Detecting Anomalies in Programmable Logic Controllers Using Unsupervised Machine Learning

Chun-Fai Chan, Kam-Pui Chow, Cesar Mak, Raymond Chan

Chapter 7

DETECTING ANOMALIES IN PROGRAMMABLE LOGIC CONTROLLERS USING UNSUPERVISED MACHINE LEARNING

Chun-Fai Chan, Kam-Pui Chow, Cesar Mak and Raymond Chan

Abstract Supervisory control and data acquisition systems have been employed for decades to communicate with and coordinate industrial processes. These systems incorporate numerous programmable logic controllers that manage the operations of industrial equipment based on sensor information. Due to the important roles that programmable logic controllers play in industrial facilities, these microprocessor-based systems are exposed to serious cyber threats.

This chapter describes an innovative methodology that leverages unsupervised machine learning to monitor the states of programmable logic controllers to uncover latent defects and anomalies. The methodology, which employs a one-class support vector machine, is able to detect anomalies without being bound to specific scenarios or requiring detailed knowledge about the control logic. A case study involving a traffic light simulation demonstrates that anomalies are detected with high accuracy, enabling the prompt mitigation of the underlying problems.

Keywords: Programmable logic controllers, anomaly detection, machine learning

1. Introduction

Supervisory control and data acquisition (SCADA) systems have been employed for decades to manage and control critical infrastructure assets. With human lives and the economy at stake, SCADA system failures – whether due to accidents or attacks – cannot be tolerated. Therefore, it is vital to detect SCADA system anomalies and implement effective mitigation strategies.

Programmable logic controllers (PLCs) are the workhorses of SCADA systems. These microprocessor-based systems implement programmable logic that processes input signals from sensors that measure system/environment state to produce output signals that are transmitted to actuators as well as other programmable logic controllers that operate and manage industrial equipment and processes. Programmable logic controllers are typically small, rugged, specialized devices designed to perform specific control tasks, often operating in harsh environments with extreme temperatures and strong vibrations. Industrial systems may have tens to hundreds of programmable logic controllers. Large infrastructure assets such as power grids and oil and gas pipelines have thousands of programmable logic controllers.

Programmable logic controllers are exposed to inadvertent and malicious threats that can impact their ability to safely operate industrial systems and facilities. The most common inadvertent threats are posed by control program implementation bugs. Malicious threats include memory read/write logic attacks [20, 21], malware worms [5, 6, 16], time bombs [1, 7], and stop and start attacks [22]. These threats make it imperative to develop security solutions for monitoring the states of programmable logic controllers to uncover latent defects and anomalies.

Unfortunately, the limited computational and storage resources of programmable logic controllers make it difficult to deploy conventional security measures such as firewalls and intrusion detection systems. Novel and efficient methodologies are required to detect anomalous controller behavior in real time, and help support prompt mitigations and forensic investigations of incidents [9, 22].

Machine learning, which has been employed with much success in intrusion and anomaly detection systems for traditional computing and networking infrastructures, is a promising approach for developing similar systems for programmable logic controllers. Supervised learning, which takes in training data with labeled outcomes, is oriented towards data clustering and classification. Unsupervised learning, which takes in unlabeled data, is geared towards outlier detection. In both cases, a mathematical model is generated from the training data and the model serves as a classifier for new data. Either model can be used for anomaly detection.

It is difficult to apply supervised learning to detect attacks on programmable logic controllers due to the lack of genuine attack data; additionally, the problem spaces (numbers of attack patterns) are large and simulating every attack pattern to generate data is infeasible. In contrast, unsupervised learning uses datasets without labels [12, 14]. A

training dataset covering normal behavior is created and normalized to construct a model that identifies outliers.

Anomaly detection is conceptually identical to outlier detection, which makes unsupervised learning ideal for the problem at hand. In fact, outlier identification is virtually equivalent to applying unary classification with respect to good cases.

