related Work

This section provides an overview of industrial control systems and discusses related research efforts.

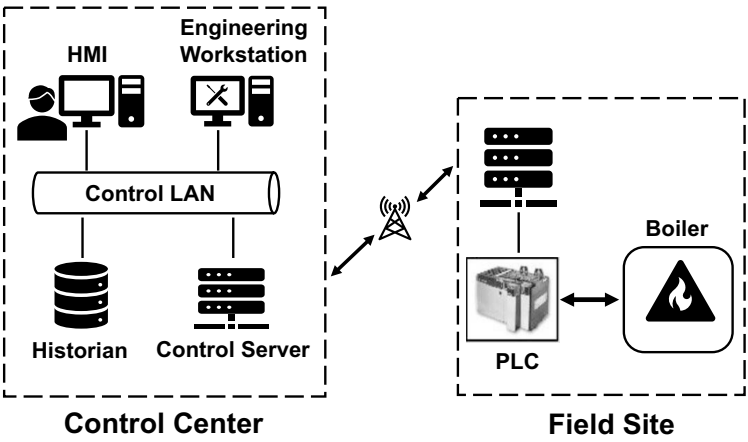


Figure 1. Industrial control system.

2.1 Industrial Control Systems

Figure 1 shows an industrial control system that operates a steam generation system. The industrial control system has a field site that houses the physical steam generation process system and a control center that operates the physical system in a safe and efficient manner. The physical process is monitored and controlled using sensors, actuators and a programmable logic controller. Water entering the boiler is converted to steam, which is transported via a pipeline. The programmable logic controller receives sensor data about parameters such as water level, pressure and temperature in the boiler. It processes the sensor data using control logic to adjust the parameter values by operating valves. The programmable logic controller also sends process state data to the control center over a network.

The control center has a human-machine interface, historian, control server and engineering workstation. The human-machine interface shows operators the current state of the physical process. The historian logs programmable controller inputs (sensor data) and outputs (control actions) for analytic and forensic purposes. The control server communicates with the field site over a network. The engineering workstation runs engineering software provided by the programmable logic controller vendor to program, configure and maintain the programmable logic controller remotely. A control engineer can create a control logic program using the engineering software and proceed to download (write) it to the programmable logic controller or upload (read) the code running on the

programmable logic controller. The IEC 61131-3 standard recommends five control logic programming languages, ladder diagrams, sequential function charts, function block diagrams, structured text and instruction lists, for writing control logic programs.

2.2 Related Work

Control logic attacks described in the research literature either target control logic code running on programmable logic controllers (also referred to as control logic injection attacks) [12, 22, 26] or compromise programmable logic controller firmware to manipulate control logic execution [10]. The attacks described in this chapter are novel in that they target control logic engines instead of the control logic code in targeted programmable logic controllers without modifying the programmable logic controller firmware.

Senthival et al. [25] have presented three types of denial of engineering operations attacks. In the first type of attack, an attacker intercepts network traffic between an engineering workstation and targeted programmable logic controller and replaces the original ladder diagram with an infected diagram or vice versa when it is downloaded or uploaded, respectively. In the second type of attack, an attacker with man-in-the-middle access replaces part of the original ladder diagram with noise when it is uploaded, causing the engineering software to crash. The third type of attack, which also crashes the engineering software, only requires the attacker to remotely download infected control logic to the targeted programmable logic controller.

Kalle et al. [13] have presented a control logic injection attack that involves four phases. The attack first compromises a programmable logic controller and steals its control logic. The stolen binary is decompiled, following which malicious logic is injected and the modified code is compiled and transferred to the programmable logic controller. The malicious logic is hidden from the engineering software by employing a virtual programmable logic controller that captures network traffic corresponding to the original logic and sends it to the engineering software when it attempts to read the modified control logic in the programmable logic controller.

McLaughlin and McDaniel [15] have presented a similar control logic injection attack. The control logic bytecode of a targeted programmable logic controller is downloaded and decompiled into a logical model in order to find mappings between the devices connected to the programmable logic controller and the variables in the control logic. The attack changes the mappings arbitrarily and downloads the control logic to the pro-

grammable logic controller to cause harm. This attack requires knowledge about industrial control system operations.

Yoo and Ahmed [27] have presented two control logic injection attacks, data execution, and fragmentation and noise padding. The data execution attack exploits the fact that a programmable logic controller does not enforce data execution prevention and transfers malicious control logic to its data blocks. The attack then changes the control flow of the programmable logic controller to execute the logic in its data blocks. In contrast, the fragmentation and noise padding attack subverts deep packet inspection by sending write requests using malicious control logic. Each write request packet contains one byte of the control logic and the rest of the packet contains noise. Every subsequent write request attempts to overwrite the memory region of the programmable logic controller that was written with noise by the previous request.

Govil et al. [11] incorporated malware in a ladder logic bomb that is inserted into the control logic of a programmable logic controller. Such a logic bomb is difficult to detect by manually validating the control logic in a programmable logic controller. The logic bomb is activated by a trigger signal or it can execute autonomously to disrupt or damage the physical system over time.

