Detecting Control System Misbehavior by Fingerprinting Programmable Logic Controller Functionality

Melissa Stockman[a], Dipankar Dwivedi[a], Reinhard Gentz[a], Sean Peisert[a,*]

[a]*Lawrence Berkeley National Laboratory, One Cyclotron Rd., Berkeley, CA, 94720, USA*

1. Introduction

Programmable Logic Controllers (PLC) are ruggedized computing devices used in process automation. They control processes such as manufacturing assembly lines, robotics, scientific instruments, and other machinery that requires some sort of logic to regulate its function. PLCs are built to be simple in function, as in the process shown in Figure 1, and also tolerant of severe conditions such as moisture, high or low temperature and dust. PLCs have existed since the 1960s, before cyberattacks in the modern sense were conceived of, and also before remote network access to PLCs was considered. Early PLCs used serial connections, and only much more modern PLCs have acquired network communication capabilities via TCP/IP in the form of Modbus known as Modbus TCP, and other, similar protocols. Because PLCs can control valuable, physical equipment, and because control systems can have physical consequences to equipment and human life, their *secure* operation is critical to maintaining *safety* [1]. False outputs can have catastrophic consequences, as Zetter [2] demonstrates. Tampering with a PLC can have disastrous effects. Therefore, knowing that the correct program is running is essential to safety and security.

Prior work has shown that non-intrusive load monitoring can be useful to infer the functionality of electrical systems [3]. Recently, it has been shown that patterns in power current signals can be used to infer activity taking place on a computing system [4, §4]. We hypothesized that power signals (specifically current and voltage) could also be used to detect such activity on a PLC. To test our hypothesis, we conducted experiments running different PLC programs. We also examined the relative importance of various features in the classification of these programs. This paper reports on our approach and our results.

This paper is organized as follows. Section 2 discusses related work on power analysis and machine learning to classify signals. Section 3 briefly describes how

*Corresponding author

Email addresses: melissa.stockman1@gmail.com (Melissa Stockman), ddwivedi@lbl.gov (Dipankar Dwivedi), rgentz@lbl.gov (Reinhard Gentz), sppeisert@lbl.gov (Sean Peisert)

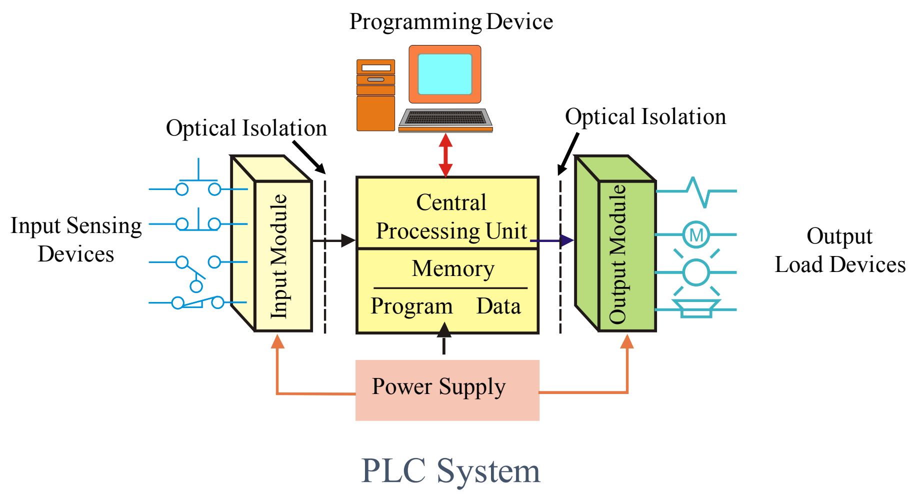


Figure 1: A PLC with inputs, outputs, a power supply, and external programming devices.

we collected data for this power analysis. Section 4 discusses various approaches to conduct the classification of time series data, while Sections 5 and 6 illustrate the experimental framework used in this study to conduct and evaluate the classification of PLC programs. Subsequently, we describe results in Section 7 followed by conclusions in Section 8. Finally, we discuss some limitations and future work in Section 9.

2. Related Work

Power analysis has long been used for non-intrusive load monitoring. Hart [3] was among the first to apply the technique for identifying physical systems by their power signatures. More recently, Gillis and Morsi [5] used a single power sensor to detect, if and which breaker in an electric system is open and closed, respectively. The task was to specify the start time of such events, with very characteristic switching signals in the data. The authors used wavelets with a supervised and unsupervised learning approach. Liebgott and Yang [6] used an active learning approach to identify the usage patterns of a set of household appliances which was similar to the previous work in that it also identified the start and end signatures in noisy measurement data.