A one-class support vector machine is a special case of a support vector machine with unary classification [17]. In this approach, data points are grouped using correlations that are computed to yield the normal state class. The region corresponding to normal state class data is used to assess if a new data point is an outlier. This approach is essentially a sophisticated regression test where the training data is processed collectively. It is especially appropriate when the training dataset mainly comprises normal state data and very little anomalous data. Indeed, the approach is well-suited to anomaly detection in programmable logic controllers because attacks are rare and attack data is hard to come by whereas normal data is readily captured during day-to-day operations.

This chapter describes a methodology that leverages unsupervised machine learning to monitor the states of programmable logic controllers to uncover latent defects and anomalies. The methodology, which employs a one-class support vector machine, can detect anomalies without being bound to specific scenarios or requiring detailed knowledge about the control logic. In addition to conventional data capture methods, the methodology leverages an additional security block in a programmable logic controller to detect anomalies [1]. The historian is also employed to store timestamped programmable logic controller state information (i.e., key memory address values) for anomaly/attack analyses and forensic investigations. A traffic light simulation case study employing a Siemens S7-1212C programmable logic controller demonstrates that anomalies are detected with high accuracy.

2. Related Work

Garitano et al. [2] have reviewed several anomaly detection methodologies and conclude that network intrusion detection systems may not be able to efficiently detect attacks on industrial control systems. Furthermore, since programmable logic controllers typically have limited computational resources, implementing host-based intrusion detection systems is generally infeasible.

Hsu et al. [4] have evaluated several machine learning algorithms on datasets comprising normal operational data from SCADA networks.

Their results demonstrate that machine learning algorithms are able to accurately detect most attacks.

Schuster et al. [10] conducted anomaly detection experiments in two plant process control networks using one-class support vector machines and isolation forest classifiers. Their studies revealed that network traffic data is inadequate for training purposes when sufficient programmable logic controller traffic is not available.

Wu and Nurse [18] have observed that valuable information can be obtained by monitoring the memory addresses of programmable logic controllers, regardless of whether the controllers were executing normally or were under attack. They also evaluated the use of a programmable logic controller logger as a forensic tool that continuously polls the memory variables in a running programmable logic controller.

Yau and Chow [20, 21] have proposed two approaches for detecting attacks on programmable logic controllers. One approach applies machine learning to logged data of pre-selected memory values of a programmable logic controller to detect abnormal operations [20]. The other approach employs a control program logic change detector that leverages anomaly detection rules to detect and record undesirable events [19].

Both the approaches require knowledge about the control logic before monitoring procedures can be applied. They may, therefore, be impractical because the personnel responsible for monitoring the security of SCADA systems are typically not involved in SCADA system development. In addition, remote monitoring of programmable logic controllers via active polling imposes network overhead that is unacceptable in industrial control system environments.

To overcome these challenges, Chan et al. [1] proposed the incorporation of a security block module to support programmable logic controller logging and attack detection capabilities. Specifically, they installed a security block (i.e., programmable logic controller code) on the device to capture selected memory content and other internal device information for monitoring purposes. This approach can help detect programmable logic controller memory read-write logic attacks with high accuracy while maintaining a low network footprint. In addition, the security block can verify the number of data blocks installed in a programmable logic controller to detect worm attacks, which is more efficient than traditional network memory address value polling method using libnodave [3] or the Siemens Step 7 library [8].

