SAND2020-1183C
Proceedings of the 2020 28th Conference on Nuclear Engineering
Joint With the ASME 2020 Power Conference
ICONE28-POWER2020
August 2-6, 2020, Anaheim, California, USA
ICONE28-POWER2020-XXXXX

ICONE2020-####

EMULATION METHODOLOGY OF PROGRAMMABLE LOGIC CONTROLLERS FOR CYBERSECURITY APPLICATIONS

Raymond Fasano, Christopher Lamb
Sandia National Laboratories
Albuquerque, NM

Mohamed El Genk, Timothy Schriener, Andrew Hahn
Institute for Space and Nuclear Power Studies and
Nuclear Engineering Department
University of New Mexico
Albuquerque, NM

ABSTRACT

A programmable logic controller (PLC) emulation methodology can dramatically reduce the cost of high-fidelity OT network emulation without compromising specific functionality. This PLC emulation methodology was developed in an ongoing effort at the University of New Mexico's Institute for Space and Nuclear Power Studies to develop an emulytic™ platform for PWR plants funded by DOE's Nuclear Engineering University Program (NEUP) in collaboration with Sandia National Laboratories. The developed PLC emulation methodology is validated to support cybersecurity analyses of operational technology (OT) networks. This PLC emulation methodology identifies and characterizes key physical and digital signatures of interest. The obtained and displayed digital signatures include the network response, traffic, and software version, while the selected physical signatures include the actuation response time and sampling rate. An extensive validation analysis is performed to characterize the signatures of the real, hardware-based PLC and the emulated PLC. These signatures are then compared to highlight differences and quantify the parameters that can be changed to optimize the emulation fidelity.

Keywords: Nuclear, Cybersecurity, PLC, Emulation

NOMENCLATURE

DHCP	dynamic host control protocol
DOE	department of energy
HITL	hardware-in-the-loop
I&C	instrumentation & control
I/O	input-output
ICS	industrial control systems
IT	informational technology
NEUP	Nuclear Energy University Program
OT	operational technology
PCAP	packet capture
PID	proportional-integral-derivative
PLC	programable logic controller
VMS	virtual machines

1. INTRODUCTION

Advanced malware attacks against industrial control systems (ICSs) such as Stuxnet, TRISIS, and CrashOverride have proven that even air gapped networks are at risk to infiltration [1-3]. For information technology (IT) systems emulated computer systems are commonly used to perform controlled cybersecurity experiments in a contained virtual environment [4]. In order to extend the same capability to OT networks emulation models for PLCs are needed. PLCs are specialized digital computers used for the autonomous operation and safety systems within critical infrastructure. Therefore, when emulating PLCs specific considerations should be made to ensure representative physical and digital response characteristics. These considerations are typically not considered when emulating VMs for IT networks due to differences in functionality and uses.

PLC emulation enables high fidelity cybersecurity and physical effect modeling of OT networks, such as the instrumentation and control (I&C) systems in nuclear power plants, without hardware-in-the-loop (HITL) integration. In a HITL setup, a physical PLC is integrated into the digital network being tested. Experiments that use PLCs as HITL are expensive, difficult to scale, and change relative to the emulated systems. Running an experiment on an entire I&C system architecture, which may include dozens of devices, can become impractical if HITL is used for every digital device on the network. A PLC emulation methodology, thus, provides a practical path forward to study the current cybersecurity posture of OT networks for critical infrastructure and aid in the design of secure architectures for future designs.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

2. BACKGROUND

Emulated computer systems, also referred to as virtual machines (VMs), are commonly used to perform controlled security experiments in a contained virtual environment in cybersecurity analyses for enterprise IT systems. The Department of Energy's (DOE) SCEPTRE framework, developed at SNL to enable cybersecurity analyses of ICSs, can start up and handle virtual network communication between a large number of VMS using ICS protocols written to specification [5]. While this framework has been used to model digital components of electrical transmission grids and solar power systems, it has not yet been applied to cybersecurity of nuclear power plants. Extending the SCEPTRE framework to modeling the safety I&C systems of nuclear power plants requires the development of emulation models for the PLCs used for autonomous control and the safety system actuation within the plant. PLCs are unique digital computers that require additional considerations when being emulated since PLCs are hard real-time systems, use networking protocols specific to industrial processes, and control physical processes utilizing a variety of inputs and outputs (I/O).

Most efforts to develop emulation models or VMs for PLCs have focused mainly on developing virtualization platforms for PLC programming and not virtualization of the computer system. For example, Chunjie and Hui have investigated developing a PLC VM based on IEC 61131-3 standard PLC programing [6]. This VM was intended to run on the PLCs themselves to create a platform that supports running common programing based on the IEC 61131-3 standard, across different hardware designs. While capable of running PLC control programming, the developed VM was not designed to replicate the characteristics of a specific PLC. Thamrin and Ismail also developed a VM for PLCs to create a development environment for PLC programming [7]. This VM development environment was planned to help train programmers in the specialized programming languages used by PLCs. The VM also focused on only running the PLC's control programming and not creating a system level virtual machine of a PLC. Gasser also developed a VM testing environment for using standardized PLC programming across different PLC hardware, similar to Chunjie and Hui [8]. Notably, Thiago Alves used OpenPLC, opensource PLC software, to study the virtualization of control systems for cybersecurity analysis [9].

