CyberRadar: A Regression Analysis Approach to the Identification of Cyber-Physical Mappings in Process Control Systems

Julian L. Rrushi
Università degli Studi di Milano and State University of New York at Binghamton
jrrushi@cs.binghamton.edu

Kyoung-Don Kang
Department of Computer Science
State University of New York at Binghamton
kang@cs.binghamton.edu

ABSTRACT

One of the attack requirements for maximizing physical damage to digitally controlled infrastructures is the identification of a mapping between program variables in a compromised control system and physical parameters related to physical processes or physical equipment. A cyber-physical mapping is quite critical from the offensive perspective as physical parameters are affected via modification of the associated program variables. The difficulty of such a reconnaissance challenge is acknowledged by control system security analysts as what they're presented with during experimental attacks is comprised of long series of random looking bytes or variable names. In this paper we provide a formal and thorough formulation of the cyber-physical mapping problem, propose a statistical approach to the identification of a cyber-physical mapping in large sets of scanning data, and further develop and demonstrate the proposed approach by applying it on a practical example, namely a network inertial attack on an electric motor.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Process Control Systems; G.3 [**Probability and Statistics**]: Correlation and Regression Analysis

General Terms

Security

Keywords

Process control systems, system reconnaissance, applied statistics

1. INTRODUCTION

The cyber attacks' potential for causing physical damage to digitally controlled physical infrastructures was demonstrated by the Idaho National Laboratories (INL) through an experimental cyber attack referred to as the Aurora generator test [10]. The experimentation carried out by INL researchers consisted of attacking the replica of a process control system [14] monitoring and controlling an electric power generator in a power plant, with the results being a violent physical destruction of the generator in question. We argue that an attack path such as that followed in the Aurora generator test on a process control system, and hence on physical equipment under its control, requires an additional reconnaissance step if compared to composite attacks on traditional general purpose computer systems, namely the identification of mappings between computer program variables and parameters of a physical process or equipment. The offensive value of such reconnaissance step is in fact acknowledged from technical discussions among security analysts researching on ethical hacking of process control systems.

We say that a computer program variable or computer memory location x_1 in a process control system is mapped to a parameter x_2 of a physical process or equipment if changing the value of x_1 causes a direct change in x_2. The aforementioned reconnaissance step is required both in the case an attacker acquires the ability to change program variables in a target process control system by gaining network access to them, and in the case in which an attacker gains system level access to the target process control system, and thus to program variables. With network access to a process control system we mean the ability of an attacker to send protocol frames to a target process control system.

Process control systems historically communicated over dedicated networks via proprietary communication protocols. Nevertheless, the process control industry has evolved into using Ethernet based TCP/IP networks and open industrial control protocols. As depicted in Figure 1, process control networks in general are interconnected with external networks, such as for example enterprise networks and Internet, via wired, wireless, or modem lines. Although control of network access into and from process control networks is enforced by firewalls, not necessarily attackers have been stopped there [12]. As we will see in the next sections, causing physical damage to physical equipment via transmission of malicious protocol frames to process control systems requires a precise specification of computer program variables that are mapped to critical parameters of this target physical equipment.

In this paper we provide a statistical technique that may be potentially employed by an attacker with network access

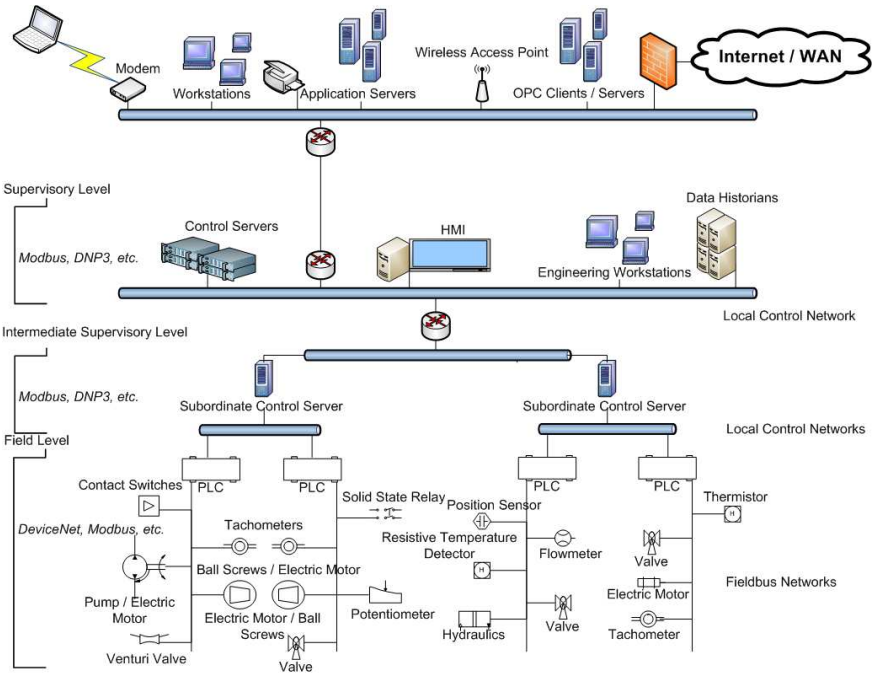

Figure 1: Setting resembling a process control network.

