A Forensic Logging System for Siemens Programmable Logic Controllers

Ken Yau, Kam-Pui Chow, Siu-Ming Yiu

Chapter 18

A FORENSIC LOGGING SYSTEM FOR SIEMENS PROGRAMMABLE LOGIC CONTROLLERS

Ken Yau, Kam-Pui Chow and Siu-Ming Yiu

Abstract Critical infrastructure assets are monitored and managed by industrial control systems. In recent years, these systems have evolved to adopt common networking standards that expose them to cyber attacks. Since programmable logic controllers are core components of industrial control systems, forensic examinations of these devices are vital during responses to security incidents. However, programmable logic controller forensics is a challenging task because of the lack of effective logging systems.

This chapter describes the design and implementation of a novel programmable logic controller logging system. Several tools are available for generating programmable logic controller audit logs; these tools monitor and record the values of programmable logic controller memory variables for diagnostic purposes. However, the logged information is inadequate for forensic investigations. To address this limitation, the logging system extracts data from Siemens S7 communications protocol traffic for forensic purposes. The extracted data is saved in an audit log file in an easy-to-read format that enables a forensic investigator to efficiently examine the activity of a programmable logic controller.

Keywords: Programmable logic controllers, forensics, logging system

1. Introduction

Critical infrastructure assets such as electricity generation plants, transportation systems and manufacturing facilities are monitored and controlled by industrial control systems [4]. Historically, industrial control systems were operated as isolated, proprietary systems with no external network connections. Thus, these systems and the critical infrastructure assets they managed were primarily exposed to internal as op-

posed to external threats. However, for reasons of convenience, modern industrial control systems use TCP/IP and wireless protocols that connect to corporate networks, vendor networks and even the Internet [12]. Additionally, industrial control systems increasingly use common embedded system platforms and commercial off-the-shelf software [4]. As a result, modern industrial control systems and the infrastructures they manage are exposed to numerous external threats, including over the Internet. Digital forensics is an important component of incident investigations involving industrial control systems. The forensic investigations provide insights to the root causes of incidents, enable the identification and prosecution of attackers, and help design appropriate security controls.

Programmable logic controllers (PLCs), which are used to automate industrial systems and processes, are important components of industrial control systems. Modern programmable logic controllers have evolved to utilize common networking standards such as IEEE 802.3 Ethernet and IEEE 802.11 Wi-Fi [1]. As a result, communicating with a programmable logic controller is similar to communicating with a commodity computer. In addition, communications suites such as libnodave and Snap7 for interfacing and exchanging data with Siemens S7 programmable logic controllers are readily available for download on the Internet. The libnodave library provides the functions needed to connect to and exchange data with Siemens S7 300/400 programmable logic controllers (it partially supports Siemens S7 1200/1500 programmable logic controllers) [5]. Snap 7 is an open source, 32/64 bit, multi-platform Ethernet communications suite for interfacing natively with Siemens S7 programmable logic controllers [8].

The popular Shodan search engine has discovered numerous industrial control systems around the world that can be accessed directly over the Internet [6]. Such a tool enables an attacker to identify a vulnerable programmable logic controller, following which the attacker could directly manipulate its logic code over the Internet. The attacker can then leverage the programmable logic controller to reach other control and network devices [6]. The likelihood of such attacks on industrial control systems makes digital forensic readiness of programmable logic controllers an important part of the security posture of critical infrastructure owners and operators.

A key challenge is that the proprietary architectures, operating systems, filesystems and data formats of programmable logic controllers make it difficult to apply traditional digital forensic tools and techniques in investigations of industrial control system incidents. Additionally,

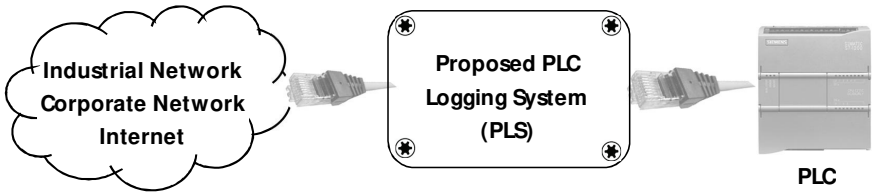

Figure 1. Programmable logic controller logging system.

programmable logic controllers operate vital industrial processes and systems that cannot be stopped for data collection and examination.

Another key challenge is inadequate logging. Several tools have been developed for generating audit logs for programmable logic controllers, but the logged information is often insufficient for forensic investigations. In general, these tools (e.g., PLC Logger [9]) monitor and capture the values of programmable logic controller memory variables, and also access programmable logic controller memory regions to record changes of memory values along with timestamps for diagnostic purposes such as tracing faults in machinery and improving system efficiency [15]. However, they do not capture crucial forensic information such as the IP address of the device that connected to a programmable logic controller, the commands sent to the programmable logic controller and the duration of the connection to the programmable logic controller.