None of these prior efforts were aimed at creating metrics to develop an PLC emulation methodology to couple an emulated PLC to a simulated process. Thus, the objective of the present work is to develop an emulation methodology for a PLC and establish metrics to validate the developed emulated PLC when coupled to a simulated process. This emulation methodology is applied to a representative open-source PLC implementation, and the emulated PLC is validated against the recorded physical and digital signatures of the real, physical PLC hardware. The validated PLC emulation methodology will be incorporated in the DOE SCEPTRE framework to support cybersecurity investigations of I&C system architectures in nuclear power plants. The next section details the developed and validated PLC emulation methodology in this work.

Table 1 outlines the general steps of the PLC emulation methodology. Emulating a PLC implies that the digital and physical behavior of the PLC is reproduced by another system through computational means. The degree of emulation is determined by the project requirements. A full emulation would reproduce the functionality of the hardware, firmware, kernel, and operating system used by the PLC. A partial emulation would replicate only some of these functionalities. Investigations of potential vulnerabilities within software programs or the computer's operating system might only require kernel and software emulation, while investigating potential exploits of vulnerabilities in a chipset's instruction set could require full emulation at the hardware and firmware levels. Several partial emulators of computer systems are available which emulate the kernel and software of a device. Full emulators, however, are far less common due to the drastic increase in complexity and computational cost.

Step	Action
1	Determine the degree of emulation that is needed
2	Use commercial/Open-source emulation software or develop the emulation that is needed
3	Based on the emulation requirements determine the physical and digital signatures of the PLC that need to be emulated
4	Benchmarked the emulated PLC against the real PLC using a representative test environment to determine the validity of the emulation
5	Collect data for both the real and emulated PLC and compare the physical and digital signatures
6	Change the emulation or configuration of the PLCs as needed until the signatures of the real and emulated PLCs converge to an acceptable range outlined by the project requirements

TABLE 1: STEPS WITHIN GENERAL PLC EMULATION METHODOLOGY

The present work requires that the emulated PLC approximate the real PLC, such that the differences between the two systems do not affect the behavior of the connected physics model and could support planned cybersecurity analysis. For the present project, this is accomplished using kernel and software emulation. In addition, the real and emulated systems should be interchangeable and not impacted by the computer hardware running the emulation.

Once a PLC emulation is successfully developed, it is validated against the real system. When obscuring the firmware and the underlining hardware of a PLC, the only way to determine if a PLC is real or emulation is to observe the digital and physical signatures of the device. Table 2 shows the

signature metrics used to validate the PLC emulation. The selected digital signature metrics of a PLC include the network response, the network traffic, and the software versions. The network response of a device quantifies how the network traffic is transmitted and received and determines the rate of data transfer. Similarly, the underlying network traffic for the system determines the type and frequency of the network data packets being transmitted and/or received. Finally, the software versions determine if the exact same software is running on the emulated and real device. If a cybersecurity flaw exists in the software, it should be exploitable on both the real and emulated PLC in exactly the same manner.

Digital Signatures
- Network response
- Network traffic
- Software versions
Physical Signatures
- Actuation response time
- Sampling rate

TABLE 2: METRICS TO COMPARE A REAL AND EMULATED PLC

The selected physical signature metrics for the PLC include the actuation response time and sampling rate (Table 2). The actuation response time is the time required for a PLC to receive data, compute an output, and send an actuation signal. Differing actuation response times would lead to different physical outcomes, by influencing the time history of the physical process. Similarly, the sampling rate of a digital controller, also referred to as the scan time, would affect the process being controlled by influencing the gain values of complex controller designs. The sampling rate of the PLC also determines whether the system is running hard real-time. In order to achieve a deterministic response PLCs are required to run hard real-time. For time critical applications designers can be confident that actuation response times will be approximately between one to two sampling periods plus network latency.

The digital signature metrics are important from a cybersecurity perspective, while the physical signature metrics determines the fidelity of the PLC's response to the connected process. Since PLCs are the interface between the physical and digital world it is paramount that the signatures of the emulated and real PLC be quantified to validate the cyber-physical coupling of the emulation.

3. TESTING METHODOLOGY

The developed emulation methodology for PLCs is performed and validated using a representative, open-source PLC architecture consisting of a Raspberry Pi 4 minicomputer running the OpenPLC software, implementing IEC 61131-3 standard programming for PLCs [10]. The Raspberry Pi is chosen because of the availability of its open source operating system to create images, the functionality of its on-board digital/analog IO, compatibility with OpenPLC, and the ability to emulate the operating system. The specifications of the real and emulated PLCs are summarized in Table 3. VMware emulation software is used to emulate an Ubuntu Server kernel and software for the emulated PLC [11]. Both the real and emulated PLCs run the same operating system and OpenPLC software. To reduce the differences between the Raspberry Pi 4 hardware and the PC running the VMware emulation software, four gigabytes of RAM and four processor cores are allocated to the emulated PLC in VMware.

System	Real	Emulated
Hardware	Raspberry Pi 4	VMware Virtual Machine
CPU	Broadcom BCM2711, Quad core Cortec-A72 (ARM v8) 64-bit SoC @ 1.5Ghz	AMD FX-8370 64-bit SoC @ 4.3Ghz (4 virtual cores)
RAM	4 Gb	4 Gb
Operating System	Ubuntu 19.10	Ubuntu 19.10
Control Software	OpenPLC Version 3	OpenPLC Version 3
Network Interface	Gigabit Ethernet	Gigabit Ethernet

TABLE 3: REAL AND EMULATED PLC SPECIFICATIONS

The testing environment used to validate the PLC emulation against the physical hardware links the real or emulated PLC to a transient simulation model running in Matlab Simulink (Fig. 1) [12]. The Simulink simulation running on a separate Linux server takes the place of an external physical process being controlled by the PLC within the testing environment. Using a simulated process complicates the comparison of the actuation response times of the real and emulated controllers due to an additional requirement of running the simulation in sync with real time when using asynchronous communication. Actual physical processes are continuous and do not add the additional complexity of getting a simulated process to run in sync with real-time. However, when doing cybersecurity research on physical processes, such as a nuclear reactor, using a simulated process is the only option due to safety and cost. The nuances of using a simulated process versus a physical process and the effect on the real and emulated PLC comparison are explained in more detail in proceeding sections.