Garcia et al. [10] have developed a model-aware rootkit that was incorporated in programmable logic controller firmware using JTAG (Joint Test Action Group) ports [20]. The rootkit generates fake, but seemingly authentic, inputs from legitimate inputs. The programmable logic controller processes the authentic inputs according to its control logic and generates output commands to actuators. However, the rootkit blocks the outputs of the programmable logic controller at the firmware level and instead sends malicious outputs generated from the fake inputs to the sensors. This deceives a control engineer who monitors the physical process via the human-machine interface.

Schuett et al. [22] have evaluated the possibility of modifying programmable logic controller firmware to execute remotely-triggered attacks. They reverse engineered programmable logic controller firmware, introduced malicious modifications and repackaged and installed the modified firmware in the programmable logic controller. The compromised firmware enabled time-based and remotely-triggered denial-of-service attacks on the programmable logic controller.

3. Attacking Control Logic Engines

This section describes the adversarial model and the attacks used in the programmable logic controller case studies.

Table 1. Attacks executed in the case studies.

Programmable Logic Controller	**MITRE ATT&CK Knowledge Base Attacks**
SEL-3505 RTAC	Network sniffing, man-in-the-middle, manipulation of control
Modicon M221	Network sniffing, unauthorized command message, loss of availability, denial of control, manipulation of control
MicroLogix 1100	Network sniffing, unauthorized command message, denial of control, manipulation of control, man-in-the-middle, denial of view
MicroLogix 1400	Network sniffing, unauthorized command message, denial of control, manipulation of control, man-in-the-middle, denial of view

The adversarial model assumes that the attacker is in the industrial control network and can communicate with the targeted programmable logic controller to launch a control logic engine attack. The attacker may use a common information technology system attack such as an infected USB drive or vulnerable webserver to infiltrate the industrial control network and proceed to execute the control logic attack on the targeted programmable logic controller. The attacker has the following capabilities in the industrial control network:

- Reading communications between the targeted programmable logic controller and engineering workstation.
- Initiating connections with the targeted programmable logic controller to send malicious messages remotely.
- Dropping or modifying messages in communications by assuming a man-in-the-middle position.

The two case studies demonstrate IEC 61131 control logic engine attacks on four programmable logic controllers. A control logic engine attack is defined as an attack that disrupts or impairs the normal functioning of a control logic engine. The case studies focus on attacks that stop control logic engines from executing control logic. Various subsets of attacks listed in the MITRE ATT&CK knowledge base [16] are employed on the programmable logic controllers (Table 1). The attacks, their MITRE ATT&CK knowledge base identifiers and descriptions are as follows:

- **Network Sniffing (T0842):** The attack sniffs network traffic to gain information about the physical process and its control strategy.
- **Man-in-the-Middle (T0830):** The attack intercepts, modifies and/or drops packets transmitted between the engineering workstation and programmable logic controller.
- **Unauthorized Command Message (T0855):** The attack sends an unauthorized command message to an industrial control system device to make it function incorrectly.
- **Loss of Availability (T0826):** The attack disrupts an industrial control system device to prevent the delivery of products and/or services.
- **Denial of Control (T0813):** The attack temporarily prevents the control of the physical process.
- **Manipulation of Control (T0831):** The attack manipulates the control of the physical process.
- **Denial of View (T0815):** The attacker disrupts or prevents an operator from viewing the status of the physical process.

4. Case Study 1: SEL-3505 RTAC

The first case study focuses on a Schweitzer Engineering Laboratories SEL-3505 Real-Time Automation Controller (RTAC) with security features such as encrypted traffic and device-level access control. The SEL-3505 RTAC incorporates a logic engine that executes its control logic. The attack, which is launched from a man-in-the-middle position between the engineering software and programmable logic controller, prevents the controller from executing the control logic in one of two ways – by modifying the packet that starts the logic engine or by dropping the packet entirely. Note that the initial communications with the controller are encrypted using transport layer security (TLS). Unencrypted communications begin only after a legitimate user has logged in, which prevents a man-in-the-middle attack.

4.1 Controller Details

The SEL-3505 RTAC is equipped with an IEC 61131 control logic engine. It has a web interface for monitoring and configuring the network interface, system logs, user accounts and security settings. A control engineer can write the control logic, configure communications protocols,

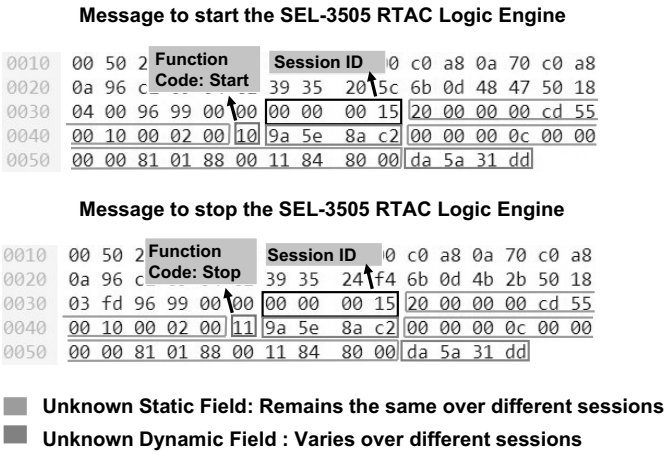


Figure 2. Messages sent to the SEL-3505 RTAC to start and stop its logic engine.

read/write projects, and start or stop the logic engine using the SEL-5033 AcSELerator RTAC software [23].