to a process control system to identify mappings between computer program variables and parameters of a physical process or equipment. Such a reconnaissance step may also result necessary from the perspective of an attacker who has acquired system access to a process control system. From that position what an attacker would normally do is replace or modify the programs that operate on variables, i.e. tags in Allen Bradley terminology, mapped to parameters of a physical process or equipment. Unless control system programmers have added comments in these programs to explain what tag is mapped to what parameter of a physical process or equipment, an attacker is required to reverse engineer these programs in order to derive such mappings. In this paper we focus on identification of these mappings from a network access perspective, and hence leave reverse engineering of control system programs as a future work.

2. TECHNICAL BACKGROUND

Without losing generality, in this paper we develop the proposed work as applied to a widely deployed type of process control system, namely a programmable logic controller (PLC) [6, 15]. The internal design of a typical PLC follows a von Neumann architecture. It is depicted in Figure 2 along with I/O modules that enable a PLC to receive input from, and send output to, physical processes usually via generation of electrical signals. In the actual context with input we mean measurements of physical process variables, such as for example the temperature in a closed container, while with output we mean generation of motion by the means of equipment that changes process variables, such as for example opening a valve to increase the water level in a tank. Input is received from sensors, i.e. devices that are embedded in physical infrastructures and that translate physical phenomena into electrical signals.

Output is sent by generating electrical signals in order to drive actuators, i.e. devices that transform electrical energy into mechanical energy usually in the form of motion. The input cards and the output cards of a PLC are connected through wiring to sensors and actuators, respectively. Some PLCs, in particular those produced in the recent years, may also communicate with sensors and actuators via a communication network referred to as fieldbus according to industrial communication protocols such as DeviceNet [16]. As shown in Figure 2 there are two kinds of sensors and actuators, namely logical and continuous, which are named after the kind of data values that they send to, and receive from, a PLC, respectively. Input and output data are referred to as logical if they consist of either `on` or `off` values. Input and output data are said to be continuous if they consist of values that are continuous in the mathematical sense.

Voltage values generated by continuous sensors are referred to as analog data, and thus are to be converted into a digital form before being processed by PLC programs. Voltage values are periodically sampled, and once acquired are processed via equations that convert them into numerical values [5]. Similarly, numerical data that PLC programs request to send to continuous actuators are to be converted beforehand into an analog form, i.e. voltage values, via digital-to-analog conversion equations. Engineers encode the logic of how a PLC should monitor and control a physical process into computer programs that are referred to as logic solving scan programs. The logic encoded in logic solving scan programs in general is devised by possibly following principles of control theory [4, 14], i.e. a discipline based on engineering and applied mathematics that deals with optimal control of the behavior of dynamical systems.

At run time logic solving scan programs, input scan programs, i.e. programs that read input values, output scan programs, i.e. programs that write output values, and any other PLC programs, such as for example fault handling

