Enhancing the Security and Forensic Capabilities of Programmable Logic Controllers

Chun-Fai Chan, Kam-Pui Chow, Siu-Ming Yiu, Ken Yau

Chapter 19

ENHANCING THE SECURITY AND FORENSIC CAPABILITIES OF PROGRAMMABLE LOGIC CONTROLLERS

Chun-Fai Chan, Kam-Pui Chow, Siu-Ming Yiu and Ken Yau

Abstract Industrial control systems are used to monitor and operate critical infrastructures. For decades, the security of industrial control systems was preserved by their use of proprietary hardware and software, and their physical separation from other networks. However, to reduce costs and enhance interconnectivity, modern industrial control systems increasingly use commodity hardware and software, and are connected to vendor and corporate networks, and even the Internet. These trends expose industrial control systems to risks that they were not designed to handle.

This chapter describes a novel approach for enhancing industrial control system security and forensics by adding monitoring and logging mechanisms to programmable logic controllers, key components of industrial control systems. A proof-of-concept implementation is presented using a popular Siemens programmable logic controller. Experiments were conducted to compare the accuracy and performance impact of the proposed method versus the conventional programmable logic controller polling method. The experimental results demonstrate that the new method yields increased anomaly detection coverage and accuracy with only a small performance impact. Additionally, the new method increases the speed of anomaly detection and reduces network overhead, enabling forensic investigations of programmable logic controllers to be conducted more efficiently and effectively.

Keywords: Programmable logic controllers, anomaly detection, forensics

1. Introduction

Industrial control systems are commonly used to manage critical infrastructure assets. In the past, industrial control systems utilized proprietary hardware and software, and were separated from other networks. Thus, the vast majority of attacks required physical assess to the systems or internal network access.

Modern industrial control systems increasingly use commodity hardware, software and networking technologies to reduce costs and enhance connectivity. In fact, most industrial control systems are now connected to vendor and corporate networks, and even the Internet. This makes them highly vulnerable to remote cyber attacks.

Attempts have been made to apply security controls and mechanisms developed for information technology (IT) infrastructures to secure and isolate industrial control systems. However, a limited number of intrusion detection systems and firewalls understand industrial control protocols. Moreover, these protection systems may not be deployable because of the limited computational resources, tight timing constraints and harsh conditions encountered in industrial control environments.

This chapter describes a novel approach for enhancing industrial control system security by adding monitoring and logging mechanisms to a programmable logic controller (PLC). A proof-of-concept implementation is presented using the popular Siemens S7 programmable logic controller with a traffic light control simulation system. Several new attacks targeting the Siemens S7 model have been released recently [3], increasing the urgency of incorporating security and forensic capabilities in the popular programmable logic controller. The experimental results demonstrate that the proposed approach provides good anomaly detection accuracy with limited impact on performance. Additionally, the implementation increases the anomaly detection speed and reduces network overhead, enabling forensic investigations of programmable logic controllers to be conducted more efficiently and effectively.

2. Programmable Logic Controllers

A programmable logic controller is a microprocessor-based system that uses programmable memory to store user-defined control programs. It is designed to operate reliably even in harsh industrial control environments. It has strict real-time constraints that mandate each execution cycle not exceed the predefined execution time. The program in a Siemens programmable logic controller is structured in the form of program blocks. The following blocks were used in the experiments conducted in this research [11]:

- **Organization Block (OB):** This type of block serves as an interface between the S7 operating system and a user program. Each organization block has a block number that indicates when it is called by the operating system. For example, OB1 is essentially the main loop of the program and is called during every execution cycle. OB100 to OB102 are called at CPU start-up to process user-defined initialization procedures prior to calling OB1. Aside from specific organization blocks that are called by the operating system, other organization blocks are executed in sequence.

- **Data Block (DB):** This type of block stores memory addresses that are shared by other program blocks and functions.

Other block structures, such as function code blocks, function blocks and system data blocks, are also components of programmable logic controller programs. However, these structures are omitted because they are not related to the experiments described in this chapter.