This chapter describes a novel programmable logic controller logging system that captures information required for digital forensic investigations. Figure 1 shows a schematic diagram of the programmable logic controller logging system. The logging system is a lightweight computer installed with a network packet analyzer that captures network traffic between a programmable logic controller and other network devices. The logging system dissects network packets and extracts potential forensic information, which it logs in the form of timestamped records. The log provides documentary evidence of the sequence of activities related to commands and data transmitted between the programmable logic controller and other network devices.

A case study involving a Siemens Simatic S7-1212C programmable logic controller is presented. The decision to focus on a Siemens Simatic S7 programmable logic controller was motivated by their widespread use around the world [1] and the fact that they were targeted successfully by the powerful and insidious Stuxnet malware. The case study uses the Siemens programmable logic controller to create two simulated control systems, a traffic light control system and a liquid mixing control system. The case study demonstrates that the analysis of packet details in

the log file based on the characteristics of S7 communications protocol yields valuable information about an attack, including the attacker's IP address, the specific actions undertaken by the attacker and the timeline.

2. Related Work

In the aftermath of the Stuxnet attack, researchers have significantly increased their efforts to discover and mitigate the vulnerabilities existing in programmable logic controllers. However, limited research has been conducted in the area of programmable logic controller forensics. An example is the work of Chan et al. [2], which focuses on the logging mechanisms of a Siemens programmable logic controller, specifically the Siemens Total Integrated Automation Portal V13 (Siemens TIA Portal). Chan and colleagues demonstrated that the Siemens logging system provides detailed information about event activities for forensic investigations. However, the system only works under two conditions. First, the incidents must be created by the workstation that runs Siemens TIA Portal. Second, the workstation with Siemens TIA Portal must not be compromised; otherwise, the logging system cannot be trusted.

Wu and Nance [15] have shown that attacks on programmable logic controllers can be determined by monitoring the memory addresses of user control programs. In particular, they identified the memory addresses used by program code, and monitored and logged the memory values to capture normal programmable logic controller behavior. The logged behavior was used to determine whether the programmable logic controller was running normally or was under attack.

Yau et al. [16] have proposed forensic solutions for programmable logic controllers. One solution involves control program logic change detection that employs user-defined rules to detect and record anomalous programmable logic controller operations. Another solution captures the values of relevant memory addresses used by a programmable logic controller in a log file [2, 13]. Machine learning techniques were applied to the logged file to identify anomalous programmable logic controller behavior.

Wu and Nance [15] and Yau et al. [16] have demonstrated that it is possible to detect anomalous programmable logic controller behavior. However, they do not capture forensic information such as the IP address of a device that connected to a programmable logic controller, the commands sent to the programmable logic controller and the duration of the connection to the programmable logic controller. The logging system described in this chapter captures vital information that enable

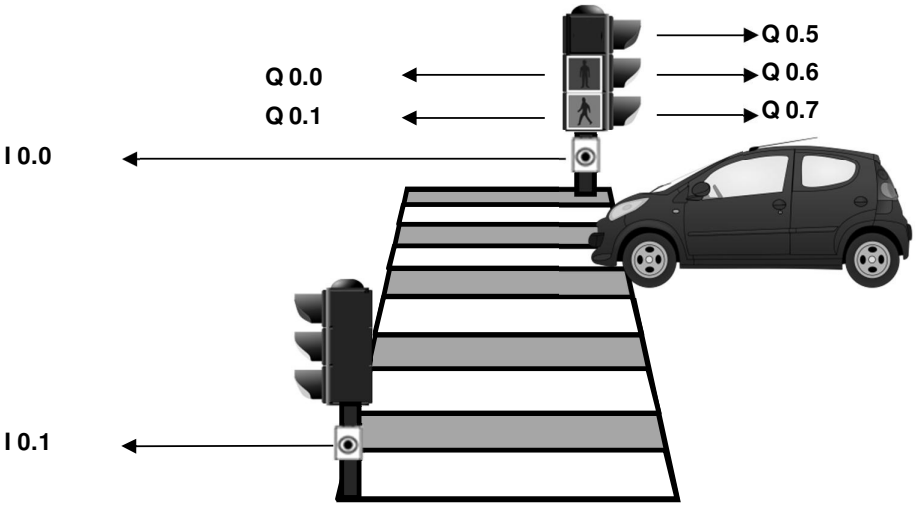

Figure 2. Input/output connections for the traffic light control system.

forensic investigators to reconstruct events and identify anomalous programmable logic controller behavior.

3. PLC Architecture and Programming

A programmable logic controller is a solid state industrial control computer with a central processor unit (CPU), memory, input/output interface and programming functionality. It can be programmed to implement functions such as control logic, sequencing, timing, arithmetic data manipulations and counting in order to monitor and control machines and processes. It accepts data and status information from devices such as switches and temperature sensors, and executes a control program stored in its memory to provide appropriate commands to devices such as valves, lights and motors [3].

Two simulated control systems, a traffic light control system [10] and a liquid mixing control system [11], were set up to demonstrate the proposed programmable logic controller logging system.