In computing, power analysis was one of the first methods to extract hidden information from computing devices. Cryptographic keys have been a particular target of such techniques [7]. In addition, computation stages have been derived from power analysis [8]. Power consumption has been exploited for a variety of other purposes including the identification of Trojans in integrated circuits [9] and to expose a wide spectrum of system-level host information in general computing container clouds [10].

Also related to our work is the use of machine learning for signal classification. Llenas et al. [11] studied the performance of machine learning models for classifying wireless signals using a sampling of the power of the signal over time. Acharya et al. [12] used a convolutional neural network (CNN) to distinguish between normal and myocardial infarction (MI) ECG signals. Most recently, Copos [4, §4] identified programs running on high-performance computing machines, applying frequency and wavelet analysis to power signatures.

Our approach is different from these existing approaches in that, to the best of our knowledge, none of these prior approaches has attempted to identify the activity running on a PLC. At the same time, our approach builds on essentially all of this prior work by leveraging both data sources (current and voltage) as well as analysis techniques.

3. Data Collection

A phasor measurement unit (PMU) is a device that measures electrical waves [13]. Specifically, it measures voltage magnitude, voltage angle, current magnitude, and current angle (i.e., a *phasor* [14]). We generated and collected the data by running different PLC programs on a single Siemens Simatic S7-1200 PLC [15] and collecting power results using a distribution-level PMU (termed a "micro-PMU" or"μPMU" [13]), manufactured by Power Standards Laboratory, that measures power signals at 512 samples per cycle, 60 cycles per second, and outputs samples at 120 samples per second — a much higher frequency than typical transmission level PMUs. We monitored the power draw of the PLC with a dedicated current loop that fed into the μPMU.

We sequentially deployed 10 different *ladder logic* programs (a graphical, low-level programming language) to the PLC that represented typical workloads (see Table 1). The programs were chosen with two criteria in mind. The first was that they should exercise different parts of the PLCś functionality i.e. networking, analog-to-digital conversion etc. We chose these programs as distinguishable from each other in a relatively major way. We then chose some programs that had overlapping PLC functionality. We did this to make our task more challenging and we were interested in determining if even small changes to the same program could be identified.

We collected and labeled μPMU data for each of the running programs. Additionally, an "idlestate" was recorded where the PLC was not running any code. This enabled us to find a baseline for our supervised learning approach. We conducted several experiments namely exp6, exp7, exp8, and exp9 at different times by running different PLC programs. These different experiment runs allowed us to design and test simple and hard problems as described below.

Goals and Threat Model. PLCs control a myriad of critically important systems including gas pipelines, electrical power grids, railroad signals, and potable water distribution. Any malicious activity targeting this device could cause damage to equipment, failure of safety systems, or reckless release of hazardous material. Attacks on a PLC could come in the form of unauthorized modifications to the firmware, configuration alteration or changing the execution control flow as described in [16]. For our work, we define misbehavior of a PLC as the intentional manipulation of the ladder logic code to adversely affect the process being controlled. This type of attack could be used to incorrectly switch railway tracks, mix incorrect amounts of chemicals, disrupt electrical substation machinery, cause tank pressure sensors to be read incorrectly, etc.

Our goal was to determine if the currently running program was the correct program. In order to do this, we needed to distinguish between major and very minor changes in the programs. Therefore some of the 10 programs were very similar to each other (i.e., a constant had a different value) while others were very different. Each program was run for 2 minutes for a total of 14,400 "rows" (120 samples/second $\times$ 120 seconds) of data containing voltage and current measurements for each.

4. Description of the Classification Problem and Approaches

The μPMU power data we collected was used to train our machine learning models. We attempted to classify the PLC programs based on the energy consumption profiles recorded by the μPMU. Since current and voltage were changing over time as the program was running, we looked at the problem of determining which PLC program was running as a time series analysis problem.

One approach to classifying time-series data is to use manually-engineered features from statistical properties of the signal. This approach typically includes examining attributes of a time series, such as minimum, maximum, mean, standard deviation, and frequency. These attributes can be used to infer properties of the time series as a whole or for some distinct window of time. However, this approach often requires some domain knowledge about the data, such as specific frequency bands and other statistical properties. Image classification problems are examples of this approach, where manually-engineered features are used by applying certain filters to the image data. Another approach to classifying time-series data is in the time domain. In contrast to using manually-engineered features for classification problems, in this approach the data is looked at, point by point, sequentially.

To classify each program using the μPMU power data, we tried several different machine learning approaches including Support Vector Machines (SVM), K-Nearest Neighbor (KNN), Random Forests (RF), and Convolutional Neural Networks (CNN). In the end, we chose RFs due to their ability to classify large datasets accurately with computational tractability, and CNNs due to their accuracy and ability to classify the data without having to use pre-built filters.

To test the performance of our models, we used two scenarios representing basic and difficult classification problems as defined in Section 5. In both scenarios we also classified programs with significant changes among themselves. The overall accuracy of each model was calculated by exact match accuracy — that is, the total number of correctly classified programs divided by the total number of all the samples.