A Siemens programmable logic controller has a real-time operating system that controls the sequence of user programs and interrupt handlers to be executed. After the CPU boots up, the operating system calls organization blocks OB100 through OB102 if they are defined. When the programmable logic controller is in the run mode, the operating system first reads the inputs from the connected devices and updates its process image input (PII) memory addresses. Following this, the programmable logic controller calls organization block OB1 as the main entry point of the user program. If an interrupt occurs during the cycle, the organization block OB1 pauses and transfers execution flow to a predefined organization block that handles the interrupt. After all the user-defined organization blocks are executed, the operating system reads the process image output (PIQ) memory addresses and send the corresponding signals to the connected devices. Figure 1 summarizes the program execution cycle of a Siemens programmable logic controller.

3. Related Work

The intensity of research in industrial control system security and forensics has increased significantly since the Stuxnet malware was discovered in 2010. Stuxnet targeted Siemens industrial control software at Iran's uranium hexafluoride centrifuge facility in Natanz, specifically, the Siemens Total Integrated Automation (TIA) Step 7 software running on Windows. The malware was able to alter the logic flow of the industrial control system by modifying its memory and hiding the changes from equipment operators [5].

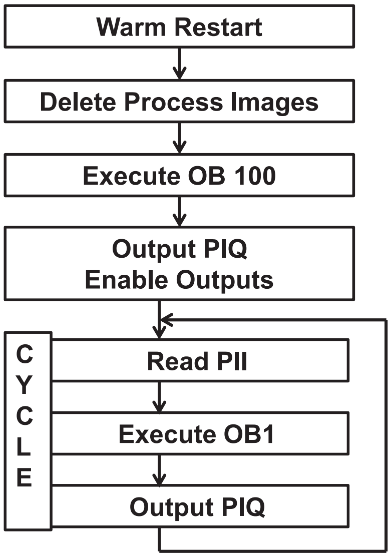

Figure 1. Program execution cycle (from [17]).

Spenneberg et al. [19] and Klick et al. [13] have demonstrated the feasibility of programmable logic controller worms and backdoors. Their malware code can infect other programmable logic controllers without leveraging personal computers and proprietary software.

The first step in programmable logic controller anomaly detection and forensics is data acquisition. Most researchers have focused on extracting data via network traffic analysis, computer and network device imaging, and remote programmable logic controller memory variable monitoring. Spyridopoulos et al. [20] and van Vliet et al. [21] have discussed the challenges of conducting forensic investigations of industrial control systems and programmable logic controllers due to the lack of security-oriented logging mechanisms that could provide valuable evidence of attacks.

Chan and Chow [4] have proposed a forensic analysis methodology that collects programmable logic controller diagnostic buffer data and metadata used in the Siemens Total Integrated Automation software [18]. However, their approach is focused on static data collection and the limited size of the diagnostic buffer may not provide complete information about an incident.

Garitano et al. [7] have reviewed various anomaly detection methodologies for industrial control systems. They argue that a network intrusion detection system may not be able to detect attacks on an industrial control system in an effective manner. Moreover, due to the limited computing resources provided by control system components, host-based intrusion detection is limited at best. Hadziosmanovic et al. [8] also argue that conventional security countermeasures such as network intrusion detection can barely identify process-related attacks involving execution code injection. This shortcoming is mainly due to the fact that network traffic generated by process-related attacks is similar to regular traffic, thereby evading common pattern-matching-based and statistics-based intrusion detection.

Wu and Nurse [22] have demonstrated that memory address monitoring can help determine if a programmable logic controller is running normally or is under attack. They also evaluated the use of a programmable logic controller as a forensic tool that continuously polls memory variables of industrial control devices and logs their values. However, remote monitoring via active polling imposes network overhead that cannot be handled in many industrial control environments.

Yau and Chow [23] have proposed two solutions for programmable logic controller anomaly detection. The first solution uses a control program logic change detector that engages a set of rules to detect and record anomalous memory address value changes. The second solution captures the values of memory addresses of interest in a log file; machine learning techniques are used to analyze the logged data and identify anomalous programmable logic controller operations [24].

Lerner et al. [15] have developed a trusted hardware system with a software simulator that predicts the output signals of a production programmable logic controller. If inconsistencies are observed between the predicted and actual outputs of the programmable logic controller, the trusted system takes over and produces the correct output signals. However, this solution is expensive due to the additional hardware and the need to reconfigure the industrial control system infrastructure.