The traffic light control system controls vehicular and pedestrian traffic at an intersection. In order to simulate the hardware configuration of the traffic light control system, the Siemens Simatic S7 programmable logic controller inputs I0.0 and I0.1 were connected to switches and the programmable logic controller outputs Q0.0, Q0.1, Q0.5, Q0.6 and Q0.7 were connected to LEDs (Figure 2).

The liquid mixing control system mixes two ingredients, such different colored paints. Two pipes at the top of the mixing tank supply the

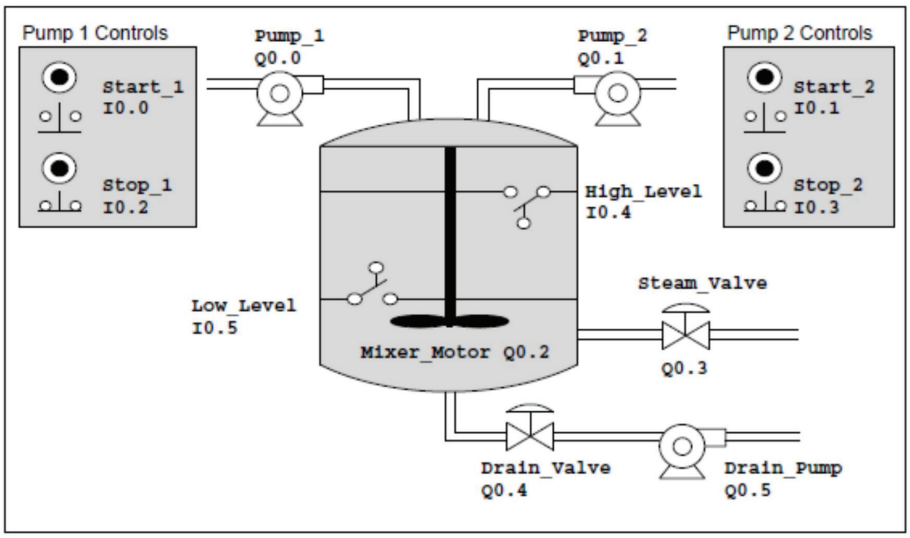

Figure 3. Input/output connections for the liquid mixing control system [11].

two ingredients. A single pipe at the bottom of the tank drains the mixture. In order to simulate the hardware configuration of the system, the Siemens Simatic S7 programmable logic controller inputs were connected to switches for the pumps and liquid level sensors. The outputs were connected to LEDs corresponding to the motor, steam/drain valves and pumps (Figure 3).

Table 1 presents the program instructions (inputs, outputs and memory bits) used by the Siemens Simatic S7 control programs for the traffic light and liquid mixing control systems.

4. Proposed Logging System

The proposed programmable logic controller logging system is implemented as a transparent proxy between an Ethernet network and the programmable logic controller. The transparency ensures that the existing network configurations are maintained. The logging system forwards all traffic except for S7 communications traffic [7].

The logging system has two processes. The first process is initiated when the logging system detects a connection request to the programmable logic controller on TCP port 102. The process captures the communications and filters potential forensic information such as IP addresses, commands and timestamps. The second process then translates and stores the information in an audit log file that is easily read and understood for forensic investigators. Table 2 shows a sample log file.

Table 1. Program instructions for the traffic light and liquid mixing control systems.

Digital Input Address	**Traffic Light Control System**	**Liquid Mixing Control System**
I0.0	Switch on right-hand side of street	Start switch for paint ingredient 1
I0.1	Switch on left-hand side of street	Start switch for paint ingredient 2
I0.2	N/A	Stop switch for paint ingredient 1
I0.3	N/A	Stop switch for paint ingredient 2
I0.4	N/A	Limit switch for maximum level
I0.5	N/A	Limit switch for minimum level
Digital Output Address	**Traffic Light Control System**	**Liquid Mixing Control System**
Q0.0	Pedestrian red light	Pump for paint ingredient 1
Q0.1	Pedestrian green light	Pump for paint ingredient 2
Q0.2	N/A	Motor for mixing paint ingredients
Q0.3	N/A	Steam for heating mixture in tank
Q0.4	N/A	Valve for draining mixture out of tank
Q0.5	Vehicle red light	Pump for draining mixture out of tank
Q0.6	Vehicle yellow light	N/A
Q0.7	Vehicle green light	N/A
Memory Bit	**Traffic Light Control System**	**Liquid Mixing Control System**
M0.0	Memory bit for switching signal after green light request from pedestrian	Memory bit for high level reached

The logging system must be trusted because it is used for evidence collection. It is vital that the system is not hacked and the data log is not altered. Therefore, the logging system should be made hacker-proof and tamper-resistant to the extent possible.

5. S7 Communications Protocol

The S7 communications protocol (S7comm) is a proprietary protocol used by the Siemens S7-300/400 family of programmable logic controllers [13]. The protocol is also partially supported by the Siemens S7-1200/1500 family of programmable logic controllers [8]. It is used for programmable logic controller programming, data exchange with programmable logic controllers, programmable logic controller data access by SCADA systems and diagnostics [13].

Table 2. Log file structure.