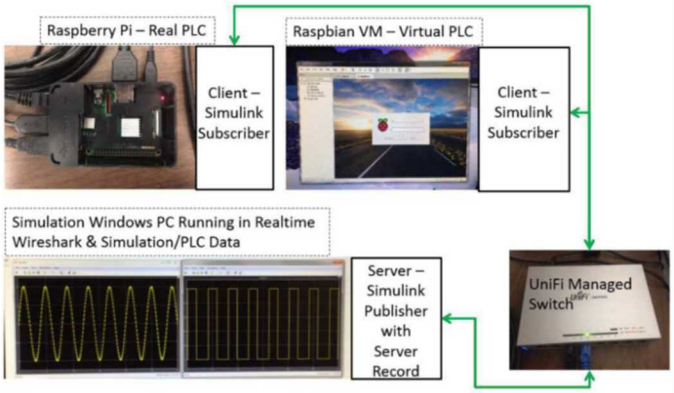

FIGURE 1: OVERVIEW OF REAL AND EMULATED PLC COMPARISON EXPERIMENTAL SETUP

An interface program is developed to handle the inter-process asynchronous communication of data values between the Simulink simulation and the remote PLC. The interface communicates state variables generated within the Simulink simulation to a python script using an S-function which communicates, using shared memory, with an external Python script (Fig. 2).

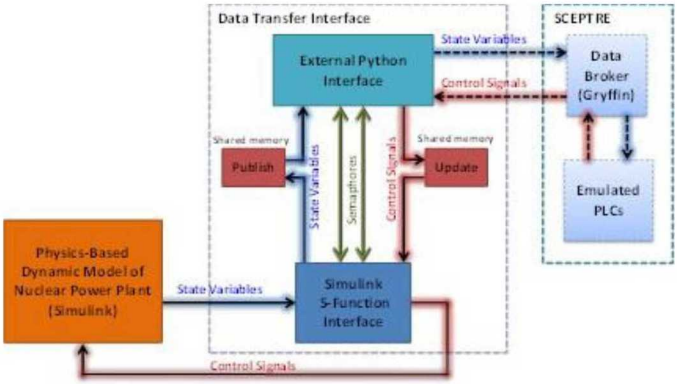

FIGURE 2: LINKING BLOCK DIAGRAM FOR TEST SYSTEM SHOWING DATA FLOW BETWEEN SIMULINK SIMULATION MODEL AND REAL/EMULATED PLCS

The python interface program creates a ModbusTCP/IP server to write or read input and output registers on either the emulated or real PLC. ModbusTCP/IP was originally a serial protocol and that was specifically created to be used with PLCs. ModbusTCP/IP is currently still used in OT networks making it an ideal protocol to benchmark the real and emulated PLCs network response characteristics. Each PLC uses OpenPLC runtime, which is compatible with ModbusTCP/IP, to take the values written to input registers and use those values to calculate a response to be written to the output registers. The inter-process communication programs used in these analyses are asynchronous with the Simulink simulation and allow for closed-loop HITL integration. Process variables are generated by the Simulink simulation, communicated to a HITL PLC (real or emulated), and the response from the PLC is piped back into the Simulink simulation to be used as an actuation signal.

Figure 1 shows the components of the test network used for testing and validation. The test network is a simple ethernet network. It consists of a Linux server, which is the computer running the Simulink model, a UniFi switch, and a PLC. For each experiment, only one PLC is connected to the UniFi switch at a time. The Simulink simulation within the benchmark testing environment uses a square wave with an amplitude of one, a period of five seconds, a pulse with of twenty percent of the period, and a zero second phase delay.

The square wave is outputted by the Linux server via ModbusTCP/IP and is sent to the UniFi switch to be routed to the real or emulated PLC. An isolated testing network was used to eliminate network routing differences between the real and emulated PLC. The local Dynamic Host Configuration Protocol (DHCP) server was run on the Linux server to handle IP address allocation. The Wireshark utility was used to capture network data traffic between the Linux server and the real or emulated PLC [13].

When the PLC receives a new value from the Linux server, the implemented ladder logic programming in OpenPLC determines if the square wave is above or below a predefined setpoint. A setpoint value of 0.5 was used to determine the state of the output register. An input value greater than 0.5 would result in a value of one, while a value less than 0.5 results in a value of zero. The total Simulink simulation length for each run was three-hundred seconds using a major time step of 50 ms. The Python interface program reads the output register of the PLC and sends the response back to the Linux server to be used in the Simulink Simulation as an actuation response signal. In this experiment the real and emulated actuation response signals were recorded in Simulink for consistency.

The following sections describe the results of the validation testing of the emulation methodology. A total of five tests were run for both the real and emulated PLC to characterize the digital and physical characteristics of each PLC independently. First in Section 3.1, an analysis is performed investigating the real-time condition of the linked Simulink simulation and PLC. Sections 4 and 5 present the validation testing of the digital signatures of the emulated PLC. Sections 6 and 7 present the validation testing for the physical signatures, showing the results of the sampling rate and actuation response time, respectively.