Researchers often assume that industrial control environments are isolated and have strong physical security protection mechanisms, implying that common attack paths would start from computers in the internal network and terminate at programmable logic controllers [2]. However, many real-world infrastructures do not have such architectures. For example, it is relatively common for remote sites to have multiple programmable logic controllers that are connected to a master station over a wireless network. Moreover, remote sites are often constrained by limited space, high humidity and extreme temperatures. Therefore, standard information technology solutions such as firewalls, intrusion detection systems and network monitoring devices may be impractical, perhaps even impossible, to implement.

4. Security and Forensic Challenges

Conventional security tools and techniques are not directly applicable to programmable logic controllers because of their unique architectures, custom operating systems, and limited memory and processing power. Ahmed et al. [1] and Folkerth [6] identify the following common security and forensic challenges facing programmable logic controllers:

- **Lack of Documentation:** Inadequate low-level documentation for programmable logic controllers adversely impacts security assessments and forensic investigations.

- **Lack of Tools:** Inadequate security and forensic tools (e.g., logging systems) are available for programmable logic controllers. Many programmable logic controllers use non-routable and proprietary serial protocols for communications that are not supported by modern security and forensic tools.

- **Availability/Always-On Requirement:** Programmable logic controller availability is always the top priority in an industrial control environment. It may be infeasible to shut down a programmable logic controller to conduct an incident response or forensic investigation.

- **Limited Computing Resources:** Limited memory and processor power make it difficult to add security functionality such as encryption, authentication and intrusion/anomaly detection. Incorporating logging functionality is also difficult; this significantly hinders security analyses and forensic investigations of incidents.

- **Large Distributed Architecture:** An infrastructure often has a large number of control devices that are dispersed over a large geographical area. Acquiring timely data for implementing security functionality may not be possible due to limited bandwidth and delays.

- **Lack of Expertise:** Limited expertise exists with regard to performing programmable logic controller forensics.

5. Proposed Solution

The proposed solution for addressing the security and forensic challenges involves the implementation of active programmable logic controller monitoring. This is accomplished by incorporating a security block in a programmable logic controller. The security block monitors

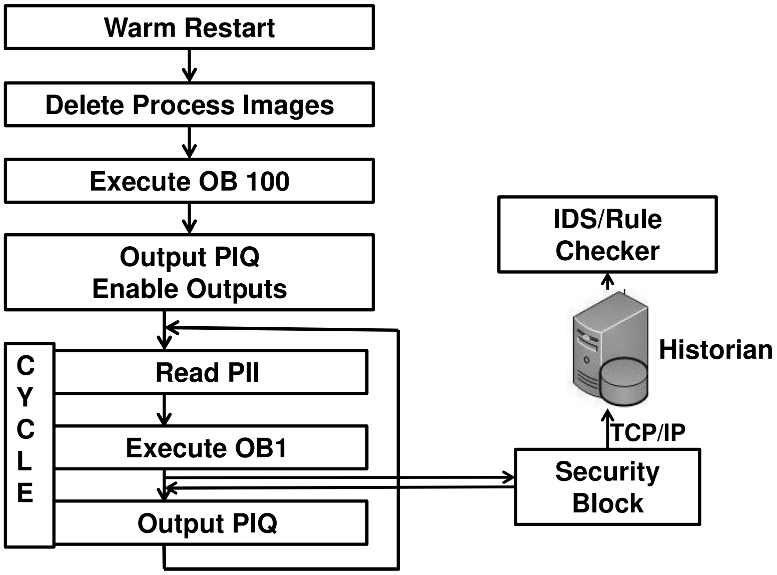

Figure 2. Program execution cycle with a security block.

critical information and system integrity, and timestamps and logs events with minimal impact on the program execution cycle of the programmable logic controller.

Figure 2 shows the new program execution cycle with the added security block. The security block is executed at the end of each scan cycle and/or before any external network communications.

Algorithm 1 specifies the security block logic. The security block implements the following three actions:

- **Critical Information Monitoring:** Lines 8–13 of the algorithm monitor the critical variables before sending information to the historian. All the digital inputs and outputs (total sixteen memory addresses) were selected as critical variables to monitor. Other memory addresses and data block variables may also be monitored.