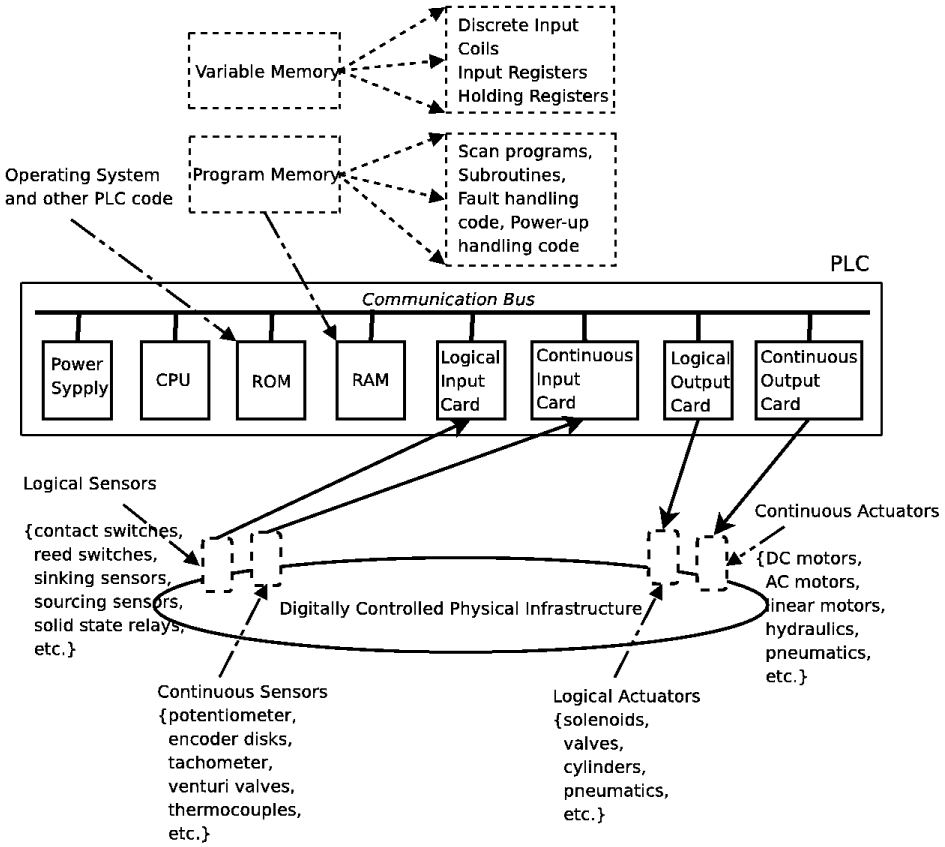

Figure 2: Organization of a typical programable logic controller.

programs, power-up handling programs, etc., are stored in a part of random access memory (RAM) referred to as program memory, as opposed to that part of RAM identified as variable memory, which is dedicated to storage of computation, input, and output data Figure 2. Logic solving scan programs are usually written in one or more programming languages of the IEC-61131-3 industrial standard [11], namely ladder diagram language, function block diagram language, structured text language, instruction list language, and sequential function chart language. The C and C++ languages are also commonly used to program PLCs. A PLC operates by periodically executing a defined sequence of scan programs known as a control loop. In common PLCs control loops are comprised of four stages and are executed a large number of times each second.

In the first stage a PLC executes code that checks the hardware and software of the PLC itself for faults. If no faults are detected, a PLC proceeds with the second stage in which it executes input scan programs. Such programs read logical and continuous input data from logical and continuous input cards, respectively, and copy these data in RAM variable memory. The memory locations in which logical and continuous input and output values are stored are designated preliminarily. A more detailed discussion of these designated locations is given later on in the next section. In the third stage a PLC executes logic solving scan programs. These programs process the input data that were previously received from sensors and that were stored in RAM variable memory, and hence produce output data that they store also in RAM variable memory in their corresponding designated locations. In the last stage a PLC executes output scan programs to propagate logical and continuous output values from RAM variable memory into logical and continuous actuators, respectively.

The principal objectives of attacks on process control systems include causing physical damage to physical equipment and sabotaging physical processes under their control. For instance, one such attack could perturb the operation parameters of an electric power generator in order to cause severe mechanical faults and hence destroy it, or manipulate the flow rate of water pumps in order to severely destabilize the parameters of a physical process in which preserving a defined water level within a container is critical to the stability and safe operation of a whole digitally controlled facility. Depending on equipment specifications and the physics behind physical processes, there is a variety of ways in which an attacker could manipulate process control systems to cause physical damage. A taxonomy of these attacks is provided by Larsen in [12]. An inertial attack consists of speeding up or slowing down at high rates heavy equipment.

An inertial attack has the potential of forcing heavy equipment to fail as in general such equipment is not tolerant to abrupt speed changes. An exclusion attack takes place when a process control system violates functional dependencies between various physical equipment, while a wear attack manipulates a process control system so as to con-

sume certain equipment components and hence reduce the life span of the equipment itself. Small variations of continuous process parameters such as electric current or fluid flow are recorded in a wave that is kept in other parts of a system. This circumstance is exploited by a resonance attack, which consists of continuously causing small variations of continuous process parameters for the purpose of increasing the size of this wave beyond acceptable limits. A surge attack is mounted by exceeding defined process parameter limits by amplitudes that go well beyond the maximal values that continuous control systems are capable of handling.
A latent abilities attack exploits latent features in off-the-shelf physical equipment. An example of a latent abilities attack provided by Larsen is to force a servomotor to run in the reverse direction, although such an action may not be part of its intended operation in a given physical infrastructure. Note that these attacks represent techniques for maximizing physical damage once an attacker has acquired network access to a process control network or system. A number of publicly known attack techniques such as those based on memory corruptions or firewall rule circumventions for system and network access, respectively, may be applied by attackers to gain access to process control systems and networks. These attack techniques come from knowledge of hacking general purpose computer systems and networks, and thus are out of the scope of this paper.

