MULTI-PLC EXERCISE ENVIRONMENTS FOR TRAINING CYBER FIRST RESPONDERS FOR INDUSTRIAL CONTROL SYSTEMS

THESIS

Joseph K. Daoud, 2LT, USA

AFIT-ENG-MS-17-M-020

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-17-M-020

MULTI-PLC EXERCISE ENVIRONMENTS FOR TRAINING CYBER FIRST RESPONDERS FOR INDUSTRIAL CONTROL SYSTEMS

THESIS

Presented to the Faculty

Department of Computer and Electrical Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Joseph K. Daoud, B.S.C.S.

2LT, USA

March 2017

AFIT-ENG-MS-17-M-020

MULTI-PLC EXERCISE ENVIRONMENTS FOR TRAINING CYBER FIRST RESPONDERS FOR INDUSTRIAL CONTROL SYSTEMS

THESIS

Joseph K. Daoud, B.S.C.S.
2LT, USA

Committee Membership:

LTC Mason Rice, PhD
Chair

Lt Col John Pecarina, PhD
Member

Mr. Stephen Dunlap
Member

AFIT-ENG-MS-17-M-020

Abstract

When systems are targeted by cyber attacks, cyber first responders must be able to react effectively, especially when dealing with critical infrastructure. Training for cyber first responders is lacking and most existing exercise platforms are expensive, inaccessible or ineffective. This paper presents a mobile training platform which incorporates a variety of programmable logic controllers into a single system which facilitates the development of the unique skills required of cyber first responders operating in the realm of industrial control systems. The platform is modeled after a jail in the northeastern United States and was developed to maximize realism. Example training scenarios are provided to address specific skills and techniques. Results show that the platform is robust enough to conduct sustained training exercises that address a curriculum that has been proposed for cyber first responders.

Table of Contents

Page

List of Figures

List of Tables

MULTI-PLC EXERCISE ENVIRONMENTS FOR
TRAINING CYBER FIRST RESPONDERS FOR INDUSTRIAL CONTROL SYSTEMS

I. Introduction

1.1 Motivation

The importance of cyber security in today's Industrial Control Systems (ICS) is widely known and has been demonstrated through several experiments and real-world attacks such as the attack on the Ukrainian power grid in December 2015 which leveraged BlackEnergy malware [10]. Those trusted to react to cyber-related ICS events, cyber first responders, must be adequately prepared for the task. In medical emergency situations, first responders are relied on to perform triage operations and stabilize victims until the victims can be transported to a dedicated medical facility. Once a victim arrives at the medical facility, physicians take over and conduct their own assessments of symptoms and diagnose the extent of the victim's injuries. Because various symptoms could have a number of underlying causes, medical students go through rigorous education and training programs which include practical exercises, clinical rotations, internships and eventually residency before they are trusted to perform these diagnoses as licensed physicians [13].

In ICS incident response, a similar process is conducted. Usually when an event occurs, the asset owner will perform initial triage-like operations to stabilize the ICS and restore operation. Asset owners will often handle initial response actions themselves because ICS malfunctions can very quickly lead to safety risks depending

on the physical process being controlled. These initial response actions often involve reverting the ICS to a semi-automated or even manually controlled state. If there is a possibility that the event is related to a cyber attack, cyber first responders from organizations like ICS-CERT should be called in to perform an assessment of the symptoms and diagnose the attack similar to physicians at a medical facility. Given the similarities between diagnosing patients in a hospital and diagnosing computer networks, it is intuitive that cyber first responders should receive similar types of hands-on training that medical students receive during their journey to becoming licensed physicians.

1.2 Research Goals and Hypothesis

As a proof of concept, this thesis presents a portable, realistic and self-contained training platform which was built at relatively low cost. This platform contains three PLCs which interact with a single set of real lights and locks to represent jail cells and a functioning mantrap. Each of the three Programmable Logic Controllers (PLCs) can be activated and deactivated via software leveraging Y-Box technology (for a detailed description of the Y-Box, see [22]).

The research goals are:

1. Integrate multiple PLCs into a mobile training platform at relatively low cost.
2. Incorporate realistic physical components into a training platform that facilitates scenarios which address training requirements for cyber first responders.

The research hypotheses are:

1. A platform like this can implement scenarios designed to address training requirements for cyber first responders.
2. Multiple PLCs can control a single set of components without interference.

3. All required components can fit into a transportable briefcase-sized platform.

1.3 Thesis Layout

The first chapter of the thesis introduces the reader to the role of cyber first responders and the type of training they should receive. Chapter two presents background information about the current state of cyber first responder training, current training environments and the framework proposed by Yoon et al. for developing effective training and evaluation scenarios. Chapter three gives details about the training platform. Chapter four gives example training scenarios that can be implemented on the training platform. Chapter five presents the results of various tests that were performed and the achievements and limitations of the research, some of which has been left for future work.

II. Background

To address the need for cyber security in today's ICS, several organizations have been created which can provide first response capabilities for assessing and reacting to cyber incidents. For these organizations to optimally perform their activities, an effective training curriculum must be in place. The following sections of this chapter discuss existing training platforms, the current state of cyber first responder training and the training and evaluation scenario framework proposed by Yoon et al.

2.1 Training Platforms for ICS First Response

The following subsections describe the challenges associated with creating ICS exercise environments and currently existing exercise environments.

2.1.1 Differences Between IT and OT Systems.

Academic institutions, government organizations and businesses emphasize cyber security by offering a variety of certifications, training courses and degree programs [16, 1, 11]. These programs can provide a cyber first responder with the skills they need to become proficient with information security. Unfortunately, many of these courses focus only on traditional information technology (IT) systems, often neglecting the realm of operations technology (OT). While there is overlap between these types of systems with regards to security, ICSs have significant differences and additional considerations that a cyber first responder should be aware of when responding to an ICS-based incident.

One of the largest distinctions between IT and OT systems are the protocols used for network communications. Shodan lists fifteen of the most common protocols that are unique to ICSs, but many more exist [18]. Each of these protocols use different

sets of parameters and operation codes which may not be compatible with any given PLC. If a cyber first responder is to be effective, she must be familiar with these unique ICS protocols, or be able to learn new protocols quickly upon encountering them.

In addition to the unique protocols, almost every PLC vendor has their own proprietary software applications used for interacting with their PLCs. This software is not typically compatible with another vendor's PLC. While there are similarities among different vendors' software, many of the intricate details are different. The differences between the software can be compared to the differences between operating systems. While Windows and Mac provide generally similar functionality, many features are implemented differently and some features may not be implemented at all. The ability to navigate the proprietary software applications that manage PLCs is a skill that cyber first responders must have.

The final difference between IT and OT systems that will be discussed here is simply the nature of the systems themselves. That is, OT systems control *physical* processes whereas IT systems do not. This means that if something goes wrong, a physical process can be interrupted which could quickly cascade to a safety risk. Cyber first responders must be cognizant of this fact while conducting their activities. Furthermore, cyber first responders should be able to understand the underlying physical process of the ICS to effectively evaluate the incident.

2.1.2 Challenges of Constructing Effective OT Exercise Environments.

Attempts to create hands-on exercises for ICS have been made, but often lack realism because some components are simulated. Even with deliberate efforts to make these simulations accurate, they ultimately cannot account for every possible scenario that a real piece of equipment could encounter and only provide a limited portrayal of

a real piece of equipment [4]. The task of creating realistic exercises is relatively easy for IT systems because the software and hardware that cyber first responders would interact with during their work already exist in classroom environments. Furthermore, these IT systems can be easily virtualized with nearly identical configurations for low cost. This capability does not exist in the OT environment. This fact, along with limitations resulting from the cost of acquiring equipment, means that quality, hands-on exercises for ICS are far less common. It should be noted that while virtualized environments still have value, an ideal hands-on exercise for training cyber first responders would incorporate a real PLC with other genuine hardware that mimic live systems. Furthermore, because cyber first responders will probably need to work with many different models of PLCs over time, experiencing a variety of PLCs during training would be beneficial.

2.1.3 Existing Platforms.

Several ICS test platforms exist, but many of them are primarily for research and development purposes and only some are available for training. This paper does not attempt to enumerate every testbed, however some examples are highlighted in the following paragraphs. Additional information about existing platforms can be found in Holm et al.[7].

Sandia National Laboratories. Sandia National Laboratories has several testbed facilities in New Mexico including the Distributed Energy Technology Laboratory (DETL), Network Laboratory, Cryptographic Research Facility, Red Team Facility and Advanced Information Systems Lab [15]. Each of these labs contain some real and some simulated Supervisory Control and Data Acquisition (SCADA) assets for research and development purposes in various domains. For example, DETL has several electrical generation and distribution capabilities which incorporate PLCs,

however the PLCs and their security are not necessarily the primary focus of research [14].

Idaho National Laboratories (INL). Idaho National Laboratories has their own facilities in Idaho Falls [8]. Their Cyber Security testbed is eventually going to be able to connect to any of their currently existing critical infrastructure testbeds including the SCADA testbed, power grid testbed, mock chemical mixing facility, wireless testbed, and physical security testbed. These testbeds comprise a full-scale critical infrastructure test range which sits on an isolated, 890 square mile location which could make for a valuable training environment. Unfortunately, most INL learning opportunities are limited to internships for students who work alongside researchers with ongoing projects [9].

National Institute of Standards and Technology (NIST). NIST proposes guidelines and recommended practices for many fields, including cybersecurity. To evaluate their own recommendations, an ICS testbed was developed [5]. The testbed attempts to emulate real ICSs so that ICS components can be evaluated with appropriate security mechanisms in place.

SANS CyberCity. One of the only physical ICS platforms built with security training in mind is the SANS CyberCity platform. The CyberCity platform is part of the SANS SEC562 course which is geared towards penetration testers and kinetic cyber effects [17]. It is a 1:87 scale city with hands-on exercises involving railway switching junctions, a water reservoir and a power grid system. While some of the systems in CyberCity are simulated, real hardware was built into the power grid system including Allen-Bradley, Siemens and Phoenix Contact PLCs [19]. All these components make CyberCity an effective training platform, but it was also very ex-

pensive to develop. Furthermore, it is not a mobile platform. While it can be accessed remotely for training, remote training denies the participant physical interaction with components and can make troubleshooting of any potential hardware failures more difficult.

2.2 Recommended Curriculum

Butts and Glover [4] propose three core areas that should be covered in an ICS training course: (i) Industrial Control System Principles; (ii) Cyber Manipulation; and (iii) Response Coordination. Each core area has recommended blocks of instruction which covers the necessary information for proficiency.

2.2.1 Industrial Control System Principles.

The first core area provides an introduction to common components of ICSs, the cyber-physical interactions of these components, communications protocols and real-world configurations. This knowledge provides the ability to communicate effectively with ICS asset owners and operators. Knowledge of these topics will enhance the trainee's ability to identify the critical components of the ICS which may be the intended targets of attack.

2.2.2 Cyber Manipulation.

The cyber manipulation core is meant to educate trainees on ICS attacking techniques. Many of these techniques are similar to what can be found in traditional IT systems, but they are framed for ICS networks and components. By framing the activities this way, techniques which have a higher precedence in ICSs are emphasized, such as pivoting. One specific, noteworthy topic in this core is the side effects that specific tools can have on ICSs. For example, network scanning tools that are gener-

ally safe to use on IT networks may cause errant behavior to certain ICS components [20].

2.2.3 Response Coordination.

The response coordination core is primarily focused on the processes of responding to a specific incident. These include coordination of internal and external responses, identification and prioritization of critical system components, identifying the severity of the incident, steps for minimizing the severity of the incident, root cause analysis and recovery. It is recommended that these concepts be taught through immersion of trainees with realistic scenarios on genuine control systems [4].

2.3 Training Scenario Criteria

Even the best training platforms have no value unless they are used appropriately. Developing realistic training scenarios is an important, but difficult task to do well. Many training courses (see [16]) rightfully place a heavy emphasis on hands-on exercises as teaching mechanisms. As with physicians, hands-on exercises with realistic scenarios are an effective technique for teaching the additional skills required for responding to ICS-related incidents.

In addition to creating realistic scenarios, it is equally important to ensure that the participants can be effectively evaluated. This is a difficult task that does not have a generally accepted right answer. In fact, most evaluations of information security professionals are performed on a case-by-case basis through activities like cyber defense competitions. These competitions are almost entirely comprised of "capture the flag" events where participants are required to gain access to a file and retrieve or plant information [21, 12]. These evaluations are fundamentally inadequate for ICSs. In IT systems, confidentiality is most often the primary concern while

integrity and availability are usually secondary. In an ICS, it is rare for confidentiality to be the primary concern. Instead, the emphasis is almost always on integrity and availability. Because capturing a flag is a compromise of confidentiality, it is generally not an adequate criterion for evaluating ICS-based activities. In ICSs, what the attacker does after capturing the flag is much more interesting. An effective evaluation of an ICS-based scenario must incorporate the physical process being controlled in some way.

Yoon et al. [23] takes advantage of the NFPA 1410 format which is used by firefighters to develop an effective framework for evaluating the readiness of cyber first responders. This framework contains specific objectives, descriptions, evaluation criteria and accompanying references for each individual scenario that is created. Furthermore, each scenario contains a designator which describes what type of scenario it is and the skills addressed by the scenario. The framework proposed by Yoon et al. is leveraged by this research to develop training scenarios with measurable evaluation criteria.

III. Multi-PLC Training Platform

This section describes the design considerations and implementation details of the platform.

3.1 Design Considerations

The purpose of this platform is to incorporate multiple models of PLCs into a single ICS platform thereby emphasizing the differences between individual PLCs.

3.1.1 Requirements.

The platform is intended to be reasonably inexpensive and mobile so that training can be conducted at any location. Using genuine components, as well as realistic programming, a replica of a jail was created within a 55.32 cm x 42.39 cm x 26.97 cm Pelican 1610 case. To enhance realism, components were selected according to the design of an actual jail in the northeastern United States. Furthermore, the PLCs' ladder logic is modeled to provide identical operation to the controller in the real jail. Any one of the three PLCs can be selected by the training administrator to be active at any given time. In summary, the multi-PLC training platform attempts to meet the following criteria:

1. incorporates of physical components.
2. incorporates of cyber manipulation principles.
3. incorporates of response coordination techniques.
4. Provides hands-on experience.
5. Implements effective training scenarios with measurable evaluation metrics for training.

Figure 1 shows the completed platform.

3.1.2 Components.

The majority of the components in the platform were ordered from online distributors. A semi-complete list of end-components is shown in Table 1. The list does not include Velcro tape, wires, screws, terminal blocks, din rail, 3-D printed components (e.g., pushbutton mounts, turnkey mount and Y-Box housing) and cables used to connect and mount components. The pushbuttons, indicator lights and turnkey replicate components that would be found on the control panel at a guard station within a real jail. Indicator lights are controlled in real jails based on inputs from a sensor on the cell door which detects whether or not the cell door is secure. Because the exercise platform does not have actual doors, this sensor is simulated in the Y-Box code allowing the Y-Box to send the PLC signals which are identical to a real sensor. This is the only simulated component in the platform.

Table 1. Components.