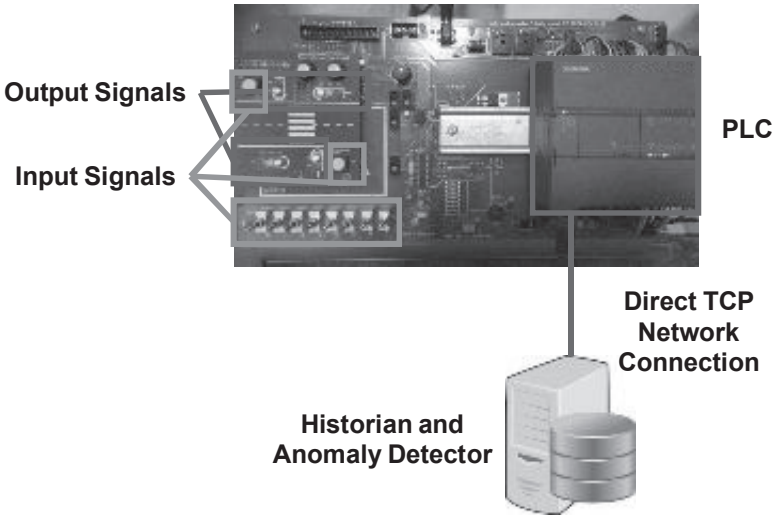

Figure 1. Experimental setup.

3. Anomaly Detection Case Study

This section describes the experimental setup and the methodology for detecting anomalous programmable logic controller operations.

3.1 Experimental Setup

Figure 1 shows the experimental setup. A Siemens S7-1212C programmable logic controller was installed with a traffic light control program that manages interactions between switches and the sequencing and durations of traffic lights. In addition to the standard traffic control light program, the programmable logic controller was equipped with a security block that transmitted input, output and memory address values to a historian via a direct TCP connection. All this information was recorded in a log file by the historian for anomaly detection and forensic analysis.

Rogue attacks on the programmable logic controller were executed by incorporating attack logic in the device. Upon receiving certain input signals, the program logic altered output signals to launch the attacks.

The objective of the experiment was to detect anomalous behavior. Events such as direct attacks, hardware failures and implementation bugs produce anomalies. By attaching timestamps to the events, anomalous situations can also be investigated retroactively by examining the data maintained by the historian.

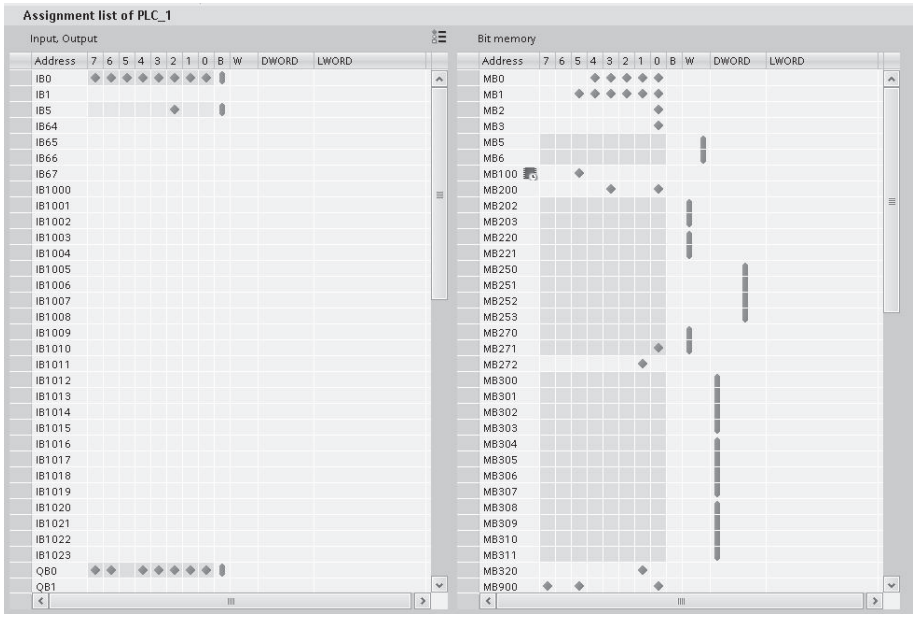

Figure 2. Assignment list for the traffic light system with a security block.

3.2 Anomaly Detection Methodology

In order to detect anomalies, it is necessary to capture adequate amounts of useful data. Since there is no prior information about the programmable logic controller logic, it is necessary to determine which memory addresses are referenced by the controller logic.