3. PROBLEM STATEMENT

If we think of offensive actions that lead to causation of physical damage to digitally controlled physical infrastructures as an ordered list of discovery and attack operations, our work addresses a discovery challenge which stands between attacks that enable an attacker to acquire access to a process control system or network, and attacks such as those proposed in [12] that are designed to maximize physical damage to physical processes and equipment. Once an attacker has gained network or system access to a PLC, he/she needs to have in hand what in our work we refer to as a cyber-physical mapping in order to be able to perform the attacks described in [12]. A cyber-physical mapping is a one-to-one correspondence between program variables that hold logical or continuous I/O values in RAM variable memory and physical process parameters or parameters that characterize the operation of physical equipment.
Consider for a moment a PLC that controls the flow of fluid in a pipe through a solenoid controlled valve. Let's suppose that this valve is a 2-way normally closed valve, i.e. the valve is closed when unenergized and opens when gets energized. This PLC will have a program variable stored in RAM variable memory, say *coil54*, which holds a logical value, namely *0* or *1*, and which is associated with the valve in question. If *coil54* is set to *1* while the valve is closed, then an output scan program in the PLC will energize the valve during the forth stage of a control loop and consequently will open it. Similarly, if the valve is open and *coil54* is set to *0*, then the output scan program will unenergize the valve and consequently will close it. Thus, an attacker may control the flow of fluid in the pipe by setting the *coil54* variable to *1* or *0* depending on whether he/she wants to allow or not allow the fluid to flow in the pipe, respectively.
In order to control the flow of fluid in the pipe an attacker needs to know that, if there are *65536* variables in a compromised PLC holding logical values, say (*coil0*, *coil1*, *coil2*, ..., *coil65535*), then it is precisely the *coil54* variable that is associated with this solenoid controlled valve. In this paper we provide a statistical approach to the identification of cyber-physical mappings from the perspective of an attacker who has acquired remote network access to a PLC, i.e. an attacker is able to transmit in a process control network protocol frames that are processed by a target PLC. Furthermore, the PLCs considered in our problem domain communicate over a process control network via a byte-oriented protocol. In this paper we develop our work as applied to PLCs whose communications are based on the Modbus industrial protocol.
Modbus is an application layer messaging protocol that enables control systems to communicate with each other in a client-server configuration within possibly different types of buses and networks [13]. The Modbus data model defines four categories of variables that hold logical or continuous I/O values. Discrete input variables hold read only single-bit data provided by logical sensors. Coil variables hold read and write single-bit data that are provided by, or are destined for, logical sensors and logical actuators, respectively. Input register variables hold read only 16-bit data that are provided by continuous sensors. Holding register variables hold read and write 16-bit data that are provided by, or are destined for, continuous sensors and continuous actuators, respectively.
Modbus defines its own addressing model in which each one of the variables of the four categories described previously is assigned an address from *0* to *65535*. Modus applications maintain a mapping between addresses of program variables as defined by the Modbus addressing model and addresses of locations in RAM variable memory where these variables are stored. These mappings are vendor specific. A Modbus protocol data unit (PDU), i.e. a protocol frame conveying information that a sending device wants a receiving device to process, is comprised of two fields, namely a function code field and a data field. Function codes indicate an operation on Modbus variables, such as write single register variable, read coil variable, etc. Function codes are encoded in one byte. Their valid values lie in the *1* to *255* range in decimal representation. The data field in a request PDU sent from a client to a server contains additional information such as Modbus addresses and the number of variables that are to be handled.
In some defined requests the function code field alone is sufficient for a server to perform a required task, therefore in these requests the data field is of zero length. The data field in a response PDU sent from a server to a client contains the data that the client had preliminarily requested via a request PDU. For example, if a master computer A controlled by human operators needs to acquire the values of four discrete input variables that are generated by logical sensors and that are stored contiguously in the RAM variable memory of a PLC B, then A sends to B a request PDU in which it specifies a function code of *0x02*, which according to the protocol specification stands for read discrete input variable, a starting address in the *0x0000* to *0xFFFF* range, which in this example will be the address of the first discrete input being asked to be read, and the number of discrete inputs that A is asking to read, namely four in this example.
In a regular transaction device B will derive from the function code the action to perform, namely read discrete input variables, will use the starting address and the number of

discrete inputs that A is asking to read for the purpose of determining the address of each one of these discrete input variables, will read their values from RAM variable memory, and will place them in the data field of a response PDU, which it then sends to device A. After acquiring network access to a PLC that communicates via the Modbus protocol, the ultimate goal of an attacker is to send one or more PDUs to a target PLC in order to manipulate logical or continuous output values. Referring to the example given earlier in this section, in order to open the solenoid controlled valve, and hence allow the fluid to flow in the pipe, an attacker would send to the target PLC a PDU with a function code of *0x05*, i.e. write single coil, and a data field comprised of the Modbus address of variable *coil54*, namely *53* in decimal or *0x35* in hexadecimal[1], and an output value of *0xFF00* [2], i.e. the value which indicates whether the coil should be set to *0* or *1*.

In order to perform attacks that maximize physical damage such as those described in [12] by sending malicious PDUs over the network to a target PLC, an attacker will try to identify the cyber-physical mapping that is in place at this target PLC. A series of Modbus scanner tools such as ModScan [3, 17] have been developed for security assessments, and some of them are publicly available. These tools enable an attacker to acquire the values of discrete input variables, coil variables, input register variables, and holding register variables, that are stored in the RAM variable memory of a target PLC. Furthermore, most of the aforementioned tools also enable an attacker to send attack frames that attempt to write to logical or continuous variables once he/she has identified the cyber-physical mapping. Modbus variables in a target PLC may be scanned several times, a process that normally produces a large set of data. The challenge consists in analyzing these data to identify cyber-physical mappings, and it is this challenge that we attack in this paper.