Component	Quantity	Component	Quantity
Cabinet Lock	5	Pushbutton	5
Relay (Electromechanical)	5	Red LED	4
Relay (Solid State)	3	Peg Board	2
Power Supply (12V)	1	Power Supply (24V)	1
Network Switch	1	Router	1
Circuit Breaker (10A)	1	Turnkey	1
Power Strip	1	CompactLogix	1
Siemens S7-300	1	ControlLogix	1
Y-Box	1		

The first PLC is a CompactLogix model L23E. Second is the Siemens S7-300 with one digital input module and one digital output module. The third PLC is a ControlLogix PLC which also contains one digital input and one digital output module. Additionally, the ControlLogix PLC does not have a built-in Ethernet or

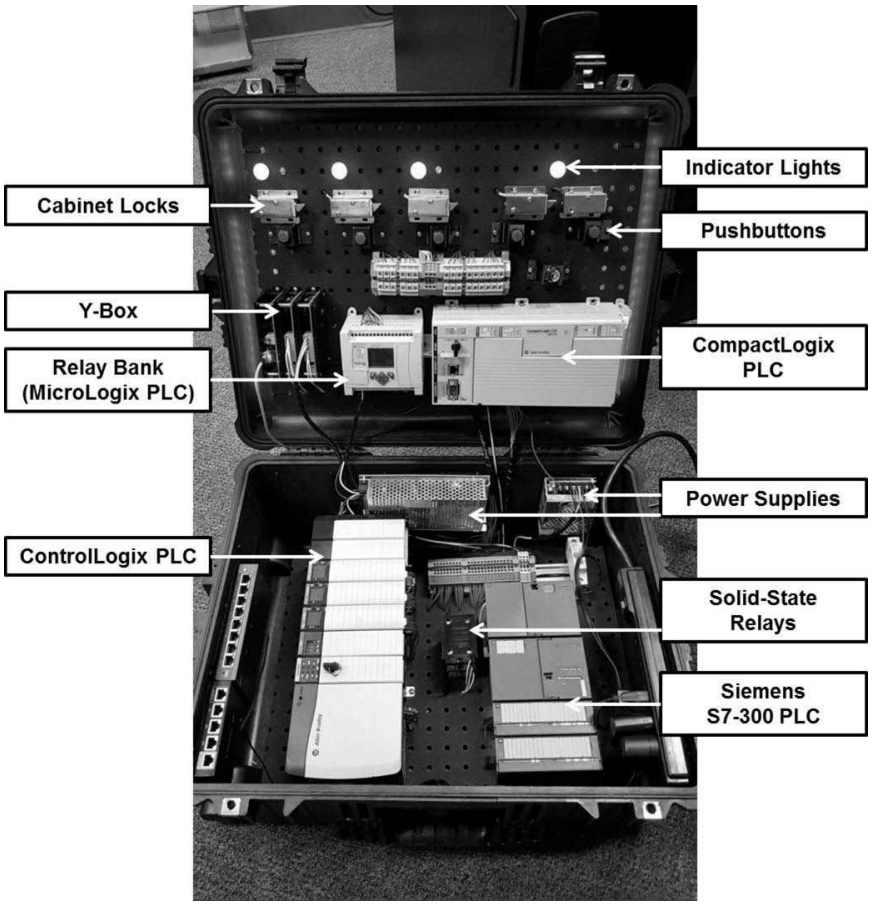


Figure 1. Completed training platform.

CPU module, therefore a Logix5555 CPU module and an EWEB Ethernet module are included in the seven-slot chassis. The Y-Box consists of a CPU module with one digital input and one digital output module. The five electromechanical relays are implemented using a Micrologix PLC.

3.1.3 Wiring.

To take full advantage of the Y-Box technology, the physical components are not wired directly to the PLC. Instead, different wiring schemes are adopted. For some applications, the Y-Box can be thought of as the "man-in-the-middle" which receives electrical signals from PLCs and other components and forwards the signals onto their destination. This wiring scheme is used for the lights and the pushbuttons, illustrated

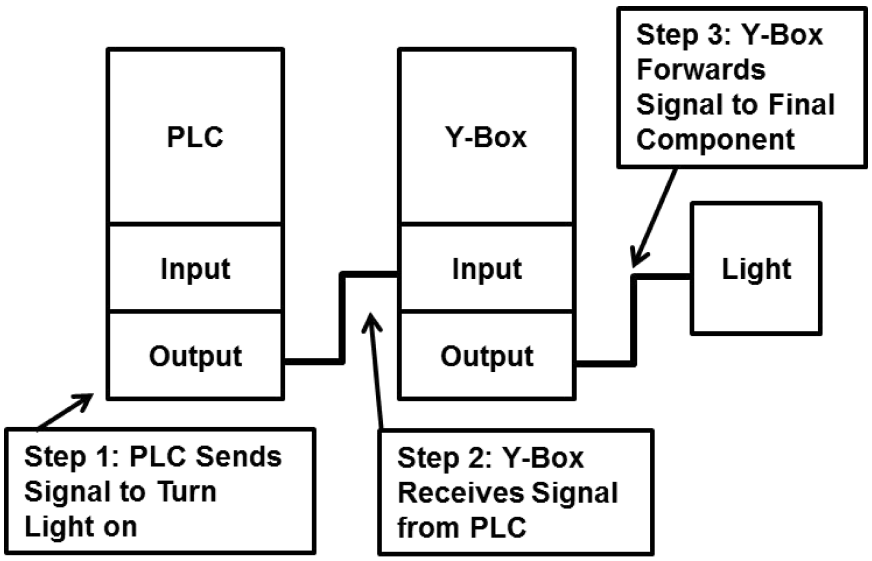


Figure 2. Wiring diagram for lights.

in Figures 2 and 3 respectively. The cabinet locks are wired differently because the Y-Box cannot provide sufficient electrical current to disengage the lock. In this case, the Y-Box is used to simply monitor the signal on the wire between the PLC and the relay which ultimately powers the lock. This is accomplished by daisy chaining the PLC outputs from the relay to the Y-Box. Figure 4 illustrates this technique.

The other wiring challenge includes connecting all three PLCs to a single set of components. This required the inputs and outputs of the three PLCs to be synchronized and wired together. Figure 5 illustrates this process for an indicator light. It shows the outputs of all three PLCs tied together, ultimately leading to a single wire which is connected the Y-Box input module.

3.1.4 PLC Selection.

The value of having three PLCs in a single platform is lost if they cannot all assume full control over the components. Once again leveraging the Y-Box technology, it is possible to control the flow of electricity into an individual PLC while denying power to the other PLCs. This task is accomplished using solid state relays controlled by

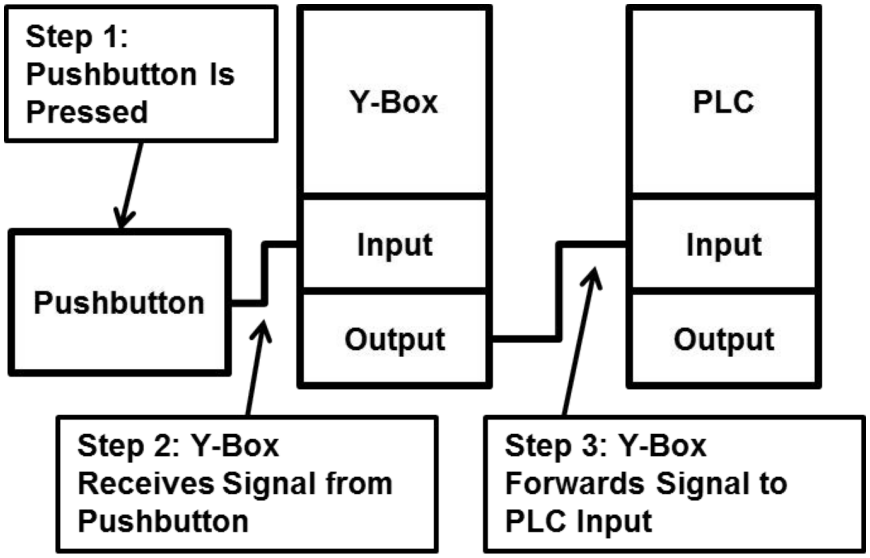


Figure 3. Wiring diagram for pushbuttons.

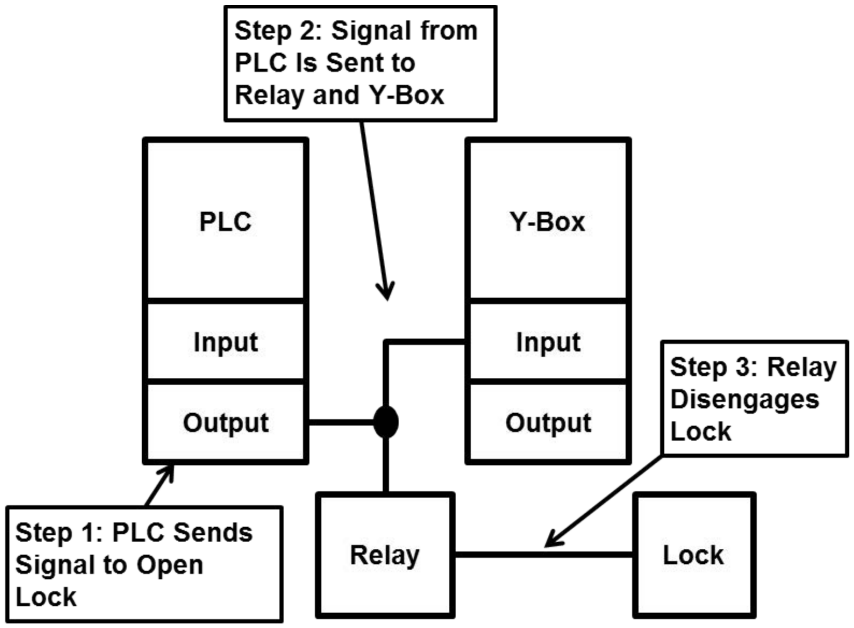


Figure 4. Wiring diagram for locks.

a Y-Box digital output. When the solid state relay receives the control signal from the Y-Box, power is allowed to flow through the relay to its corresponding PLC subsequently activating that PLC. This is the case for the ControlLogix and Siemens S7-300 PLCs. The CompactLogix PLC is slightly different from the other two because it operates on 24V direct current. In this case, the relay instead controls power to a

24V power supply which in turn powers the CompaactLogix PLC. The wiring of the relays is illustrated in Figure 6. Note that Figure 6 is simplified for the CompactLogix PLC and does not show the 24V power supply.

3.2 Exercise Layout

One possible layout for an exercise is shown in Figure 7. The following is a description of the functions of each segment of the tables in Figure 7.

3.2.1 White Cell Table.

An effective white cell should be aware of all activities performed by the training participants. The simulation terminal is a machine running Y-Box software implemented in Python. The monitoring terminal runs network monitoring software and is connected to a mirrored port on the switch to capture all traffic during the exercise. During the exercise, the white cell should watch the engineering workstation, the human-machine interface, the network traffic and the participants themselves. Furthermore, the white cell should watch the Y-Box software. If a training scenario involved malware which fooled the human-machine interface, it would be difficult for the white cell to maintain awareness of the state of the physical system during the participant's activities. The Y-Box overcomes this issue and is aware of the true state of the locks, lights and pushbuttons. Figure 8 shows the Y-Box software view of the system with the physical reality of the system as well as the PLC's perspective of the system. Figure 8 also shows the PLC selection buttons which dictate which PLC is active at any given time. It should be noted that the buttons in the software are capable of overriding the physical components in the case, providing the white cell with the ability to manage all aspects of the exercise at all times.

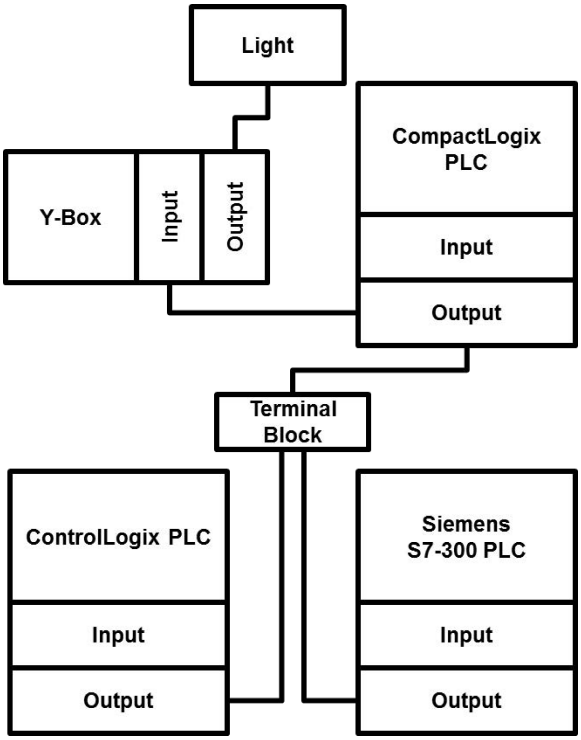


Figure 5. Wiring diagram for PLC inputs and outputs.

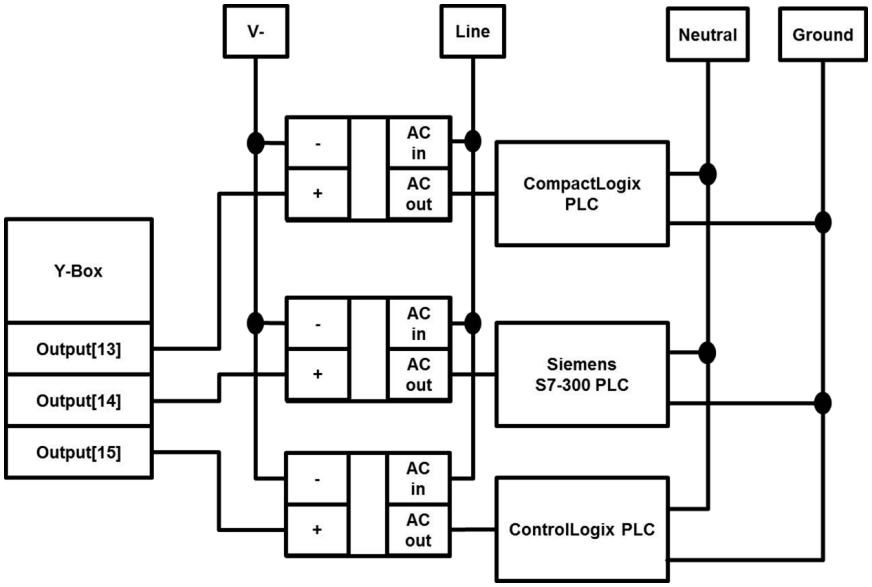


Figure 6. Wiring diagram for PLC selection.

3.2.2 Platform Table.

An engineering workstation and a human-machine interface are running beside the platform. The training platform is contained in a Pelican 1610 case. Within the

case is a fully-functioning replica of a guard station panel that closely mimics what would be found in an actual jail. Additionally, the case contains five cabinet locks, three of which represent jail cells and two of which serve as a mantrap. Each of the jail cells has a corresponding light indicating whether or not the cell is secure. The mantrap has only one light indicating its state and is secure if and only if both of its doors are closed and locked. The PLCs are connected to a network switch housed within the base of the Pelican case.

3.2.3 Exercise Participant Table.

Training participants are seated at the exercise participant table within sight of the training platform as shown in Figure 7. Laptops are provided with standard security tools (e.g., Kali Linux and Security Onion virtual machines), as well as virtual machines containing the necessary proprietary software applications to interact with the PLCs. Participants may also bring any tools which they feel are appropriate for the exercise. From their table, they are connected to a network switch within the Pelican case and are free to interact with the platform to complete their assigned task. Note that the layout can be rearranged to accommodate different rooms or table sizes, and additional network switches can be added to accommodate more participants.

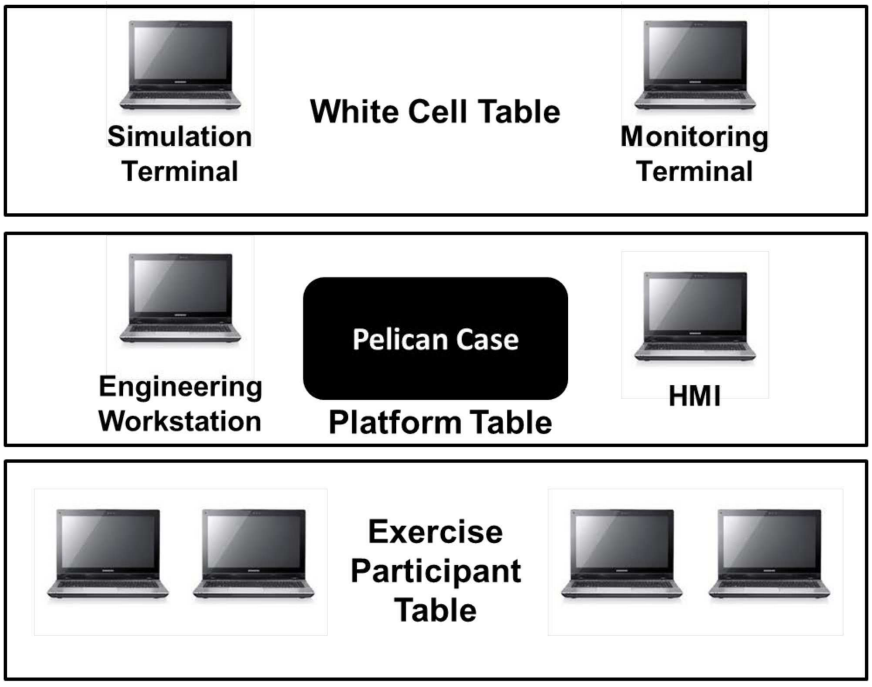


Figure 7. Exercise layout.