If the source code of the traffic light program is available, the code can be loaded into TIA [15], an integrated development environment for Siemens programmable logic controllers, which creates an assignment list that contains all the referenced memory addresses. Figure 2 shows the assignment list for the traffic light program. Because the security block was configured to use memory block addresses MB200 to MB900, these addresses were deemed to be irrelevant and were, therefore, ignored in the data capture.

If the source code is not available, then it is necessary to capture the contents of memory addresses during normal operation. The memory capture process is repeated for small memory blocks until the contents of all the memory addresses have been captured. Next, the memory addresses whose contents do not change are eliminated based on the assumption that their inactivity implies that they have no impact on programmable logic controller behavior. The assumption is reasonable

for programs that do not flip and restore memory addresses during a cycle, and have no external dependencies. This turned out to be the case for the traffic light simulation program.

Since the source code was available in the experiment, the Siemens TIA integrated development environment was used to identify the memory addresses of interest in the programmable logic controller.

After the memory addresses have been identified, several approaches can be used to capture information about programmable logic controller status. One approach is to use a network sniffer or mirror port in a network device to capture network traffic to and from the programmable logic controller. Another approach is to actively poll memory address values using an external program [19]. Yet another approach is to use a security block to transmit internal programmable logic controller data [1] to a historian.

The approach adopted in this work was to capture and analyze input and output signals and memory values using a security block. One reason is that, in many real-world deployments (as in the case of the traffic control experiment), a programmable logic controller has minimal external network traffic – because it is directly connected to input/output ports, not all signals generate network traffic traces during normal operation. In addition, stateful information about programmable logic controller operations may not be transferred to an external device such as a historian for storage. Thus, network traffic captures alone would not provide adequate information about the programmable logic controller.

Since different combinations of memory address values may represent different program states, it is important to ensure that the captured values are consistent within a programmable logic controller execution cycle. However, using an external program (e.g., Snap7 [8]) over a network to query memory address values does not ensure their consistency due to network latency and programmable logic controller operating system delays. In addition, continuously polling multiple memory addresses imposes overhead on a programmable logic controller that may degrade its performance.

These challenges are overcome using a security block to produce a consistent snapshot of memory values in every cycle. Other advantages of the security block over active polling are higher levels of correlation between memory addresses, less lag and missing state data, and accurate timestamp information for forensic analyses.

Since the data transferred from a security block is in the form of a tokenized byte stream, the byte stream has to be converted back to its original data types (e.g., integer and boolean) for input to a machine learning model.

The popular OCSVM outlier detection machine learning model [10] was employed to detect anomalies. The specific OCSVM model used was from Scikit-learn [13], an open-source software package that provides several machine learning libraries written in Python. The formatted input data was provided to the OCSVM model libraries.

The OCSVM parameters were optimized to increase model accuracy before it underwent training and testing. This was achieved by applying a portion of the original dataset to conduct an iterative search for the best parameters.

The following optimized parameter settings were employed:

- **Kernel:** This parameter specifies the non-linear function used by the support vector machine to project the hyperspace to a higher dimension. The optimal rbf kernel setting was used.

- **Degree:** This parameter specifies the degree of the polynomial kernel function. The optimal degree setting of four was used.

- **Coef0:** This parameter is not significant for the rbf kernel. The optimal setting of zero was used.

- **Nu:** This parameter specifies the maximum number of training examples that can be misclassified and the minimum fraction of training examples for the support vector. The optimal setting of 0.001 was used.

The next step involved the generation of anomalous events and data collection. Programs were executed to inject attack traffic into the programmable logic controller to simulate real attacks. Normal and attack data collected by the historian were used as samples.

The following metrics for benchmarking the accuracy of machine learning techniques [11] were used to assess the effectiveness of anomaly detection:

- **Precision:** Precision is defined as the ratio of true positives to the sum of true positives and false positives ($= \frac{TP}{TP+FP}$). It measures the ability of a classifier to not misclassify negative samples as positive samples. The precision ranges from one (best) to zero (worst).