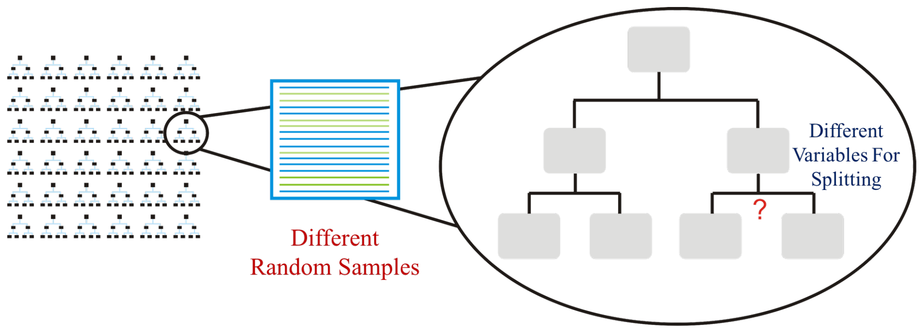


Figure 2: A Schematic of a Random Forest Classifier Random Forest Classifier.

$$\frac{1}{n}\sum_{i=1}^{n} I(Y_i = Z_i)$$

where I is the indicator function.

4.1. Random Forest (RF)

We selected the random forest classifier due to its computational efficiency on large datasets and its ability to handle a large number of input variables as well as its ability to generalize well. Additionally, random forests show the importance of features in the classification which would assist us in deciding which features to keep in our models.

To best describe the random forest classifier, we first describe a decision tree classifier. Decision tree classifiers [17] are simple yet powerful models which employ a divide and conquer approach to classification. Data is recursively partitioned into sections based on the best split which separates out one class. The right side of Figure 2 shows a magnified decision tree.

Random Forests are collections of these decision trees as shown on the left side of Figure 2. For each sample of data, a number of decision trees' results are aggregated. The final output is then the class that was predicted the most by the individual decision trees. For our Random Forest model, we leveraged the RandomForestClassifier [18] as part of the scikit-learn package [19] with default parameters.

4.2. Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are designed to recognize patterns in images directly from the pixel representation of an image [20]. We decided to try this approach on our dataset, since the current magnitude over time can be thought of as a "picture" of the running PLC program. The input values are related positionally to each other, i.e., nearby values in the time-series of current magnitude are extremely related.

A CNN, in contrast to RF, does not require complex feature engineering. Data can be input "as is" into the classifier. This is key because a highly accurate model can be trained without the need for domain expertise regarding the PLC programs. The training phase learns “filters” which become more complex as the data propagates through deeper and deeper layers. CNNs recognize simple patterns in the initial layers building up to much more complex patterns in the final layers. They extract local features by constraining the reactive region of the hidden layers to small local patches. Passing through the layers, neurons become related to each other, and some become more influential than others. Figure 3 shows a typical CNN.

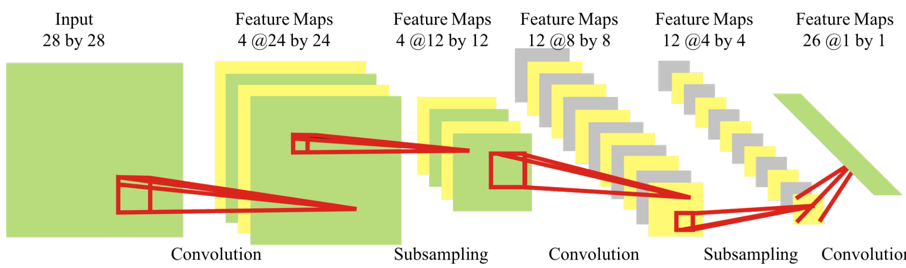


Figure 3: A Schematic of a typical Convolutional Neural Network.

For any machine learning model, it is important to guard against overfitting the data. That is, it is necessary to avoid creating a model that is too highly tuned to the idiosyncrasies of the training dataset and hence does not perform well on new data. For CNNs, using a “dropout layer” randomly selects neurons so as not to continually use the most influential ones as predicting the final output. This guards against overfitting by allowing the network to learn different views of the data.

We used Tensorflow [21], an open source library developed by Google for dataflow programming of highly computational applications to implement our neural network model. The CNN was composed of three layers: two convolutional layers, then a fully connected layer. The “Softmax” activation function was used. It maps the output to c classes as a set of probabilities. The highest probability class is assigned as the predicted class.