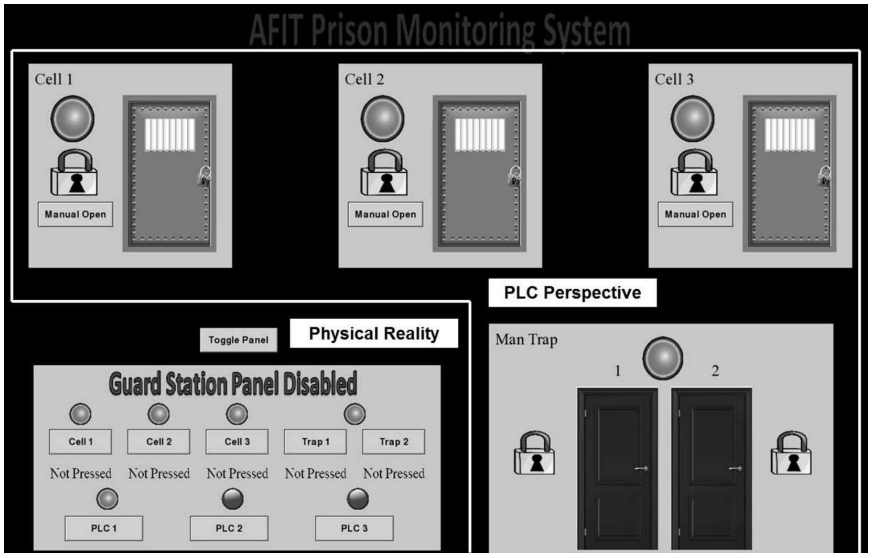


Figure 8. White cell view.

IV. Training Scenario - Segmentation of an ICS Network

One of the most important steps in securing an ICS is to properly segment the ICS network [20]. This simple task is an effective example for demonstrating different implementations of similar features among various PLCs. For this reason, a beginner-level scenario was designed for the multi-PLC training platform.

Because this scenario is meant to demonstrate differences between the PLCs' implementations, the scenario is simplified in several ways. First, the initial IP addresses of all three PLCs are the same (192.168.108.205). The new IP addresses that the participants are meant to load onto the PLCs are also the same (10.1.4.205). Next, the participants need not concern themselves with whether the changing of the IP address will impact functionality of other components in the ICS. For this scenario, it is assumed that all other issues regarding components that are dependent on the PLC's IP address have already been addressed. More difficult scenarios can be developed to demonstrate second and third order effects that can occur from this process.

The final simplification of the scenario is that there are no password protections on any of the files. In a real-world environment, it is reasonable to expect that a cyber first responder would be provided the necessary access by the asset owner to perform her duties. While there are required credentials in the ControlLogix administrative web server, they have been reset to the factory default credentials for demonstration purposes.

The scenario is implemented using the framework proposed by Yoon et al. which was briefly described at the end of Chapter Two [23].

- *Objective*: Isolate a PLC that is located on an improperly segregated network.
- *Description*: The participant uses the necessary software and techniques to change a PLC IP address from 192.168.108.205 to a new IP address of 10.1.4.205.

- *Type*: Network reconfiguration.
- *Evaluation Criteria*:
  - Identify relevant software within five minutes.
  - Identify appropriate technique for updating IP address within ten minutes.
  - Update and confirm the new IP address within fifteen minutes.
  - Perform all activities with minimal PLC downtime.
- *Reference*: NIST SP 800-82, Rockwell Automation EWEB module documentation, Siemens S7-300 documentation and Rockwell Automation CompactLogix documentation.

4.1 Segmentation Using a CompactLogix PLC

The first PLC that the participant interacts with is the CompactLogix PLC. Updating the IP address for this PLC is completed using the following steps:

1. Open the appropriate RSLogix5000 project file and access the Ethernet module properties.
2. Under the "Port Configuration" tab, enter the new IP address into the appropriate field and click "set." Confirm the update in the dialogue windows that appear.
3. Ensure connectivity to the new IP address (this may require routing or changing the IP address of the engineering workstation).

Step one requires the identification of the RSLogix5000 software. The second step demonstrates the technique to identify and update the IP address. The final step ensures that the PLC is available. The CompactLogix is capable of having its

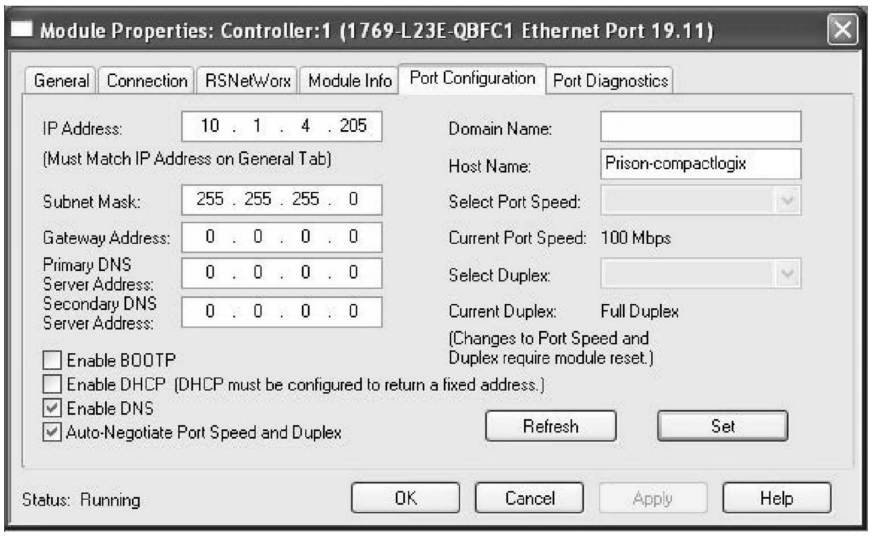


Figure 9. Updating the IP field in the RSLogix5000 software for the CompactLogix PLC.

IP address updated without downtime, thus the participant should receive a lower evaluation if the PLC resets or faults. Figure 9 shows the relevant dialogue window for updating the IP address.

4.2 Segmentation Using a Siemens PLC

After completing the assigned task on the CompactLogix PLC, control of the platform is switched to the Siemens PLC by the instructor. Once the PLC has finished booting, the participant must again perform the task of changing the PLC's IP address to an isolated subnet. This is the first time that the participant is truly exposed to the differences between the PLCs. The programming environment for the Siemens PLC is different from the CompactLogix. The steps required to complete the task on the Siemens PLC are outlined below:

1. Open the SIMATIC project file.

2. Access the "HW Config" in the SIMATIC software and navigate to the "object properties" of the PN-IO module.

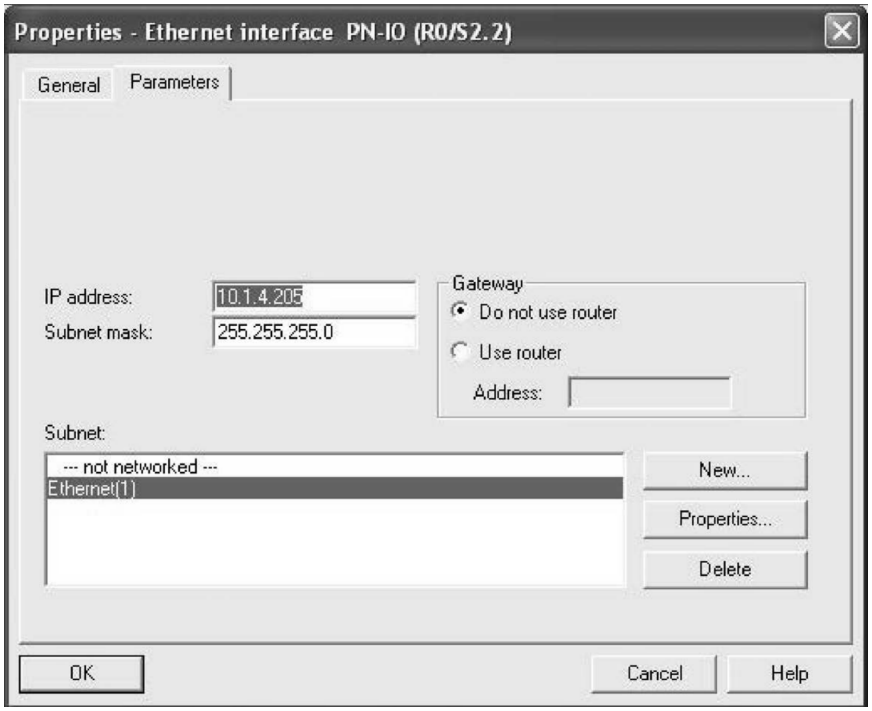


Figure 10. Updating the IP field in the SIMATIC software for the S7-300 PLC.

3. Under the "General" tab, select "Properties" and enter the new IP address as shown in Figure 10.

4. Download the new configuration to the PLC using the *old* IP address as the target station as shown in Figure 11.

5. Ensure connectivity to the new IP address (this may require routing or changing the IP address of the engineering workstation).

Step one requires the identification of the SIMATIC software. Steps two through four involve identifying the appropriate technique to update the IP address. Step five ensures that the PLC completed the download successfully with minimal downtime.

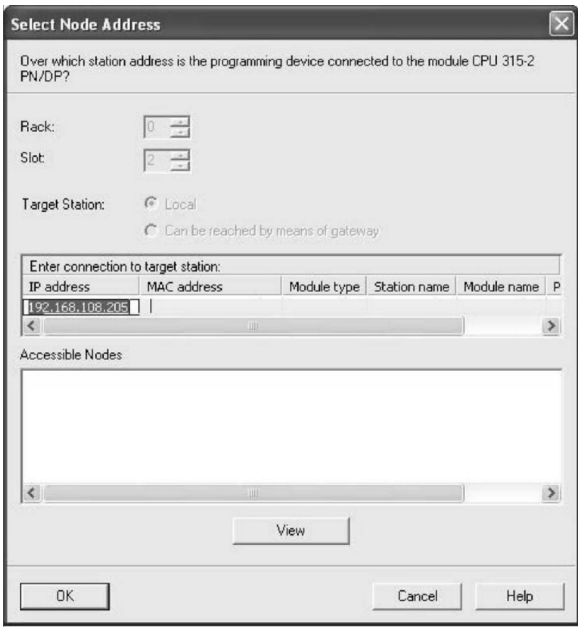


Figure 11. Setting the target IP for the S7-300.

4.3 Segmentation Using a ControlLogix PLC

The participant now performs the required tasks using an implementation that is unique to the ControlLogix PLC. The ControlLogix PLC is equipped with a 1756-EWEB Enhanced Web Server Module which provides an administrative web interface to manage the PLC. The necessary steps for completion are as follows:

1. Open a web browser and navigate to the IP address of the PLC.
2. Open the network configuration tab, input the new IP address to the appropriate field and apply the changes.
3. Confirm that the new address is correct. Upon completion, a message will be displayed notifying the participant of the new IP address.
4. Ensure connectivity to the new IP address (this may require routing or changing the IP address of the engineering workstation).

Step one requires that the participant to identify the web interface provided by

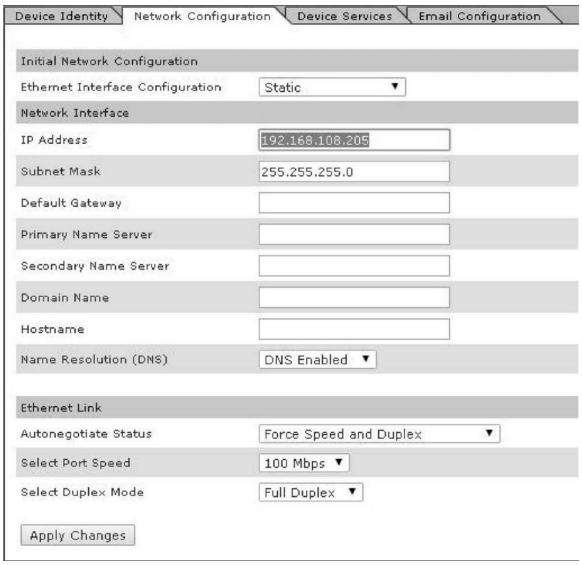


Figure 12. Updating the IP field from the ControlLogix administrative page.

the EWEB module. Step two identifies the appropriate technique and performs the update. Steps three and four confirm that the change was successful. There should be no downtime in the completion of this task with the ControlLogix PLC.

4.4 Scenario Selection and Alternate Scenario Possibilities

The network segmentation scenario was chosen because it effectively demonstrates how different PLCs will often require different techniques to perform the same task. These differences emphasize the value of a cyber first responder having experience on a variety of PLCs. The scenario incorporates several of the training curricula proposed by Butts and Glover in a proven format as described by Yoon et al. It should be noted that segmenting a network is only one of a variety of tasks that a cyber first responder may need to complete in their line of work and it is not necessarily intended to be a particularly difficult example. Other examples such as modifying ladder logic, updating firmware and applying patches will also have processes unique to different PLCs and vendors with varying levels of difficulty. Because the platform incorporates real PLCs, each of these scenarios, as well as other scenarios involving

more advanced topics, could be implemented with minimal reconfiguration of the platform. The following scenarios showcase the flexibility of the multi-PLC platform:

4.4.1 Analysis of Malicious Implant in PLC Firmware.

- *Objective*: Reverse engineer firmware to identify and analyze malicious implant
- *Description*: The participant uses the necessary software and techniques to extract PLC firmware from the device and identifies malicious code given a correct version of the firmware. The participant then determines the exact functionality and purpose of the malicious code.
- *Type*: Reverse engineering.
- *Evaluation Criteria*:
  - Identify malicious code within 45 minutes.
  - Restore the PLC firmware within 20 minutes.
  - Analyze malicious code within 90 minutes.
- *Reference*: Rockwell Automation ControlLogix documentation, Siemens S7-300 documentation, Rockwell Automation CompactLogix documentation.

The reverse engineering scenario further emphasizes the differences between the PLCs by requiring participants to extract and analyze firmware from the device (see [2] for details on the reverse engineering process for ICSs). It also brings up an equally important point that there are often similarities between some PLCs. Specifically, the CompactLogix and ControlLogix PLCs have very similar firmware, despite being different PLC models. The reverse engineering scenario can be implemented with

different malware of varying levels of complexity to accommodate participants' capabilities.

4.4.2 Digital Forensics of a Malfunctioning PLC.

- *Objective*: Determine the cause of a malfunctioning PLC's behavior.
- *Description*: The participant uses the necessary software and techniques to identify the root cause of the PLC's behavior.
- *Type*: Digital forensics.
- *Evaluation Criteria*:
  - Collect sufficient data to perform forensics within 30 minutes.
  - Identify the cause of the malfunction within 45 minutes.
  - Identify corrective action within 60 minutes.
- *Reference*: Rockwell Automation ControlLogix documentation, Siemens S7-300 documentation, Rockwell Automation CompactLogix documentation.

The digital forensics scenario accomplishes similar tasks to the reverse engineering scenario by showing that the process for conducting digital forensics on ICSs is identical for different PLCs (see [6] for details on this process). Despite using the same process, the data being analyzed (e.g., Ladder Logic code, network traffic and log files) will still be different because of operational differences between the PLCs. These operational differences mean that a cyber first responder in a real-world situation will have to focus on specific, contextualized pieces of information to effectively analyze the root cause of a malfunctioning PLC's behavior. The difficulty of this scenario can be modulated by inducing different types of PLC malfunctions ranging

from simple faults to advanced malware infections. The scenario can also be repeated multiple times with different malfunctions to increase the training participant's exposure to various malfunctions.