3.1 REAL-TIME CONDITION

As addressed previously, a simulated process is used to provide process variables to the real and emulated PLCs instead of a physical process. Therefore, the simulation model attempts to run in sync with real time by using custom code in the Python interface program to ensure that both the real and emulated controllers receive the same values from the simulation at the same rate. Real-Time synchronization or a consistent implementation of real-time sync is required for comparing the real PLC and emulated PLC when using asynchronous communication. If the simulated process operates at a different

rate, it will affect the actuation response time of the controller. For example, if the simulation for the real PLC runs at a slower rate relative to the simulation for the emulated PLC, it is possible that the real PLC will record faster actuation response times. Furthermore, in applications were the response of transient physics models is important, real-time sync of the simulation becomes paramount. Accurate timing of asynchronous signals enables reproducibility of the physical response of transient physics models coupled with real or emulate PLCs. Validity of system response is out of the scope of this work, making the real-time sync implementation less significant allowing for an approximate real-time implementation to be acceptable for a comparison.

To investigate how the timestep size impacted the real-time synchronization, Simulink simulations were run with major time steps of 25, 50, 100, 250, and 500ms. It was found that the deviation from real-time sync and the major time step were inversely proportional to each other. The computer running the simulation has a fundamental limit of how fast it can run the simulation. If the computational time is greater than the major time step no real-time sync software implementation will be able to overcome this limitation. Given the computation resources of the Linux server used, 50 ms was chosen as a representative major time step for more complex physics models with acceptable real-time sync performance given the external Python interface used.

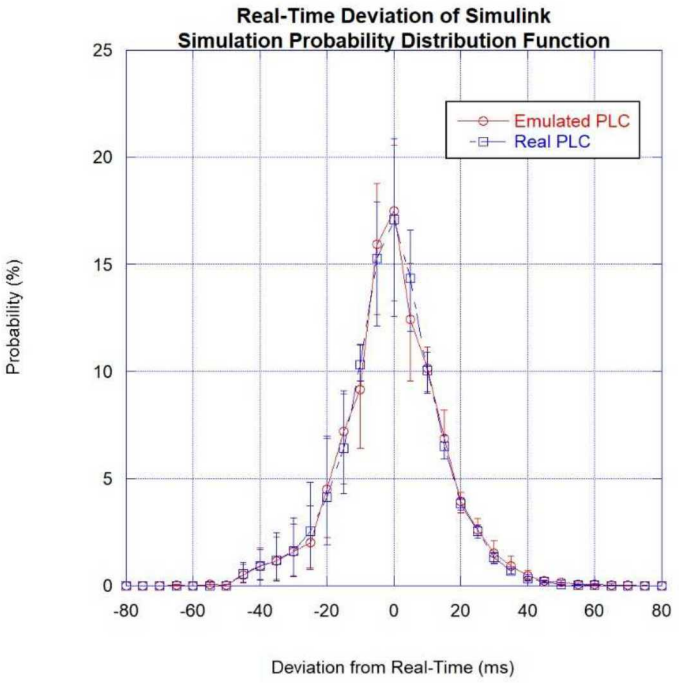

FIGURE 3: SIMULINK SIMULATION DEVIATION FROM REAL-TIME SYCN

Figure 3 shows the real-time deviation of the Simulink simulated process, calculated by subtracting wall time from simulation time, when communicating the real and emulated PLC. Positive deviation from real time is the Simulink simulation running slower than real-time and negative deviation is faster. Error bars are the first and third quartile of the data collected.

On average the Simulink simulation had a deviation from real-time of -2.3ms (14.85 sigma) when communicating with the emulated PLC and an average deviation of -2.66ms (14.29 sigma) when communicating with the real PLC. The real-time deviation between the real and emulated PLC are within statistics of each other enabling the controllers to be compared without the simulated process effecting the validity of the comparison.

4. NETWORK RESPONSE

The network response characteristics of the emulated and real PLCs using a representative industrial protocol are collected using Wireshark to capture the network traffic between the Linux server and the PLC. Wireshark is run on the Linux server for all simulation cases. The network response of the emulated and real PLCs is evaluated by comparing the ModbusTCP/IP packet round-trip time and the network statistics for the ModbusTCP/IP socket communication between the Linux server and real or emulated PLC. The network round trip time is important because the slowest exchange of ModbusTCP/IP packets determines the physical limit of outputting the data by the simulation to the PLC and fingerprinting the devices on the network. Therefore, the lower limit of the sampling time for the PLC is that of the slowest communication of data from the server to the client. In other environments, the communication method may be different from the ModbusTCP/IP communication used here, but the underlining principle of quantifying the rate at which data can be transferred to quantify the network response remains pertinent.

The ModbusTCP/IP protocol uses query and response packets to read and write holding registers on PLCs. Figure 4 shows the network response to query-response and Fig. 5 shows the response-query packet pairs for the real and emulated PLCs. The Python Interface program along with network latency controlled the rate at which query-response and response-query packets were sent and received. As seen in Fig. 4 the emulated PLC has a higher probability of sending and receiving ModbusTCP/IP packets at a faster rate than the real PLC, suggesting an overall faster Query-Response network response. Overall the emulated PLC has a faster response time with an average of 0.73 ms than the real PLC with an average of 1.78 ms. Interestingly, the response-query round-trip time shows that the real and emulated PLC had virtually the same response characteristics, Fig. 5. For response-query packet round-trip times the emulated PLC has an average round-trip time of 23.56 ms and the real PLC has an average round-trip time of 21.87 ms.