The SEL-3505 RTAC uses the ex-GUARD [17] system to control task execution. All tasks that are not approved by the whitelist are blocked [23]. The device communicates with the AcSELerator software on port 5432. Most communications, including session establishment, user authentication and reading/writing projects from/to the device, are encrypted using TLS encryption. However, after a user logs in, the SEL-3505 RTAC opens port 1217 and starts a second communications channel for sharing its state in real time. Surprisingly, the communications on port 1217 are unencrypted.

4.2 Vulnerabilities and Attacks

Exploration of the SEL-3505 RTAC communications internals revealed that it receives unencrypted commands on port 1217 to start and stop the logic engine. Additionally, the packets carrying the commands could be identified. Reverse engineering the commands enabled the understanding of function codes and other fields such as session IDs.

Figure 2 shows the two request packets sent by the AcSELerator software to the SEL-3505 RTAC to start and stop its logic engine. The session ID is incremented by three in each new session and the function codes for starting and stopping the logic engine are `0x10` and `0x11`, respectively. The other messages remain the same in different sessions

```
Input: TCP Packet
1. if (packet_src == AcSELerator & packet_dst == RTAC & packet_port == 1217)
    2. if (packet_payload_contains (static_fields))
        3. Modify/Drop
```

Figure 3. Pseudocode for the Ettercap filters.

(identified as unknown static fields) and their semantics are not required to launch attacks.

The following attacks listed in the MITRE ATT&CK knowledge base were launched in the case study:

- **Network Sniffing (T0842):** Network sniffing is the first step in finding vulnerabilities to launch subsequent attacks. Since the sniffer is in the same network as the engineering workstation and programmable logic controller, network traffic can be captured and subsequently analyzed to gain information needed to launch attacks. The information includes the port on which unencrypted communications are sent as well as specific message fields that are used to design Ettercap filters [8].

- **Man-in-the-Middle (T0830):** After sniffing the network traffic and identifying the packets responsible for starting/stopping the logic engine, Ettercap filters are used to poison the ARP caches of the target machines and assume a man-in-the-middle position between the AcSELerator software and SEL-3505 RTAC. The contents of packets sent by the engineering software to the device are then modified or the packets are simply dropped.

- **Manipulation of Control (T0831):** The man-in-the-middle position is leveraged to stop the control logic engine of the SEL-3505 RTAC from executing its control logic, which impacts the control of the physical process.

The information obtained via network sniffing was leveraged to create two Ettercap filters (DropFilter and Start-StopFilter) that identify messages containing the logic engine start and stop commands. The commands in the packets sent to the SEL-3505 RTAC can be modified (i.e., by converting start to stop and vice versa) and the packets transmitted or the packets could be simply dropped to stop the logic engine or prevent a control engineer from accessing the logic engine. Figure 3 shows pseudocode corresponding to the Ettercap filters.

The filters first identify messages sent between the AcSELerator software and SEL-3505 RTAC using their IP address and port 1217, respectively. As shown in Figure 2, large numbers of packets containing the start and stop commands are the same in different sessions (termed as unknown static fields). The filters search these static bytes in the TCP payloads of the messages to identify the right messages. After correctly identifying a message, DropFilter uses the `drop()` command to drop the message. Similarly, Start-StopFilter changes the function code located at index 15 in the TCP payload from start (`0x10`) to stop (`0x11`) and vice versa.

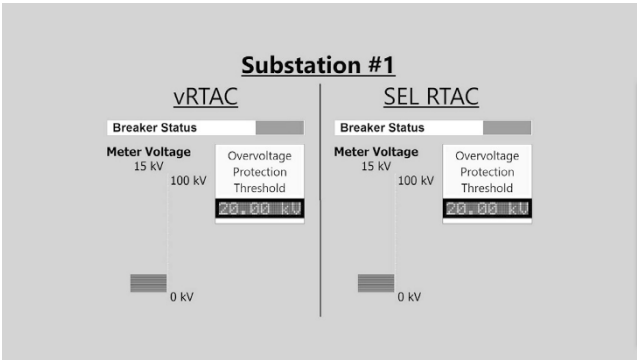
4.3 Experimental Evaluation

The control logic engine attack was evaluated on a power substation testbed comprising an SEL-3505 RTAC connected to an engineering workstation, circuit breaker and emulated voltage measurement device designed to behave as a voltmeter. The SEL-3505 RTAC was configured to open the circuit breaker when the voltmeter reports a voltage level higher than a specified threshold. Expensive power equipment could be damaged or destroyed if the circuit breaker does not open when the voltage rises beyond the threshold.

