Remotely Inferring Device Manipulation of Industrial Control Systems via Network Behavior

Georgios Lontorfos
Johns Hopkins University
Information Security Institute
glontor1@jhu.edu

Kevin D. Fairbanks*
United States Naval Academy
fairbank@usna.edu

Lanier Watkins*
Johns Hopkins University
Information Security Institute
lanier.watkins@jhuapl.edu

William H. Robinson
Vanderbilt University
william.h.robinson
@vanderbilt.edu

***Abstract*— This paper presents preliminary findings on a novel method to remotely fingerprint a network of Cyber Physical Systems and demonstrates the ability to remotely infer the functionality of an Industrial Control System device. A monitoring node measures the target device's response to network requests and statistically analyzes the collected data to build and classify a profile of the device's functionality via machine learning. As ICSs are used to control critical infrastructure processes such as power generation and distribution, it is vital to develop methods to detect tampering. A system employing our measurement technique could discover if an insider has made unauthorized changes to a device's logic. Our architecture also has advantages because the monitoring node is separate from the measured device. Our results indicate the ability to accurately infer (i.e., using a tunable threshold value) discrete ranges of task cycle periods (i.e., CPU loads) that could correspond to different functions**.

***Index Terms*—cyber-physical systems, tampering, security, network traffic analysis, processor workload, machine learning, device fingerprinting**

I. INTRODUCTION

Industrial control systems (ICSs) are pervasive throughout commercial and national critical infrastructure. These systems can be used to monitor, control, and automate various manufacturing and distribution processes. With the advent of cyber-attacks that have real-world effects, such as Stuxnet and an attack on a German steel mill's control system [7], it has become evident that ensuring the security of these control systems is of great importance. As noted in [6], an ICS can vary based upon the implementation. Distributed Control Systems (DCS) and Programmable Logic Controller (PLC) operated systems are typically deployed on a local network within a worksite. Supervisory control and data acquisition (SCADA) systems are generally deployed over great distances and can tolerate the data loss and delays associated with various communications methods. While the scope and size of an ICS may vary, these systems share key components.

PLCs can be viewed as the workhorses of an ICS. While they were originally designed to perform functions such as opening and closing switches, relays, and valves, modern devices can perform more complex operations. Remote Terminal Units (RTUs) allow remote data acquisition and control functions. These devices can feature wired or wireless network interfaces and can be deployed in places that are difficult to access. In some cases, a PLC with network access can function as an RTU and is referred to as such; however, an RTU normally has more advanced functionality than a PLC and can control multiple processes. The distinction between these devices is further blurred by integrated products that combine both, such as the Broderson RTU32S or RTU32 [8]. A Master Terminal Unit (MTU) is typically used to control RTUs and PLCs in a SCADA system.

Our contribution focuses on inferring the processor load on a RTU device through network measurements. In our implementation, we solicit network traffic (i.e., via ICMP ping), capture and analyze ICMP replies from an ABB RTU560, and we demonstrate the use of machine learning to infer the task cycle period (i.e., frequency of scanning of inputs, executing PLC code, and updating outputs). This approach takes advantage of the fact that packet response times will vary based on a RTU's task cycle period. In particular, the CPU load required to perform the task or set of tasks is inferred. While this profile does not reveal the exact functionality of the device, it can be used to detect unsanctioned changes to the device's logic.

The remote inference technique has several benefits. As it does not execute code on the ICS device, the approach does not depend on the device for data storage or direct processing power. Instead, a separate monitoring node is used to measure and process the data for several ICS devices. These data are then combined into a fingerprint of the ICS network. The technique can also be used to identify highly loaded devices that may be critical to the function of a system. We believe that applying this technique to a network of ICS devices is feasible because ICS devices have a focused purpose when compared to general-purpose computers. Since there is no end user performing a variety of actions (e.g., streaming web content, playing a game, using a word processor), the profile of an ICS device is likely to remain fairly stable until it is programmed to perform a different task. Because of this constrained behavior, our approach can be used to develop a fingerprint for the operational task cycle periods of an ICS network, through the measurement and analysis of the network behavior by its constituent devices. It is then possible to determine when this fingerprint has been altered. It is our intention to expand upon

*corresponding authors

Authorized licensed use limited to: IEEE Xplore. Downloaded on October 19,2024 at 14:09:46 UTC from IEEE Xplore. Restrictions apply.

978-1-4673-6773-8/15/$31.00 ©2015 IEEE

this preliminary research to create a remote intrusion detection system that produces forensic artifacts.