Date/Time	Source IP Address	Protocol	PLC Command	PLC Memory Value Change
01 Jan 2017 10:05pm	192.168.0.10	TCP	Establish connection	N/A
01 Jan 2017 10:10pm	192.168.0.10	S7comm	WRITE	Set Output Q0.7 to TRUE from FALSE
01 Jan 2017 11:00pm	192.168.0.10	S7comm	CPU STOP	N/A
01 Jan 2017 11:30pm	192.168.0.10	TCP	Close connection	N/A

Table 3. S7comm commands.

No.	Command
1	Data Read/Write
2	Cyclic Data Read/Write
3	Directory Information
4	System Information
5	Block Move
6	PLC Control
7	Date and Time
8	Security
9	Programming

The Ethernet implementation of S7comm is based on ISO TCP (RFC 1006), which is block-oriented. Each block is called a protocol data unit (PDU). The S7comm protocol is function-oriented or command-oriented, i.e., each transmission contains a command or a reply to a command. If a command or reply does not fit inside a single protocol data unit, it is split across multiple protocol data units [8].

Table 3 shows the nine categories of S7comm commands. Each command has four components: (i) header; (ii) parameters; (iii) parameter data; and (iv) data block.

The first two components of an S7comm command (header and parameters) are mandatory; the other components are optional. Figure 4 presents the protocol encapsulation structure followed by S7 Telegram, ISO on TCP and TCP/IP.

S7comm data is inserted in the payloads of connection-oriented transport protocol (COTP) packets. As shown in Figure 5, the first byte is always `0x32`, which corresponds to the protocol identifier [13].

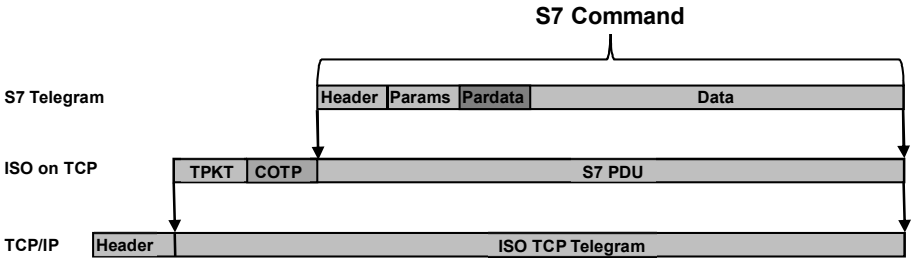

Figure 4. Protocol encapsulation [8].

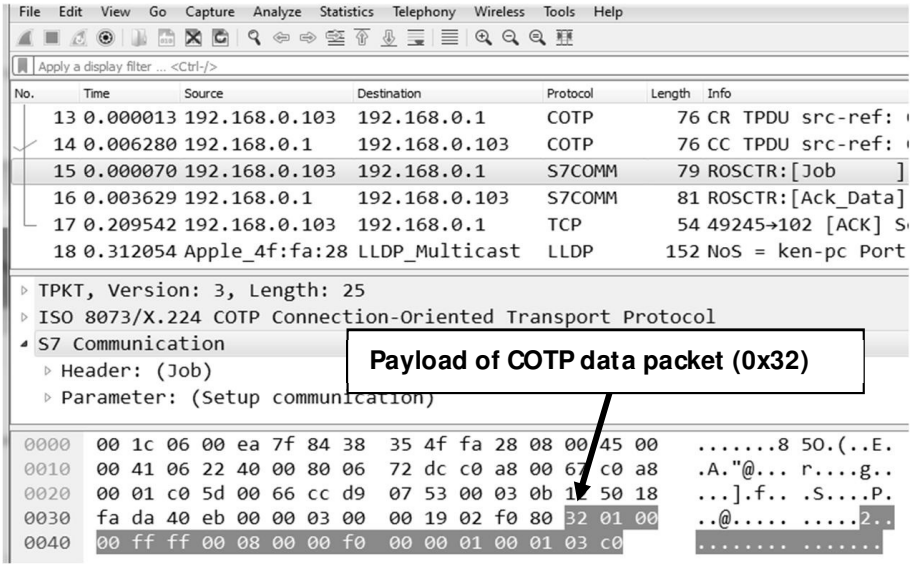

Figure 5. Data packets.

A programmable logic controller connection had to be established in order to collect S7comm protocol traffic. The following steps were involved in establishing a connection to the S7 programmable logic controller [13]:

- A connection was established to the programmable logic controller using its IP address and TCP port 102.
- A connection was established at the ISO layer (COTP connect request). The destination transport services activity point (TSAP) data has two bytes. The first byte of the destination TSAP data specifies the communications type (1 = PG (programming console); 2 = OP (Siemens HMI panel)). The second byte specifies the slot and rack numbers (position of the programmable logic