The Query-Response characteristics are dictated by the individual hardware for each PLC, while the Response-Query characteristics are dependent on the Python interface code. In this case the emulated PLC hardware is faster than the emulated PLC leading to a noticeable difference in the Query-Response characteristics. Overall however, the major time step used in the Simulink process simulator was 50 ms. Relative to the magnitude of the major time step the difference in network

response characteristics has a negligible effect on the physical process.

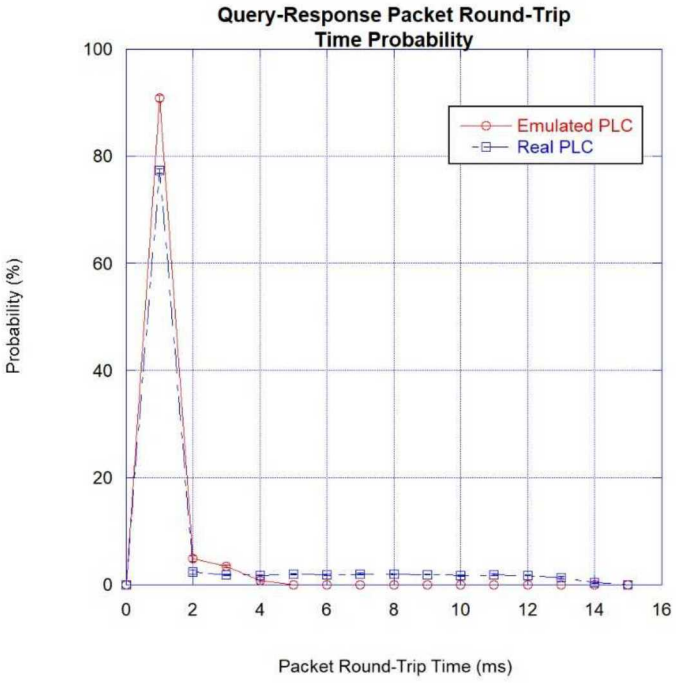

FIGURE 4: QUERY-RESPONSE MODBUSTCP/IP PACKET ROUND-TRIP TIME

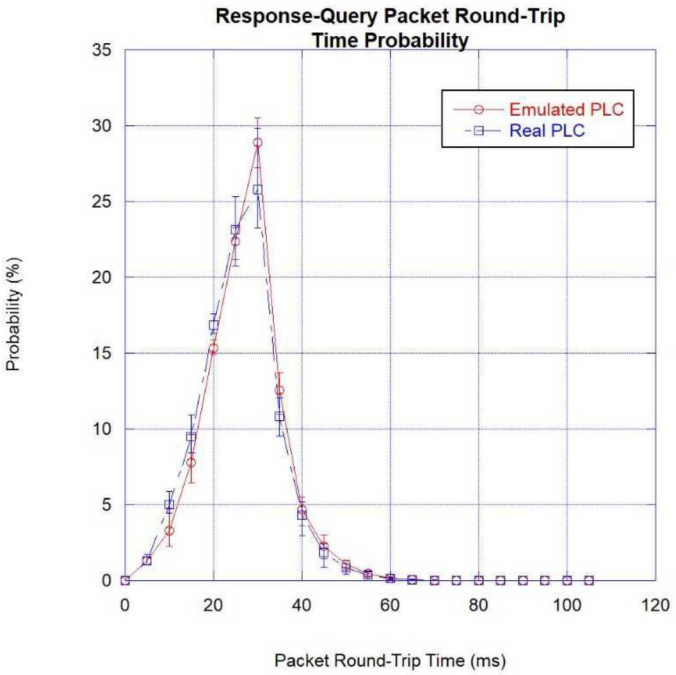

FIGURE 5: RESPONSE-QUERY MODBUSTCP/IP PACKET ROUND-TRIP TIME

From a digital perspective the network response characteristics are significantly different such that network monitoring would be able to distinguish the real and emulated PLC.

5. NETWORK TRAFFIC

The collected packet capture (PCAP) data is analyzed to determine the types of packets sent during the experiment. The data for all experiments was combined for the real and emulated PLC to decrease the differences in the time of day each experiment is ran. It is possible that some programs will send out network communication based on the time of day or the current state of the operating system. The numbers and categories of data packets collected by the Wireshark utility are given in Table 4 for the emulated and real PLC tested. The vast majority (> 99%) of the collected data packets are TCP/IP packets. The fraction of the total packets collected in each category between the real and emulated PLCs differed by < 1%.

PLC Type	Packet Type	Average	STD	Percentage of Total (%)
Real	TCP	25666.60	1946.36	99.86
	UDP	35.20	8.15	0.14
Emulated	TCP	27455.40	1266.14	99.84
	UDP	43.60	16.88	0.16

TABLE 4: NETWORK TRAFFIC CAPTURE FOR THE REAL AND EMULATED PLC

The network traffic signatures for the emulated PLC are determined to be comparable to the real PLC with the exception of the bandwidth; the emulated PLC had a slightly higher bandwidth. The collected PCAP data in Table 4 shows however, that the emulated PLC can use the same network protocols and generate packets of the same data types and with similar proportions to the total level of network traffic as the real PLC. These similarities in types and proportions of network traffic are important for the capability of the emulated PLC to represent the real PLC in cyber-security investigations.