- **Recall:** Recall is defined as the ratio of true positives to the sum of true positives and false negatives ($= \frac{TP}{TP+FN}$). It measures the ability of a classifier to identify all the positive samples. The recall ranges from one (best) to zero (worst).

Table 1. Classification results for various performance metrics.

	Records	**Precision**	**Recall**	**F1 Score**
Training Set	922,162	1.00	0.94	0.97
Testing Set 1	505,041	0.97	0.93	0.94
Testing Set 2	588,573	0.95	0.94	0.94
Testing Set 3	471,207	0.97	0.95	0.96

- **F1 Score:** The F1 score is a weighted average of precision and recall ($= \frac{TP+FN}{TP+FP}$). The higher the score, the better the ability of a classifier to detect negative samples while maintaining a low false positive rate. The F1 score ranges from one (best) to zero (worst).

Table 1 shows the anomaly detection results for the training set and three testing sets. Good results were obtained. The precision for the three testing sets ranges from 0.95 to 0.97; the recall ranges from 0.93 to 0.95; and the F1 score ranges from 0.94 to 0.96.

4. Discussion

In industrial control environments, it is difficult to obtain attack data and little, if any, details are available about the internal logic of programmable logic controllers. Since adequate amounts of normal operational data are available, the solution to detecting anomalies caused by attacks is to employ a machine learning technique create a model of normal behavior and use the trained model to identify anomalous behavior. The experimental results demonstrate that the trained detector was able to recognize normal behavior with a low error rate. Thus, it would be effective as a monitoring mechanism for detecting unknown attacks and unanticipated failures.

The experiments assumed that the training dataset contained only normal scenarios, without any anomalous events. This requires the number of normal scenarios in the dataset to be substantial enough to be distinguishable from anomalous outliers. The experiments revealed that insufficient amounts of training data about normal scenarios yield high false positive rates. Therefore, a large normal dataset must be used during the training phase.

Another observation is that the model parameters have large impacts on the accuracy of detection. A previous study with the simulated traffic light system [21] revealed that the default parameter settings yield modest results. In contrast, the experiments described in this chapter

demonstrate that good results are obtained by using a small dataset in an iterative search for optimal model parameters and then applying the model with the optimized parameters to larger datasets for training and testing.

The logging mechanism implemented by the historian maintains precise timestamps of programmable logic controller memory status. The timestamped information coupled with the trained anomaly detection model can significantly advance forensic investigations. For example, when the anomaly detection model triggers an alert with a concrete timestamp, a forensic investigator can narrow down the time and duration of the incident, and look up and recreate the programmable logic controller memory status and behavior using the data stored by the historian.

However, the proposed approach has some limitations. First, the analysis of memory addresses is not scalable. If large numbers of memory addresses are used by a programmable logic controller, then a filtering mechanism would be required to reduce the number of features considered by the machine learning model. Second, if a programmable logic controller stores its state data on an external device during its execution cycle, this data must be obtained and verified to ensure accurate detection. Finally, unsupervised learning requires a large and rich normal dataset for model training.

5. Conclusions

The lack of genuine attack data and the difficulty in generating simulated attack data render unsupervised learning well-suited to developing anomaly detection systems for programmable logic controllers. The proposed anomaly detection methodology, which employs a one-class support vector machine, accurately detects anomalies without being bound to specific scenarios or requiring detailed knowledge about the control logic. The methodology leverages an additional security block in a programmable logic controller to detect anomalies and employs the historian to store timestamped programmable logic controller state information (i.e., key memory address values) to support anomaly/attack analyses and forensic investigations. Experimental results with a traffic light simulation system employing a Siemens S7-1212C programmable logic controller demonstrate that anomalies are detected with high accuracy.

Future research will focus on implementing increased state awareness based on live programmable logic controller memory analysis to enhance anomaly detection. Efforts will also concentrate on tuning the unsu-

pervised learning methodology to enhance performance metrics such as precision, recall and the F1 score.