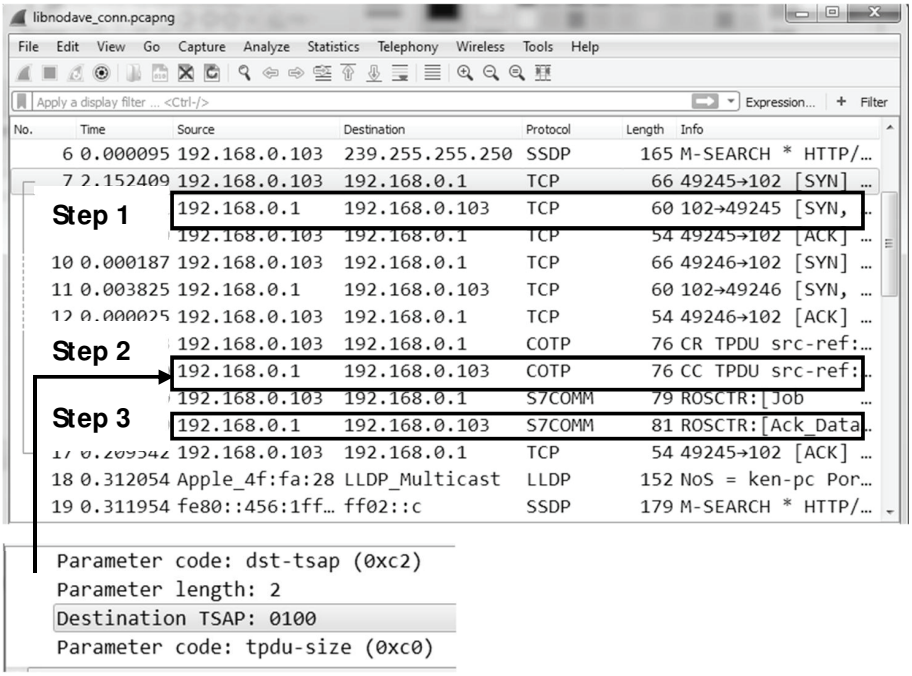

Figure 6. Establishing the S7 programmable logic controller connection.

controller). The slot number is coded in bits 0-4 while the rack number is coded in bits 5-7.

- A connection was established at the S7comm layer (`s7comm.param.func = 0xf0`; setup communications). Details regarding the S7-comm protocol (e.g., protocol data unit size) were negotiated.

Figure 6 shows a Wireshark capture of the S7 programmable logic controller connection steps.

6. Creating Audit Log Records

To demonstrate the data logging methodology, a Siemens Simatic S7 1212C programmable logic controller was used to set up two simulated control systems. A computer was installed with the Snap7 software to create anomalous programmable logic controller behavior. Another computer was installed with Wireshark and the S7comm Wireshark dissector plugin [14] to capture programmable logic controller activities. Figure 7 shows the experimental setup.

The experiments involved two parts. In the first part, the traffic light control system was connected to a wireless access point. In the second

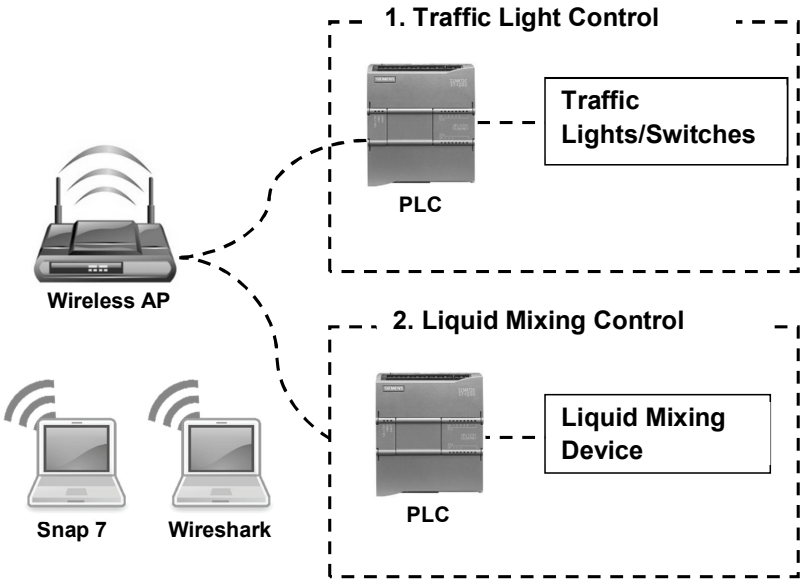

Figure 7. Experimental setup.

part, the traffic light control system was replaced with the liquid mixing control system. Four common programmable logic controller requests were employed: (i) CPU START; (ii) CPU STOP; (iii) READ; and (iv) WRITE.

6.1 Traffic Light Control System

The following steps were involved in sending CPU STOP/START requests to the programmable logic controller:

- Wireshark was started to capture network packets.
- Snap7 established a connection to the programmable logic controller.
- Snap7 sent the CPU STOP request to the programmable logic controller.
- Snap7 sent the CPU START request to the programmable logic controller.
- Snap7 closed its connection to the programmable logic controller.
- Wireshark was stopped and the captured packets were saved in a log file.