II. Related Work

Our approach to monitoring industrial control system (ICS) devices for anomalies is similar to a technique used by Watkins et al. [1][2], where they ping general-purpose nodes and analyze the results for deviations in response time due to contention for shared resources inside of the node. They explain in detail how this approach can be implemented for different applications, such as remote detection of bitcoin mining bots and passive resource discovery. They also design and execute experiments to validate that a rise in the average memory access in a Linux-based system due to high utilization in the node is the culprit for the delays observed in the ICMP replies. Our work differs in that we focus on VxWorks-based ICS devices (i.e., ABB RTU560) for various industrial applications. Even though Linux and VxWorks are different types of operating systems, they both use TCP/IP networking, and thus both require access to the CPU to create ICMP responses. So when ICMP requests come into either node with CPU utilization, the responses from both nodes will experience a delay of some magnitude due to contention for the CPU.

Some network intrusion detection methods are similar in that they analyze network traffic from the ICS devices. These methods are heavily focused on examining the network traffic for anomalies in the way protocols are used, or looking for known signatures of protocol misuse. Some techniques attempt to build protection against certain types of attacks [9], such as:

- *Unauthenticated Command Execution*: A lack of authentication between the MTU and RTU can be used by attackers to forge packets that can be directly sent to multiple RTUs.
- *DOS*: Using similar logic, an attacker can create fake packets, always impersonating the MTU to strain the resources of the RTUs.
- *Replay-Attacks*: As most systems do not implement anti-replaying defenses, adversaries are able to re-use legitimate Modbus/DNP packets.
- *Man-in-the-Middle attacks*: An adversary gains access to the production network, intercepts, and modifies the legal packets sent by the MTU.

Some of the most popular approaches build on top of existing systems and decompose the system in terms of its basic components, like PLCs, RTUs, and control servers, as well as information flows [11]. They also: (1) keep track of critical states and known vulnerabilities related to those components, (2) attempt to analyze the command response traffic that is generated by the system, (3) trace certain behavioral profiles, (4) feed a virtual system with real-time recorded commands, and (5) perform checks on whether there are alerts raised [12]. Other proposed IDS solutions focus on introducing fast and lightweight methods utilizing algorithms that keep the rate of false positives low [10].

Our proposed scanning method differs in that it uses probing requests and performs the assessment locally on the monitoring node. This approach does not depend on unsolicited network traffic that could be generated by a particular application. For the solicited network traffic, the technique does not depend on the type, content, or length of the response packets that will be received from the RTU devices. It evaluates the state of the target devices based on their response timings.

III. RTU Architecture and VxWorks Scheduling

The ABB RTU560 contains: (1) a webserver, (2) a software enabled PLC function, (3) Ethernet 10/100 BaseT LAN interface and TCP/IP protocol stack, (4) a 32-bit 133 MHz ELAN520 586 main processing unit (MPU) and memory system, (5) a VxWorks operating system (OS) Version 6.9.4.1, and (6) other modules and protocol stacks. Architecturally, the RTU is similar to a general-purpose node; however, its VxWorks OS is a real-time operating system (RTOS) specifically designed for use in embedded applications. In this paper, we focus on the fact that the MPU is a shared resource in the RTU, and thus some of the basic theories developed by Watkins et al. [1][2] are applicable.

Specifically, VxWorks uses a preemptive round-robin scheduler [3]; however, for interrupts, VxWorks runs interrupt service routines (ISR) outside of any other task's context. Thus, there is no context switching needed to handle interrupts [4]. Since the CPU is involved in creating the ICMP response packets that result from the ICMP request interrupts, the existing CPU utilization in the RTU induces delays into the resulting ICMP reply network traffic. Certain levels of CPU utilization introduce distinctive patterns of delays into the ICMP network traffic. In Figure 1, we see that the RTU560 has only one CPU that must be shared by all of the processes involved in allowing the RTU to function. When the PLC function is operational, significant CPU utilization occurs depending on the PLC code and the task cycle frequency. In this paper, we fix the PLC code, and we investigate the ability to detect varying task cycle frequency. We have observed that the task cycle frequency induces delays into the ICMP responses due to: (1) the contention for the CPU by the PLC operations and (2) the creation of ICMP response packets.

Authorized licensed use limited to: IEEE Xplore. Downloaded on October 19,2024 at 14:09:46 UTC from IEEE Xplore. Restrictions apply.