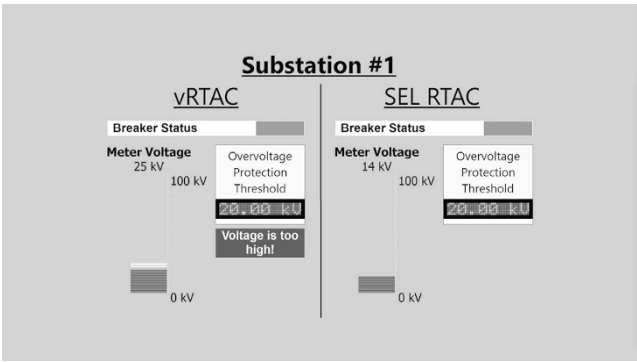
The system is monitored from a human-machine interface as shown in Figure 4(a). Note that vRTAC shows the ground truth of the system even when the SEL-3505 RTAC has failed. Specifically, it shows the inconsistency between the true state and the state reported by the device during the attack.

When the SEL-3505 RTAC starts up, it automatically enables the logic engine, so the first step is to stop the logic engine. This is a typical operation when system maintenance or reprogramming is performed. At this point, ARP spoofing attacks are launched against the SEL-3505 RTAC and engineering workstation using the Ettercap packet filters. When an operator sends the command to start the logic engine, it is intercepted and the function code is modified or the packet is dropped before it reaches the SEL-3505 RTAC.

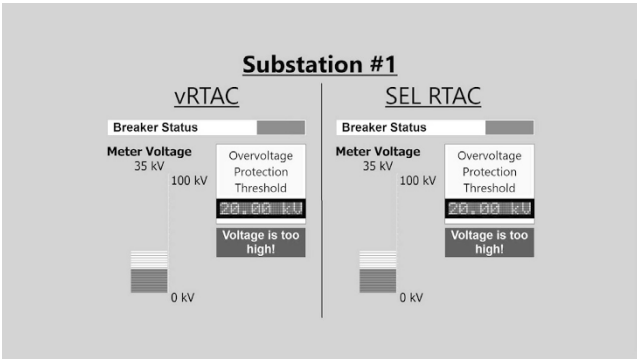
Since the logic engine fails to start even after the operator sends multiple start commands, the device is unable to control or monitor the power system. During this time, if the voltmeter detects a high voltage, such as from a short circuit in the system, a controller is not in place to open the circuit breaker. Therefore, power is allowed to flow and reach critical transformers, potentially damaging or destroying them and causing a power outage. This state is shown in Figure 4(b). Specifically, the



(a)



(b)



(c)

Figure 4. Impact of the SEL-3505 RTAC control engine attack.

SEL-3505 RTAC no longer reports an up-to-date voltage value and the breaker is not opened when the voltage rises above the threshold.

After the Ettercap attack ends, an operator can restart the SEL-3505 RTAC logic engine. The SEL-3505 RTAC is then able to detect the high voltage from the voltmeter and the circuit breaker is opened. Figure 4(c) shows the state after the logic engine is enabled. Although the breaker is opened, the action would be too late to prevent equipment damage.

5. Case Study 2: Traditional Controllers

The second case study focuses on three traditional programmable logic controllers without built-in security features: Schneider Electric Modicon M221 and Allen-Bradley MicroLogix 1100 and 1400 devices. The three programmable logic controllers differ from the SEL-3505 RTAC programmable logic controller used in the first case study in two ways. First, the programmable logic controllers do not have separate logic engines; instead, their processors assume the role. Depending on the programmable logic controller, it needs to be in the run mode or be given a start controller command to execute the control logic. Second, unlike the SEL-3505 RTAC, communications are not encrypted. The control logic engine is attacked by crafting a malicious message that is sent to the programmable logic controller to remotely change its state.

5.1 Case Study 2(a): Modicon M221

The Schneider Electric Modicon M221 device is a nano programmable logic controller designed to control manufacturing processes. A control engineer can write control logic, monitor a physical process and control the state of the Modicon M221 device using the vendor-provided SoMachine Basic engineering software. SoMachine Basic supports two IEC 61131 control logic languages, ladder diagrams and instruction lists. A control engineer downloads the control logic program to an M221 device by writing to its memory. The engineer can also start or stop the execution of control logic by the logic engine via SoMachine Basic. Communications between an M221 device and SoMachine Basic, which employ a proprietary protocol on port 502, are not encrypted. The proprietary protocol is encapsulated in Modbus/TCP. An M221 device only allows one connection at a time.

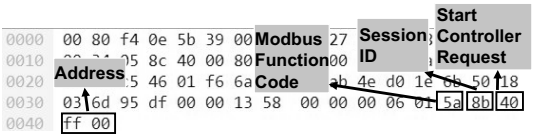
Vulnerabilities and Attacks. The principal vulnerability is that communications between an M221 device and its engineering software are not encrypted. Two other vulnerabilities are that the M221 device states that execute and disable the execution of the control logic pro-

gram can be changed remotely via the engineering software and an M221 device can only have a single connection to the engineering software at a time.