References

[1] C. Chan, K. Chow, S. Yiu and K. Yau, Enhancing the security and forensic capabilities of programmable logic controllers, in *Advances in Digital Forensics XIV*, G. Peterson and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 351-367, 2018.

[2] I. Garitano, R. Uribeetxeberria and U. Zurutuza, A review of SCADA anomaly detection systems, *Proceedings of the Sixth International Conference on Soft Computing Models in Industrial and Environmental Applications*, pp. 357–366, 2011.

[3] T. Hergenhahn, libnodave (`sourceforge.net/projects/libnodave`), 2014.

[4] J. Hsu, D. Mudd and Z. Thornton, Project Report – SCADA Anomaly Detection, Department of Electrical and Computer Engineering, Mississippi State University, Mississippi State, Mississippi (`www.ece.uah.edu/~thm0009/icsdatasets/MSU_SCADA_Final_Report.pdf`), 2014.

[5] S. Karnouskos, Stuxnet worm impact on industrial cyber-physical system security, *Proceedings of the Thirty-Seventh Annual Conference of the IEEE Industrial Electronics Society*, pp. 4490–4494, 2011.

[6] J. Klick, S. Lau, D. Marzin, J. Malchow and V. Roth, Internet-facing PLCs as a network backdoor, *Proceedings of the IEEE Conference on Communications and Network Security*, pp. 524–532, 2015.

[7] Langner, A time bomb with fourteen bytes, Dover, Delaware (`www.langner.com/2011/07/a-time-bomb-with-fourteen-bytes`), July 21, 2011.

[8] D. Nardella, Step 7 Open Source Ethernet Communications Suite, Bari, Italy (`snap7.sourceforge.net`), 2016.

[9] S. Nazir, S. Patel and D. Patel, Assessing and augmenting SCADA cyber security: A survey of techniques, *Computers and Security*, vol. 70, pp. 436-454, 2017.

[10] F. Schuster, F. Kopp, A. Paul and H. Konig, Attack and fault detection in process control communications using unsupervised machine learning, *Proceedings of the Sixteenth International Conference on Industrial Informatics*, pp. 433–438, 2018.

[11] Scikit-learn Project, scikitlearn.metrics: Metrics (scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics), 2016.

[12] Scikit-learn Project, Novelty and Outlier Detection (scikit-learn.org/stable/modules/outlier_detection.html#outlier-detection), 2017.

[13] Scikit-learn Project, sklearn.svm.OneClassSVM (scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html), 2017.

[14] Scikit-learn Project, An Introduction to Machine Learning with scikitlearn (scikit-learn.org/stable/tutorial/basic/tutorial.html), 2018.

[15] Siemens, Totally Integrated Automation Portal, Nuremberg, Germany, 2019.

[16] R. Spenneberg, M. Bruggemann and H. Schwartke, PLC-blaster: A worm living solely in the PLC, presented at *Black Hat USA*, 2016.

[17] Wikipedia, One-Class Classification (en.wikipedia.org/wiki/One-class_classification), 2018.

[18] T. Wu and J. Nurse, Exploring the use of PLC debugging tools for digital forensic investigations of SCADA systems, *Journal of Digital Forensics, Security and Law*, vol. 10(4), pp. 79–96, 2015.

[19] K. Yau and K. Chow, PLC forensics based on control program logic change detection, *Journal of Digital Forensics, Security and Law*, vol. 10(4), pp. 59–68, 2015.

[20] K. Yau and K. Chow, Detecting anomalous programmable logic controller events using machine learning, in *Advances in Digital Forensics XIII*, G. Peterson and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 81–94, 2017.

[21] K. Yau, K. Chow, S. Yiu and C. Chan, Detecting anomalous behavior of PLCs using semi-supervised machine learning, *Proceedings of the IEEE Conference on Communications and Network Security*, pp. 580–585, 2017.

[22] E. Yilmaz and S. Gonen, Attack detection/prevention system against cyber attacks on industrial control systems, *Computers and Security*, vol. 77, pp. 94–105, 2018.