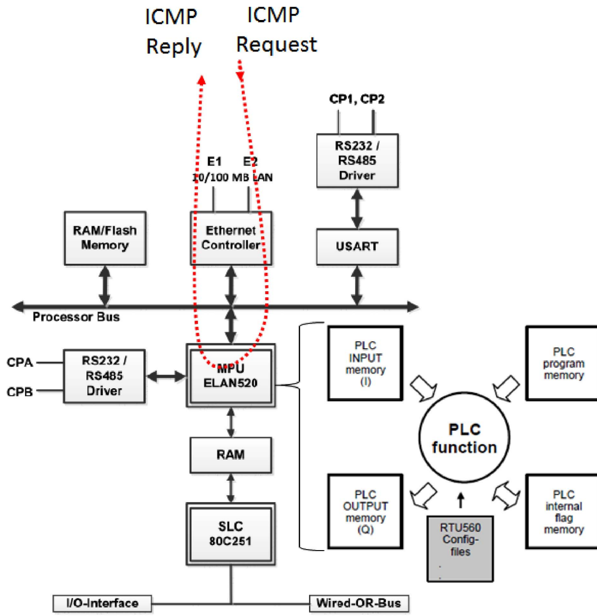

Figure 1: ABB RTU560 architecture with identified PLC function [14][15]

IV. RTU Operations and Terminology

Operationally, the ABB RTU560 is used to communicate directly with industrial devices via its PLCs or communicate with other RTUs to collect or report data. In this paper, we are concerned with the use of the RTU for communicating directly with industrial devices. We are specifically concerned with two parameters: (1) **task cycle period** – the frequency for running the task cycles, and (2) **task cycle duration** – the time it takes to actually run the task cycles. The task cycle duration should always be smaller than the pre-defined task cycle period, otherwise the task cycle cannot complete. A watchdog timer can be set to throw an error if this violation occurs. The watchdog timer is set near the task cycle period, and if the task cycle duration exceeds this time, then an alert is generated. During the task cycle, the following actions must occur: (1) scanning of inputs, (2) executing of PLC code, and (3) updating of outputs. All of these activities affect the CPU load of the RTU. Generally, the CPU load of the RTU is a function of task cycle period and task cycle duration. Practically, the task cycle duration is dominated by the time to execute PLC code. In our experiments (Section VI), we use the same unmodified PLC code, vary the task cycle period, and then demonstrate the ability to infer the task cycle period. In future work, we plan to investigate the effects of varying the PLC code.

V. Inference Algorithm for the Task Cycle Period

We propose a **Task Cycle Period Inference Algorithm** (Algorithm 1) capable of inferring the task cycle period for each RTU connected to a switch in the network. The algorithm requires a monitor node to be connected directly to each switch. The monitor node's job is to take measurements from the RTUs connected to each switch by sending probing requests (i.e., ICMP pings). Then, it will infer their CPU utilization and thus their task cycle period by feeding features from the captured ICMP replies into a previously trained machine learning method.

Variables and functions from the algorithm are provided in Table I. Specifically, the algorithm works by calculating the ICMP response times from the ICMP request timestamps and the associated ICMP reply timestamps, and then stores those results in a matrix denoted as PT_{ni}. After calculating all of the ICMP responses, it will extract feature vectors by taking the mean, standard deviation, variance, harmonic mean, and geometric mean of each row in the matrix PT_{ni}, and the result is stored in FV_{nj}. Next, the algorithm feeds each row of FV_{nj} into a previously trained WEKA Random Forest Trees [5] instance. Using the features per IP address, the WEKA Random Forest Trees instance decides the task cycle period where each set of features belong, and passes this information along in a row matrix RNn as shown in the algorithm below. A collection of these row matrices per switch in the network yields an Operational Task Cycle Fingerprint (OTCF) for an ICS network.

TABLE I. Description of Task Cycle Period Inference Algorithm's variables and functions

Variables and Functions	Description
S_n	Row matrix of available nodes (IP addresses) per switch
n	Node index
i	Packet number
j	Feature number
PT_{NI}	Stores ICMP response values
FV_{ni}	Stores features extracted from ICMP responses
Avg_RT_{ni}	Row matrix of average Response Times
WEKA-RFT()	WEKA machine learning instance
ping	Unix ping command
mean(), stdev(), var(), har_mean(), geo_mean()	Average, standard deviation, variance, harmonic mean, and geometric mean of the response time values per row
$fv1_n$, $fv2_n$, $fv3_n$, $fv4_n$ $fv5_n$	Temporarily vectors used to hold features
ICMP_resp_time()	Calculates ICMP response time from packets
1Dto2D()	Converts 1D feature vectors into a 2D matrix