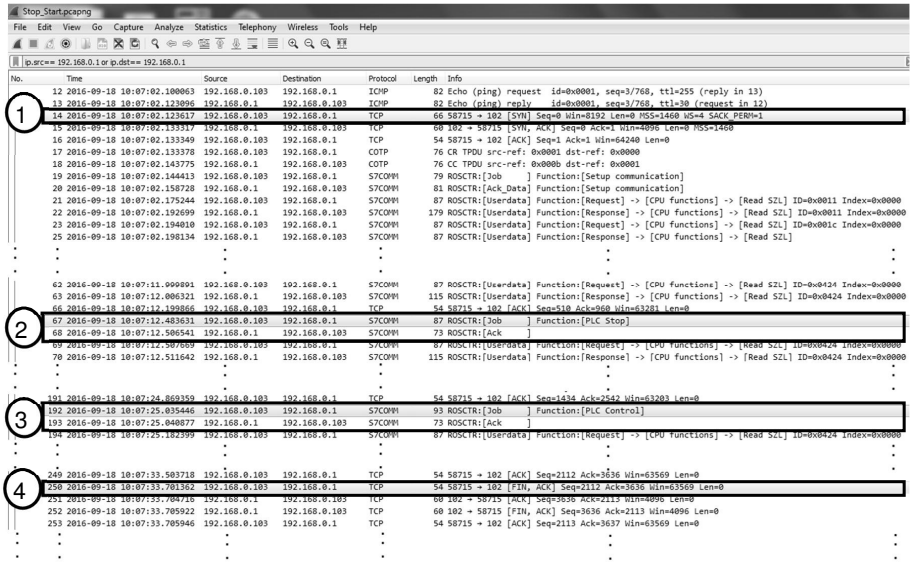

Figure 8. Programmable logic controller STOP and START requests.

- The packets with the programmable logic controller IP address `[192.168.0.1]` were filtered for analysis.

Figure 8 shows the captured log file associated with the programmable logic controller STOP and START requests. Analysis of the log file yields the following reconstruction of programmable logic controller activities:

- `10:07:02am, 18 Sep 2016:` A computer `[192.168.0.103]` established a connection to the programmable logic controller `[192.168.0.1]`.
- `10:07:12am, 18 Sep 2016:` The computer sent a CPU STOP request to the programmable logic controller to stop the traffic light system.
- `10:07:25am, 18 Sep 2016:` The computer sent a CPU START request to the programmable logic controller to re-start the traffic light system.
- `12:07:46pm, 18 Sep 2016:` The computer closed its connection to the programmable logic controller.

Based on the data in Table 1, the memory values of the programmable logic controller outputs at `12:07:46pm, 18 Sep 2016` indicate that all the pedestrian and vehicle lights were turned on. The programmable

logic controller operation appears to be anomalous because an attempt was made to turn on all the traffic lights at the same time.

The following steps were involved in sending READ and WRITE requests to the programmable logic controller:

- Wireshark was started to capture network packets.
- Snap7 established a connection to the programmable logic controller.
- Snap7 read the values of inputs (I0.0 to I0.7), outputs (Q0.0 to Q0.7) and memory bits (M0.0 to M0.7) from the programmable logic controller.
- Snap7 wrote the value 1 to inputs (I0.0 to I07), outputs (O0.0 to 0.7) and memory bits (M0.0 to M0.7) of the programmable logic controller.
- Snap7 closed its connection to the programmable logic controller.
- Wireshark was stopped and the captured packets were saved in a log file.
- The packets with the programmable logic controller IP address `[192.168.0.1]` were filtered for analysis.

Figure 9 shows the captured log file associated with the programmable logic controller READ and WRITE requests. The following controller activities can be reconstructed in time sequence upon analyzing the log file:

- `12:07:28pm, 18 Sep 2016:` A computer `[192.168.0.103]` established a connection to the programmable logic controller `[192.168.0.1]`.
- `12:07:31pm, 18 Sep 2016:` The log reveals that the computer read memory values from the programmable logic controller (shown in Figure 9):
  - I0.7 to I0.0: `0x03 [0000 0011]`
  - Q0.7 to Q0.0: `0x03 [0000 0011]`
  - M0.7 to M0.0: `0x00 [0000 0000]`
- `12:07:44pm, 18 Sep 2016:` The log reveals that the programmable logic controller memory values were altered as follows (shown in Figure 9):