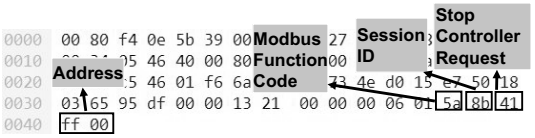
The following attacks listed in the MITRE ATT&CK knowledge base were launched in the case study:

- **Network Sniffing (T0842):** Network sniffing enables the communications between the SoMachine Basic engineering software and M221 device to be captured. Protocol information obtained by analyzing the captured traffic can be leveraged in subsequent attacks.
- **Unauthorized Command Message (T0855):** Protocol information obtained by analyzing the captured network traffic is used to create a message that stops an M221 device from executing its control logic program. Since the communications are not encrypted, any crafted message can be sent to an M221 device remotely.
- **Loss of Availability (T0826):** An M221 device allows only a single connection at a time. Therefore, an attack that does not close the session that was established to send the crafted message, could prevent a control engineer from connecting to and communicating with the targeted M221 device.
- **Denial of Control (T0813):** An M221 device allows only a single connection at a time. Therefore, an attack that does not close the session that was established to send the crafted message, could prevent a control engineer from using the targeted M221 device to control the physical process.
- **Manipulation of Control (T0831):** Stopping the control logic engine of an M221 device from executing its control logic impacts the control of the physical process.

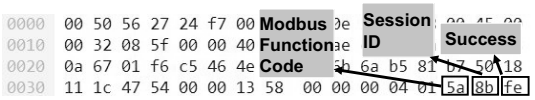
Differential analysis as well as manual analysis were employed to reverse engineer the proprietary M221 device protocol and identify the packets sent by the SoMachine Basic software to start and stop the logic engine of an M221 device. Figures 5(a) and 5(b) show the packets. The function codes `0x40` and `0x41` are used to start and stop an M221 logic engine, respectively. As shown in Figure 5(c), an M221 device sends a success message to the SoMachine Basic software to acknowledge its change of state. This information was leveraged to write a Python script that establishes a session with an M221 device and sends crafted messages to start and stop the execution of its control logic.



(a) Request message to start the Modicon M221 device.



(b) Request message to stop the Modicon M221 device.



(c) Response message with success function code.

Figure 5. Messages sent to start and stop the Modicon M221 device.

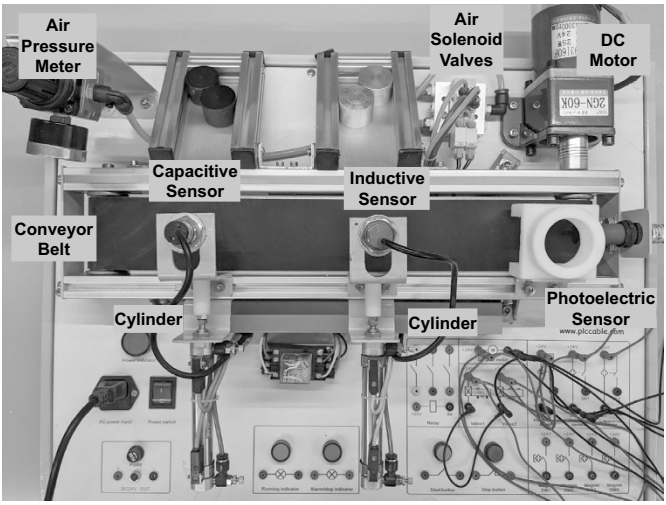


Figure 6. Top view of the conveyor belt testbed.

Experimental Evaluation. The control logic engine attack was evaluated using the conveyor belt testbed shown in Figure 6. The conveyor

belt, which sorts different types of objects with the help of sensors and actuators, is controlled by an M221 device. The SoMachine Basic software runs on a Windows 7 virtual machine while the attack scripts run on an Ubuntu 16.04 virtual machine.

It is assumed that the attacker has infiltrated the network containing the M221 device and engineering workstation. The attacker uses network scanning to identify the IP address of the M221 device. Next, the attacker initiates a Modbus protocol session with the M221 device and sends it a stop controller request. Upon receiving the request, the M221 device stops executing its control logic, which halts the conveyor belt. However, the attacker does not terminate the Modbus session. Because the M221 device can only have one session at a time, a control engineer is unable to communicate with the M221 device and issue a start controller request to get the conveyor belt operational.

5.2 Case Study 2(b): MicroLogix 1100 and 1400

The Allen-Bradley MicroLogix 1100 and 1400 programmable logic controllers belong to same family and have many similarities. Both the devices are monitored and controlled via the RSLogix 500 engineering software and use the unencrypted PCCC protocol encapsulated in EtherNet/IP to communicate with the RSLogix 500 software [24]. The RSLogix 500 software supports ladder diagrams for writing control logic. Upon connecting with a MicroLogix device, a control engineer can upload the control logic (read the control logic on the device) or download new control logic (write to device memory). The two MicroLogix devices have three modes of operation, Run, Program and Remote, that are set and changed using a command line interface. In the Run mode, a device executes the control logic and controls a physical process. To access or change the control logic, a control engineer places the device in the Program mode, which also pauses the logic engine. For operational ease, the device is often placed in the Remote mode, which enables a control engineer to change the mode from Run to Program and vice versa remotely using the engineering software.