VI. Experimental Evaluation

Our goal is to demonstrate our method in a realistic environment; therefore, we emulate the main components in an industrial control system (ICS) field site. These main components are essentially the RTU or the PLC (our ABB RTU560 contains both) and the physical device or devices it is controlling. Per [15], PLCs are used in both Supervisory control and data acquisition (SCADA) and Distributed control systems (DCS) implementations as a local controller within a

Authorized licensed use limited to: IEEE Xplore. Downloaded on October 19,2024 at 14:09:46 UTC from IEEE Xplore. Restrictions apply.

supervisory control scheme. DCS are used primarily for the control of ICS in the same region for industries such as oil refineries, water and wastewater treatment, electric power generation plants, chemical manufacturing plants, automotive production, and pharmaceutical processing facilities [15]. Similarly, SCADA is used in disparate regions for centralized data collection and control for industries such as water distribution and wastewater collection systems, oil and natural gas pipelines, electrical utility transmission and distribution systems, and rail and other public transportation systems [15]. Networks for both of these are illustrated in Figures 2 and 3. Note that in 2015, the modems that appear in these figures would be replaced with TCP/IP networks. Also note that for both figures, the network that contains the PLC is a local area network (LAN) and not a wide area network (WAN); therefore, there is no issue with large delays interfering with the results of our delay-sensitive method.

Algorithm 1

Input: S_n, A matrix of IP addresses of devices under every switch.
Output: RN[n], Matrix of inferred task cycle periods per IP address

01: i=0
02: j=0
03: **for** each compute node in S_n **do**
04: **ping** *compute node* 100 times
05: **while** i< 100 do
06: PT_{ni} = ICMP_resp_time(i)
07: i++
08: **end while**
09:
10: **end for**
11: **for** each *compute node* in PT_{ni} **do**
12: $fv1_n$ = mean(PT_{ni})
13: $fv2_n$ = stdev(PT_{ni})
14: $fv3_n$ = var(PT_{ni})
15: $fv4_n$ = har_mean(PT_{ni})
16: $fv5_n$ = geo_mean(PT_{ni})
17: **end for**
18: FV_{nj} = 1Dto2D($fv1_n$,$fv2_n$,$fv3_n$,$fv4_n$,$fv5_n$)
19: RN_n = WEKA_RFT(FV_{nj})
20: **end for**
21: **return** RN_n

A. Experimental Setup

Our test-bed hardware and software consists of: (1) an ABB RTU560 running on firmware version 11.2, on VxWorks 6.9.4.1., (2) four Intel Core i5-based laptops, one used as a monitor node and the other three used to place a load on the switches, (3) and two switches. In addition, (4) Wireshark was used to capture ICMP replies, (5) WEKA version 3.7.11 Random Forest Trees was trained and used to categorize vectors of ICMP replies, and (6) customized configurations were used to place workloads (~15%, ~30%, ~50%, or ~70%) on the RTU's CPU.

The ABB RTU560 has a built-in software PLC. Also, we use fixed PLC code supplied by the vendor that simulates PLC activity by having it scan its inputs at a specific task cycle frequency and perform calculations (which induces a CPU load) similar to PLC stimulus from a real physical device (i.e., pump, switch, generator). This approach allows us to emulate a real field site where PLCs on a LAN are connected to physical devices to control industrial processes. We also lightly load the LAN with network traffic to emulate a realistic environment. The intent is not to simulate traffic patterns; instead, the intent is to emulate a lightly loaded networked ICS network.

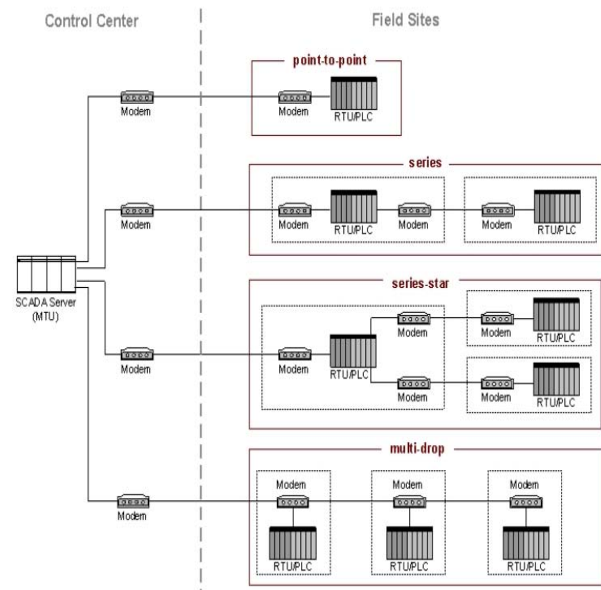

Figure 2: Basic SCADA communication topologies [16]