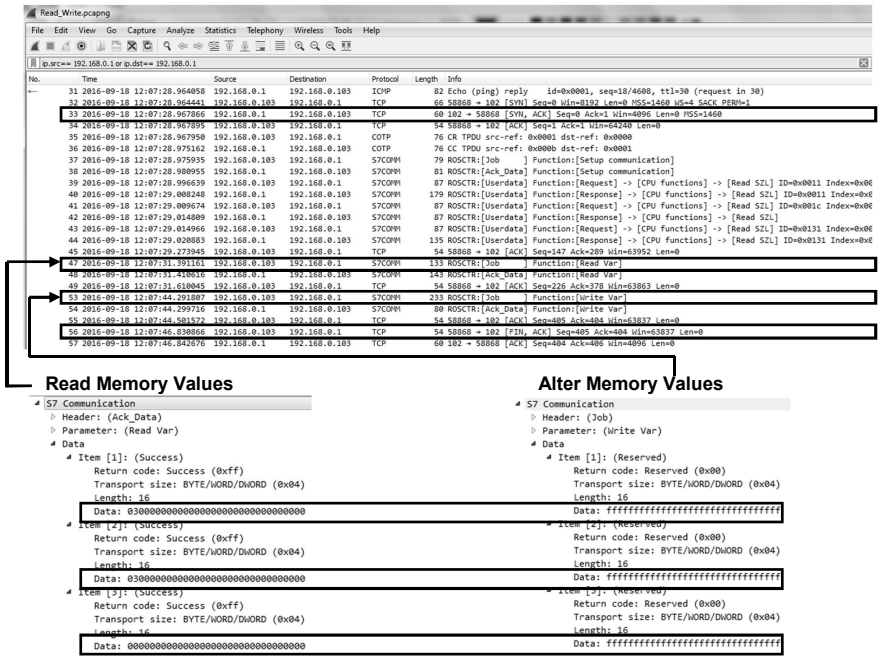

Figure 9. Programmable logic controller READ and WRITE requests.

  - I0.7 to I0.0: `0xff [1111 1111]`
  - Q0.7 to Q0.0: `0xff [1111 1111]`
  - M0.7 to M0.0: `0xff [1111 1111]`

- `12:07:46pm, 18 Sep 2016:` The computer closed its connection to the programmable logic controller.

Based on the data in Table 1, the memory values of the programmable logic controller outputs at `12:07:44pm, 18 Sep 2016` indicate that all the pedestrian and vehicle lights were turned on. The programmable logic controller operation appears to be anomalous because an attempt was made to turn on all the traffic lights at the same time.

6.2 Liquid Mixing Control System

The following steps were involved in sending CPU WRITE requests to the programmable logic controller.

- Wireshark was started to capture network packets.
- Snap7 established a connection to the programmable logic controller.

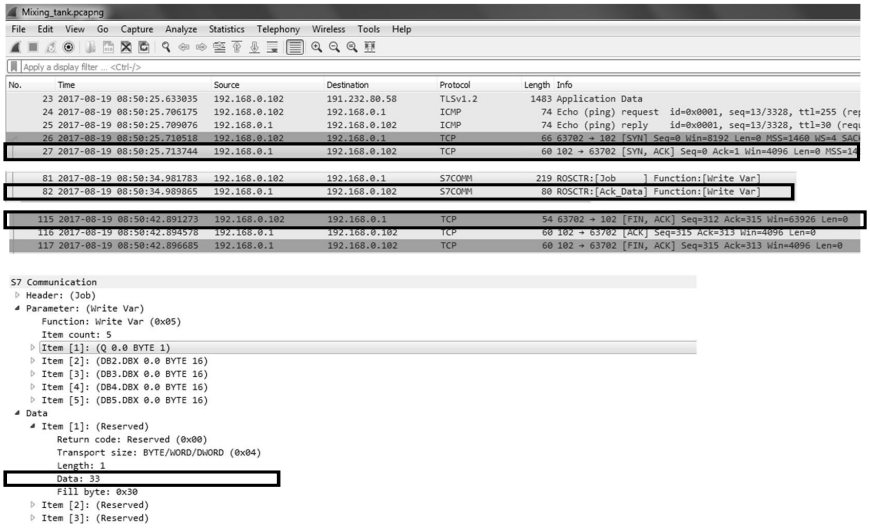

Figure 10. Programmable logic controller WRITE requests.

- Snap7 wrote the value 1 to outputs (O0.0 to 0.7) of the programmable logic controller.
- Snap7 closed its connection to the programmable logic controller.
- Wireshark was stopped and the captured packets were saved in a log file.
- The packets with the programmable logic controller IP address `[192.168.0.1]` were filtered for analysis.

Figure 10 shows the captured log file associated with the programmable logic controller WRITE requests. The following controller activities can be reconstructed in time sequence upon analyzing the log file:

- `08:50:25pm, 19 Aug 2017:` A computer `[192.168.0.102]` established a connection to the programmable logic controller `[192.168.0.1]`.
- `08:50:34pm, 19 Aug 2017:` The log reveals that the programmable logic controller memory values were altered as follows (shown in Figure 10):

  Q0.7 to Q0.0: `0x33 [0011 0011]`
- `12:07:42pm, 19 Aug 2017:` The computer closed its connection to the programmable logic controller.

Based on the data in Table 1, the memory values of the programmable logic controller at `08:50:34pm, 19 Aug 2017` indicate that the pump for paint ingredient 1 and the pump for paint ingredient 2 were turned on; meanwhile, the drain valve and drain pump were also turned on. The programmable logic controller operation appears to be anomalous because an attempt was made to turn on all the valves and pumps at the same time.