- **System Integrity Monitoring:** Lines 14–19 of the algorithm monitor system integrity by tracking the number of installed data blocks. Monitoring the number of data blocks is necessary because modern malware is capable of spreading to programmable logic controllers without the use of conventional computer workstations or laptops. The proof-of-concept malware implementations described in [3, 19] require additional data blocks for network communications and malware program code storage. By checking the

Algorithm 1: Security block logic.

```
 1 Initialize:
 2 for each critical-value do
 3     Store critical-value in data block of security block
 4 end
 5 number-data-blocks $\leftarrow$ 0
 6 send-alert $\leftarrow$ false
 7 During each execution cycle:
 8 for each critical-value do
 9     if current-critical-value $\neq$ security-block-value then
10         security-block-value $\leftarrow$ current-critical-value
11         send-alert $\leftarrow$ true
12     end
13 end
14 for each data-block-address do
15     if first-byte of the data-block is readable then
16         number-data-blocks $\leftarrow$ number-data-blocks + 1
17         send-alert $\leftarrow$ true
18     end
19 end
20 if send-alert = true then
21     Query current-system-time
22     Format current-system-time to byte stream
23     Invoke network system call to send current-system-time
24     and every security-block-value to historian
25 end
```

number of data blocks, it is possible to detect unauthorized modifications to the programmable logic controller structure.

- **Event Logging:** Lines 20–24 of the algorithm query the system for the time and format the timestamp to a byte stream. The monitored information is timestamped and sent via TCP/IP to a historian for long-term storage.

6. Experimental Methodology and Results

This section describes the experiments undertaken to evaluate the proposed security and forensic solution for programmable logic controllers.

6.1 Experimental Setup

The experiments employed a Siemens S7-1200 model programmable logic controller with the 1214C CPU and firmware version v4.0. The

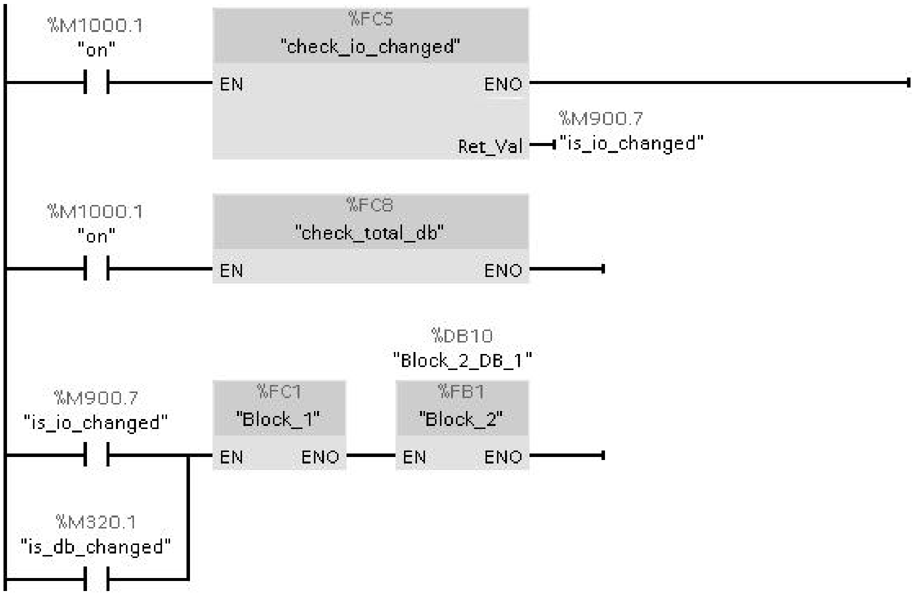

Figure 3. Ladder logic of the security block.

programmable logic controller was loaded with a traffic light control simulation program.

The security block in the programmable logic controller was implemented using the structured control language (SCL) and ladder diagram (ladder logic) language. The Total Integrated Automation Portal v13 was used to add the security block (organization block OB32767), the last organization block in the programmable logic controller. Figure 3 shows the ladder logic of the security block. The security block only executes after all the other organization blocks have executed.