4. CYBER-PHYSICAL MAPPING DISCOVERY

4.1 General Approach

The values assumed by continuous program variables in PLCs at least in part depend on two factors that are unreachable via a network access to a target PLC. The former factor is control logic that engineers encode into logic solving scan programs, while the latter comprises physical parameters of internal architecture or configuration of physical equipment. Without information on these factors it appears as difficult to tell which continuous program variables are the ones of interest in large sets of data produced by scanning the RAM variable memory of target PLCs over a network. We have found persistent statistical relations between program variables, and hence indicate that these statistical relations are a potential mechanism that an attacker may leverage to identify program variables of interest, and thus derive their Modbus addresses. More precisely, if a linear association between program variables of interest is in place, we propose the degree of linear association as an instrument for identifying these variables of interest.

Our first movement consists of finding out whether a program variable of interest in a testing PLC is linearly correlated with other program variables that are acquirable via network scanning. The investigation proceeds with assessing whether the program variable of interest and candidate program variables, i.e. program variables that the program variable of interest may be potentially linearly correlated with, follow a Gaussian distribution [8]. If that is the case, then we apply the least squares regression [2] method to estimate the regression line that characterizes the linear relationship between these program variables. More precisely, we estimate the intercept and slope of this regression line, and hence build the regression line itself along with a scatter plot displaying values of the program variables under investigation, i.e. a set of data acquired from the RAM variable memory of a testing PLC. We then use the regression line and the scatter plot to characterize the degree of linear association between these program variables, namely estimate their linear regression coefficient.

When it comes to conducting an experimental attack, we employ a protocol scanner such as ModScan to acquire values of program variables from the RAM variable memory of a target PLC over a network. We then estimate the degrees of linear association between all program variables, which is, we estimate their linear regression coefficients. The identification of a program variable of interest in a target PLC takes place when defined program variables are found to have a regression coefficient equal to the regression coefficient between the program variable of interest and candidate program variables as preliminarily estimated on the testing PLC.

4.2 A Practical Example: Network Inertial Attack on an Electric Motor

Our experimentation objective is to carry out a network inertial attack on an alternating current (AC) induction motor [9] that is controlled by a PLC. For such a purpose we need to find out which program variable in the RAM variable memory of the target PLC is mapped to a physical parameter that is used to set the speed of this AC induction motor. The AC induction motor is comprised of a rotating center that is referred to as a rotor, and a fixed external part that is known as a stator. The stator of an AC induction motor has a number of windings that form multiple poles, i.e. pairs of windings, and may use permanent magnets to set up an opposing magnetic field. As AC polyphase current passes through windings on the stator, it creates a rotating magnetic field in the stator.

This rotating magnetic field induces currents in the rotor conductors. These currents in turn interact with the rotating magnetic field in the stator, with the result being the creation of a torque that rotates the rotor. Our target PLC uses a continuous sensor, namely a battery powered stroboscopic tachometer, to measure the rotational speed of the rotor. For currents to be induced in rotor conductors, the magnetic field in the stator must rotate relative to these ro-

[1]In the Modbus addressing model coil variables are addressed starting at zero. Thus, the address of the first coil variable is *0*, the address of the second coil variable is *1*, and so on. Note that we're assuming that the hypothetical variable *coil54*, which might as well have been called *v1* or *p286*, is the 54th coil variable in the RAM variable memory block dedicated to storage of coil variables.

[2]In Modbus the output value *0x0000* requests the coil to be *0* (off), while the output value *0xFF00* requests the coil to be *1* (on).

IR[16]	IR[53]	IR[18]	IR[69]	HR[19685]	HR[20008]	HR[18610]	HR[65530]
702.5	1884.0	1205.3	685.2	63.9	36.5	42.1	49.6
803.8	1977.0	903.9	679.2	55.4	39.2	41.6	52.4
901.8	1782.0	1306.9	722.4	55.8	38.3	45.2	62.3
904.1	1608.0	1004.8	763.2	67.3	45.8	48.6	60.1
1004.7	1884.0	1407.8	735.6	57.8	48.1	46.3	59.2
903.1	1977.0	1409.4	796.8	58.1	49.3	51.4	57.3
1004.9	1782.0	1408.3	868.8	61.8	51.5	57.4	57.9
809.6	1608.0	1598.3	817.2	48.9	58.3	53.1	61.4
1208.8	1782.0	1203.9	890.2	38.9	61.8	59.1	63.8
803.5	1608.0	957.5	945.6	48.6	47.5	63.8	65.0

Table 1: Excerpt from the data set acquired through ModScan from a target PLC.

tor conductors. If the rotor rotates with a rotational speed that is equal to the rotational speed of the magnetic field in the stator, then the magnetic field in the stator won't rotate relative to the rotor conductors, and hence no currents will be induced in the rotor conductors and no torque will be created.