V. Results

The following subsections describe the results of the platform development.

5.1 Hardware Verification

Initial debugging of the wiring, Y-Box code and PLC code involved interacting with the physical components mounted in the Pelican case and confirming that the Y-Box and the PLC behaved as intended. This process revealed that some of the variables had been coded incorrectly into the Ladder Logic. These variables needed their memory addresses reassigned to correct their mapping to the PLC inputs and outputs. The Y-Box software was also verified, confirming the behavior of the physical components and that the software was capable of overriding the physical components to control the case autonomously.

5.2 Reliability Test

Table 2. Reliability test results.

Controller	Trials	Failures
CompactLogix	50	0
Siemens S7-300	50	0
ControlLogix	50	0
Totals	150	0

After confirming that the components were behaving correctly, an automated Python script tested the reliability of the platform. This test attempts the following steps:

1. Select PLC.

2. Power up selected PLC.

3. Wait 25 seconds for PLC to activate.

4. Test all buttons, locks and lights for functionality.

5. Shut down PLC.

6. Reset Y-Box parameters.

Initial runs of the test encountered failures because the Python test code sent commands too quickly. This denied the Y-Box adequate time to update its inputs and outputs. The resolution of this issue was to include "wait" commands of 25 seconds for the PLC to boot and varying amounts of time between 0.4 and 2.0 seconds for other functions (e.g., button presses, indicator light updates and lock status updates). Step four is the key component to this test. This step starts with the first jail cell and simulates a button press. The script then checks that the PLC responds appropriately before repeating the process for the other two cells. Next, the test code evaluates the mantrap by testing every possible combination of button presses and confirming the responses. Finally, it simulates a button press on cell one again with the panel disabled. In this situation, the lock should not disengage and the test is considered a failure if it does. The implemented wiring scheme with the Y-Box allows electrical signals to be sent to the PLC without having to receive signals from the buttons. Furthermore, the state of the panel's turnkey can be overridden by the Y-Box itself. These capabilities allow each of the functions to be simulated by the Y-Box alone. Subsequently, the tests can be fully automated in a manner that is transparent to the PLC since the PLC receives the same signals as it would under normal operation. To prevent failures during one iteration from impacting the results of the next iteration, all Y-Box values are reset to a default value in step six. 150 total iterations were completed, testing each PLC 50 times. The results of the test are shown in Table 2.

5.3 Timing

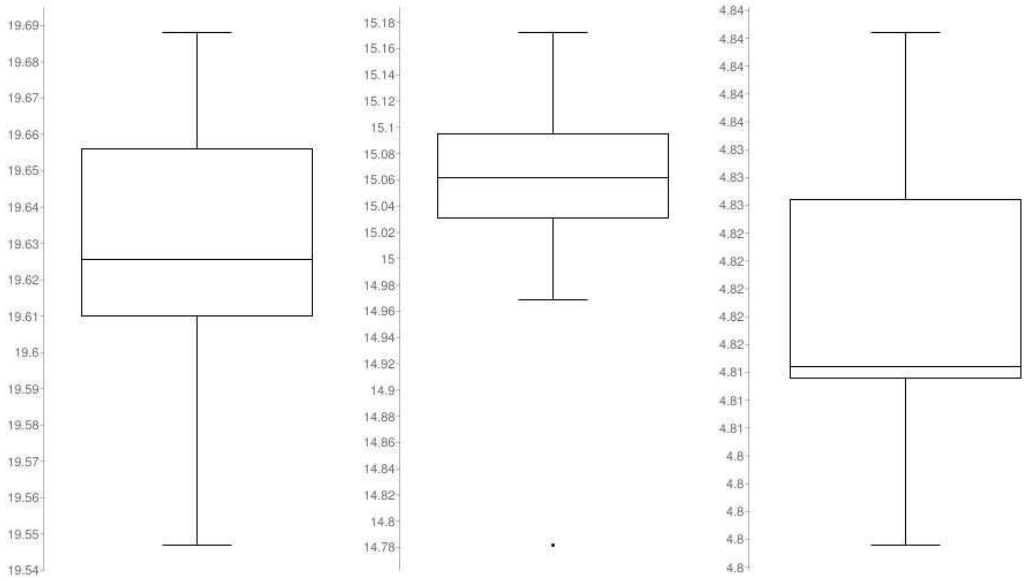
Table 3. PLC startup times (in seconds).

Controller	Min	Max	Mean	Std Dev
CompactLogix	19.547	19.688	19.629	0.030
Siemens S7-300	14.782	15.172	15.060	0.062
ControlLogix	4.797	4.843	4.816	0.010

Incorporating multiple PLCs into one single platform is useless if switching between PLCs takes a prohibitively long amount of time. Ideally, control of the system should be able to be switched from one PLC to another within the amount of time that it takes for the participant to be prepared for the next task. To evaluate this metric, the time required for each PLC to fully power up was measured and recorded by an automated Python script which performed the following steps:

1. Select PLC.
2. Send power to PLC and begin timer.
3. Send input command to PLC.
4. Wait for PLC to react to input, stop timer upon completion.
5. Shut down PLC.
6. Reset Y-Box parameters.

For this test, it is only necessary to examine the amount of time that it takes for the PLC to become responsive to an input. Table 3 and Figure 13 present the results of this test which were also determined over the course of 150 trials (50 per PLC). The results show that the PLCs had significantly different boot times, but were very consistent across all iterations. Note that Figure 13 has different Y-axis scales for each subfigure.



(a) CompactLogix PLC. **(b) Siemens S7-300 PLC.** **(c) ControlLogix PLC.**

Figure 13. Box plots of PLC timing results.

5.4 Functional Analysis Criteria

The multi-PLC training platform attempts to meet the following criteria:

1. incorporates of physical components.
2. incorporates of cyber manipulation principles.
3. incorporates of response coordination techniques.
4. Provides hands-on experience.
5. Implements effective scenarios with measurable evaluation metrics for training.

The replication of the jail system, including the PLC running realistic Ladder Logic, pushbuttons, locks, lights and the turnkey, provides the training participant with a view of some of the physical components involved in a real-world system. This addresses the need for cyber first responders to understand the underlying physical processes involved in ICS.

By creating scenarios which incorporate concepts such as reverse engineering and digital forensics, cyber manipulation principles can be effectively taught to cyber first responders. Training participants will be exposed to topics like access vectors, vulnerability analysis, implanting malware, manipulating physical processes and defensive mechanisms, all of which are considered cyber manipulation principles by Butts and Glover [4].

Skills involved in response coordination include the ability to prioritize system components, knowing how to identify attacks and understanding the steps required to appropriately defend and restore. Each of these skills are practiced in some way by the scenarios described in this work, and can be enhanced by designing alternate scenarios using the existing platform.

The entire training platform is self-contained, and all relevant software is contained within easily recoverable virtual machines. This provides the opportunity for training participants to have full access to the system for hands-on exercises without being concerned with causing damage to an existing system. Cyber first responders will benefit most from hands-on exercises where they can gain experience with realistic systems that incorporate real hardware.

Using the framework proposed by Yoon et al., training scenarios can be created that include specific, measurable evaluation criteria. The scenarios provided in this paper would be easy to expand to accommodate different skill levels. Furthermore, entirely new scenarios could be created relatively easily to achieve different training goals without the need for significant modification to the training platform.

5.5 Limitations

The multi-PLC platform does have some limitations. First, the jail system platform does not incorporate analog components. Analog signals are more complicated

than digital signals from a programming perspective and may enhance the experience for training participants. Another limitation of the platform is the scale of the replica system. In a full-sized jail, there are more components such as additional doors and alarms. This design choice leads to the trade-off between the cost and scale of the platform. Finally, the platform is limited to the use of one PLC at any given time.

VI. Conclusion

This chapter summarizes the conclusions made by the research.

6.1 Conclusions of Research

Effective ICS platforms are necessary for cyber first responder training. Unfortunately, most test beds today are meant for research and development and are not available for training purposes. Furthermore, test beds that have been built for training tend to be very expensive and often substitute simulations in place of genuine components. Ideal test beds contain full-scale, fully operational ICS with effective training scenarios that address the unique skills required for cyber first responders to operate in these environments. The cost of such a test bed is prohibitively high; however, the multi-PLC training platform presented in this thesis is capable of addressing many of these unique skills at a fraction of the cost. Furthermore, the multi-PLC training platform is capable of supporting any number of scenarios ranging from basic examples such as changing an IP address to advanced scenarios involving digital forensics and reverse engineering. The variety of PLCs in the platform provides opportunities to experience different protocols and programming environments. Much akin to the education techniques of the medical profession, the hands-on exercises provided by the multi-PLC training platform allow cyber first responders to gain a variety of experience to help them in their future work.

6.2 Research Hypotheses

The first research hypothesis was that a platform could be built which implements scenarios designed to address training requirements for cyber first responders. As shown by chapter four as well as section 5.4, the platform built for this research

confirms the first hypothesis.

The second hypothesis was that multiple PLCs can control a single set of components without interference. By cleverly wiring the components in the system together as described in section 3.1.3, this hypothesis is also confirmed.

The final hypothesis was that all required components can fit into a transportable briefcase-sized platform. While more complex systems could be implemented in larger platforms, the platform created for this research was able to incorporate an ICS with three PLCs as well as other physical components into a briefcase-sized Pelican case thereby confirming the third hypothesis.

6.3 Significance of Research

The value of this research is primarily in the proof of concept where three PLCs are tied into a single set of components for training purposes. This technique can be used in future platforms to save money on components and diversify training with different brands or even generations of PLCs. Being able to run a single ICS on a selectable generation of PLC is valuable because PLCs are built to be reliable for many years before they need replaced. The long life cycle of PLCs dictates that cyber first responders must be able to interact with PLCs from any generation, thus demonstrating the value of this technique.

MULTI-PLC EXERCISE ENVIRONMENTS FOR
TRAINING CYBER FIRST RESPONDERS FOR INDUSTRIAL CONTROL SYSTEMS

A. Live Exercise Scenarios

A live exercise was conducted at Fort Gordon with members of the Cyber Protection Brigade using a previous version of the training platform. The following offense-oriented scenarios were completed by the participants. Note that the scenario format was adapted to include completion requirements as opposed to evaluation criteria. Furthermore, references were not provided to the participants.

A.1 Narrative

To help guide the activities of the participants through their completion of the scenarios, the following narrative was provided:

An asset to the intelligence community is being held in a jail in a foreign nation. The location of the jail prohibits US forces from conducting a raid to retrieve the asset and therefore cyber is the best option. An informant within the prison claims that the asset is being held in cell 2, and that in order to reach the exit, the asset will also have to navigate a mantrap. The informant has also stated that while he is willing to perform certain actions on our behalf, he will not perform any actions for us that will lead him to become compromised. Once the imprisoned asset reaches the exit, a team of friendly forces will be there to retrieve him. In this operation, we are not concerned with the victims ability to attribute the attack to the US, and the

timing of the exploit is not important since the team outside will remain in place to retrieve the asset for as long as needed. We do care however about protecting our exploits for use in the future and thus careful bookkeeping should be performed for cleaning up and making digital forensics more difficult after the attack is complete.

A.2 Reconnaissance and Enumeration of the Prison's Corporate Network

- *Objective*: Gather as much intelligence as possible about the prison and its networks.
- *Description*: Through reconnaissance, the team discovers the types of systems used by the prison. This information is compared with the results of the network enumeration to discover that some components are hidden from the corporate network.
- *Type*: Intelligence gathering.
- *Completion Requirements*:
  - Team completes system reconnaissance
  - Team completes network enumeration
  - Team appropriately manages team members
  - Team performs activities without alerting system administrators.
  - Documentation of results found during recon and enumeration.

A.3 Gain Access to the Prison's ICS Network

- *Objective*: Gain access to the ICS network by bridging the networks using a raspberry pi.

- *Description*: A team of attackers finds access to a foreign prisons corporate network. While conducting reconnaissance, it became clear that a bridge between the corporate and ICS networks did not exist and that one would need to be created. The team has available an informant inside the prison who is capable of planting a tool that will connect the networks for them. The team must ensure that the raspberry pi provided to them is correctly configured to connect to the networks. Furthermore, the team must generate clear instructions for the informant, who is not tech savvy, which leads to the proper configuration inside the prison to provide the team access.

- *Type*: Gaining unauthorized access.

- *Completion Requirements*:

  - Team completes system reconnaissance.
  - Team completes network enumeration.
  - Team appropriately manages team members.
  - Team performs activities without alerting system administrators.
  - Documentation of results found during recon and enumeration.

A.4 Reconnaissance and Enumeration of the Prison's ICS Network

- *Objective*: Discover all relevant programmable logic controllers and other interesting devices.

- *Description*: The team of attackers uses network mapping tools and other information sources in order to determine which machines on the network are relevant for the attack.

- *Type*: Intelligence gathering.
- *Completion Requirements*:
  - Discover and Identify PLC.
  - Discover and Identify HMI workstation.
  - Discover and Identify Engineering workstation.
  - Discover potential vulnerabilities.
  - Team appropriately manages team members.
  - Team performs activities without alerting system administrators.
  - Documentation of results found during recon and enumeration.

A.5 Exploit Planning

- *Objective*: Research the vulnerabilities found in the previous phase, then develop and test exploits prior to deployment.
- *Description*: Using information found in the previous phase, the team will develop an exploit that can free the imprisoned asset quickly. The team does not need to concern itself with attribution or covertness while developing this exploit since the guards have been paid off to allow the asset and only the asset to escape. If any other prisoners are freed, the guards will not allow anyone to escape an1d the mission will be a failure.
- *Type*: Exploit development.
- *Completion Requirements*:
  - Fully develop and test the exploit

  - Effectiveness of exploit
  - Team appropriately manages team members
  - Team performs activities without alerting system administrators
  - Documentation of results found during recon and enumeration

A.6 Exploit Execution and Exfiltration

- *Objective*: Deploy the exploit developed in the previous step and reset the system to its pre-exploitation state.
- *Description*: The team downloads the malware to the PLC or the HMI or both resulting in the asset escaping from the prison. The team also performs whatever activities needed to cover its tracks in the system in order to make digital forensics as difficult as possible, protecting any exploits for future use.
- *Type*: Exploitation and exfiltration.
- *Completion Requirements*:
  - Attack complete
  - Exfiltration complete
  - Degree to which system is cleaned
  - Manner in which system is cleaned
  - Team efficiently manages resources

B. Software Code

Some example code for the Y-Box and the PLCs is included in the following sections. Note that not all code that was used is included in this work.

B.1 Y-Box

The following subsections provide the code implementing most of the functionality of the Y-Box. This code is all written in Python.

B.1.1 Main.