A computer workstation with Netcat as a TCP server was deployed as the historian. The received data was multiplexed to a rule checker program written in Python. The rule checker parsed the data and applied the detection rules suggested by Yau and Chow [23].

Another computer workstation was used for performance benchmarking. It employed the conventional approach to poll the relevant memory address values of the programmable logic controller using the libnodave open source library [10]. The libnodave client, written in C, connected to the programmable logic controller via ISO-TSAP-over-TCP/IP. All the collected information was timestamped according to the clock on the computer workstation.

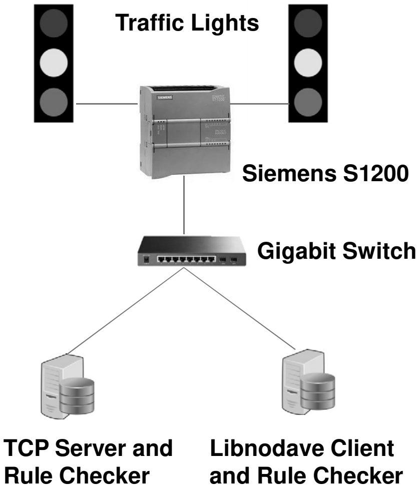
Traffic Lights
Siemens S1200
Gigabit Switch
TCP Server and Rule Checker
Libnodave Client and Rule Checker

Figure 4. Experimental setup.

Figure 4 shows the experimental setup. The programmable logic controller and the two computer workstations were connected using a gigabit switch.

6.2 Attacks

The following attacks were executed to compare the detection accuracy of the security block implementation versus the conventional monitoring method:

- **Memory Read/Write Logic Attack:** This attack attempted to read and alter the values of memory variables in the running programmable logic controller.

  A conventional method for monitoring the attack is to poll the memory address values and compare them with the expected values. The libnodave client was employed to implement the conventional method as a benchmark. It sent about 180 packets per second to query the current value of a memory variable. The current value was compared against the previous value to determine if there was an unexpected change to the memory variable. The conventional method was able to detect a change to a single variable within 100 ms. In contrast, the security block could detect changes to multiple variables within just 10 ms and it sent just one packet after a change was detected.

- **Malware Worm Attack:** This attack simulated the injection of a programmable logic controller worm as in [3, 19] by pushing one or more data blocks and an organization block to the programmable logic controller.

  In order to detect if an additional data block had been added to the programmable logic controller, the total number of installed data blocks was counted and compared against the expected number of data blocks. The maximum number of data blocks available in the test device is 60,000. Unfortunately, the S7-1200 device does not provide an API or a system function call to identify the number of installed data blocks as in the case of the earlier S7-300 model. As a workaround, the first byte of every possible data block was examined and the number of times a value could be successfully read was counted.

  In the conventional monitoring method, libnodave was used to send a probe message to query for the first byte in a data block. If the data could be read, then it was possible to infer that the specific data block was installed. This process was repeated for all possible data blocks from DB0 to DB59999. Finally, the total number of installed blocks was compared against the expected number of blocks to determine if there was a change in the number of data blocks installed.

  In the case of the security block method, a problem was encountered because examining all the data blocks in one execution cycle exceeded the hard execution time constraint of 150 ms. Therefore, the workaround checked a fixed number of data blocks in a single execution cycle and the remaining checks were performed in subsequent cycles until all 60,000 data blocks were checked. This implementation satisfies the execution time constraint and also enables the performance impact imposed by the security block to be controlled.

- **Time Bomb Attack:** This attack inserted malware that triggered at a particular time. Upon being triggered, the malware sent an output signal and reverted to its normal state during the next scan cycle. This attack was difficult to detect because the anomalous output signal was active for a very short period of time. Moreover, the program was not changed in any other way. No additional network traffic was generated because the output device was not connected to the monitoring network (so network sniffing would not capture any anomalous traffic).

In the conventional and proposed methods, a particular output address was monitored as in the case of the memory read/write logic attack. The time bomb was set to trigger every five seconds and change an output address value for exactly one cycle. In the next cycle, the time bomb changed the output address value back to its original value.

The monitoring was conducted for five hours and the number of alerts triggered by each method was recorded. The conventional monitoring method using libnodave triggered around 850 alerts whereas the security block method triggered 3,600 alerts. Thus, the sensitivity of the security block method is four times better than conventional active polling using libnodave.