Table 1: PLC Program Description

	Networking	ADC	Digital out	Description
Idlestate				CPU in stop state
program 3	x			Reads from Modbus, runs a function on the data and returns decision via Modbus
program 4			x	Makes LEDs blink (1s period)
program 5		x	x	Read switch status and displays on build in LED (all off)
program 6	x	x		Reads analog value and sends it via modbus
program 7 client	x	x		version 2 of program 6
program 7 server	x			version 2 of program 3
program 8	x			version 3 of program 3 (debug variables present in program)
program 9	x			PROFINET Client
program 10	x			version 3 of program 3 (release version/no debug variables)

5. Experimental Scenarios

Our experiments were broken up into two different scenarios. These scenarios were of particular interest for testing our hypothesis and being able to use the current approach in monitoring potential real-time manipulation of the PLC. We describe them in the following subsections.

5.1. Scenario 1

First, we combined all datasets (experiment runs 6, 7, 8, and 9) together and used the result of 5-fold cross validation as the performance indicator. We considered this scenario a useful starting point. Combining all datasets into one big dataset, and subsequently using cross-validation led to higher accuracy than Scenario 2. This was due to the fact that cross-validation's random selection of the training set contained a small amount of data from each run with its specific random noise, thereby letting the classifier learn the random information for that run. This approach would perform well in an online situation where training data would continuously be added to update the model.

5.2. Scenario 2

Scenario 2 involved training the classifier on three separate datasets (e.g., experiment runs 6, 7, and 8) and testing on the fourth dataset (e.g., experiment 9), i.e., 4-fold cross validation with completely different datasets. This problem was more complex than Scenario 1 because experiments were carried out at different times of the day and different days, and each dataset was subject to influence by external factors such as voltage fluctuations and temperature. This scenario was used to test the robustness of a fixed model that could be trained once and used statically any time in the future without the need for additional online training data. In this scenario, we report the performance measures as the average accuracy achieved for individual classifications of each dataset while training on the rest of the three datasets.

6. Classification of PLC Programs for Different Scenarios

These scenarios posed significant challenges in classifying PLC programs. Considering the complexity of the classification problem at hand, both time and frequency domains were deemed necessary for our analysis. Therefore, in order to detect subtle differences between PLC programs, we tested our scenarios in both the time and frequency domains individually. This allowed us to more granularly tune our machine learning models' metaparameters.

The μPMU power data was a time series of electrical information collected from the power draw of the attached PLC. It included current magnitude and angle, and voltage magnitude and angle. The data was labeled for each PLC program run, plus the "idlestate" as described in Section 3.

6.1. Feature Engineering

We applied feature engineering techniques to the data including rolling averages, time-lagged windows and Butterworth filters.

The rolling average data was created by calculating the average of the data points over a window of time. This window was then slid through the entire original dataset to create a new dataset. Rolling averages have the effect of smoothing the data. Because it averages all features in a window of time, it removes the small variations between successive time intervals which could be due to noise. This allowed our machine learning models to more readily focus on the signal rather than the noise.

For time-lagged windows, we combined consecutive values of the current magnitude to form a much larger row of input features we defined as a "lag window."

Let T be a time series of length n, a lag window W_x of T is a sampling of length $w < n$ of contiguous positions from T, such that $W_t = t_x, ...t_{x+w-1}$ for $1 <= x <= n - w + 1$.

We flattened m contiguous current samples into vectors. Each component of the vector was a current magnitude at consecutive moments in time. We can think of the m values of current magnitude as an m dimensional vector and note that within this m dimensional space only a small number of "points" are associated with a particular PLC program run.

Because different PLC programs exhibit varying frequencies within certain bands in the spectrum, we used the Butterworth filter—a type of signal processing filter designed to mask unwanted frequencies, and known to give an especially flat frequency response in the passband [22].

Each of these techniques created an alignment-free framework which allowed for the fact that the beginning and end points of the program runs were not necessarily precisely aligned with the recorded start time. This was due to the fact that each program was started manually and the measurement granularity of the μPMU was in $\frac{1}{120}$ths of a second.

6.2. PLC Program Classification in Time Domain

In the time domain, for scenario 1, we used current magnitude and angle, and voltage magnitude and angle measurements. For scenario 2 we used only current magnitude and angle, as we noted that these measurements are determined by the PLC itself and are not dominated by the surrounding environment since the PLC only consumes $\approx 3W$ as opposed to other possible noisy consumers in the measurement environment that may consume hundreds of watts.

In scenario 1 we used each set of timestamped values of these features, as a separate row of input. We also applied rolling averages to these features. For scenario 2, we applied rolling averages as well as a lag window.

Through heuristics, we determined that the optimal size for the lag window for our data was approximately 6 seconds ($m \approx 720$) and a window size of 20 gave the best for the rolling average. That being said, this result is for our data, which, as with all data, has noise of various kinds. Other datasets may have different ideal lag windows and window sizes. In order to identify such datasets, procedures and guidelines are discussed in more detail elsewhere [23].