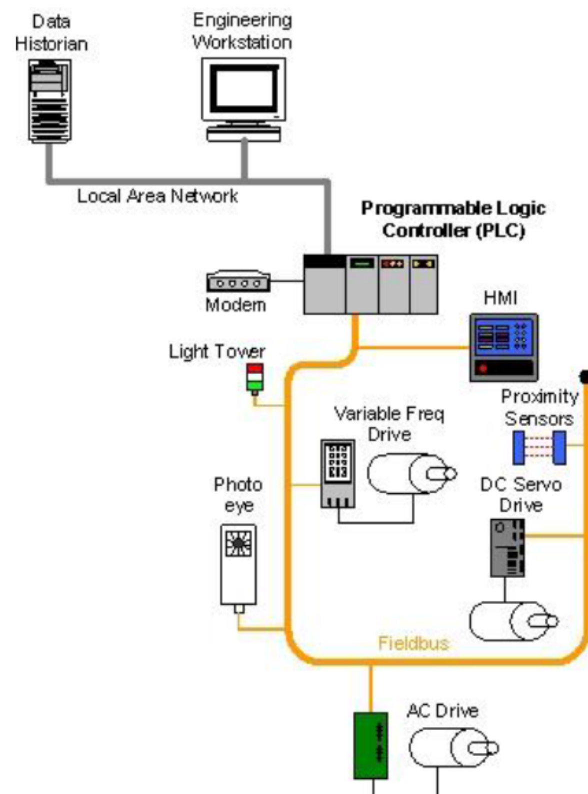

Figure 3: DCS implementation example [16]

B. *Experimental Procedure*

The monitor node was used to launch the applications and scripts that push the configuration to the RTU. We worked with

Authorized licensed use limited to: IEEE Xplore. Downloaded on October 19,2024 at 14:09:46 UTC from IEEE Xplore. Restrictions apply.

the RTU vendor to develop four configurations files that contain PLC code to induce a workload on the RTU even though the RTU is not connected to an industrial device. We achieved ~15%, ~30%, ~50% and ~70% CPU utilization by fixing the PLC code and modifying the task cycle period to 100 ms, 40 ms, 30 ms, and 20 ms respectively. We verified the CPU utilization by using the RTU's web interface statistics page.

Our experimental procedure is separated into three parts: (1) No-hop statistically analyzed experiments, (2) multi-hop statistically analyzed experiments, and (3) no-hop machine learning-based experiments.

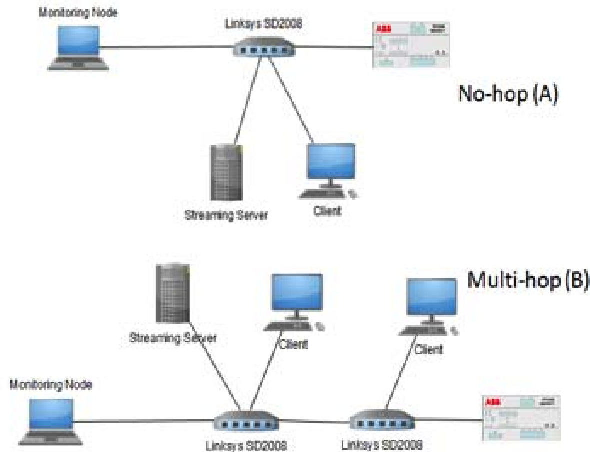

Figure 4: (a) No-hop experimental setup (b) Multi-hop experimental setup

C. *No-hop Statistically Analyzed Experiments*

The no-hop statistically analyzed experiments were done to determine if a correlation exists between network traffic and RTU CPU load (i.e., task cycle period). Our approach was as follows. We implemented an Ethernet-based local area network (LAN) consisting of three laptops, and an ABB RTU560 connected to one switch. Then, a video file was streamed from one laptop to another to place a load the LAN (See Figure 4a). Then, we pinged from the monitor node to the RTU 10,000 times at different CPU loads (~15%, ~30%, ~50%, and ~70%), then captured and stored the ICMP replies into separate pcap files. We exported the ICMP response times from each pcap file into Microsoft Excel and took the average. The results are illustrated in Table II.