6.3 Performance Impact

The performance impact of the security block was tested using the Siemens Total Integrated Automation Professional software v13 SP1. The software was executed on a computer workstation that was connected to the programmable logic controller via TCP/IP using a Profinet interface. The online and diagnostics feature of the software was employed to capture the shortest, current and longest cycle times for the following test cases:

- **Test Case 1:** This test case involved a control benchmark without the security block.
- **Test Case 2:** This test case employed the security block, but without a status change.
- **Test Case 3:** This test case involved frequent manual status changes by modifying the input signal rapidly (more than fifteen status changes per second).
- **Test Case 4:** This (stress) test case changed the input signal every cycle (approximately 150 updates per second).
- **Test Case 5:** This test case employed an optimized security block that did not perform timestamp and I/O string conversions.

Table 1 shows the performance impacts introduced by the security block for the five test cases. The security block added about 5 ms performance impact to the existing program under normal usage (Test Cases 2 and 3) and about 10 ms under stress testing (Test Case 4). The experimental results are consistent with the performance benchmarking performed by Klick et al. [13]. Most of the cycle time (80%) used by the

Table 1. Performance impacts introduced by the security block.

Test Case	Shortest Cycle Time (ms)	Current Cycle Time (ms)	Longest Cycle Time (ms)
1	1	1	5
2	3	6	7
3	3	6	7
4	3	9	12
5	3	6	8

security block was employed to check for new data blocks. The performance impact can be optimized by adjusting the number of data blocks checked per cycle.

7. Discussion

The incorporation of a security block in the programmable logic controller offers several advantages over conventional passive or external monitoring:

- The security block provides access to internal programmable logic controller information, functions and addresses; some of this information is not accessible using a network protocol. For example, there is no easy way to access the internal system clock at a certain point in execution, or intermediate execution states or interrupts during an execution cycle.

- The performance impact of the security block can be adjusted. This is because the number of steps required can be computed and the time required can be measured like other programmable logic controller functions; the steps can then be distributed over multiple execution cycles to adjust the performance impact. In contrast, the conventional polling approach is handled by the programmable logic controller operating system with undocumented time requirements.

- The security block method has a higher detection accuracy rate than the conventional polling method. For example, in the case of the time bomb attack, some memory addresses changes were missed by the conventional method. The security block method provides more complete data, which facilitates the implementation of additional anomaly detection functionality.

- Events identified by the security block method can be crafted like syslog messages and matched against simple regular expressions. These messages can be parsed by intrusion detection or situational awareness and event management systems to trigger alerts.
- Compared with the conventional memory variable polling method, the security block method only sends traffic when certain conditions are met, avoiding the network overhead incurred by continuous monitoring.
- The security block method does not require network traffic captures and network traffic analysis. Neither special network equipment nor the reconfiguration of a mirror port to redirect network traffic for sniffing are required.
- The security block method is well suited to programmable logic controllers that are deployed individually in remote locations. Accurate timestamps provided by the programmable logic controllers facilitate the creation of event timelines in forensic investigations.
- The security block method facilitates the implementation of security and forensic functionality. For example, the security block could be enhanced to configure the reading of memory address values on the fly and to report when changes are detected.

However, the security block method has certain limitations. Cases exist were this solution may not be appropriate or may require careful consideration before being applied. One example is when a programmable logic controller must operate continuously and cannot be taken down to introduce the security block. Other examples are when the programmable logic controller program is developed by a third-party and modifications to the main program block are not allowed. Finally, selecting the information to be monitored requires domain-specific expertise; too much monitoring can negatively impact performance.

Another drawback of the security block method is that additional information is sent to the network. This information is exposed to passive network sniffing attacks as in the case of the conventional method that polls memory variables.

This research has focused exclusively on the Siemens S7-1200 programmable logic controller. Other programmable logic controller models have different instruction sets. For example, the S7-300 series uses TEST_DB to detect the creation of a new data block; this may, in fact, improve the performance impact of the security block.

The current security block method is unable to combat malware that injects code in organization block OB1 without using a data block [14].