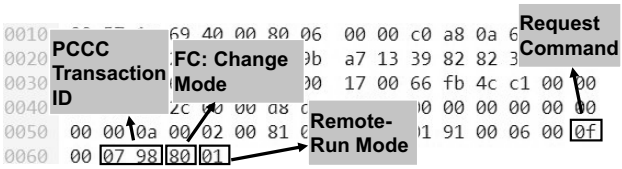
Vulnerabilities and Attacks. The ability to change the operational modes of MicroLogix 1100 and 1400 devices is exploited. Specifically, changing the mode to Program pauses control logic execution and the device waits for an operator to update its control logic and configuration.

The following attacks listed in the MITRE ATT&CK knowledge base were launched in the case study:

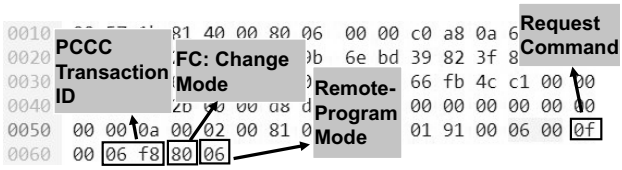
- **Network Sniffing (T0842):** Network sniffing provides valuable information about the communications protocol used by the MicroLogix devices and RSLogix 500 engineering software. Analysis of the captured traffic helps identify the packets responsible for changing the device state to the Program mode, which pauses control logic execution.
- **Unauthorized Command Message (T0855):** The information obtained by network sniffing is leveraged to craft a message that remotely changes the MicroLogix device state from the Run mode to the Program mode.
- **Denial of Control (T0813):** Pausing the control logic program prevents a control engineer from controlling the physical process.
- **Manipulation of Control (TO831):** Pausing the control logic program impacts the control of the physical process.
- **Man-in-the-Middle (TO830):** The change to the MicroLogix device state is hidden from the control engineer by poisoning the ARP caches of the engineering software and MicroLogix device to achieve a man-in-the-middle position, following which the MicroLogix device state is modified from Program to Run whenever the engineering software requests the device state.
- **Denial of View (T0815):** The man-in-the-middle attack deceives the control engineer who assumes that the MicroLogix device is still in the Run mode and is controlling the physical process.

Manual reverse engineering was employed to identify the request messages sent by the RSLogix 500 software to change the device mode to Remote-Run or Remote-Program. Figures 7(a) and 7(b) show the messages sent to place MicroLogix devices in the Remote-Run and Remote-Program modes. The function code `0x80` changes the mode, following which the function codes `0x01` or `0x06` place the devices in the Remote-Run or Remote-Program modes, respectively.

It was also determined that RSLogix 500 software periodically requests the device status. Figure 8(a) shows that the function code (FC) `0x03` is used to request device status. Differential analysis of the response messages for the Remote-Run and Remote-Program modes revealed that a function code of `0x21` is used for the Remote-Run mode

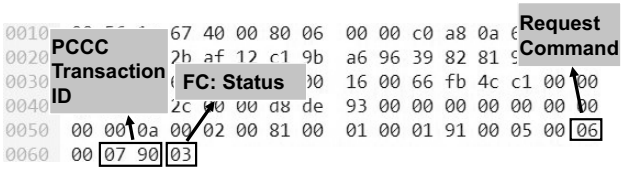


(a) Request message to set device to the Remote-Run mode.

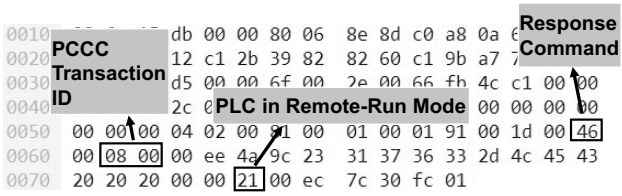


(b) Request message to set device to the Remote-Program mode.

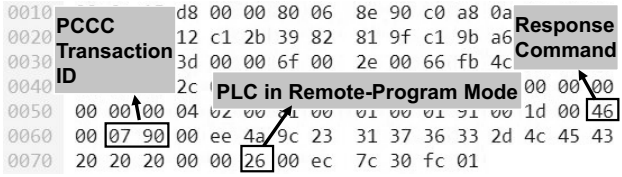
Figure 7. Messages sent to change the mode of the MicroLogix devices.



(a) Request message to device for current status.



(b) Response message from device in the Remote-Run mode.



(c) Response message from device in the Remote-Program mode.

Figure 8. Status request to and response messages from the MicroLogix devices.

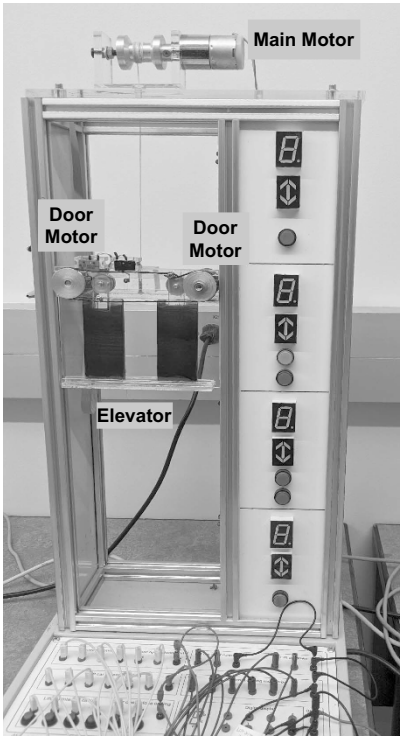


Figure 9. Front view of the elevator testbed.