Consequently the rotational speed of the rotor has to be different, i.e. usually smaller, than the rotational speed of the magnetic field in the stator. The former speed is referred to as actual rotational speed, while the latter speed is referred to as synchronous speed. In this paper we use ω and τ to denote the actual rotational speed of an AC induction motor and the synchronous speed of such motor, respectively. The difference between the synchronous speed and the actual rotational speed of an AC induction motor is referred to as magnetic slip, which in this paper we denote with δ. From the physics behind the operation of an AC induction motor we know that the actual rotational speed of an AC induction motor is controlled via the applied voltage frequency, which in this paper we denote with γ. Taking into account that the actual rotational speed of the AC induction motor as reported by the tachometer is a continuous input value, by referring to the Modbus specification we derive that the target PLC will use an input register variable to hold ω in RAM variable memory.

Furthermore, since the applied voltage frequency is a continuous output value, we derive that the PLC will use a holding register variable to hold γ. The equation that relates γ with τ is:

$$\gamma = \frac{p\tau}{120} \tag{1}$$

where p is the number of poles in the AC induction motor. Furthermore, when defining the magnetic slip we stated that:

$$\tau = \omega + \delta \tag{2}$$

If load is a torque that opposes to the rotation of the rotor, then with ν we denote the nameplate speed at full load [9]. The magnetic slip is related to the nameplate speed at full load according to the following equation:

p	ω	τ	l	ν	δ	γ
4	1246.3	1884.0	0.9	1175.4	637.7	62.8
4	1255.6	1977.0	0.9	1175.4	721.4	65.9
4	1236.1	1782.0	0.9	1175.4	545.9	59.4
4	1218.7	1608.0	0.9	1175.4	389.3	53.6
4	1205.8	1479.0	0.9	1175.4	273.2	49.3
4	1178.8	1209.0	0.9	1175.4	30.2	40.3
4	1203.7	1458.0	0.9	1175.4	254.3	48.6
4	1222.6	1647.0	0.9	1175.4	424.4	54.9
4	1197.7	1398.0	0.9	1175.4	200.3	46.6
4	1186.0	1281.0	0.9	1175.4	95.0	42.7

Table 2: A sample of values of physical parameters that characterize the operation of an AC induction motor studied in laboratory settings.

$$\delta = l\,(\tau - \nu) \tag{3}$$

where l denotes the load. Plugging of equation (3) into equation (2) yields:

$$\tau = \omega + l\,(\tau - \nu) \tag{4}$$

Plugging equation (4) into equation (1) yields:

$$\gamma = \frac{pl\,(\tau - \nu)}{120} + \frac{p}{120}\omega \tag{5}$$

In order to manipulate the speed of the AC induction motor we need to know the Modbus address of the holding register variable that holds γ. As we know that ω is held by an input register variable and γ lies in a holding register variable, we use ModScan to acquire the values of all input register variables and holding register variables. Table 1 provides an excerpt from the full scan of the target PLC. For the sake of clarity, IR and HR stand for input register variable and holding register variable, respectively. The Modbus address of each scanned variable is given in square brackets. By having used in equation (5) each scanned input register variable

as a possible ω, and each scanned holding register variable as a possible γ, we would have reached a match between the holding register variable that is mapped to γ and the input register variable that is mapped to ω. Nevertheless, as we're operating via a network we have no visibility into the physical parameters p, δ, τ, ν, and l. Consequently equation (5) is not directly applicable to the identification of a holding register variable that is mapped to γ by trying to correlate its values with values of ω.
We now describe the proposed statistical approach as applied to identification of the Modbus address of a holding register variable that is mapped to the applied voltage frequency in a target PLC, which in turn controls an AC induction motor. In laboratory settings we study an AC induction motor characterized by values of physical parameters p, δ, τ, ν, and l, chosen randomly among those available. Our thesis is that, although the internal architecture and configuration of a testing AC induction motor may be totally different than the internal architecture and configuration of an AC induction motor controlled by the target PLC, some program variables that are mapped to physical parameters such as applied voltage frequency and actual rotational speed exhibit hidden but calculable statistical relations. Furthermore, these statistical relations are persistent among electric motors, even though their internal architectures and configurations may differ to large degrees.