Fortunately, the programmable logic controller used in this research employs the S7-Comm Plus v3.0 protocol, which is equipped with additional digest checking. Thus, it appears that the programmable logic controller is not vulnerable to the code injection attack, unless the latest protocol has been hacked [9].

8. Conclusions

This chapter has presented a novel security block method for detecting memory variable changes that may affect programmable logic controller integrity, and for supporting incident response and forensic investigations. Experiments that evaluated the proposed method against the conventional programmable logic controller polling method reveal that it provides increased accuracy and reduced (and adjustable) performance impact. Additionally, the proposed method introduces minimal network traffic overhead and detects attacks that are rarely identified by other approaches. However, although the method offers promising anomaly detection and forensic functionality, implementing it in a production environment requires careful study to prevent disruptions to the industrial control system and the infrastructure assets it operates.

Future research will focus on applying the proposed method to other programmable logic controller models. Research will also concentrate on detecting program code changes caused by worm and code injection attacks. Furthermore, machine learning techniques will be employed to enhance the coverage and accuracy of attack detection.

References

[1] I. Ahmed, S. Obermeier, M. Naedele and G. Richard, SCADA systems: Challenges for forensic investigators, *IEEE Computer*, vol. 45(12), pp. 44–51, 2012.

[2] B. Akyol, H. Kirkham, S. Clements and M. Hadley, A Survey of Wireless Communications for the Electric Power System, Technical Report PNNL-19084, Pacific Northwest National Laboratory, Richland, Washington, 2010.

[3] D. Beresford, Exploiting Siemens Simatic S7 PLCs, presented at *Black Hat USA*, 2011.

[4] R. Chan and K. Chow, Forensic analysis of a Siemens programmable logic controller, in *Critical Infrastructure Protection X*, M. Rice and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 117–130, 2016.

[5] N. Falliere, L. O'Murchu and E. Chien, W32.Stuxnet Dossier, Version 1.4, Symantec, Mountain View, California, 2011.

[6] L. Folkerth, Forensic Analysis of Industrial Control Systems, InfoSec Reading Room, SANS Institute, Bethesda, Maryland, 2015.

[7] 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.

[8] D. Hadziosmanovic, D. Bolzoni and P. Hartel, A log mining approach for process monitoring in SCADA, *International Journal of Information Security*, vol. 11(4), pp. 231–251, 2012.

[9] C. Hao, New PLC worm virus and its countermeasures (in Chinese), NSFOCUS, Santa Clara, California (`blog.nsfocus.net/worm-plc-strategy`), September 12, 2016.

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

[11] C. Jones, *STEP 7 Programming Made Easy in LAD, FBD and STL: A Practical Guide to Programming S7300/S7-400 Programmable Logic Controllers*, Patrick-Turner Publishing, Atlanta, Georgia, 2013.

[12] 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.

[13] 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.

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

[15] L. Lerner, Z. Franklin, W. Baumann and C. Patterson, Application-level autonomic hardware to predict and preempt software attacks on industrial control systems, *Proceedings of the Forty-Fourth Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 136–147, 2014.

[16] Mice Engineering, Yuen Long Sewage Treatment Works System, Hong Kong, China (`www.miceeng.com/eng/project_id=yuenlong.htm`), 2008.

[17] Siemens, SITRAIN – Training for Industry Worldwide, Nuremberg, Germany (`www.sitrain-learning.siemens.com`), 2018.

[18] Siemens, Totally Integrated Automation Portal, Nuremberg, Germany (`www.siemens.com/global/en/home/products/automation/industry-software/automation-software/tia-portal.html`), 2018.

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

[20] T. Spyridopoulos, T. Tryfonas and J. May, Incident analysis and digital forensics of SCADA and industrial control systems, *Proceedings of the Eighth IET International System Safety Conference Incorporating the Cyber Security Conference*, 2013.

[21] P. van Vliet, M. Kechadi and N. Le-Khac, Forensics in industrial control systems: A case study, *Proceedings of the Workshop on the Security of Cyber-Physical Systems; Conference on Cybersecurity of Industrial Control Systems*, pp. 147–156, 2016.

[22] 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.

[23] 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.

[24] 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, Heidelberg, Germany, pp. 81–94, 2017.