The following code is the main program.

```
import pygame
from pygame.locals import *
import guard_station
import prison_cell
import time
import _thread
import Ybox
import mantrap
import indicator
import time

#
    ##############################################################################

# Authors: Evan Plumley, Joseph Daoud and Jeff Guion
```

```
#
# This programruns the prison cell simulation displaying the
    value
# of the PLC outputs and providing the PLC inputs
#
#
#
    ##########################################################################


class PrisonSim:
    def __init__(self):
        self._running = True
        self.screen = None
        self.size = self.width, self.height = 1400, 950
        self.ybox = Ybox.Ybox()
        self.buttonclick1 = False
        self.buttonclick2 = False
        self.buttonclick3 = False
        self.buttonclick4 = False
        self.buttonclick5 = False
        self.buttonclick6 = False
        self.buttonclick7 = False
        self.buttonclick8 = False
        self.buttontoggle1 = False
        self.buttontoggle2 = False
```

```
        self.buttontoggle3 = False
        self.buttontoggle4 = False
        self.buttontoggle5 = False
        self.buttontoggle6 = False
        self.buttontoggle7 = False
        self.buttontoggle8 = False
        self.PLC1power = False
        self.PLC2power = False
        self.PLC3power = False

    def on_init(self):
        #Initialize screen
        pygame.init()
        self.font = pygame.font.SysFont('Times', 25)
        pygame.display.set_caption('Ybox Simulation')
        self.screen = pygame.display.set_mode(self.size,
            pygame.HWSURFACE | pygame.DOUBLEBUF)
        self.screen.fill((black))

        x_padding = 42.5
        y_padding = 95

        cell_panel_width = 325
        cell_panel_height = 300

        #create the guard station panel with number of cells
```

```
self.guard_station_panel = guard_station.GuardStation
    (self, 5, 50, 575)

#create each cell at given location
cell_one_panel = prison_cell.PrisonCell(self, 1,
    x_padding, y_padding)
cell_two_panel = prison_cell.PrisonCell(self, 2, 5*
    x_padding+cell_panel_width, y_padding)
cell_three_panel = prison_cell.PrisonCell(self, 3, 9*
    x_padding+2*cell_panel_width, y_padding)

#add all cells to list
self.cell_door_panels = [cell_one_panel,
    cell_two_panel, cell_three_panel]

#add mantrap display
self.mantrap = mantrap.ManTrap(self, 3*x_padding+2*
    cell_panel_width, (2.2 * y_padding) +
    cell_panel_height)

#add the title of the program
main_title_dimensions = ((615, 70))
main_title_icon = pygame.image.load ("images/title.
    png")
main_title_icon2 = pygame.image.load ("images/title.
    png")
```

```
main_title_icons = [main_title_icon, main_title_icon2
    ]
main_title = indicator.Indicator(self, 390, 10,
    main_title_icons , main_title_dimensions)


# set the inital states
self.ybox.sendWrite(1,0,1) #light indicators green
self.ybox.sendWrite(1,1,0)
self.ybox.sendWrite(1,2,1)
self.ybox.sendWrite(1,3,0)
self.ybox.sendWrite(1,4,1)
self.ybox.sendWrite(1,5,0)
self.ybox.sendWrite(1,6,0)
self.ybox.sendWrite(1,7,0)
self.ybox.sendWrite(1,8,0)
self.ybox.sendWrite(1,9,1) #panel enabled
self.ybox.sendWrite(1,10,1)
self.ybox.sendWrite(1,11,1)
self.ybox.sendWrite(1,12,1) #test
self.ybox.sendWrite(1,13,0)
self.ybox.sendWrite(1,14,0)
self.ybox.sendWrite(1,15,0)

pygame.display.update()
try:
```

```
            _thread.start_new_thread(timedReads, (self,))
        except Exception as e:
            print(e)
        self._running = True

    #Handle all events
    def on_event(self, event):
        if event.type == pygame.QUIT:
            self._running = False
        else: #Determine if button was clicked
            for i, cell_btn in enumerate(self.
                guard_station_panel.cell_btns):
                if 'click' in cell_btn.handleEvent(event):
                    try:
                        clickButton(self,i)
                    except Exception as e:
                        print(e)

            for i, cell in enumerate(self.cell_door_panels):
                if 'click' in cell.key_btn.handleEvent(event)
                    :
                    try:
                        openDoor(self,i)
                    except Exception as e:
                        print(e)
```

```
            if 'click' in self.guard_station_panel.
               disable_btn.handleEvent(event):
                self.guard_station_panel.disable_clicked =
                   True

            if 'click' in self.guard_station_panel.PLC1_BTN.
               handleEvent(event):
                self.guard_station_panel.PLC1_clicked = True

            if 'click' in self.guard_station_panel.PLC2_BTN.
               handleEvent(event):
                self.guard_station_panel.PLC2_clicked = True

            if 'click' in self.guard_station_panel.PLC3_BTN.
               handleEvent(event):
                self.guard_station_panel.PLC3_clicked = True

            pygame.display.update()



def on_loop(self):
    pass
def on_render(self):
    pass
def on_cleanup(self):
```

```
        self.ybox.sendWrite(1,0,0)
        self.ybox.sendWrite(1,1,0)
        self.ybox.sendWrite(1,2,0)
        self.ybox.sendWrite(1,3,0)
        self.ybox.sendWrite(1,4,0)
        self.ybox.sendWrite(1,5,0)
        self.ybox.sendWrite(1,6,0)
        self.ybox.sendWrite(1,7,0)
        self.ybox.sendWrite(1,8,0)
        self.ybox.sendWrite(1,9,0)
        self.ybox.sendWrite(1,10,0)
        self.ybox.sendWrite(1,11,0)
        self.ybox.sendWrite(1,13,0)
        self.ybox.sendWrite(1,14,0)
        self.ybox.sendWrite(1,15,0)
        pygame.quit()
        self.ybox.closePort()


#start program
def on_execute(self):
    if self.on_init() == False:
        self._running = False

    while( self._running ):
        for event in pygame.event.get():
```

```
                self.on_event(event)
            self.on_loop()
            self.on_render()
        self.on_cleanup()

#pushes button for two seconds then releases it
def clickButton(self, i):
    #door one button
    if i == 0 and self.buttonclick1 == False:
        self.buttonclick1 = True
        self.buttontoggle1 = True
    elif i == 0 and self.buttonclick1 == True:
        self.buttonclick1 = False
        self.buttontoggle1 = True

    #door two button
    elif i == 1 and self.buttonclick2 == False:
        self.buttonclick2 = True
        self.buttontoggle2 = True
    elif i == 1 and self.buttonclick2 == True:
        self.buttonclick2 = False
        self.buttontoggle2 = True

    #door three button
    elif i == 2 and self.buttonclick3 == False:
        self.buttonclick3 = True
```

```
            self.buttontoggle3 = True
        elif i == 2 and self.buttonclick3 == True:
            self.buttonclick3 = False
            self.buttontoggle3 = True

        #door four button
        elif i == 3 and self.buttonclick4 == False:
            self.buttonclick4 = True
            self.buttontoggle4 = True
        elif i == 3 and self.buttonclick4 == True:
            self.buttonclick4 = False
            self.buttontoggle4 = True

        #door five button
        elif i == 4 and self.buttonclick5 == False:
            self.buttonclick5 = True
            self.buttontoggle5 = True
        elif i == 4 and self.buttonclick5 == True:
            self.buttonclick5 = False
            self.buttontoggle5 = True

        #PLC 1 Button
        elif i == 5 and self.buttonclick6 == False:
            self.buttonclick6 = True
            self.buttontoggle6 = True
        elif i == 5 and self.buttonclick6 == True:
```

```
            self.buttonclick6 = False
            self.buttontoggle6 = True

        #PLC 2 Button
        elif i == 6 and self.buttonclick7 == False:
            self.buttonclick7 = True
            self.buttontoggle7 = True
        elif i == 6 and self.buttonclick7 == True:
            self.buttonclick7 = False
            self.buttontoggle7 = True

        #PLC 3 Button
        elif i == 7 and self.buttonclick8 == False:
            self.buttonclick8 = True
            self.buttontoggle8 = True
        elif i == 7 and self.buttonclick8 == True:
            self.buttonclick8 = False
            self.buttontoggle8 = True

        else:
            print("Something went wrong in the lock reads1")



#method to open the door via the manual key button
```

```
def openDoor(self, i):
    if self.cell_door_panels[i].doorClosed == True:
        self.cell_door_panels[i].cell_door.change_state()
        self.cell_door_panels[i].doorClosed = False

    elif self.cell_door_panels[i].doorClosed == False:
        self.cell_door_panels[i].cell_door.change_state()
        self.cell_door_panels[i].doorClosed = True

#method to monitor the PLC and chnage the display accordingly

def timedReads(self):
    past = int(round(time.time() * 1000)) #getting starting
       milisecond time to execute reads from the ybox
    while True:
        present = int(round(time.time() * 1000)) #getting
           present time to comapre to past
            #check to see if 100 milliseconds have  passed
        if present - past >= 100:
            past = present

            ########################################
            #set button clikc values and write accordingly
            #
               ##################################################
```

```
if self.buttonclick1 == True and self.
    buttontoggle1 == True:
      self.ybox.sendWrite(1,1,1)
      self.buttontoggle1 = False
elif self.buttonclick1 == False and self.
    buttontoggle1 == True:
      self.ybox.sendWrite(1,1,0)
      self.buttontoggle1 = False

if self.buttonclick2 == True and self.
    buttontoggle2 == True:
      self.ybox.sendWrite(1,3,1)
      self.buttontoggle2 = False
elif self.buttonclick2 == False and self.
    buttontoggle2 == True:
      self.ybox.sendWrite(1,3,0)
      self.buttontoggle2 = False

if self.buttonclick3 == True and self.
    buttontoggle3 == True:
      self.ybox.sendWrite(1,5,1)
      self.buttontoggle3 = False
elif self.buttonclick3 == False and self.
    buttontoggle3 == True:
      self.ybox.sendWrite(1,5,0)
      self.buttontoggle3 = False
```

```
if self.buttonclick4 == True and self.
    buttontoggle4 == True:
     print("here99")
     self.ybox.sendWrite(1,8,1)
     self.buttontoggle4 = False
elif self.buttonclick4 == False and self.
    buttontoggle4 == True:
     self.ybox.sendWrite(1,8,0)
     self.buttontoggle4 = False

if self.buttonclick5 == True and self.
    buttontoggle5 == True:
     self.ybox.sendWrite(1,7,1)
     self.buttontoggle5 = False
elif self.buttonclick5 == False and self.
    buttontoggle5 == True:
     self.ybox.sendWrite(1,7,0)
     self.buttontoggle5 = False

################################################
#read the lock value for all doors from the PLC
    and then react
################################################
for i in range(0, len(self.cell_door_panels)):
    #aqcuiring the actual channel number
```

```
if i == 0:
    readnum = 0
elif i == 1:
    readnum = 3
elif i == 2:
    readnum = 7
else:
    print("Something went wrong in the lock
        reads2")



resp = self.ybox.sendRead(0, readnum)
readnum = str(readnum)
if resp == ("r0," + readnum + ",1") and self.
    cell_door_panels[i].lockClosed == True: #
    checks for a lock state change
     self.cell_door_panels[i].lock_indicator.
         change_state() #open lock
     self.cell_door_panels[i].lockClosed =
         False #lock open state flag
     #make sure I dont unecesariily change the
          state due to the manual key
     if self.cell_door_panels[i].doorClosed ==
         True:
```

```
                self.cell_door_panels[i].cell_door.
                    change_state()
                self.cell_door_panels[i].doorClosed =
                    False
            pygame.display.update()


        elif resp == ("r0,"+ readnum +",0") and self.
            cell_door_panels[i].lockClosed == False:
            self.cell_door_panels[i].lock_indicator.
                change_state() #open lock
            self.cell_door_panels[i].lockClosed =
                True #lock open state flag
            #make sure I dont unecesariily change the
                state due to the manual key
            if self.cell_door_panels[i].doorClosed ==
                False:
                self.cell_door_panels[i].cell_door.
                    change_state()
                self.cell_door_panels[i].doorClosed
                    = True
            pygame.display.update()

    #Checking for changes for the indicator light (
        Door secure sensor outputs)
    for i in range(0, 3):
```

```
#mapping to secure lights for the plc
if i == 0:
    writenum = 0
elif i == 1:
    writenum = 2
elif i == 2:
    writenum = 4

if (self.cell_door_panels[i].doorClosed ==
   False or self.cell_door_panels[i].
   lockClosed == False) and self.
   cell_door_panels[i].indicatorLight == True
   :
    self.cell_door_panels[i].
       cell_door_indicator.change_state()#
       change to red
    self.cell_door_panels[i].indicatorLight =
        False
    self.ybox.sendWrite(1,writenum,0) # turn
       the PLC light
    pygame.display.update()

elif (self.cell_door_panels[i].doorClosed ==
   True and self.cell_door_panels[i].
   lockClosed == True) and self.
   cell_door_panels[i].indicatorLight ==
```

```
            False:
                self.cell_door_panels[i].
                    cell_door_indicator.change_state()#
                    change to green
                self.cell_door_panels[i].indicatorLight =
                     True
                self.ybox.sendWrite(1,writenum,1) # turn
                    the PLC light
                pygame.display.update()


##########################################
# read prison guard button statuses just for cell
    doors and mantrap
##########################################
for i in range(0, len(self.guard_station_panel.
    button_pushed)):
    #aqcuiring the actual channel number
    if i == 0:
        readnum = 1
        writenum = 1
    elif i == 1:
        readnum = 4
        writenum = 3
    elif i == 2:
        readnum = 6
```

```
            writenum = 5
        elif i == 3:
            readnum = 10
            writenum = 8
        elif i == 4:
            readnum = 12
            writenum = 7
        else:
            print("Something went wrong in the button
                reads3")

        resp2 = self.ybox.sendRead(0, readnum)
        readnum = str(readnum)

        if resp2 == ("r0,"+ readnum +",1") and self.
            guard_station_panel.button_pushed[i] ==
            False:
                self.guard_station_panel.btn_statuses[i].
                    change_state()
                self.guard_station_panel.button_pushed[i]
                    = True
                self.ybox.sendWrite(1,writenum,1)
                pygame.display.update()
        elif resp2 == ("r0,"+ readnum + ",0") and
            self.guard_station_panel.button_pushed[i]
            == True:
```

```
                self.guard_station_panel.btn_statuses[i].
                    change_state()
                self.guard_station_panel.button_pushed[i]
                    = False
                self.ybox.sendWrite(1,writenum,0)
                pygame.display.update()



##################################################
#Read guard station light statuses
##################################################
for i in range(0, 4):
    #aqcuiring the actual channel number
    if i == 0:
        readnum = 2
    elif i == 1:
        readnum = 5
    elif i == 2:
        readnum = 8
    elif i == 3:
        readnum = 13
    else:
        print("Something went wrong in the button
            reads4")
```

```
        resp3 = self.ybox.sendRead(0, readnum)
        readnum = str(readnum)
        if resp3 == ("r0,"+ readnum +",1") and self.
            guard_station_panel.light_green[i] ==
            False:
              self.guard_station_panel.lights[i].
                  change_state()
              self.guard_station_panel.light_green[i] =
                   True
              pygame.display.update()

        elif resp3 == ("r0,"+ readnum +",0") and self
            .guard_station_panel.light_green[i] ==
            True:
              self.guard_station_panel.lights[i].
                  change_state()
              self.guard_station_panel.light_green[i] =
                   False
              pygame.display.update()

##############################################
# Read the locks for the man trap and react
    appropriatley
##############################################
```

```
#lock reads for trap door one
##################################
readnum = 9
writenum = 10
readnum_2 = 11
writenum_2 = 11
resp4 = self.ybox.sendRead(0, readnum)
resp5 = self.ybox.sendRead(0, readnum_2)
readnum = str(readnum)
readnum_2 = str(readnum_2)
#all reads done for both above

if resp4 == ("r0," + readnum + ",1") and self.
    mantrap.lock1Closed == True: #checks for a
    lock state change
      self.mantrap.lock_indicator1.change_state() #
          open lock
      self.mantrap.lock1Closed = False #lock open
          state flag
      self.mantrap.trap_door1.change_state()
      self.mantrap.door1Closed = False
      self.ybox.sendWrite(1,writenum,0) # trap door
           in unsecure
      if self.mantrap.indicatorLight == True:
          self.mantrap.indicatorLight = False
```

```
                self.mantrap.secure_indicator.
                    change_state()
            pygame.display.update()


        elif resp4 == ("r0," + readnum + ",0") and self.
            mantrap.lock1Closed == False: #checks for a
            lock state change
                self.mantrap.lock_indicator1.change_state() #
                    open lock
                self.mantrap.lock1Closed = True #lock open
                    state flag
                self.mantrap.trap_door1.change_state()
                self.mantrap.door1Closed = True
                self.ybox.sendWrite(1,writenum,1) # trap door
                     in unsecure
                if resp5 == "r0,11,0" and self.mantrap.
                    indicatorLight == False:
                    self.mantrap.indicatorLight = True
                    self.mantrap.secure_indicator.
                        change_state()
            pygame.display.update()

        #lock reads for trap door 2
        ##############################
```

```
if resp5 == ("r0," + readnum_2 + ",1") and self.
    mantrap.lock2Closed == True: #checks for a
    lock state change
        self.mantrap.lock_indicator2.change_state() #
            open lock
        self.mantrap.lock2Closed = False #lock open
            state flag
        self.mantrap.trap_door2.change_state()
        self.mantrap.door2Closed = False
        self.ybox.sendWrite(1,writenum_2,0) # trap
            door in unsecure
        if self.mantrap.indicatorLight == True:
            self.mantrap.indicatorLight = False
            self.mantrap.secure_indicator.
                change_state()
        pygame.display.update()

elif resp5 == ("r0," + readnum_2 + ",0") and self
    .mantrap.lock2Closed == False: #checks for a
    lock state change
        self.mantrap.lock_indicator2.change_state() #
            open lock
        self.mantrap.lock2Closed = True #lock open
            state flag
        self.mantrap.trap_door2.change_state()
        self.mantrap.door2Closed = True
```

```
            self.ybox.sendWrite(1,writenum_2,1) # trap
                door  secure
            if resp4 == "r0,9,0" and self.mantrap.
                indicatorLight == False:
                self.mantrap.indicatorLight = True
                self.mantrap.secure_indicator.
                    change_state()
            pygame.display.update()



        #read the key and execute only if the key holds
            the power to do so
        resp6 = self.ybox.sendRead(0, 14)
        if resp6 == ("r0,14,1") and self.
            guard_station_panel.panel_enabled == False and
             self.guard_station_panel.panel_keyControl ==
            True:
             self.ybox.sendWrite(1,9,1)
             self.guard_station_panel.enablePanel()
             pygame.display.update()
        if resp6 == ("r0,14,0") and self.
            guard_station_panel.panel_enabled == True and
            self.guard_station_panel.panel_keyControl ==
            True:
```

```
            self.ybox.sendWrite(1,9,0)
            self.guard_station_panel.disablePanel()
            pygame.display.update()

        #read the simulation disable buttton and take
            control power form the key
        if self.guard_station_panel.disable_clicked ==
            True and self.guard_station_panel.
            panel_enabled == False:
            self.guard_station_panel.panel_keyControl =
                False
            self.guard_station_panel.disable_clicked =
                False
            self.ybox.sendWrite(1,9,1)
            self.guard_station_panel.enablePanel()
            pygame.display.update()

        if self.guard_station_panel.disable_clicked ==
            True and self.guard_station_panel.
            panel_enabled == True:
            self.guard_station_panel.panel_keyControl =
                False
            self.guard_station_panel.disable_clicked =
                False
            self.ybox.sendWrite(1,9,0)
            self.guard_station_panel.disablePanel()
```

```
            pygame.display.update()

        #return control power to the key if the sim and
            key match
        if self.guard_station_panel.panel_keyControl ==
            False and resp6 == ("r0,14,1") and self.
            guard_station_panel.panel_enabled == True:
                self.guard_station_panel.panel_keyControl =
                    True

        if self.guard_station_panel.panel_keyControl ==
            False and resp6 == ("r0,14,0") and self.
            guard_station_panel.panel_enabled == False:
                self.guard_station_panel.panel_keyControl =
                    True

        #PLC power switching
        if self.guard_station_panel.PLC1_clicked == True
            and not self.PLC1power:
                if self.PLC2power == True:
                        self.ybox.sendWrite(1,14,0)
                        self.PLC2power = False
                if self.PLC3power == True:
                        self.ybox.sendWrite(1,15,0)
                        self.PLC3power = False
                time.sleep(1)
```

```
            self.ybox.sendWrite(1,13,1)
            self.PLC1power = True
            self.guard_station_panel.enablePLC1()
            pygame.display.update()
            self.guard_station_panel.PLC1_clicked = False
        elif self.guard_station_panel.PLC1_clicked ==
            True and self.PLC1power:
            self.ybox.sendWrite(1,13,0)
            self.PLC1power = False
            self.guard_station_panel.disablePLC()
            pygame.display.update()
            self.guard_station_panel.PLC1_clicked = False

        if self.guard_station_panel.PLC2_clicked == True
            and not self.PLC2power:
            if self.PLC1power == True:
                self.ybox.sendWrite(1,13,0)
                self.PLC1power = False
            if self.PLC3power == True:
                self.ybox.sendWrite(1,15,0)
                self.PLC3power = False
            time.sleep(1)
            self.ybox.sendWrite(1,14,1)
            self.PLC2power = True
            self.guard_station_panel.enablePLC2()
            self.guard_station_panel.PLC2_clicked = False
```

```
elif self.guard_station_panel.PLC2_clicked ==
    True and self.PLC2power:
        self.ybox.sendWrite(1,14,0)
        self.PLC2power = False
        self.guard_station_panel.disablePLC()
        pygame.display.update()
        self.guard_station_panel.PLC2_clicked = False

if self.guard_station_panel.PLC3_clicked == True
    and not self.PLC3power:
        if self.PLC1power:
            self.ybox.sendWrite(1,13,0)
            self.PLC1power = False
        if self.PLC2power:
            self.ybox.sendWrite(1,14,0)
            self.PLC2power = False
        time.sleep(1)
        self.ybox.sendWrite(1,15,1)
        self.PLC3power = True
        self.guard_station_panel.enablePLC3()
        self.guard_station_panel.PLC3_clicked = False
elif self.guard_station_panel.PLC3_clicked ==
    True and self.PLC3power:
        self.ybox.sendWrite(1,15,0)
        self.PLC3power = False
        self.guard_station_panel.disablePLC()
```

```
                pygame.display.update()
                self.guard_station_panel.PLC3_clicked = False


if __name__ == "__main__" :

    green = (200,0,0)
    white = (255, 255, 255)
    black = (0,0,0)
    grey = (200, 200, 200)
    dark_grey = (140, 140, 140)
    light_blue = (0, 0, 255)
    dark_blue = (0, 0, 150)

    prisonSim = PrisonSim()
    prisonSim.on_execute()
```