(Figure 8(b)) whereas `0x26` is used for the Remote-Program mode (Figure 8(c)). This information was used to create a program that initiates a session with a targeted MicroLogix device and sends the mode change messages to place in the device in the Remote-Program mode, which pauses the execution of its control logic. Since the RSLogix 500 software periodically requests device status, the change in the device mode would be detected by an operator. To deceive the operator, an Ettercap filter was employed to detect the status response message, following which the Remote-Program function code in the message is changed to Remote-Run.

Experimental Evaluation. Since the MicroLogix 1100 and 1400 devices use the same communications protocol and function codes, the logic engine attack was only executed on a MicroLogix 1400 device. Figure 9 shows the elevator testbed used in the evaluation. The elevator has four floors. An elevator user can select a floor from inside or out-

side the elevator that is input to the MicroLogix 1400 device, which moves the elevator to the selected floor. RSLogix 500 software running on a Windows 7 virtual machine (engineering workstation) communicates with the MicroLogix 1400 device. Attacks are launched from an Ubuntu 18.04.3 LTS machine. As in the other case studies, the MicroLogix 1400 programmable logic controller, engineering workstation and attacker machine are in the same network.

The first attack step is to assume a man-in-the-middle position using ARP poisoning. Following this, a session is established with the MicroLogix 1400 device that controls the elevator. Next, a request message is send to the MicroLogix 1400 device to change its mode to Remote-Program, which causes the device to stop executing its control logic, thereby halting elevator operation. An Ettercap filter is used to detect a status response message sent by an elevator control operator and change the Remote-Program function code in the message to Remote-Run. Thus, the operator is unaware that the attack has rendered the elevator non-operational.

6. Mitigation

The principal problem is the lack of encrypted communications. The Modicon M221 and MicroLogix 1100 and 1400 programmable logic controllers do not have encryption. Although the SEL-3505 RTAC employs TLS encryption for most of its communications, the communications that occur on port 1217 are not encrypted. This makes it easier for an attacker to reverse engineer the protocol and launch a number of attacks.

The Modicon M221 and MicroLogix 1400 devices implement password authentication to prevent the control logic from being read and, in some cases, written by unauthorized users. However, an attacker can change the device state to prevent the control logic from running without any authentication. Therefore, it is important to also use password protection when issuing commands to change the device state.

The Modicon M221 device also has a default feature that enables users to connect to it without any authentication. It allows a connection to one user at a time, exposing itself to attacks that only require a successful connection to the device. Specifically, the attacker connects with the device, pauses the execution of its control logic and keeps the session active to prevent a legitimate user from accessing the device. Such attacks can be defeated by requiring a password to establish a connection with the device.

Man-in-the-middle attacks on all the programmable logic controllers considered in the case studies can be prevented using DHCP snooping and ARP inspection [1].

7. Conclusions

Most of the attacks described in the literature inject malicious control logic into programmable logic controllers. In contrast, this chapter has presented a novel type of attack that targets the control logic engine that is responsible for running the control logic. These attacks disable control logic engines by exploiting inherent features such as starting and stopping the engines, and changing the operating modes of the programmable logic controllers. Case studies involving control engine attacks on programmable logic controllers that manage power substation, conveyor belt and elevator testbeds demonstrate the significance of the control logic engine attack vector as well as the impacts of control engine attacks on physical systems.

Acknowledgements

This chapter has been authored by UT-Battelle LLC under Contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The research was partially supported by the Virginia Commonwealth Cyber Initiative.

References

[1] H. Adjei, T. Shunhua, G. Agordzo, Y. Li, G. Peprah and E. Gyarteng, SSL stripping technique (DHCP snooping and ARP spoofing inspection), *Proceedings of the Twenty-Third International Conference on Advanced Communications Technology*, pp. 187–193, 2021.

[2] I. Ahmed, S. Obermeier, M. Naedele and G. Richard III, SCADA systems: Challenges for forensic investigators, *IEEE Computer*, vol. 45(12), pp. 44–51, 2012.

[3] I. Ahmed, S. Obermeier, S. Sudhakaran and V. Roussev, Programmable logic controller forensics, *IEEE Security and Privacy*, vol. 15(6), pp. 18–24, 2017.

[4] I. Ahmed, V. Roussev, W. Johnson, S. Senthivel and S. Sudhakaran, A SCADA system testbed for cybersecurity and forensic research and pedagogy, *Proceedings of the Second Annual Industrial Control System Security Workshop*, pp. 1–9, 2016.

[5] A. Ayub, H. Yoo and I. Ahmed, Empirical study of PLC authentication protocols in industrial control systems, *Proceedings of the IEEE Security and Privacy Workshops*, pp. 383–397, 2021.

[6] S. Bhatia, S. Behal and I. Ahmed, Distributed denial-of-service attacks and defense mechanisms: Current landscape and future directions, in *Versatile Cybersecurity*, M. Conti, G. Somani and R. Poovendran (Eds.), Springer, Cham, Switzerland, pp. 55–97, 2018.