6.3. PLC Program Classification in Frequency Domain

We converted time domain signals into the frequency domain using Discrete Fourier Transform (DFT) [24, 25]. We used individual time series describing a particular feature for a specific PLC program (e.g., the current magnitude for idlestate), and subsequently, we computed frequencies using DFT. Liaw et al. [26] demonstrated that the accuracy of the RF classifier depends on how uncorrelated trees are in the forest. The more uncorrelated trees are in the forest, the more accurate the RF classifier. Therefore, to remove correlations between trees as well as noise, and separate signals so that the individual trees are strong, we used rolling averages and Butterworth filters. Rolling averages (also known as moving averages) reduce the noise in the signals because of the smoothing effect of averages, while Butterworth filters are more versatile and remove unwanted frequencies with a ripple free response [22]. Filter windows were chosen based on the exhaustive search technique. For example, the RF classifier was tested for multiple filter windows (sizes) that were slid through the spectrum.

7. Results and Discussion

We discuss our results from Tables 2 and 3 separately for frequency and time domains.

We also discuss the confusion matrices that show the errors in our predictions. Columns are the predictions for each PLC program (or the "idlestate"). For example, in Figure 4a, the first column shows all samples predicted to be "idlestate", the second column shows all samples predicted as r_code10, etc. Rows represent the actual PLC program that was running (or the "idlestate"). The top row shows all samples where the PLC was actually in the "idlestate." Moving along the row, the mispredictions for "idlestate," and which programs it was mispredicted as, are shown in the corresponding column. The matrix gives a summary of all mispredictions. All non-zero values outside the diagonal are incorrect predictions. A model with perfect prediction would have a confusion matrix where all values not on the diagonal are zero.

We display the confusion matrices as heat maps in order to illustrate the fact that even in the cases of some wrong predictions, the majority of predictions fall into the correct class. This is important because if the model is used over a 2 minute window of time, instead of each 0.2 seconds, accuracy would be 100%. We show our accuracy results based on the stricter time constraint to show that our approach can be used to detect a program change within 0.2 seconds of its occurrence.

Table 2: Performance of the Random forest Classifier for two scenarios

	Scenario 1		Scenario 2		
	Without Roll. Avg.	With Roll. Avg.	Without Filters	With Filters[a]	With Filters[b]
All Programs	70.17%	97.7%	11.2%	24.6%	28%
4 Prog. States	77.7%	99.08%	24.2%	28.3%	83%

Filters[a] – low pass (with normalized cutoff frequency -2.5E-06)
Filters[b] – a low pass (with cutoff frequency -2.5E-06) cascaded with a bandpass filter (4th order, low cutoff 45 Hz, high cutoff 55 Hz)

7.1. Frequency Domain

It is clear from Table 2 that the RF classifier performed better for scenario 1 than scenario 2. For scenario 1, the RF classified more than 70% of programs accurately when we trained the classifier using all the datasets. Furthermore, the RF classifier's performance improved from 70% to 77% when a rolling average window with a triangular window size of 120 samples (data worth 1 sec) was used in the frequency domain. The improved performance of the classifier can be ascribed to the rolling average filter that reduced the noise in the signals. Similarly, when we used only four program states for classification, the RF classifier identified approximately 97.7% to 99.08% of the programs accurately with and without rolling average filters (Figures 4a and 4b), respectively. Correctly predicted programs are shown along the diagonal. The misclassified programs ($\sim$ 3%; for all programs) are spread across other cells and do not show any pattern, which shows that the RF classifier performed consistently. This particular scenario was considered as a simple problem, and the RF classifier performed remarkably. Indeed, when the classifier did not perform effectively, it was because of the noise in the dataset. Hence, using a rolling average filter improved the classifier's performance significantly.

Scenario 2 was considered a hard problem, because here we trained the classifier on a dataset (combining three different datasets) and testing on a completely new dataset (fourth dataset). In this scenario, the RF classifier performed poorly and was able to identify programs accurately only 11% and 24% for for Scenario 1 and Scenario 2, respectively. However, when we used a low pass Butterworth filter, the RF classifier showed slight improvements from 11% to 24% and from 24% to 28% for Scenario 1 and Scenario 2, respectively.

The classifier performed poorly in identifying all programs (programs with major and minor differences). We then tested with a low pass Butterworth filter cascaded with a band pass Butterworth filter. This improved accuracy to 83% for the four program states (programs with major differences) (Figure 5b).