B.1.2 Guardstation.

The following code handles the interactions with the buttons in the guard station panel.

```
import pygame
from pygame.locals import *
import indicator
```

```
import pygbutton

class GuardStation:

    def __init__(self, sim=None, num_cells=0, x=0, y=0):
        if sim:
            screen = sim.screen
            font = sim.font
            self.cell_btns = []
            self.lights = []
            self.PLC_lights = []
            self.btn_statuses = []
            self.button_pushed = []
            self.light_green = []
            self.panel_enabled = True
            self.panel_keyControl = True #enables the key o be overidden
            self.disable_clicked = False
            self.PLC1_clicked = False
            self.PLC2_clicked = False
            self.PLC3_clicked = False
            btn_height = 40;
            btn_width = 100;
            cell_padding_x = 25
            cell_padding_y = 60
            grey = (200, 200, 200)
            dark_grey = (140, 140, 140)
```

```
white = (255, 255, 255)
black = (0,0,0)

title_height = 25
light_height = 35
light_width = 35
panel_width = btn_width * num_cells + (num_cells+1)*cell_paddin
panel_height = btn_height + 2*cell_padding_y + title_height +

guard_display_panel = pygame.draw.rect(screen, (grey), (x, y,


#flag for cell button toggle
for i in range(0, num_cells):
    self.button_pushed.append(False)

#flag for light to toggle
for i in range(0, 4):
    self.light_green.append(True)

#Title
#screen.blit(font.render('Guard Station Panel', True, (black))

#guard station image loading
green_icon = pygame.image.load("images/greenLightAlt.png")
red_icon = pygame.image.load("images/redLightAlt.png")
```

```
pressed_icon = pygame.image.load("images/pressed.png")
notPressed_icon = pygame.image.load("images/notpressed.png")
panelEnabled = pygame.image.load("images/panelenabled.png")
panelDisabled = pygame.image.load("images/panelDisabled.png")


cell_start_x = x + cell_padding_x
cell_start_y = y + cell_padding_y+title_height+ (0.5 * light_he
light_start_x = x + cell_padding_x + (btn_width*0.5) - (light_w
light_start_y = y + cell_padding_y
status_start_x = x + cell_padding_x
status_start_y = y + cell_padding_y + title_height+ (0.5 * ligh

#disable button to the right of the panel
self.disable_btn = pygbutton.PygButton((cell_start_x + (panel_w
self.disable_btn.draw(screen)


#create enabled/disbaled indicator
panel_icons = [panelEnabled, panelDisabled]
icon_dimensions = ((400,45))
self.enable_label = indicator.Indicator(sim, x + (0.18 * panel_

for i in range(0, num_cells):
    cell_num = i+1
    if cell_num == num_cells-1:
```

```
        button_text = "Trap 1"
    elif cell_num == num_cells:
        button_text = "Trap 2"
    else:
        button_text = "Cell %s" % cell_num

    cell_btn = pygbutton.PygButton((cell_start_x, cell_start_y
    cell_btn.draw(screen)

    #Door light
    if i == num_cells - 2:
        light_icons = [green_icon, red_icon]
        light_dimensions = ((35,35))
        light = indicator.Indicator(sim, light_start_x + (0.63
        self.lights.append(light)
    elif i == num_cells -1:
        pass
    else:
        light_icons = [green_icon, red_icon]
        light_dimensions = ((35,35))
        light = indicator.Indicator(sim, light_start_x, light_s
        self.lights.append(light)

    #button indicator
    status_icons = [notPressed_icon, pressed_icon]
    status_dimensions = ((100, 20))
```

```
        status = indicator.Indicator(sim, status_start_x, status_st

        #adding all objects to respective lists
        self.cell_btns.append(cell_btn)
        self.btn_statuses.append(status)

        #adjusting placements for next iteration of icon and buttor
        light_start_x += btn_width + cell_padding_x
        cell_start_x += btn_width + cell_padding_x
        status_start_x += btn_width + cell_padding_x

    #Light indicators for PLC buttons
    light_icons = [red_icon, green_icon]
    light_dimensions = ((35,35))
    self.light1 = indicator.Indicator(sim, 150, 770, light_icons,
    self.light2 = indicator.Indicator(sim, 350, 770, light_icons,
    self.light3 = indicator.Indicator(sim, 550, 770, light_icons,

    #New buttons for PLC switching
    self.PLC1_BTN = pygbutton.PygButton((100, y - cell_padding_y +
    self.PLC1_BTN.draw(screen)

    self.PLC2_BTN = pygbutton.PygButton((300, y - cell_padding_y +
    self.PLC2_BTN.draw(screen)

    self.PLC3_BTN = pygbutton.PygButton((500, y - cell_padding_y +
```

```
        self.PLC3_BTN.draw(screen)

def disablePanel(self):
    if self.panel_enabled == True:
        self.panel_enabled = False
        self.enable_label.change_state()


def enablePanel(self):
    if self.panel_enabled == False:
        self.panel_enabled = True
        self.enable_label.change_state()


def enablePLC1(self):
        self.light1.change_state(1)
        self.light2.change_state(0)
        self.light3.change_state(0)
        pygame.display.update()


def enablePLC2(self):
        self.light1.change_state(0)
        self.light2.change_state(1)
        self.light3.change_state(0)
        pygame.display.update()
```

```
    def enablePLC3(self):
            self.light1.change_state(0)
            self.light2.change_state(0)
            self.light3.change_state(1)
            pygame.display.update()

    def disablePLC(self):
            self.light1.change_state(0)
            self.light2.change_state(0)
            self.light3.change_state(0)
            pygame.display.update()
```

B.1.3 Ybox.

The following code handles communications with the Y-Box

```
import time
import msvcrt
from pygame.locals import *
import serial
import _thread

#############################################################################
# Author: Evan Plumley and Joseph Daoud
# Date: 6/20/2016
# version: 1.0
```

```
#
#The Ybox class
# dependencies: pygame, pyserial
################################################################################


class Ybox(object):
    def __init__(self):
        self.comMap = {'r10':'R1,0\n', 'r11':'R1,1\n', 'r12':'R1,2\n', 'r13
                       'r15':'R1,5\n','r16':'R1,6\n','r17':'R1,7\n','r18':'R1
                       'r111':'R1,11\n', 'r111':'R1,11\n', 'r112':'R1,12\n',
                       'r115':'R1,15\n', 'r00':'R0,0\n', 'r01':'R0,1\n', 'r02
                       'r05':'R0,5\n','r06':'R0,6\n','r07':'R0,7\n','r08':'R0
                       'r010':'R0,10\n', 'r011':'R0,11\n', 'r012':'R0,12\n',
                       'r015':'R0,15\n', 'w100':'W1,0,0\n', 'w101':'W1,0,1\n'
                       'w120':'W1,2,0\n', 'w121':'W1,2,1\n', 'w130':'W1,3,0\n
                       'w141':'W1,4,1\n', 'w150':'W1,5,0\n', 'w151':'W1,5,1\n
                       'w170':'W1,7,0\n', 'w171':'W1,7,1\n', 'w180':'W1,8,0\n
                       'w1100':'W1,10,0\n', 'w1101':'W1,10,1\n', 'w1110':'W1,1
                       'w1121':'W1,12,1\n', 'w1130':'W1,13,0\n', 'w1131':'W1,1
                       'w1150':'W1,15,0\n', 'w1151':'W1,15,1\n'}
        self.ser = serial.Serial(
            port= 'COM4',
            baudrate=115200,
            parity=serial.PARITY_NONE,
            stopbits=serial.STOPBITS_ONE,
```

```
            bytesize=serial.EIGHTBITS,
            timeout = 1
            )
        time.sleep(1)
        print ("Initialize Complete")



def readLine(self):
    line = self.ser.readline().decode()
    line = line.strip ()
    return line

def sendRead(self, slot, channel):
    msg = self.comMap['r' + str(slot) + str(channel)]
    self.ser.write(msg.encode())
    line = self.ser.readline().decode()
    line = line.strip ()
    return line

def sendWrite(self, slot, channel, value):
    #getter = 'w' + str(slot) + str(channel) + str(value)
    msg = "W" + str(slot) + "," + str(channel) + "," + str(value) + "\r
    self.ser.write(msg.encode())
    line = self.ser.readline().decode()
    line = line.strip ()
```

```
        return line

    def readAll(self, slot):
        slotstr = str(slot)
        msg = 'R0' + slotstr + 'A'
        self.ser.write(msg.encode())
        line = self.ser.readline().decode()
        line = line.strip ()
        return line

    def closePort(self):
        self.ser.close()
        print("serial port closed")
```

B.2 Ladder Logic

The following subsections contain the logic files for each of the PLCs.

B.2.1 CompactLogix.