We pilot our analysis towards identification of statistical relations between the holding register variable that is mapped to γ and the input register variable that is mapped to ω. More precisely, we are interested in their degree of linear association as measured by a linear correlation coefficient denoted with r. In laboratory settings we use ModScan to acquire values of γ and ω over a defined period of time from a testing PLC that controls an AC induction motor. Given that we have installed and configured the testing PLC along with the AC induction motor, we know the Modbus address of the holding register variable and the Modbus address of the input register variable that are mapped to γ and ω, respectively. Scanned values of γ and ω along with values of p, δ, τ, ν, and l of the AC induction motor in laboratory settings are given in Table 2. Let's analyze the values of γ and ω in Table 2 through linear regression [1, 7] by first checking whether these values follow a Gaussian distribution [8].
Let $\bar{\gamma}$ and $\bar{\omega}$ denote the mean average of γ and the mean average of ω, respectively. We have:

$$\bar{\gamma} = \left(\frac{\sum_{i=1}^{10} \gamma}{10} \right) = 52.41 \tag{6}$$

$$\bar{\omega} = \left(\frac{\sum_{i=1}^{10} \omega}{10} \right) = 1215.13 \tag{7}$$

The normal density curves for γ and ω are depicted in Figure 3. In our analysis we consider γ as a dependent variable, and ω as an independent variable. Note that we are not assuming causality between these two program variables in this order. Let a and b denote the intercept and slope, respectively, in the linear relation between γ and ω. Then we have:

$$\gamma = a + b\ \omega \tag{8}$$

where a and b are estimated via least squares regression [2] as shown in the following equations:

$$b = \left(\frac{\sum_{i=1}^{10} \left((\omega_i - \bar{\omega}) (\gamma_i - \bar{\gamma}) \right)}{\sum_{i=1}^{10} (\omega_i - \bar{\omega})^2} \right) = 0.33 \tag{9}$$

$$\bar{\gamma} = a + b\ \bar{\omega} \Rightarrow a = \bar{\gamma} - b\ \bar{\omega} = -352.6 \tag{10}$$

Thus, the linear relation between γ and ω is represented by:

$$\gamma = (-352.6) + 0.33\ \omega \tag{11}$$

The scatter plot and linear regression line for this relation are depicted in Figure 4. Let $\hat{\gamma}$ denote the values of γ estimated by the linear regression line of Figure 4. The correlation coefficient r measuring the degree of linear association between γ and ω is estimated by the following equation:

$$r = \left(\sqrt{\frac{\sum_{i=1}^{10} (\hat{\gamma}_i - \bar{\gamma})^2}{\sum_{i=1}^{10} (\gamma_i - \bar{\gamma})^2}} \right) = 1 \tag{12}$$

Armed with a quantification of a statistical relation between γ and ω we return to our problem in Table 1, namely how to identify the Modbus address of a holding register variable that is mapped to physical parameter γ in the target PLC. The solution to our problem consists of measuring the degree of linear association between each holding register variable and each input register variable in Table 1. The holding register variable mapped to γ and the input register variable mapped to ω will be those whose correlation coefficient r is 1. Table 3 provides the correlation coefficients of holding register variables and input register variables measured upon data that were gathered from scanning the target PLC. From these estimations we can see that the Modbus address of the holding register variable that is mapped to γ is *18610*. Further, the Modbus address of the input register variable that is mapped to ω is *69*.[3]

5. CONCLUSION AND FUTURE WORK

The degree of linear association among continuous program variables that follow a Gaussian distribution proves to be a viable mechanism for identifying a cyber-physical mapping. In this paper we provide an applied regression analysis approach for revealing a cyber-physical mapping in large sets

[3]Their correlation coefficient is slightly less than 1, namely 0.99, due to a series of roundings of numbers that were performed during the mathematical estimations.

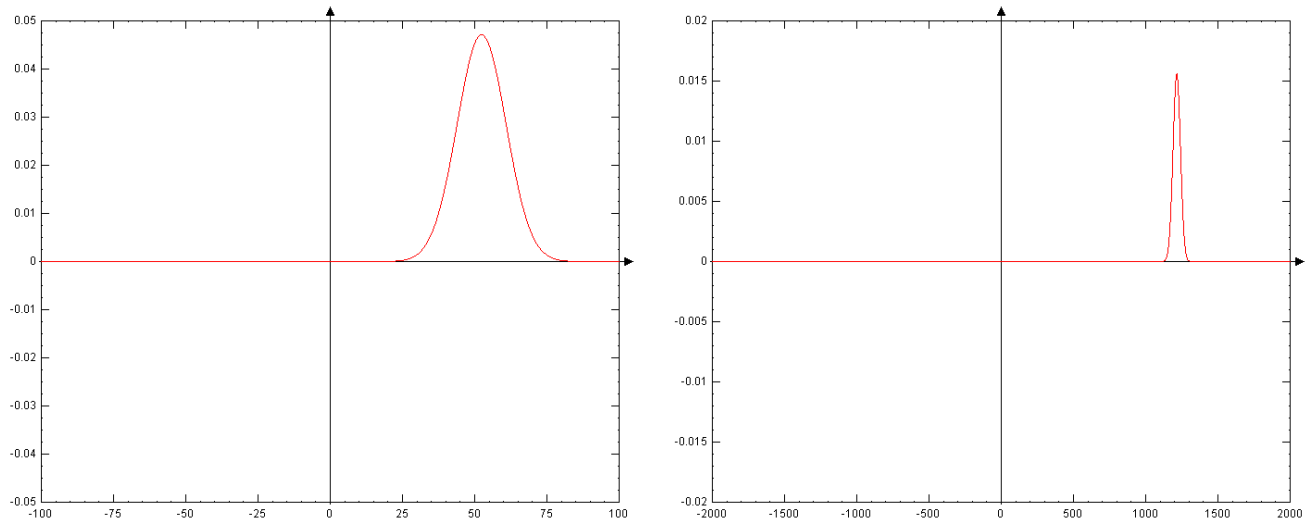

Figure 3: Normal density curves for applied voltage frequency γ and actual motor rotational speed ω, left and right respectively, in which the standard deviation of γ is 8.46751 and the standard deviation of ω is 25.40254