Figure 6a compares frequency contents computed for the time series of the current magnitude across four program states for Scenario 1. Here, we combined all the datasets as described in Section 5. It is clear from Figure 6a that the frequency contents show different signatures across datasets for different programs;

therefore, the RF classifier performed effectively for Scenario 1. Similarly, Figures 6b and 6c compare frequency contents computed for the time series of the current magnitude across four program states for Scenario 2. Figure 6b shows frequencies when a low pass Butterworth filter was applied, while Figure 6c shows frequencies when we filtered signals using low pass and band pass Butterworth filters. It is clear from the frequency contents (Figure 6b) that there is no distinguishable pattern for the RF to detect. For example, r_code9 shows different amplitudes for each of the different datasets. Therefore, it is hard for the classifier to perform effectively using these features. Furthermore, Figure 6c demonstrates that there are frequency bands across the spectrum where the classifier can grow strong trees, as frequency contents can be distinguished between programs (e.g., PLC programs). Accordingly, the classifier performed relatively better with two filters despite Scenario 2 being a hard problem.

7.2. Time Domain

As shown in Table 3, for scenario 1, the performance of the RF model in the time domain had 89% accuracy without rolling average and 97% with rolling average using all the available μPMU features (current magnitude and angle, voltage magnitude and angle). The accuracy with only 4 program states rose to 95% without rolling average and 99% with rolling average. When using completely different datasets for training and testing in scenario 2, the accuracy dropped drastically to 20% and 30% with and without rolling average respectively. This was due to the fact that many of the programs were too similar to distinguish between. When reducing the PLC programs down to those that were significantly different, the RF model achieved a respectable 71% with rolling average and 76% with lag-windowed magnitude.

Figure 7 shows the confusion matrix/heat map for scenario 1 for all programs using rolling averages. As can be seen, the mispredictions are distributed throughout the matrix indicating that there was not a general confusion between any two particular programs and that our technique could be used over some longer window of time to achieve 100% accuracy.

Figure 8a shows the heatmap for scenario 2 using lag windows. This model performed relatively well at 76% accuracy.

For the CNN model, we only used lag windows and did not perform rolling averages. We did this because the CNN we used was originally designed for image classification, thus we wanted our inputs to be similar to that of an image. For detecting all 10 programs, the CNN did not perform well, (40% in scenario 1 and 30% in scenario 2). We explain this with the fact that the random noise in each experiment is larger than the signature change due to the minimal program changes. However, the CNN performed the best overall in both scenarios for 4 program states at 84%. Of note is that the CNN performed the same on the 4 program states in both scenarios. In this scenario the changes in programs were significant enough to clearly identify each program.

Figure 8b shows that the majority of misclassifications occurred due to r_code7client being predicted incorrectly as r_code9. This may indicate that portions of r_code9 are similar to r_code7server (i.e., they both use the networking function at some point) but not overwhelmingly so, since a preponderance of the samples were correctly classified as r_code7server.

8. Conclusion

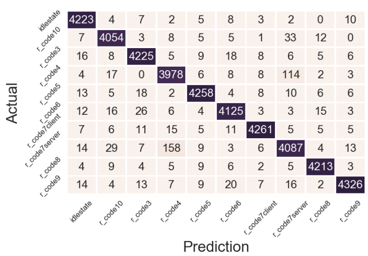
Our framework and experiments show that the power signature can identify PLC programs as the output of a μPMU using the two machine learning approaches of random forests and convolutional neural networks. Our accuracy on a single dataset of 10 PLC programs that included programs which were very similar, reached 99.08%. Using data from completely separate runs, we could still detect major program changes at 84% accuracy.

In practical terms, once the models are developed, implementation to secure an actual system is straightforward and does not require domain knowledge. It only entails attaching a μPMU to the PLC and collecting data for a short period of time (approximately 5 minutes because a PLC's cyclic program ensures a signature after a small interval). The model training takes another 30 minutes. Once training is complete, the model is installed with its associated monitoring program which runs constantly, looking for and alerting on detection of anomalies. The model does not change unless the PLC program is changed.

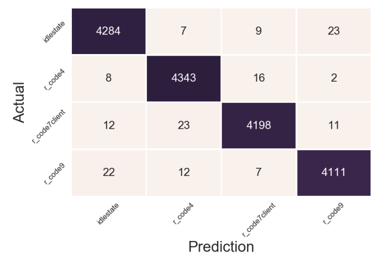
9. Limitations and Future Work

This study demonstrated the potential for classification of PLC programs both in time and frequency domains. We showed that different filters could help improve predictions of PLC programs in the frequency domain. Similarly, time domain also demonstrated a tremendous potential in the classification of various programs. However, data collected on different days resulted in some incoherency in signals of the same program between datasets. Moreover, RF and CNN classifiers were not able to identify programs with minor differences effectively. These issues could be addressed in the future work. It is also desirable to include more complexity in data by using more than one PLC in future studies to evaluate the robustness of our method. Future work may involve taking advantage of time and frequency domains together by combining the two domains. Future work may also include designing a specific filter in the frequency domain for a particular problem set. In the time domain, it would be interesting to explore how different deeper CNNs would perform when we include more features.