D. *Multi-hop Statistical Analyzed Experiments*

The multi-hop statistically analyzed experiments were done to determine if the correlation mentioned in Section 6.2.1 would be masked by the addition of another hop in the network. Our approach is the same as in the no-hop experiments, except an additional switch and an additional laptop is used to add one hop to the LAN and to stream a video file across this additional hop (See Figure 4b). The results of this experiment are displayed in Table III.

E. *No-hop Machine Learning-based Experiments*

The no-hop machine learning-based experiments were done to determine if a machine learning method could be trained to correlate network traffic and RTU CPU load (i.e., task cycle period). Our approach is the same as in the no-hop experiments except instead of averaging the response times exported from each pcap, we take the average, standard deviation, variance, harmonic mean, and geometric mean for every set of 99 ICMP response values from the OSI Layer 3 (i.e., ICMP response time) of the packets in the pcap files. We also use the same set of previously mentioned statistics on the inter-packet spacing values of the OSI Layer 1 timestamps (i.e., ICMP reply inter-packet spacing). The results are 10 features extracted from the OSI Layer 1 and Layer 3 packets for each 100 measurement taken on the RTU. These features are fed into WEKA Random Forest Tree algorithm, and WEKA standard 10-fold cross-validation [13] is used to train and test the classifier. Basically, this is done to decrease the variation in the performance evaluation process. Operationally, WEKA implements 10-fold cross validation by, *Dividing the dataset into 10 parts (i.e., folds), holding out each part in turn, and averaging the results such that each data point in the dataset is used once for testing and 9 times for training* [13]. Our classification results appear in Table IV.

VII. RESULTS AND DISCUSSION

TABLE II. No-hop statistical experiments.

No-Hop Experiments		
CPU Load	**Task Cycle Period (ns)**	**Average Response Time (ms)**
~15%	100	0.967
~30%	40	0.982
~50%	30	0.986
~70%	20	1.011

TABLE III. Multi-hop statistical experiments

Multi-Hop Experiments		
CPU Load	**Task Cycle Period (ns)**	**Average Response Time (ms)**
~15%	100	1.003
~30%	40	1.008
~50%	30	1.010
~70%	20	1.020

In our no-hop statistically analyzed experiments, we determined that the average ICMP response of the ABB RTU560 is correlated with its CPU load (i.e., task cycle period). That is, the larger the average ICMP response values, then the higher the RTU CPU load and the lower the task cycle period, and vice versa. In our multi-hop statistically analyzed experiments, we determined that the correlation between the average ICMP response and the RTU's CPU load is inferable across multiple hops. Finally, in our no-hop machine learning-based experiments, we demonstrated that a machine learning

Authorized licensed use limited to: IEEE Xplore. Downloaded on October 19,2024 at 14:09:46 UTC from IEEE Xplore. Restrictions apply.

method could be trained to correlate network traffic and RTU CPU load. Also by using information from OSI Layer 1 and Layer 3, better classification is achieved. Since our method evaluates the ICMP ping response behavior of a real RTU device, during the time we are collecting the solicited ICMP replies, we have to realistically consider only the dominant behavior of the device. Therefore, we have determined empirically that by looking for a minimum threshold of 51% of the test patterns, we can use machine learning to very accurately identify task cycle periods given network traffic. Understandably so, in Table IV, the largest misclassification in the confusion matrix is between the machine learning class adjacent to the correct choice; however, our threshold holds in all of our experiments. The probability distribution functions (PDFs) in Figure 5 illustrate the uniqueness of the network traffic behavior that allows each CPU load level to be identifiable.

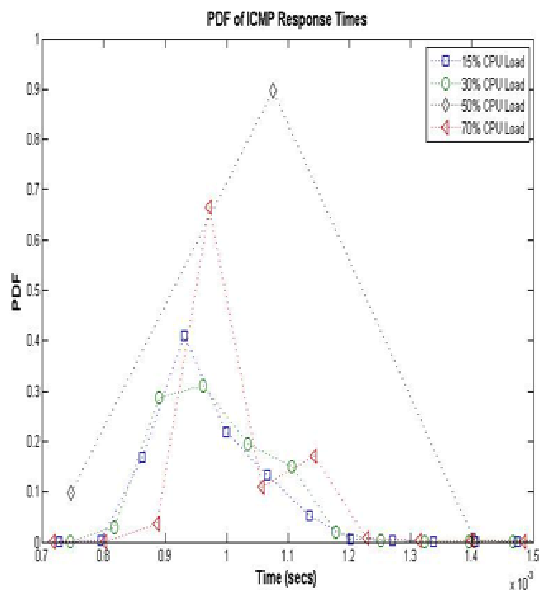

Figure 5: Probability distribution function (PDF) of ICMP response times

TABLE IV. No-hop machine learning-based experiments.