	$IR[16]$	$IR[53]$	$IR[18]$	$IR[69]$
$HR[19685]$	-0.41	0.16	-0.05	-0.54
$HR[20008]$	0.64	-0.36	0.43	0.71
$HR[18610]$	0.4	-0.54	0.05	0.99
$HR[65530]$	0.49	-0.66	0.1	0.72

Table 3: Measurements of the degree of linear association between values of holding register variables and values of input register variables that were scanned from the memory of a target PLC.

of scanning data. We also demonstrate the efficiency of the proposed approach by applying it in the form of a network inertial attack on an AC induction motor. A cyber-physical mapping challenge holds also for system level access in addition to network level access. In this regard we are investigating on the potential of the proposed approach along with symbolic execution techniques for identifying cyber-physical mappings in PLC programs written in one of the PLC programming languages of the IEC 61131-3 standard. Furthermore, given the potential of linear correlation coefficients for being used as a reconnaissance mechanism in malicious cyber attacks on digitally controlled physical infrastructures, we are also investigating on deception algorithms to leverage malicious use of regression analysis into a malicious activity sensing mechanism.

Acknowledgments

This work was supported in part by NSF grant CNS-0614771. Julian L. Rrushi was supported in part on a doctoral scholarship from Università degli Studi di Milano, and in part on a research scholarship from (ISC)^2. Opinions, findings, and conclusions expressed in this paper are those of the authors only.

6. REFERENCES

[1] R. Berk. *Regression Analysis: A Constructive Critique*. Sage Publications, 2004.

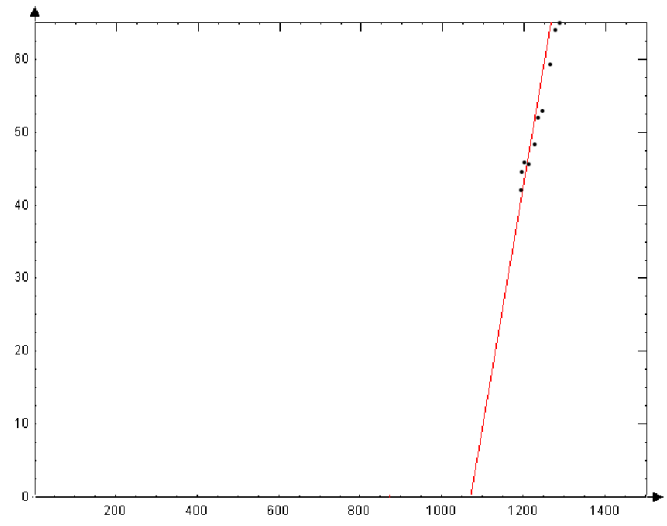

Figure 4: Scatter plot and linear regression line for the statistical relation between γ and ω.

[2] A. Bjorck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[3] M. Bristow. ModScan: A SCADA MODBUS Network Scanner. *DEFCON 16*.

[4] W. Brogan. *Modern Control Theory*. Prentice Hall, 1990.

[5] M. Demler. *High-Speed Analog-to-Digital Conversion*. Academic Press, 1991.

[6] K. Erickson. *Programmable Logic Controllers: An Emphasis on Design and Application*. Dogwood Valley Press, 2005.

[7] D. Freedman. *Statistical Models: Theory and Practice*. Cambridge University Press, 2005.

[8] R. Herrnstein and C. Murray. *The Bell Curve: Intelligence and Class Structure in American Life*.

Free Press, 1994.
[9] A. Hughes. *Electric Motors & Drives*. Newnes, 2005.
[10] IBM Internet Security Systems. X-Force Threat Insight Monthly. *IBM Website*.
[11] International Electrotechnical Commission. IEC 61131, Programmable controllers - Part 3: Programming languages. *IEC Website*.
[12] J. Larsen. Breakage. *Blackhat Federal 2008*.
[13] Modbus Organization. Modbus Application Protocol Specification. June 2004.
[14] N. Nise. *Control Systems Engineering*. Wiley, 2007.
[15] F. Petruzella. *Programmable Logic Controllers*. Career Education, 2004.
[16] Rockwell Automation. DeviceNet Adaptation of CIP.
[17] WinTECH Software. ModScan Tool. *WinTECH website*.