Ensuring cybersecurity typically involves identifying threats in real-time and from a variety of different possible origins and threat vectors. Moreover, *actionable* cybersecurity requires higher order defense than detecting simple anomalies to identify that something is wrong. We have demonstrated how machine learning algorithms can be applied to monitor certain classes of threats to operational technology devices controlling cyber-physical systems. However, future research will undoubtedly be useful in uncovering solutions to additional classes of cyber attacks.

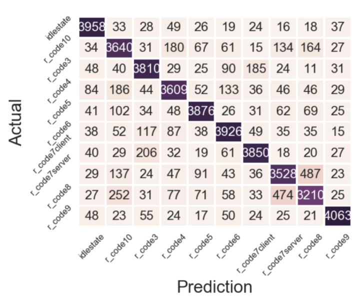


(a) Predicting all the PLC programs using frequencies. Before computing frequencies, the time series data were smoothed using a rolling average filter.

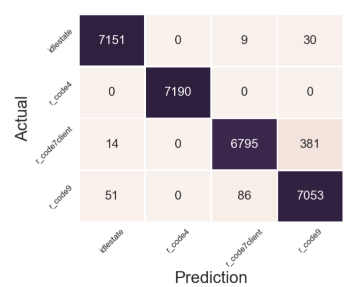


(b) Predicting only four PLC program states using frequencies. Before computing frequencies, the time series data were smoothed using a rolling average filter.

Figure 4: Scenario 1 – Confusion matrices for Scenario 1 in frequency domain

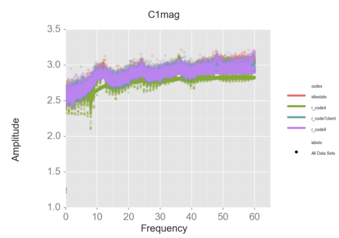


(a) Predicting all the PLC programs using frequencies. Before computing frequencies, the time series data were filtered using Butterworth Filters; a low band filter was cascaded with a band pass filter.

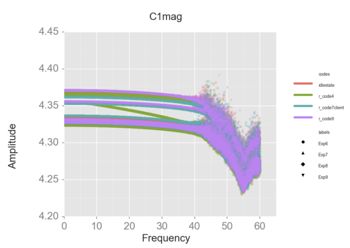


(b) Predicting only four PLC program states using frequencies. Before computing frequencies, the time series data were filtered using Butterworth Filters; a low band filter was cascaded with a band pass filter.

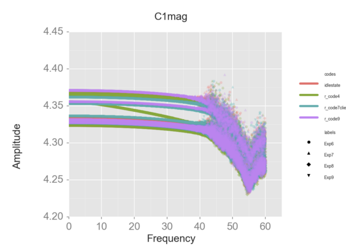
Figure 5: Scenario 2 – Confusion matrices for Scenario 2 in frequency domain



(a) Scenario 1 – Comparing the frequency contents across four program states for the current magnitude. Before computing frequencies, the time series data were smoothed using a rolling average filter.



(b) Scenario 2 – Comparing the frequency contents across four program states for the current magnitude. The time series data were filtered using a low pass Butterworth Filter before computing frequencies.



(c) Scenario 2 – Comparing the frequency contents across four program states for the current magnitude using Butterworth Filters, a low pass filter cascaded with a band pass filter.

Figure 6: Comparing frequency contents for the current magnitude with different filtering approaches.

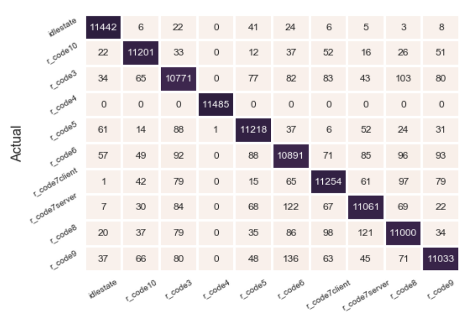
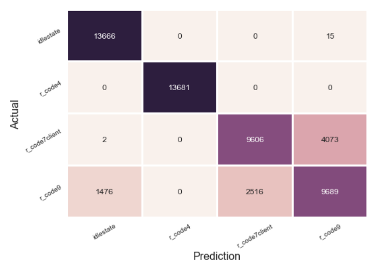


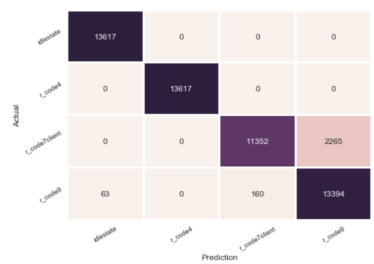
Figure 7: Scenario 1 – Time domain RF using rolling averages on all programs.