		Predicted CPU Load			
		~15%	**~30%**	**~50%**	**~70%**
Actual CPU Load	**~15%**	**65**	29	4	1
	~30%	31	**57**	3	8
	~50%	4	5	**72**	18
	~70%	6	10	21	**62**

Table IV is the confusion matrix from the WEKA Random Forest Tree algorithm. It displays the number of ICMP reply vectors out of a total of 99 that were properly classified. The proper classes are on the diagonals, highlighted in yellow, with the off-diagonals of each row being misclassifications. We posit that because adjacent classes are similar (i.e., 15% and 30% are similar and 50% and 70% are similar), it will be difficult to separate them with a low rate of error. By combining these similar classes the error rate goes down significantly; however, the resolution of the method decreases as well. By only looking for the location of the majority of the instances, the resolution can be increased while maintaining accuracy.

VIII. Assumptions and Limitations

In this paper, we have investigated the ability to use network measurements as a method to infer CPU load by modifying the task cycle period while keeping the PLC code constant; however in future work, we will consider also modifying the PLC code. The task cycle period essentially denotes how often the PLCs communicate with connected industrial devices; therefore, a smaller task cycle period corresponds to very frequent exchanges between the PLCs and the industrial devices, which probably indicates more precise measurements or operations. Our assumption is that each task cycle period used in our experiment signifies a different type of industrial function. If this cycle is modified from the original setting without authorization, then our measurement and classification technique would detect the change, possibly finding a malicious insider. Another assumption is that by taking multiple measurements from the RTU, we can more accurately capture the overall behavior of the device. We apply a 51% threshold to the classes in the confusion matrix of the machine learning method. In other words, the inference of the machine learning method is based on the class where 51% or more of the measurement values reside. The method is shown in Algorithm 2. In our implementation, the size of the iap_array is 4 to represent the load levels were measured.

In previous work [1], the CPU load of general-purpose nodes were inferred using ICMP, TCP, and UDP network traffic on a network of 50 real nodes via Deterlab. This work demonstrates that the inference method works well in a local area network (LAN) where the switch is lightly loaded. To emulate this light loading on the switch in our test-bed, video streaming is employed. As evidence of light loading, there was no packet loss in this environment. It is assumed that a lightly loaded network is reasonable for most well-designed LANs. If the network becomes heavily congested, then it is likely that most services will deteriorate, including the inference method.

In this work, the focus has been placed on inference over a single-hop. Some preliminary research has been conducted in multiple-hop environments that suggest similar results can be obtained in more complex environments; however, that subject matter is not explored here. As a proof-of-concept, and to simplify the testing and development of our algorithm, we believe single-hop inference is an appropriate first step.

Authorized licensed use limited to: IEEE Xplore. Downloaded on October 19,2024 at 14:09:46 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2

Input: cmatrix, A confusion matrix
Output: pc, Predicted Load Class. -1 means no conclusive decision

```
01: var iap_array[4]
02: //Inter-Arrival Pattern array
03:
04: iap_array=count(cmatrix)
05: //count() returns number of patterns per class
06: //majority() returns class with majority patterns
07:
08: for each Actual Class do
09:         if (majority(iap_array) == class_15% )
10:                 pc = 15
11:         else if (majority(iap_array) == class_30%)
12:                 pc = 30
13:         else if (majority(iap_array) == class_50%)
14:                 pc = 50
15:         else if (majority(iap_array) == class_70%)
16:                 pc = 70
17:         else pc = -1
18: end for
19: return pc
```

We investigated the use of most of the algorithms implemented in WEKA 3.7.11, and the results were roughly similar. We chose Random Forest Tree, because it yielded consistent results, and it was trainable in a reasonable timeframe (e.g., less than a minute for 100s of trees). Other methods will be further explored in future work. Machine learning was used with the understanding that not all of the ping responses will be representative of the state of the device; therefore we will not see 100% conforming behavior. We make the classification decision based on whether a majority of the response behavior conforms to a single predicted class, as noted in Algorithm 1. In this work, metrics such as false-positives are not applicable to the confusion matrix produced by WEKA, since Algorithm 2 refines it to single predicted class per actual class. If the predicted class is not consistent with the actual class, then the prediction is incorrect; therefore, the metric, percent of correct predictions would apply. Based on our results in Table IV, our percent of correct predictions would be 100%.