6. SAMPLING RATE

The rate at which a PLC samples the controlled process is a very important parameter based on classical control theory, and in terms of determining if the PLC is fast enough to monitor the physical process in [14]. The sampling rate of the real and emulated PLC is measured using an iterative algorithm implemented in Python that writes a value to a register, waits for the PLC to execute its ladder logic, and records the time it takes the PLC to submit a new control action based on the new input. This method assumes that the python script is running at a much faster rate than the ladder logic program and that the computational time required to record the elapsed time is negligible relative to the sampling rate. These are valid assumptions for the current testing setup. The real and emulated PLCs are tested with sampling rates specified in the OpenPLC program of 1, 25, 50, 100, 250, and 500 ms, and the actual sampling rate is recorded using the Python sampling rate algorithm to check for any deviations from the ideal specified rate.

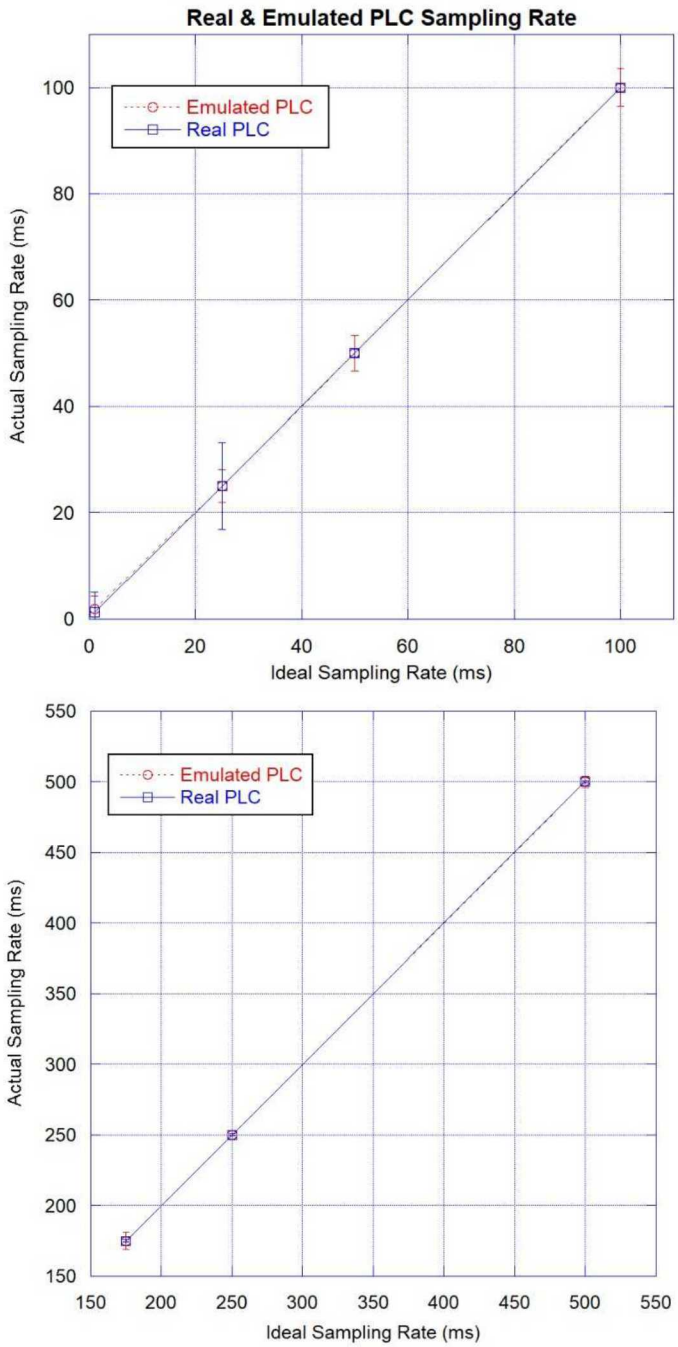

FIGURE 6: SAMPLING RATE FOR THE REAL AND EMULATED PLC (FIGURE 6A TOP AND FIGURE 6B BOTTOM)

Figures 6a and 6b present the results of the sampling rate analyses for the emulated and real PLCs, respectively. The sampling rate results for 1 –100 ms are shown in Fig. 6a, with the results for sampling rates of 175-500 ms shown in Fig. 6b. The actual and ideal sampling rates are nearly equivalent in the mean for both the emulated and real PLCs from 1-500 ms. Comparing the variance in the actual measured sampling rate, for sampling times above 50 ms the real PLC experienced less variance than the emulated PLC. The difference in the variance between the emulated and real PLCs decreased as the sampling rate increased, with the smallest variance observed at a sampling rate of 250ms.

These results for the sampling rate suggest that the emulated PLC matches the same average sampling rate as the real PLC. In order to achieve the most consistent results and enable the best comparison between controllers a sample rate of $\geq$ 250 ms should be used. In these analyses a sampling rate of 250 ms is used for the real and emulated PLC across all experiments.

7. ACTUATION RESPONSE TIME

An important physical signature to measure for quantifying the performance of a PLC is the actuation response time. The actuation response time is defined here as the time it takes for the PLC to execute a control action based on the inputs being sampled from the process being controlled. To determine if the observed actuation response deviates from an ideal response, internal logic in the Simulink simulation model is used to generate a 'true' response signal. This ideal 'true' signal is recorded internally by Matlab Simulink and is not dependent on the signals transmitted to and received from the PLC.