[7] T. Chen and S. Abu-Nimeh, Lessons from Stuxnet, *IEEE Computer*, vol. 44(4), pp. 91–93, 2011.

[8] Ettercap Project, Ettercap (`www.ettercap-project.org`), 2021.

[9] N. Falliere, L. O'Murchu and E. Chien, W32.Stuxnet Dossier, Version 1.4, Symantec, Mountain View, California, 2011.

[10] L. Garcia, F. Brasser, M. Cintuglu, A. Sadeghi, O. Mohammed and S. Zonouz, Hey, my malware knows physics! Attacking PLCs with a physical-model-aware rootkit, *Proceedings of the Twenty-Fourth Annual Network and Distributed System Security Symposium*, 2017.

[11] N. Govil, A. Agrawai and N. Tippenhauer, On ladder logic bombs in industrial control systems, in *Computer Security*, S. Katsikas, F. Cuppens, N. Cuppens, C. Lambrinoudakis, C. Kalloniatis, J. Mylopoulos, A. Anton and S. Gritzalis (Eds.), Springer, Cham, Switzerland, pp. 110–126, 2018.

[12] R. Johnson, Survey of SCADA security challenges and potential attack vectors, *Proceedings of the International Conference on Internet Technology and Secured Transactions*, 2010.

[13] S. Kalle, N. Ameen, H. Yoo and I. Ahmed, CLIK on PLCs! Attacking control logic with decompilation and virtual PLCs, *Proceedings of the Network and Distributed System Security Symposium Workshop on Binary Analysis Research*, 2019.

[14] N. Kush, E. Foo, E. Ahmed, I. Ahmed and A. Clark, Gap analysis of intrusion detection in smart grids, *Proceedings of the Second International Cyber Resilience Conference*, pp. 38–46, 2011.

[15] S. McLaughlin and P. McDaniel, SABOT: Specification-based payload generation for programmable logic controllers, *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 439–449, 2012.

[16] MITRE Corporation, ATT&CK for Industrial Control Systems, Bedford, Massachusetts (`collaborate.mitre.org/attackics/index.php/Main_Page`), 2021.

[17] Office of Electricity Delivery and Energy Reliability, exe-GUARD, DOE/OE-0009, U.S. Department of Energy, Washington, DC (www.energy.gov/sites/prod/files/2017/04/f34/SEL_Exe-guard_FactSheet.pdf), 2012.

[18] S. Qasim, J. Lopez and I. Ahmed, Automated reconstruction of control logic for programmable logic controller forensics, in *Information Security*, Z. Lin, C. Papamanthou and M. Polychronakis (Eds.), Springer, Cham, Switzerland, pp. 402–422, 2019.

[19] S. Qasim, J. Smith and I. Ahmed, Control logic forensics framework using a built-in decompiler of engineering software in industrial control systems, *Forensic Science International: Digital Investigation*, vol. 33(S), article no. 301013, 2020.

[20] M. Rais, R. Awad, J. Lopez and I. Ahmed, JTAG-based PLC memory acquisition framework for industrial control systems, *Forensic Science International: Digital Investigation*, vol. 37(S), article no. 301196, 2021.

[21] M. Rais, Y. Li and I. Ahmed, Spatiotemporal G-code modeling for secure FDM-based 3D printing, *Proceedings of the Twelfth ACM/IEEE International Conference on Cyber-Physical Systems*, pp. 177–186, 2021.

[22] C. Schuett, J. Butts and S. Dunlap, An evaluation of modification attacks on programmable logic controllers, *International Journal of Critical Infrastructure Protection*, vol. 7(1), pp. 61–68, 2014.

[23] Schweitzer Engineering Laboratories, SEL-3505/SEL-3505-3 Real-Time Automation Controller (RTAC), Pullman, Washington (selinc.com/products/3505), 2021.

[24] S. Senthivel, I. Ahmed and V. Roussev, SCADA network forensics of the PCCC protocol, *Digital Investigation*, vol. 22(S), pp. S57–S65, 2017.

[25] S. Senthivel, S. Dhungana, H. Yoo, I. Ahmed and V. Roussev, Denial of engineering operations attacks on industrial control systems, *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pp. 319–329, 2018.

[26] R. Sun, A. Mera, L. Lu and D. Choffnes, SoK: Attacks on Industrial Control Logic and Formal Verification-Based Defenses, arXiv: 2006.04806 (arxiv.org/abs/2006.04806), 2020.

[27] H. Yoo and I. Ahmed, Control logic injection attacks on industrial control systems, in *ICT Systems Security and Privacy Protection*, G. Dhillon, F. Karlsson, K. Hedstrom and A. Zuquete (Eds.), Springer, Cham, Switzerland, pp. 33–48, 2019.

[28] H. Yoo, S. Kalle, J. Smith and I. Ahmed, Overshadow PLC to detect remote control logic injection attacks, in *Detection of Intrusions and Malware, and Vulnerability Assessment*, R. Perdisci, C. Maurice, G. Giacinto and M. Almgren (Eds.), Springer, Cham, Switzerland, pp. 109–132, 2019.