IX. Conclusion

In this paper, we have reported preliminary results suggesting that the CPU load of an RTU can be inferred based upon its network response behavior. This research relies on inter-packet spacing and ICMP response time as a source of data. Our implementation uses the ping command to solicit these data that are captured and processed by a monitoring node. Machine learning was then used to build classifiers for the CPU load. One of the advantages of this approach is that the measurements and classification do not depend upon the RTU performing any specialized local routine (e.g., antivirus scanning). Furthermore, the CPU load for the RTU can be inferred without knowledge of the logic that it is currently executing or the network protocol with which it is communicating. Therefore, this technique could be applied if the network traffic were encrypted. In other words this technique does not depend on the native protocols of the ICS network, and thus is applicable to a wide variety of ICS networks. In the future, more complex experiments will be conducted using multiple RTUs, and we will also vary the PLC code that is executed.

ICS cyber threats equate to threats to corporate and national infrastructure when one considers the variety of manufacturing, distribution, and automated systems controlled by ICS networks. Ideally, this research can be used as a component in an anomaly-based intrusion detection system that takes advantage of the fact that an ICS generally exhibits predictable behavior when compared to general-purpose information technology systems.

References

[1] L. Watkins, W. H. Robinson, and R. A. Beyah, "A passive solution to the CPU resource discovery problem in cluster grid networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 2000-2007, 2011.

[2] L. Watkins, W. H. Robinson, and R. Beyah, "Using network traffic to infer hardware state: A kernel-level investigation," accepted to *ACM Transactions on Embedded Computing Systems*, 2015.

[3] M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R.J. Bril, "Towards hierarchical scheduling in VxWorks," In *International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, pp. 63 – 72, July 2008.

[4] Vxworks. (2015). Available: http://www.cs.ru.nl/lab/vxworks/vxworksOS.html

[5] WEKA. (2015). Available: http://www.cs.waikato.ac.nz/ml/weka/

[6] K. Stouffer, J. Falco, and K. Scarfone, "Guide to Industrial Control Systems (ICS) Security" *National Institute of Standards and Technology*, SP 800-82, 2011.

[7] K. Zetter. "A Cyberattack has caused confirmed physical damage for the second time ever," *Wired*, January 8th, 2015. Available: http://www.wired.com/2015/01/german-steel-mill-hack-destruction/

[8] Brodersen. Available: http://brodersensystems.com/products/plcrtu-series/

[9] B. Zhu and S. Sastry, "SCADA-specific intrusion detection/prevention systems: A survey and taxonomy," *in 1st Workshop on Secure Control Systems (SCS),* 2010.

[10] S. Parthasarathy and D. Kundur, "Bloom filter based intrusion detection for smart grid SCADA," in *25th IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*, 2012.

[11] M.-K. Yoon and G. F. Ciocarlie. "Communication pattern monitoring: Improving the utility of anomaly detection for industrial control systems," *NDSS Workshop on Security of Emerging Networking Technologies (SENT)*, 2014.

[12] I. Fovino, A. Carcano, M. T. De Lacheze, and A. Trombetta, "Modbus/DNP3 state-based intrusion detection system," in *24th IEEE International Conference Advanced Information Networking and Applications (AINA)*, Perth, WA, 2010.

Authorized licensed use limited to: IEEE Xplore. Downloaded on October 19,2024 at 14:09:46 UTC from IEEE Xplore. Restrictions apply.

[13] WEKA 10-Fold Cross Validation. (2015). Available: http://www.cs.waikato.ac.nz/ml/weka/mooc/dataminingwithweka/transcripts/Transcript2-5.txt

[14] ABB. (2015). RTU500 Series Remote Terminal Unit: Function Description Release 11. Available:
http://www.scribd.com/doc/243709820/E500-FD-Rel11-Part1-Overview#scribd

[15] ABB. (2015). RTU560 Data Sheet DIN Rail RTU 560CID11. Available:
http://www09.abb.com/global/scot/scot258.nsf/veritydisplay/1dc5c9b5bb1ae98b83257a55002d6ab3/$file/E560_CID11_DS.pdf

[16] NIST. (2015). NIST Special Publication 800-82: Guide to Industrial Control Systems Security.Available:
http://csrc.nist.gov/publications/drafts/800-82r2/sp800_82_r2_second_draft.pdf

Authorized licensed use limited to: IEEE Xplore. Downloaded on October 19,2024 at 14:09:46 UTC from IEEE Xplore. Restrictions apply.