Thus, the ideal response represents a control action with zero time lag relative to the process simulation. The ideal response is compared against the generated responses of the real or emulated PLC to quantify any time lag in response time due to network communication or differences in PLC computational time. First, the state variable signal received by the PLCs is compared to the original signal generated by the Simulink model to verify that the real and emulated PLCs are receiving the same and correct input data. The results show that each the real and emulated PLC, for all tests, received the same square wave signal generated by the Simulink model.

Once the PLC receives the square wave input, the ladder logic implemented in OpenPLC determines if the current square wave value is above or below the setpoint of 0.5. If the square wave value is above 0.5, the command output is a value of one, and if it is below 0.5, the output is zero. Therefore, the command signal creates an output signal that is also a square wave over time. The controller response signal transmitted by the connected PLC is recorded within the running Simulink simulation and compared to the ideal baseline signal to identify the response time lag. This time lag in the response of the PLC is unavoidable due to the Python interface code being asynchronous and the sampling rate of the PLC, in this case the sampling rate is 250 ms.

The actuation response for the emulated and real PLCs are shown in Fig. 7. The controller response for both the emulated and real PLCs consistently lags the ideal response signal as expected. However, a non-physical response of the PLCs is observed due to the real-time sync Python interface code used. The sampling rate of the PLCs is 250ms, therefore it is impossible for the PLCs to respond to an external signal faster

than 250 ms. In this case response signals are observed for both the real and emulated PLC at 150ms and 200ms. As shown in the Section 6 the sampling rates for both PLCs are highly consistent and are near real-time with-in ~ ±10ms at a sampling rate of 250 ms. The Python interface code's real-time sync code is also consistent but deviates from real-time up to ±50ms (Fig. 3). When the real-time deviation accumulates such that the Simulink simulation is running slower than real-time, relative to the PLC, it is possible for the PLC to respond faster than physically possible. This non-physical result is only applicable to when time dependent processes are being simulated and communicating asynchronously with external devices.

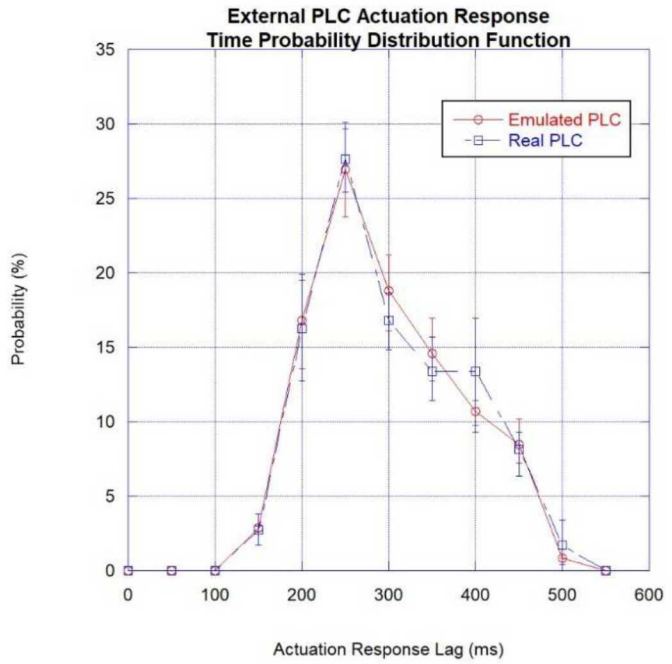

FIGURE 7: ACTUATION RESPONSE OF THE REAL AND EMULATED PLC

For the purposes of this report although a non-physical result was observed from the simulated processes perspective the relative actuation response between the real and emulated PLC are in good agreement. The emulated PLC had a slightly faster actuation response time with an average response time of 280.59ms, compared to an average response time of 284.06ms for the real PLC. The results suggest that each controller can be used interchangeably in cyber-physical emulation experiments without compromising the outcomes of the physical process being controlled.

As noted in Section 3.1, the inter-process Python code used in this analysis is asynchronous with the Simulink simulation. This is for better validation comparison between the real and emulated PLCs. The delay in the actuation response signal generated by the OpenPLC programming can be characteristic to the PLCs computing hardware and software. A synchronous inter-process communication method could be employed to eliminate the time lag between the observed and ideal actuation response signals by ensuring that the Simulink simulation will wait for the PLC control signal before continuing to the next timestep. The minimum size of the simulation timestep to be used, however, will still be limited by the network response time for the communication between the PLC and server PC running the process simulation model. A synchronous interface code also has the added advantage of being able to run faster than real-time.

8. CONCLUSION

This work developed a PLC emulation methodology and applies it to validating an emulation of a Raspberry 4 as the benchmarking hardware for a PLC running the open-source OpenPLC program. The chosen PLC emulation uses the VMware software to emulate the software of the physical device and employs the same OpenPLC control program software and network communication protocols as the real PLC. The validation testing analyses investigates the emulated and real PLCs linked to a process simulation model running within Matlab Simulink. A Simulink S-function and Python interface script were used to handle the inter-process communication between Simulink and the OpenPLC control program running on both the emulated and real PLCs. Characterizing the physical and digital signatures within the framework of the PLC emulation methodology enables the development of a PLC emulation for OT cybersecurity testing without the need for HITL. This methodology can be applied to more industry representative PLCs such as Allen Bradley, Rockwell, Siemens, ect. depending on project requirements.

The validation testing analyses have shown that the real and emulated PLCs, if properly configured, can have digital and physical signatures that approximate each other. Using the same software on both systems implies that any vulnerabilities in the OS, applications, and communication protocols used on the real system would be reflected in the emulated system and that the internal logic programming is highly consistent between the two.