7. Experimental Results and Discussion

In the experiments, four common programmable logic controller requests, CPU START, CPU STOP, READ and WRITE were identified by packet analysis using Wireshark with the S7 dissector plugin. The sequences with timestamps related to programmable logic controller connection establishments and requests were also captured in the log file. Based on the log file and the programmable logic controller application, an investigator could reconstruct the anomalous programmable logic controller operations. In addition to the four programmable logic controller requests, other programmable logic controller activities were also be revealed via packet analysis, including programmable logic controller program uploads and downloads. The upload and download commands have the S7comm function parameter (first parameter byte) of `0x1A` [7]. The download commands provide valuable information about the timeline of programmable logic controller program updates.

The experiments used Wireshark with the S7 dissector plugin to capture and analyze S7 packets for forensic purposes. However, due to the large volume of packets, it is infeasible to capture all the information related to the packets for analysis. Therefore, the proposed programmable logic controller logging system uses Wireshark with the S7 dissector plugin and an add-on feature to capture packets selectively. When the logging system detects a connection request sent to the programmable logic controller on TCP port 102, it starts capturing the communications. Likewise, when the logging system detects a disconnection request sent to programmable logic controller on TCP port 102, it stops capturing the communications.

Since the raw data capture is disorganized, a logging system process converts the captured packet information to a human-readable format that is stored in an audit file. Thus, a forensic investigator would only have to examine the audit file. This reduces the effort required by the investigator, who would not be expected to be a control system expert.

Since the ISO-TSAP packets that encapsulate the proprietary Siemens S7comm protocol are sent in plaintext, it is relatively simple to reverse

engineer them and make modifications as needed. This characteristic of ISO-TSAP packets enables attackers to replicate operator activities involving programming and management, including turning off the CPU, disabling memory protection and uploading new project files to the programmable logic controller [1]. On one hand, attackers leverage ISO-TSAP to monitor and interfere with programmable logic controller operations. On the other hand, the proposed logging system leverages ISO-TSAP to capture valuable information about attacker activities for forensic investigations.

Several tools are available for creating audit logs for programmable logic controllers, but the information they capture is insufficient in forensics investigations. Specifically, the audit logs usually capture the values of relevant memory addresses used by programmable logic controller programs to support debugging and troubleshooting. Crucial forensic information is always missing, including the IP address of the device that connected to the programmable logic controller, the commands (e.g., READ data, WRITE data and DOWNLOAD program) sent to the programmable logic controller and the duration of the connection to the programmable logic controller.

Finally, the logging system incorporate two processes in order to minimize its impact on programmable logic controller operations. The first process is responsible for capturing packets. The second process analyzes the captured packets and stores forensically-relevant data in a human-readable format in the audit log file. Because this process is time consuming, it is executed as a batch process instead of a real-time process to enhance the efficiency of the logging system.

8. Conclusions

Current programmable logic controller logging tools provide insufficient information for digital forensic investigations. To address this limitation, the logging system described in this chapter analyzes network traffic between a Siemens Simatic S7 programmable logic controller and network devices based on the Siemens S7 communications protocol to record evidence of the sequence of activities related to commands and data exchanged between the programmable logic controller and other network devices. The log provides valuable information about attacks, including the attacker IP addresses, specific actions and timelines. The decision to focus on a Siemens Simatic S7 programmable logic controller was motivated by their widespread use [1] and the fact that they were targeted successfully by the insidious Stuxnet malware.

Future research will focus on developing a production logging system for industrial control system environments. Attempts will also be made to expand the logging capabilities to handle other popular industrial control protocols such as Modbus and DNP3 for various programmable logic controller models.

References

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

[2] 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.

[3] T. Cruz, J. Barrigas, J. Proenca, A. Graziano, S. Panzieri, L. Lev and P. Simoes, Improving network security monitoring for industrial control systems, *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, pp. 878–881, 2015.

[4] European Union Agency for Network and Information Security, Critical Infrastructures and Services, Heraklion, Greece (`enisa.europa.eu/topics/critical-information-infrastructures-and-services`), 2017.

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

[6] J. Klick, S. Lau, D. Marzin, J. Malchow and V. Roth, Internet-facing PLCs – A new back orifice, presented at *Blackhat USA*, 2015.

[7] J. Malchow, D. Marzin, J. Klick, R. Kovacs and V. Roth, PLC Guard: A practical defense against attacks on cyber-physical systems, *Proceedings of the IEEE Conference on Communications and Network Security*, pp. 326–334, 2015.

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

[9] PLC-Logger Project, PLC-Logger and Analyzer (`sourceforge.net/projects/plclogger`), 2014.

[10] Siemens, SIMATIC S7-300 Programmable Controller Quick Start, Primer, Preface, C79000-G7076-C500-01, Nuremberg, Germany, 1996.

[11] Siemens, SIMATIC S7-200 Programmable Controller System Manual, 6ES7298-8FA01-8BH0, Edition 08/2005, Nuremberg, Germany, 2005.

[12] 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.

[13] T. Wiens, S7 Communications (s7comm), *Wireshark Wiki* (`wiki.wireshark.org/S7comm`), 2016.

[14] T. Wiens, S7comm Wireshark Dissector Plugin (`sourceforge.net/projects/s7commwireshark`), 2017.

[15] 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.

[16] 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.

[17] 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.

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