The following code is running in the CompactLogix PLC.

```
(*******************************************

Import-Export
Version    := RSLogix 5000 v19.01
Owner      := user,
Exported   := Tue Feb 14 14:53:53 2017
```

```
   Note:  File encoded in UTF-8.  Only edit file in a program
          which supports UTF-8 (like Notepad, not Wordpad).

********************************************)
IE_VER := 2.10;

CONTROLLER Prison (ProcessorType := "1769-L23E-QBFC1",
                   Major := 19,
                   TimeSlice := 20,
                   ShareUnusedTimeSlice := 1,
                   RedundancyEnabled := 0,
                   KeepTestEditsOnSwitchOver := 0,
                   DataTablePadPercentage := 50,
                   SecurityCode := 0,
                   SFCExecutionControl := "CurrentActive",
                   SFCRestartPosition := "MostRecent",
                   SFCLastScan := "DontScan",
                   SerialNumber := 16#c01d_8398,
                   MatchProjectToController := No,
                   CanUseRPIFromProducer := No,
                   InhibitAutomaticFirmwareUpdate := 0)
        MODULE Controller (Parent := "Controller",
                           ParentModPortId := 1,
                           CatalogNumber := "1769-L23E-QBFC1",
                           Vendor := 1,
                           ProductType := 14,
```

```
                    ProductCode := 89,
                    Major := 19,
                    Minor := 13,
                    PortLabel := "RxBACKPLANE",
                    ChassisSize := 4,
                    Slot := 0,
                    Mode := 2#0000_0000_0000_0001,
                    CompatibleModule := 0,
                    KeyMask := 2#0000_0000_0001_1111)
END_MODULE

MODULE LocalENB (Parent := "Controller",
                 ParentModPortId := 1,
                 CatalogNumber := "1769-L23E-QBFC1 Ethernet Port",
                 Vendor := 1,
                 ProductType := 12,
                 ProductCode := 191,
                 Major := 19,
                 Minor := 11,
                 PortLabel := "RxBACKPLANE",
                 Slot := 1,
                 NodeAddress := "192.168.108.203",
                 Mode := 2#0000_0000_0000_0000,
                 CompatibleModule := 0,
                 KeyMask := 2#0000_0000_0000_0000)
END_MODULE
```

```
MODULE Local (Parent := "Controller",
              ParentModPortId := 1,
              CatalogNumber := "CompactBus",
              Vendor := 1,
              ProductType := 12,
              ProductCode := 71,
              Major := 19,
              Minor := 11,
              PortLabel := "RxBACKPLANE",
              ChassisSize := 7,
              Slot := 3,
              CommMethod := 805306369,
              Mode := 2#0000_0000_0000_0001,
              CompatibleModule := 0,
              KeyMask := 2#0000_0000_0001_1111)
                ExtendedProp := [[[ ___<public><ConfigID>901</Config
END_MODULE

MODULE Discrete_Inputs (Parent := "Local",
                        ParentModPortId := 1,
                        CatalogNumber := "Embedded IQ16F",
                        Vendor := 1,
                        ProductType := 7,
                        ProductCode := 320,
                        Major := 3,
```

```
                    Minor := 1,
                    PortLabel := "RxBACKPLANE",
                    Slot := 1,
                    Mode := 2#0000_0000_0000_0001,
                    CompatibleModule := 1,
                    KeyMask := 2#0000_0000_0001_1111)
                ExtendedProp := [[[ ___<public><ConfigID>240</Config
                ConfigData  := [16,102,1,34,34,0,0,0];
                CONNECTION Input (Rate := 5000,
                                  EventID := 0)
                           InputData  := [0,3256];
                           InputForceData := [0,0,0,0,-72,12,
                END_CONNECTION

END_MODULE

MODULE Discrete_Outputs (Parent := "Local",
                         ParentModPortId := 1,
                         CatalogNumber := "Embedded OB16",
                         Vendor := 1,
                         ProductType := 7,
                         ProductCode := 322,
                         Major := 3,
                         Minor := 1,
                         PortLabel := "RxBACKPLANE",
                         Slot := 2,
```

```
                    Mode := 2#0000_0000_0000_0001,
                    CompatibleModule := 1,
                    KeyMask := 2#0000_0000_0001_1111)
                ExtendedProp := [[[ ___<public><ConfigID>150</Config
                ConfigData  := [18,102,1,0,0,0,0,0];
                CONNECTION Output (Rate := 5000,
                                   EventID := 0)
                            InputData  := [0,0];
                            InputForceData := [0,0,0,0,0,0,-72
                            OutputData  := [810];
                            OutputForceData := [0,0,0,0,0,0,0,
                END_CONNECTION

END_MODULE

MODULE Analog_IO (Parent := "Local",
                  ParentModPortId := 1,
                  CatalogNumber := "Embedded IF4XOF2",
                  Vendor := 1,
                  ProductType := 10,
                  ProductCode := 120,
                  Major := 1,
                  Minor := 1,
                  PortLabel := "RxBACKPLANE",
                  Slot := 3,
                  Mode := 2#0000_0000_0000_0001,
```

```
                    CompatibleModule := 1,
                    KeyMask := 2#0000_0000_0001_1111)
                  ExtendedProp := [[[ ___<public><ConfigID>350</Config
                  ConfigData  := [24,102,1,0,0,0,0,0,0,0,0];
                  CONNECTION Output (Rate := 5000,
                                     EventID := 0)
                                InputData  := [0,0,0,0,0,0,0,0,0,0];
                                InputForceData := [0,0,0,0,0,0,0,0
                                OutputData  := [0,0];
                                OutputForceData := [0,0,0,0,0,0,0,
                  END_CONNECTION

END_MODULE

MODULE Counters (Parent := "Local",
                 ParentModPortId := 1,
                 CatalogNumber := "Embedded HSC",
                 Vendor := 1,
                 ProductType := 109,
                 ProductCode := 70,
                 Major := 1,
                 Minor := 1,
                 PortLabel := "RxBACKPLANE",
                 Slot := 4,
                 Mode := 2#0000_0000_0000_0001,
                 CompatibleModule := 1,
```

```
                KeyMask := 2#0000_0000_0001_1111)
               ExtendedProp := [[[ ___<public><ConfigID>100</Config
               ConfigData  := [100,102,1,256,0,0,0,0,0,2147483647,
,0,0];
          CONNECTION Output (Rate := 5000,
                                 EventID := 0)
                              InputData  := [0,0,0,0,32,-4096,0
                              InputForceData := [0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                              OutputData  := [0,0,0,0,0,0,0,0
                              OutputForceData := [0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
          END_CONNECTION

END_MODULE

TAG
          DOOR1_SECURE OF Local:2:O.Data.1 (RADIX := Decimal);
          DOOR1_UNLOCK OF Local:2:O.Data.0 (RADIX := Decimal);
          DOOR2_SECURE OF Local:2:O.Data.3 (RADIX := Decimal);
          DOOR2_UNLOCK OF Local:2:O.Data.2 (RADIX := Decimal);
          DOOR3_SECURE OF Local:2:O.Data.5 (RADIX := Decimal);
          DOOR3_UNLOCK OF Local:2:O.Data.4 (RADIX := Decimal);
          HMI : BOOL[32] (RADIX := Decimal) := [2#0,2#0,2#0,2#0,2#0,2
```

```
                    ,2#0,2#0,2#0,2#0,2#0];
        IN_DOOR1_SECURE_SENSOR OF Local:1:I.Data.3 (RADIX := Decim
        IN_DOOR2_SECURE_SENSOR OF Local:1:I.Data.4 (RADIX := Decim
        IN_DOOR3_SECURE_SENSOR OF Local:1:I.Data.5 (RADIX := Decim
        IN_INNER_DOOR_SECURE_SENSOR OF Local:1:I.Data.10 (RADIX :=
        IN_LOCKDOWN OF Local:1:I.Data.6 (RADIX := Decimal);
        IN_OPEN_INSIDE_BTN OF Local:1:I.Data.8 (RADIX := Decimal);
        IN_OPEN_OUTSIDE_BTN OF Local:1:I.Data.9 (RADIX := Decimal)
        IN_OUTER_DOOR_SECURE_SENSOR OF Local:1:I.Data.11 (RADIX :=
        IN_UNLOCK_DOOR1_BTN OF Local:1:I.Data.0 (RADIX := Decimal)
        IN_UNLOCK_DOOR2_BTN OF Local:1:I.Data.1 (RADIX := Decimal)
        IN_UNLOCK_DOOR3_BTN OF Local:1:I.Data.2 (RADIX := Decimal)
        KEY_ON OF Local:1:I.Data.7 (RADIX := Decimal);
        MANTRAP_SECURE OF Local:2:O.Data.8 (RADIX := Decimal);
        PANEL_ENABLE OF Local:2:O.Data.9 (RADIX := Decimal);
        UNLOCK_INSIDE OF Local:2:O.Data.6 (RADIX := Decimal);
        UNLOCK_OUTSIDE OF Local:2:O.Data.7 (RADIX := Decimal);
END_TAG

PROGRAM MainProgram (MAIN := "MainRoutine",
                     MODE := 0,
                     DisableFlag := 0)
        TAG
        END_TAG

        ROUTINE MainRoutine
```

```
                                        N: [XIC(IN_UNLOCK_DOOR1_BTN) XIC(K
                                        N: XIC(IN_DOOR1_SECURE_SENSOR)OTL(I
                                        N: [XIC(IN_UNLOCK_DOOR2_BTN) XIC(K
                                        N: XIC(IN_DOOR2_SECURE_SENSOR)OTL(I
                                        N: [XIC(IN_UNLOCK_DOOR3_BTN) XIC(K
                                        N: XIC(IN_DOOR3_SECURE_SENSOR)OTL(I
                                        N: [XIC(IN_OPEN_INSIDE_BTN) XIC(KE
                                        N: [XIC(IN_OPEN_OUTSIDE_BTN) XIC(K
                                        N: XIC(IN_INNER_DOOR_SECURE_SENSOR
                                        N: XIC(KEY_ON)OTE(PANEL_ENABLE);
                    END_ROUTINE

END_PROGRAM

TASK MainTask (Type := CONTINUOUS,
               Rate := 10,
               Priority := 10,
               Watchdog := 500,
               DisableUpdateOutputs := No,
               InhibitTask := No)
                MainProgram;
END_TASK

CONFIG ASCII(XONXOFFEnable := 0,
             DeleteMode := 0,
             EchoMode := 0,
```

```
                    TerminationChars := 65293,
                    AppendChars := 2573,
                    BufferSize := 82) END_CONFIG

CONFIG ControllerDevice END_CONFIG

CONFIG CST(SystemTimeMasterID := 0) END_CONFIG

CONFIG DF1(DuplicateDetection := 1,
                    ErrorDetection := BCC Error,
                    EmbeddedResponseEnable := 0,
                    DF1Mode := Pt to Pt,
                    ACKTimeout := 50,
                    NAKReceiveLimit := 3,
                    ENQTransmitLimit := 3,
                    TransmitRetries := 3,
                    StationAddress := 0,
                    ReplyMessageWait := 5,
                    PollingMode := 1,
                    MasterMessageTransmit := 0,
                    NormalPollNodeFile := "<NA>",
                    NormalPollGroupSize := 0,
                    PriorityPollNodeFile := "<NA>",
                    ActiveStationFile := "<NA>",
                    SlavePollTimeout := 3000,
                    EOTSuppression := 0,
```

```
                    MaxStationAddress := 31,
                    TokenHoldFactor := 1,
                    EnableStoreFwd := 0,
                    StoreFwdFile := "<NA>") END_CONFIG

CONFIG ExtendedDevice END_CONFIG

CONFIG FaultLog END_CONFIG

CONFIG FileManager END_CONFIG

CONFIG ICP END_CONFIG

CONFIG PCCC END_CONFIG

CONFIG Redundancy END_CONFIG

CONFIG SerialPort(BaudRate := 19200,
                          Parity := No Parity,
                          DataBits := 8 Bits of Data,
                          StopBits := 1 Stop Bit,
                          ComDriverId := DF1,
                          PendingComDriverId := DF1,
                          RTSOffDelay := 0,
                          RTSSendDelay := 0,
                          ControlLine := No Handshake,
```

```
                                PendingControlLine := No Handshake,
                                RemoteModeChangeFlag := 0,
                                PendingRemoteModeChangeFlag := 0,
                                ModeChangeAttentionChar := 27,
                                PendingModeChangeAttentionChar := 27,
                                SystemModeCharacter := 83,
                                PendingSystemModeCharacter := 83,
                                UserModeCharacter := 85,
                                PendingUserModeCharacter := 85,
                                DCDWaitDelay := 0) END_CONFIG

CONFIG TimeSynchronize(Priority1 := 128,
                                                Priority2 := 128,
                                                PTPEnable := 0) END_CONFIG

CONFIG UserMemory END_CONFIG

CONFIG WallClockTime(LocalTimeAdjustment := 0,
                                        TimeZone := 0) END_CONFIG

END_CONTROLLER
```

B.2.2 Siemens S7-300.

The following code is running in the S7-300 PLC.

```
ORGANIZATION_BLOCK OB 1
TITLE = "Main Program Sweep (Cycle)"
```

```
VERSION : 0.1


VAR_TEMP
  OB1_EV_CLASS : BYTE ; //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event
  OB1_SCAN_1 : BYTE ;   //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of O
  OB1_PRIORITY : BYTE ; //Priority of OB Execution
  OB1_OB_NUMBR : BYTE ; //1 (Organization block 1, OB1)
  OB1_RESERVED_1 : BYTE ;       //Reserved for system
  OB1_RESERVED_2 : BYTE ;       //Reserved for system
  OB1_PREV_CYCLE : INT ;        //Cycle time of previous OB1 scan (millisec
  OB1_MIN_CYCLE : INT ; //Minimum cycle time of OB1 (milliseconds)
  OB1_MAX_CYCLE : INT ; //Maximum cycle time of OB1 (milliseconds)
  OB1_DATE_TIME : DATE_AND_TIME ;       //Date and time OB1 started
END_VAR
BEGIN
NETWORK
TITLE =

      A     "IN_UNLOCK_DOOR1_BTN";
      A     "KEY_ON";
      O     "HMI_OPEN_DOOR1";
      O     "HMI_OPEN_ALL";
      =     L     20.0;
      A     L     20.0;
      AN    "IN_LOCKDOWN";
```

```
      =     "DOOR1_UNLOCK";
      A     L     20.0;
      BLD   102;
      R     "DOOR1_SECURE";
NETWORK
TITLE =

      A     "IN_DOOR1_SECURE_SENSOR";
      S     "DOOR1_SECURE";
NETWORK
TITLE =

      A     "IN_UNLOCK_DOOR2_BTN";
      A     "KEY_ON";
      O     "HMI_OPEN_DOOR2";
      O     "HMI_OPEN_ALL";
      =     L     20.0;
      A     L     20.0;
      AN    "IN_LOCKDOWN";
      =     "DOOR2_UNLOCK";
      A     L     20.0;
      BLD   102;
      R     "DOOR2_SECURE";
NETWORK
TITLE =
```

```
      A     "IN_DOOR2_SECURE_SENSOR";
      S     "DOOR2_SECURE";
NETWORK
TITLE =

      A(    ;
      A     "IN_UNLOCK_DOOR3_BTN";
      A     "KEY_ON";
      O     "HMI_OPEN_DOOR3";
      )     ;
      AN    "IN_LOCKDOWN";
      O     "HMI_OPEN_ALL";
      =     "DOOR3_UNLOCK";
      R     "DOOR3_SECURE";
NETWORK
TITLE =

      A     "IN_DOOR3_SECURE_SENSOR";
      S     "DOOR3_SECURE";
NETWORK
TITLE =

      A(    ;
      A     "IN_OPEN_OUTSIDE_BTN";
      A     "KEY_ON";
      O     "HMI_OPEN_MANTRAP_INNER";
```

```
        )       ;
        A       "IN_INNER_DR_SEC_SENSOR";
        AN      "IN_OPEN_INSIDE_BTN";
        =       "UNLOCK_OUTSIDE";
        R       "MANTRAP_SECURE";
NETWORK
TITLE =

        A(      ;
        A       "IN_OPEN_INSIDE_BTN";
        A       "KEY_ON";
        O       "HMI_OPEN_MANTRAP_OUTER";
        )       ;
        A       "IN_OUTER_DR_SEC_SENSOR";
        AN      "IN_OPEN_OUTSIDE_BTN";
        =       "UNLOCK_INSIDE";
        R       "MANTRAP_SECURE";
NETWORK
TITLE =

        A       "IN_OUTER_DR_SEC_SENSOR";
        A       "IN_INNER_DR_SEC_SENSOR";
        S       "MANTRAP_SECURE";
NETWORK
TITLE =
```

```
      A       "KEY_ON";
      =       "ENABLE_PANEL";
END_ORGANIZATION_BLOCK
```

B.2.3 ControlLogix.