Table 3: Time Domain Performance

	Scenario 1		Scenario 2	
	Without Roll. Avg.	With Roll. Avg.	Roll. Avg.	Lag Windowed
RF all programs	89%	97%	20%	30%
RF 4 prog. states	95%	99%	71%	76%
CNN all programs	40%	NA	NA	30%
CNN 4 prog. states	84%	NA	NA	84%



(a) Time domain RF using lag windows on 4 program states.



(b) Time domain CNN using lag windows on 4 program states.

Figure 8: Scenario 2 – Confusion matrices for Scenario 2 time domain

Acknowledgments

This work was supported by the Laboratory for Telecommunication Sciences and has been authored by authors at Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors of this work.

[1] C. McParland, S. Peisert, A. Scaglione, Monitoring Security of Networked Control Systems: It's the Physics, IEEE Security & Privacy 12 (6) (2014) 32–39.

[2] K. Zetter, Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon, Broadway books, 2014.

[3] G. W. Hart, Nonintrusive appliance load monitoring, Proceedings of the IEEE 80 (12) (1992) 1870–1891.

[4] B. Copos (*Advisor: Sean Peisert)*, Modeling Systems Using Side Channel Information, Ph.D. thesis, University of California, Davis (2017).

[5] J. M. Gillis, W. G. Morsi, Non-intrusive load monitoring using semi-supervised machine learning and wavelet design, IEEE Transactions on Smart Grid 8 (6) (2017) 2648–2655. `doi:10.1109/TSG.2016.2532885`.

[6] F. Liebgott, B. Yang, Active learning with cross-dataset validation in event-based non-intrusive load monitoring, in: 2017 25th European Signal Processing Conference (EUSIPCO), 2017, pp. 296–300. `doi:10.23919/EUSIPCO.2017.8081216`.

[7] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: Advances in cryptology—CRYPTO'99, Springer, 1999, pp. 789–789.

[8] Y. Carmeli, On bugs and ciphers: New techniques in cryptanalysis, Ph.D. thesis, Technion-Israel Institute of Technology, Faculty of Computer Science (2015).

[9] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, B. Sunar, Trojan Detection Using IC Fingerprinting, in: Proceedings of the IEEE Symposium on Security and Privacy, 2007, pp. 296–310.

[10] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, H. Wang, Containerleaks: emerging security threats of information leakages in container clouds, in: Proceedings of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2017, pp. 237–248.

[11] A. M. Llenas, J. Riihijärvi, M. Petrova, Performance Evaluation of Machine Learning Based Signal Classification Using Statistical and Multiscale Entropy Features, in: Proceedings of the 2017 IEEE Wireless Communications and Networking Conference (WCNC), 2017.

[12] U. R. Acharya, H. Fujita, S. L. Oh, Y. Hagiwara, J. H. Tan, M. Adam, Application of deep convolutional neural network for automated detection of myocardial infarction using ECG signals, Information Sciences 415 (2017) 190–198.

[13] Power Standards Laboratory, PQube Phasor Measurement Unit, `http://pqubepmu.com/`.

[14] A. G. Phadke, Synchronized phasor measurements in power systems, IEEE Computer Applications in Power 6 (2) (1993) 10–15.

[15] Siemens Simatic S7-1200 PLC, `https://www.siemens.com/global/en/home/products/automation/systems/industrial/plc/s7-1200.html`.

[16] A. Abbasi, M. Hashemi, Ghost in the PLC: Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack, in: Proceedings of Black Hat Europe, Black Hat, 2016.

[17] R. Kohavi, R. Quinlan, Decision Tree Discovery, in: Handbook of Data Mining and Knowledge Discovery, University Press, 1999, pp. 267–276.

[18] scikit-learn - Forests of randomized trees, `http://scikit-learn.org/stable/modules/ensemble.html#forest`.

[19] scikit-learn - Machine Learning in Python, `http://scikit-learn.org/stable/`.

[20] Y. LeCun, Y. Bengio, Convolutional networks for images, speech, and time series, in: M. A. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, MA, USA, 1998, pp. 255–258.

[21] TensorFlow - An open-source software library for Machine Intelligence, `https://www.tensorflow.org`.

[22] S. Butterworth, On the theory of filter amplifiers, Wireless Engineer 7 (6) (1930) 536–541.

[23] K. M. Tan, R. A. Maxion, "Why 6?" Defining the Operational Limits of stide, an Anomaly-Based Intrusion Detector, in: Proceedings of the 2002 IEEE Symposium on Security and Privacy, IEEE, 2002, pp. 188–201.

[24] J. W. Cooley, J. W. Tukey, An algorithm for the machine calculation of complex fourier series, Mathematics of computation 19 (90) (1965) 297–301.

[25] E. Ziegel, Numerical recipes: The art of scientific computing (1987).

[26] A. Liaw, M. Wiener, et al., Classification and regression by randomforest, R news 2 (3) (2002) 18–22.