The testing effort characterized the digital signatures of the network response and types and proportions of network traffic recorded between the PLC and server PC running the Simulink process simulation. The analyses of the network response showed that the emulated PLC has a slightly higher bandwidth capacity relative to the real PLC, however this was found to not have a significant impact on the relative communication speeds relative to the real PLC. The difference in bandwidth did influence the recorded network response and traffic, with the emulated PLC transmitting slightly more data packets than real PLC. Although the total number of packets was found to be different, the PCAP analyses showed that the emulated PLC generated the same type of network traffic as observed using the real PLC. When looking at the relative proportions of the data packets the real and emulated PLC did not have a difference of greater than 1% between TCP/IP and UDP/IP packet transmission.

Validation analyses investigating the sampling rate of OpenPLC Runtime found that on average the sampling rate set

in OpenPLC matched the ideal sampling rate. For sample rates < 250 ms however, the variance of the sampling rate is significant. The smallest difference in the sampling rates of the emulated and real PLC systems occurred for a sampling rate of 250 ms.

Finally, the physical signatures of the actuation response time and sampling rate are also characterized for the real PLC and compared against the developed PLC emulation. The validation testing analyses showed that the difference between actuation signal response times of the real and emulated PLC was found to be within 2%, Table 5. Nonphysical actuation response times were also observed due to the real-time implementation of the simulated process. This result suggests that future cyber-physical emulation experiments using asynchronous communication with time dependent processes should ensure that the simulated process runs in real-time ensuring that the cumulative error is never greater than one major time step. This complication can be avoided by using synchronous communication, but additional programming would be required to incorporate time dependent controllers, such as proportional-integral-derivative (PID) controllers.

PLC Type	Signature		Average (ms)	STD (ms)
Real	Network Response	QR	1.78	3.19
		RQ	21.87	9.04
	Network Traffic	TCP/IP	25666.60	1946.36
		UDP/IP	35.20	8.15
	Sampling Rate		249.99	0.70
	Actuation Response Time		284.06	79.03
Emulated	Network Response	QR	0.73	0.61
		RQ	23.56	8.75
	Network Traffic	TCP/IP	27455.40	1266.14
		UDP/IP	43.60	16.88
	Sampling Rate		250.00	3.35
	Actuation Response Time		280.59	77.10

TABLE 5: SUMMARY OF DIGITAL AND PHYSICAL SIGNATURE DATA

The validation testing analyses demonstrate PLC emulation with actuation responses and sampling rates which are nearly indistinguishable from the real PLC to the process simulation being controlled. From a cybersecurity perspective, this PLC emulation runs the same software, communicates using the same communication protocols, and generates the same types and proportions of network traffic data types as the real device. Employment of the developed PLC emulation methodology shows the importance of selecting the proper configuration parameters to ensure that the emulated PLC behaves comparable to the real system, confirming the need for detailed characterization and comparison of the physical and digital signatures as is performed in this work. Special attention should be given to the time dependencies of the PLC, such as the sampling rate or how the PLC interfaces with the simulated process, to ensure representative behavior of the process being modeled.

ACKNOWLEDGEMENTS

This research is being performed using funding received from the DOE Office of Nuclear Energy's Nuclear Energy University Program under Contract No. Nu-18-NM-UNM-050101-01. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. DOE's National Nuclear Security Administration under contract DE-NA-0003525. The views expressed in the article do not necessarily represent the views of the U.S. DOE or the United States Government.

REFERENCES

[1] Falliere, N., Murchu, L., and Chien, E., 2011, "W32 Stuxnet Dossier," Symantec.
[2] Dragos, 2017, "TRISIS-Hatman Malware Analysis of Safety Systems Targeted Malware."
[3] Dragos, 2017, "CrashOverride Analysis of the Threat to Electric Grid Operators."
[4] Nance, K., Hay, B., Dodge, R., Seazzu, A., and Burd, S., 2009, "Virtual Laboratory Environments: Methodologies for Educating Cybersecurity Researchers," Methodological Innovations Online.
[5] Camacho-Lopez, T. R., 2016, "SCEPTRE," Electricity Subsector Coordinating Council & Government Executives MeetingAlbuquerque, NM.
[6] Zhou, C., and Chen, H., 2009, "Development of a PLC Virtual Machine Orienting IEC 61131-3 Standard," 2009 International Conference on Measuring Technology and Mechatronics Automation, pp. 374-379.
[7] Thamrin, N., and Ismail, M., 2011, "Development of Virtual Machine for Programmable Logic Controller (PLC) by Using STEPS Programming Method," IEEE International Conference on System Engineering and Technology, IEEE.
[8] Gasser, T., 2013, "Virtual Machine and Code Generator for PLC-Systems."
[9] Alves, T. R., Das, R., and Morris, T., 2016, "Virtualization of Industrial Control System Testbeds for Cybersecurity," ICSS.
[10] Alves, T. R., Buratto, M., de Souza, F. M., and Rodrigues, T. V., 2014, "OpenPLC: An Open Source Alternative to Automation," Ieee Glob Humanit C, pp. 585-589.
[11] VMWare, 2019, "VMWare Workstation."
[12] MathWorks, 2019, "Matlab & Simulink R2019b," Natick, Massachusetts, United States.
[13] Combs, G., 2019, "Wirshark."
[14] Nise, N. S., 2015, Control Systems Engineering.