The following code is running in the ControlLogix PLC.

```
(*******************************************

  Import-Export
  Version    := RSLogix 5000 v16.04
  Owner      := user,
  Exported   := Tue Feb 14 14:50:57 2017
*******************************************)
IE_VER := 2.7;

CONTROLLER Prison (ProcessorType := "1756-L55",
                   Major := 16,
                   TimeSlice := 20,
                   ShareUnusedTimeSlice := 1,
                   RedundancyEnabled := 0,
                   KeepTestEditsOnSwitchOver := 0,
                   DataTablePadPercentage := 50,
                   SecurityCode := 0,
                   SFCExecutionControl := "CurrentActive",
                   SFCRestartPosition := "MostRecent",
                   SFCLastScan := "DontScan",
```

```
            SerialNumber := 16#0049_e688,
            MatchProjectToController := No,
            InhibitAutomaticFirmwareUpdate := 0)
MODULE Local (Parent := "Local",
              ParentModPortId := 1,
              CatalogNumber := "1756-L55",
              Vendor := 1,
              ProductType := 14,
              ProductCode := 51,
              Major := 16,
              Minor := 3,
              PortLabel := "RxBACKPLANE",
              ChassisSize := 7,
              Slot := 0,
              Mode := 2#0000_0000_0000_0000,
              CompatibleModule := 0,
              KeyMask := 2#0000_0000_0001_1111)
END_MODULE

MODULE EthernetIP (Parent := "Local",
                   ParentModPortId := 1,
                   CatalogNumber := "1756-ENBT/A",
                   Vendor := 1,
                   ProductType := 12,
                   ProductCode := 58,
                   Major := 4,
```

```
                    Minor := 1,
                    PortLabel := "RxBACKPLANE",
                    Slot := 1,
                    NodeAddress := "192.168.108.205",
                    Mode := 2#0000_0000_0000_0000,
                    CompatibleModule := 1,
                    KeyMask := 2#0000_0000_0001_1111)
END_MODULE

MODULE DI_1 (Parent := "Local",
             ParentModPortId := 1,
             CatalogNumber := "1756-IB16",
             Vendor := 1,
             ProductType := 7,
             ProductCode := 11,
             Major := 3,
             Minor := 1,
             PortLabel := "RxBACKPLANE",
             Slot := 2,
             CommMethod := 536870913,
             ConfigMethod := 8388609,
             Mode := 2#0000_0000_0000_0000,
             CompatibleModule := 1,
             KeyMask := 2#0000_0000_0001_1111)
                ConfigData  := [28,16,1,0,0,0,1,1,1,1,0,0,0,0,6553
                CONNECTION StandardInput (Rate := 20000,
```

```
                                                    EventID := 0)
                                            InputData  := [0,0];
                                            InputForceData := [0,0,0,0,0,0,0,0
                    END_CONNECTION

END_MODULE

MODULE DO_1 (Parent := "Local",
             ParentModPortId := 1,
             CatalogNumber := "1756-OB8",
             Vendor := 1,
             ProductType := 7,
             ProductCode := 18,
             Major := 3,
             Minor := 1,
             PortLabel := "RxBACKPLANE",
             Slot := 3,
             CommMethod := 536870913,
             ConfigMethod := 8388609,
             Mode := 2#0000_0000_0000_0000,
             CompatibleModule := 1,
             KeyMask := 2#0000_0000_0001_1111)
                ConfigData  := [28,18,1,0,0,0,0,0,0,0];
                CONNECTION Standard (Rate := 20000,
                                     EventID := 0)
                                    InputData  := [0,0];
```

```
                                    InputForceData := [0,0,0,0,0,0,0,0
                                    OutputData  := [0];
                                    OutputForceData := [0,0,0,0,0,0,0,
                    END_CONNECTION

END_MODULE

MODULE DO_2 (Parent := "Local",
             ParentModPortId := 1,
             CatalogNumber := "1756-OB8",
             Vendor := 1,
             ProductType := 7,
             ProductCode := 18,
             Major := 3,
             Minor := 1,
             PortLabel := "RxBACKPLANE",
             Slot := 4,
             CommMethod := 536870913,
             ConfigMethod := 8388609,
             Mode := 2#0000_0000_0000_0000,
             CompatibleModule := 1,
             KeyMask := 2#0000_0000_0001_1111)
                ConfigData  := [28,18,1,0,0,0,0,0,0,0];
                CONNECTION Standard (Rate := 20000,
                                     EventID := 0)
                                    InputData  := [0,0];
```

```
                                        InputForceData := [0,0,0,0,0,0,0,0
                                        OutputData := [0];
                                        OutputForceData := [0,0,0,0,0,0,0,
                    END_CONNECTION

END_MODULE

TAG
        DOOR1_SECURE OF Local:3:O.Data.1 (RADIX := Decimal);
        DOOR1_UNLOCK OF Local:3:O.Data.0 (RADIX := Decimal);
        DOOR2_SECURE OF Local:3:O.Data.3 (RADIX := Decimal);
        DOOR2_UNLOCK OF Local:3:O.Data.2 (RADIX := Decimal);
        DOOR3_SECURE OF Local:3:O.Data.5 (RADIX := Decimal);
        DOOR3_UNLOCK OF Local:3:O.Data.4 (RADIX := Decimal);
        ENABLE_PANEL OF Local:4:O.Data.1 (RADIX := Decimal);
        HMI : BOOL[32] (RADIX := Decimal) := [2#0,2#0,2#0,2#0,2#0,2
                                ,2#0,2#0,2#0,2#0,2#0];
        IN_DOOR1_SECURE_SENSOR OF Local:2:I.Data.3 (RADIX := Decim
        IN_DOOR2_SECURE_SENSOR OF Local:2:I.Data.4 (RADIX := Decim
        IN_DOOR3_SECURE_SENSOR OF Local:2:I.Data.5 (RADIX := Decim
        IN_INNER_DOOR_SECURE_SENSOR OF Local:2:I.Data.10 (RADIX :=
        IN_LOCKDOWN OF Local:2:I.Data.6 (RADIX := Decimal);
        IN_OPEN_INSIDE_BTN OF Local:2:I.Data.8 (RADIX := Decimal);
        IN_OPEN_OUTSIDE_BTN OF Local:2:I.Data.9 (RADIX := Decimal)
        IN_OUTER_DOOR_SECURE_SENSOR OF Local:2:I.Data.11 (RADIX :=
        IN_UNLOCK_DOOR1_BTN OF Local:2:I.Data.0 (RADIX := Decimal)
```

```
        IN_UNLOCK_DOOR2_BTN OF Local:2:I.Data.1 (RADIX := Decimal)
        IN_UNLOCK_DOOR3_BTN OF Local:2:I.Data.2 (RADIX := Decimal)
        KEY_ON OF Local:2:I.Data.7 (RADIX := Decimal);
        MANTRAP_SECURE OF Local:3:O.Data.7 (RADIX := Decimal);
        UNLOCK_INSIDE OF Local:4:O.Data.0 (RADIX := Decimal);
        UNLOCK_OUTSIDE OF Local:3:O.Data.6 (RADIX := Decimal);
END_TAG

PROGRAM MainProgram (MAIN := "MainRoutine",
                     MODE := 0,
                     DisableFlag := 0)
                TAG
                END_TAG

                ROUTINE MainRoutine
                                N: [XIC(IN_UNLOCK_DOOR1_BTN) XIC(K
                                N: XIC(IN_DOOR1_SECURE_SENSOR)OTL(I
                                N: [XIC(IN_UNLOCK_DOOR2_BTN) XIC(K
                                N: XIC(IN_DOOR2_SECURE_SENSOR)OTL(I
                                N: [XIC(IN_UNLOCK_DOOR3_BTN) XIC(K
                                N: XIC(IN_DOOR3_SECURE_SENSOR)OTL(I
                                N: [XIC(IN_OPEN_INSIDE_BTN) XIC(KE
                                N: [XIC(IN_OPEN_OUTSIDE_BTN) XIC(K
                                N: XIC(IN_INNER_DOOR_SECURE_SENSOR
                                N: XIC(KEY_ON)OTE(ENABLE_PANEL);
                END_ROUTINE
```

```
        END_PROGRAM

        TASK MainTask (Type := CONTINUOUS,
                       Rate := 10,
                       Priority := 10,
                       Watchdog := 500,
                       DisableUpdateOutputs := No,
                       InhibitTask := No)
                        MainProgram;
        END_TASK

CONFIG ASCII(XONXOFFEnable := 0,
                      DeleteMode := 0,
                      EchoMode := 0,
                      TerminationChars := 65293,
                      AppendChars := 2573,
                      BufferSize := 82) END_CONFIG

CONFIG ControllerDevice END_CONFIG

CONFIG CST(SystemTimeMasterID := 0) END_CONFIG

CONFIG DF1(DuplicateDetection := 1,
                     ErrorDetection := BCC Error,
                     EmbeddedResponseEnable := 0,
```

```
                    DF1Mode := Pt to Pt,
                    ACKTimeout := 50,
                    NAKReceiveLimit := 3,
                    ENQTransmitLimit := 3,
                    TransmitRetries := 3,
                    StationAddress := 0,
                    ReplyMessageWait := 5,
                    PollingMode := 1,
                    MasterMessageTransmit := 0,
                    NormalPollNodeFile := "<NA>",
                    NormalPollGroupSize := 0,
                    PriorityPollNodeFile := "<NA>",
                    ActiveStationFile := "<NA>",
                    SlavePollTimeout := 3000,
                    EOTSuppression := 0,
                    MaxStationAddress := 31,
                    TokenHoldFactor := 1,
                    EnableStoreFwd := 0,
                    StoreFwdFile := "<NA>") END_CONFIG

CONFIG ExtendedDevice END_CONFIG

CONFIG FaultLog END_CONFIG

CONFIG FileManager END_CONFIG
```

```
CONFIG ICP END_CONFIG

CONFIG PCCC END_CONFIG

CONFIG Redundancy END_CONFIG

CONFIG SerialPort(BaudRate := 19200,
                  Parity := No Parity,
                  DataBits := 8 Bits of Data,
                  StopBits := 1 Stop Bit,
                  ComDriverId := DF1,
                  PendingComDriverId := DF1,
                  RTSOffDelay := 0,
                  RTSSendDelay := 0,
                  ControlLine := No Handshake,
                  PendingControlLine := No Handshake,
                  RemoteModeChangeFlag := 0,
                  PendingRemoteModeChangeFlag := 0,
                  ModeChangeAttentionChar := 27,
                  PendingModeChangeAttentionChar := 27,
                  SystemModeCharacter := 83,
                  PendingSystemModeCharacter := 83,
                  UserModeCharacter := 85,
                  PendingUserModeCharacter := 85,
                  DCDWaitDelay := 0) END_CONFIG
```

```
CONFIG UserMemory END_CONFIG

CONFIG WallClockTime(LocalTimeAdjustment := 0,
                     TimeZone := 0) END_CONFIG

END_CONTROLLER
```

Bibliography

1. Air Force Institute of Technology, Cyber Operations Program Description, Department of Electrical and Computer Engineering, Graduate School of Engineering and Management, Wright-Patterson Air Force Base, Ohio (www.afit.edu/ENG/programs.cfm?p=4&a=pd), 2014.

2. Z. Basnight, J. Butts, J. Lopez and T. Dube, Firmware Counterfeiting and Modification Attacks on Programmable Logic Controllers, M.S. Thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright–Patterson Air Force Base, Ohio, 2013.

3. A. Bauer and I. Byock, Talking with Your Doctor about Prognosis, American Society of Clinical Oncology, San Diego, California (www.cancer.net/blog/2014-08/talking-your-doctor-about-prognosis), 2014.

4. J. Butts and M. Glover, How industrial control system security training is falling short, in *Critical Infrastructure Protection IX*, M. Rice and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 135–149, 2015.

5. R. Candell, T. Zimmerman and K. Stouffer, An Industrial Control System Cybersecurity Performance testbed, NISTIR 8089, National Institute of Standards and Technology, Gaithersburg, Maryland, 2015.

6. L. Folkerth, Forensic Analysis of Industrial Control Systems, InfoSec Reading Room, SANS Institute, Bethesda, Maryland (www.sans.org/reading-room/whitepapers/forensics/forensic-analysis-industrial-control-systems-36277), 2015.

7. H. Holm, M. Karresand, A. Vidstrom and E. Westring, A Survey of Industrial Control System testbeds, Swedish Defense Research Agency (FOI),

Sweden (www.springer.com/cda/content/document/cda_downloaddocument/9783319265018-c2.pdf?SGWID=0-0-45-1532903-p177788982), 2015.

8. Idaho National Laboratory, INL Cyber Security Research: Defending the Network Against Hackers, Idaho Falls, Idaho (www4vip.inl.gov/research/inl-cyber-security-research), 2016.

9. Idaho National Laboratory, University Partnerships, Idaho Falls, Idaho (www.inl.gov/inl-initiatives/education/), 2016.

10. Industrial Control Systems Cyber Emergency Response Team, Cyber-Attack Against Ukrainian Critical Infrastructure, Alert (IR-ALERT-H-16-056-01), U.S. Department of Homeland Security, Washington, DC, 2016.

11. International Information System Security Certification Consortium, Information Security Certification Programs, Clearwater, Florida (www.isc2.org/credentials/default.aspx), 2016.

12. National Collegiate Cyber Defense Competition, 2016 Rules, San Antonio, Texas (www.nationalccdc.org/index.php/competition/competitors/rules), 2016.

13. M. Poag, Medical Student Education Program in Psychiatry, New York University School of Medicine, New York University, New York (www.med.nyu.edu/psych/education/medical-student-education), 2016.

14. Sandia National Laboratories, Distributed Energy Technology Laboratory, Albuquerque, New Mexico (energy.sandia.gov/wp-content/gallery/uploads/DETL_Factsheet_SAND2010-3643_Aug2011.pdf), 2016.

15. Sandia National Laboratories, SCADA Testbeds, Albuquerque, New Mexico (energy.sandia.gov/energy/ssrei/gridmod/

cyber-security-for-electric-infrastructure/scada-systems/testbeds/), 2016.

16. SANS Institute, ICS Training Courses, Bethesda, Maryland (ics.sans.org/training/courses), 2016.

17. SANS Institute, SEC562: CyberCity Hands-On Kinetic Cyber Range Exercise, Bethesda, Maryland (www.sans.org/course/cybercity-hands-on-kinetic-cyber-range-exercise), 2016.

18. Shodan, Industrial Control Systems (www.shodan.io/explore/category/industrial-control-systems), 2016.

19. E. Skoudis, How to build a completely hackable city in five steps: And why you should build your skills in this arena, presented at *Sans Pen Test Hackfest*, 2013.

20. K. Stouffer, J. Falco and K. Scarfone, Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.

21. University of Texas at San Antonio, Cyber Panoply, San Antonio, Texas (www.cyberpanoply.com/index.html), 2012.

22. J. Yoon, Framework for Evaluating the Readiness of Cyber First Responders for Industrial Control Systems, M.S. Thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 2016.

23. J. Yoon, S. Dunlap, J. Butts, M. Rice and B. Ramsey, Evaluating the readiness of cyber first responders responsible for critical infrastructure protection, *International Journal for Critical Infrastructure Protection*, vol. 13, pp. 19–27, 2016.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23–03–2017	Master's Thesis	Aug 2015 — Mar 2017

4. TITLE AND SUBTITLE

Multi-PLC Exercise Environments for Training ICS First Responders

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Daoud, Joseph, K, 1LT, USA

5d. PROJECT NUMBER

17G310

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-17-M-020

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of Homeland Security ICS-CERT
POC: Neil Hershfield, DHS ICS-CERT Technical Lead ATTN: NPPD/CSC/NCSD/US-CERT Mailstop: 0635 245 Murray Lane, SW, Bldg 410, Washington, DC 20528 Email: ics-cert@dhs.gov phone: 1-877-776-7585

10. SPONSOR/MONITOR'S ACRONYM(S)

DHS ICS-CERT

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

When systems are targeted by cyber attacks, cyber first responders must be able to react effectively, especially when dealing with critical infrastructure. Training for cyber first responders is lacking and most existing exercise platforms are expensive, inaccessible or ineffective. This paper presents a mobile training platform which incorporates a variety of programmable logic controllers into a single system which facilitates the development of the unique skills required of cyber first responders operating in the realm of industrial control systems. The platform is modeled after a jail in the northeastern United States and was developed to maximize realism. Example training scenarios are provided to address specific skills and techniques. Results show that the platform is robust enough to conduct sustained training exercises that address a curriculum that has been proposed for cyber first responders.

15. SUBJECT TERMS

Indusctrial Control Systems, Cyber First Responders, Training Platform

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Barry E. Mullins, AFIT/ENG
U	U	U	U	121	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x7979; barry.mullins@afit.edu