Chapter 5

A FIRMWARE VERIFICATION TOOL FOR PROGRAMMABLE LOGIC CONTROLLERS

Lucille McMinn and Jonathan Butts

Abstract Current supervisory control and data acquisition (SCADA) systems do not have adequately tailored security solutions. Programmable logic controllers (PLCs) in SCADA systems are particularly vulnerable due to a lack of firmware auditing capabilities. Since a PLC is a field device that directly connects to a physical system for monitoring and control, a compromise of its firmware could have devastating consequences. This paper describes a tool developed specifically for verifying PLC firmware in SCADA systems. The tool captures serial data during firmware uploads and verifies it against a known good firmware executable. It can also replay captured data and analyze firmware without the presence of a PLC. The tool does not require any modifications to a SCADA system and can be implemented on a variety of platforms. These features, along with the ability to isolate the tool from production systems and adapt it to various architectures, make the tool attractive for use in diverse SCADA environments.

Keywords: Programmable logic controllers, firmware verification

1. Introduction

The critical infrastructure depends on secure, reliable supervisory control and data acquisition (SCADA) systems that provide critical control, communication and monitoring capabilities over geographically dispersed locations [4, 5]. Because SCADA systems are increasingly interconnected via unsecured networks, security solutions have focused on creating logical and physical boundaries between systems and the network layer [16, 21]. However, even with network isolation, additional attack ingress points have manifested themselves. Indeed, attackers are increasingly leveraging non-traditional means to compromise SCADA systems [18].

Current attack response and mitigation tools are inadequately tailored to SCADA systems [26]. In many cases, the available resources are IP network tools that have been adapted to SCADA environments (e.g., packet capture tools, general operating system analysis tools and network-based intrusion detection systems). Although these tools and systems provide protection in a broad sense, tailored security solutions are needed to address emerging threats specific to SCADA systems. Perhaps the most pressing concern is verifying the proper operation of field devices, such as programmable logic controllers (PLCs), that directly monitor and control physical systems. These devices typically operate "below" the network layer and have few security mechanisms. As demonstrated by Stuxnet, manipulation of these devices can have direct physical consequences.

This research describes a tool that helps validate PLC firmware. Firmware, in the most basic sense, is fixed microcode that provides a bridge between the hardware and programmable software on a device. An attacker who can gain access to and manipulate firmware has full control over the functionality of the device and can potentially mask actions from detection. The tool described in this paper helps validate firmware and ensure that any attempt to alter the firmware is detected. The tool is adaptable and portable. Its ability to quickly and safely interface to a computer in order to analyze data sent to a PLC makes it a viable security application in diverse SCADA environments.

2. Background

Most critical infrastructure protection strategies leverage traditional network security constructs. Firewalls and intrusion detection or prevention systems are used to implement a defense-in-depth security strategy. Many SCADA-system-specific solutions engage encryption to achieve confidentiality [9, 17], but such approaches make it more difficult to audit network traffic [19]. Other security solutions often require the modification or addition of system components, which can hinder real-time performance or may be infeasible for large-scale SCADA systems in the field [2, 13, 15, 20].

While network defense is critical to security, network-based attacks are unlikely to be the primary method of exploitation in the future. Stuxnet infiltrated a non-networked environment via a nontraditional vector – a USB device [6]. Indeed, because critical infrastructure assets are high value targets, an advanced persistent threat can be expected to find an input vector to compromise even the most secure system [8, 10, 25]. Considering the variety of non-network-based input vectors, security must be applied beyond the network layer [18].

In SCADA systems, PLCs are field devices that directly connect to physical equipment. The devices control equipment and report data about their operation to remote monitoring stations. The PLCs themselves are typically monitored via a remote human machine interface [3]. As demonstrated by Stuxnet and other recent attacks [7, 23], a PLC under the control of a malicious entity can have devastating effects.

A PLC presents three main targets for attack: hardware, firmware and logic program. Hardware is the lowest layer of abstraction and, at some level, must be trusted. Hardware security requires a trusted supply chain or methods for thoroughly testing the device. In this research, we consider firmware, which includes the PLC operating system, to be the lowest electronically-modifiable layer of a PLC. Note that some PLCs do not have modifiable firmware and others have additional modifiable levels such as a reprogrammable BIOS. Nevertheless, the basic methodology presented in this paper can be used for these architectures, where the lowest modifiable layer is synonymous with firmware.

This research specifically focuses on devices with modifiable firmware and logic program layers. Logic program modifications alter PLC functionality and can be performed fairly easily by accessing the PLC management software. Manipulations of PLC programs, however, can be identified during an inspection. On the other hand, PLC firmware modification is the most intrusive and least detectable attack – there are no easy methods to extract and verify the firmware after it is loaded on a PLC [22]. As shown in Figure 1, the problem is exacerbated by the many potential input vectors that enable firmware alteration (e.g., programming computers, SCADA control systems and firmware update software). Indeed, any access point to a PLC or access to firmware code to be uploaded is an avenue for altering PLC firmware.

3. Tool Design and Evaluation

A tool for verifying that a source device is sending unmodified firmware to a PLC must have three primary features: (i) ability to capture communications data; (ii) ability to analyze captured data; and (iii) ability to determine the validity of the firmware. The tool, which is positioned between the sending device and the PLC, must capture communications data without impacting PLC operations. After the data capture, the tool must analyze the data to determine if the firmware is unmodified.

In the most basic form, the identification of modified firmware is accomplished by comparing the firmware under test with a known good firmware version. For our purposes, we assume a known good baseline version is available for comparison. Note that simple hash comparison for firmware is not as straightforward as checking for modified files in a traditional operating system. Indeed, the requirements to capture and analyze the data as it is being loaded and then perform the comparison render firmware validation nontrivial.

Another important feature is to emulate PLC communications and verify the unmodified firmware independently without the need for a PLC. This type of independent verification is critical to tool portability as it enables implementation on a generic personal computer or mobile computing device for multiple PLC and firmware instances.

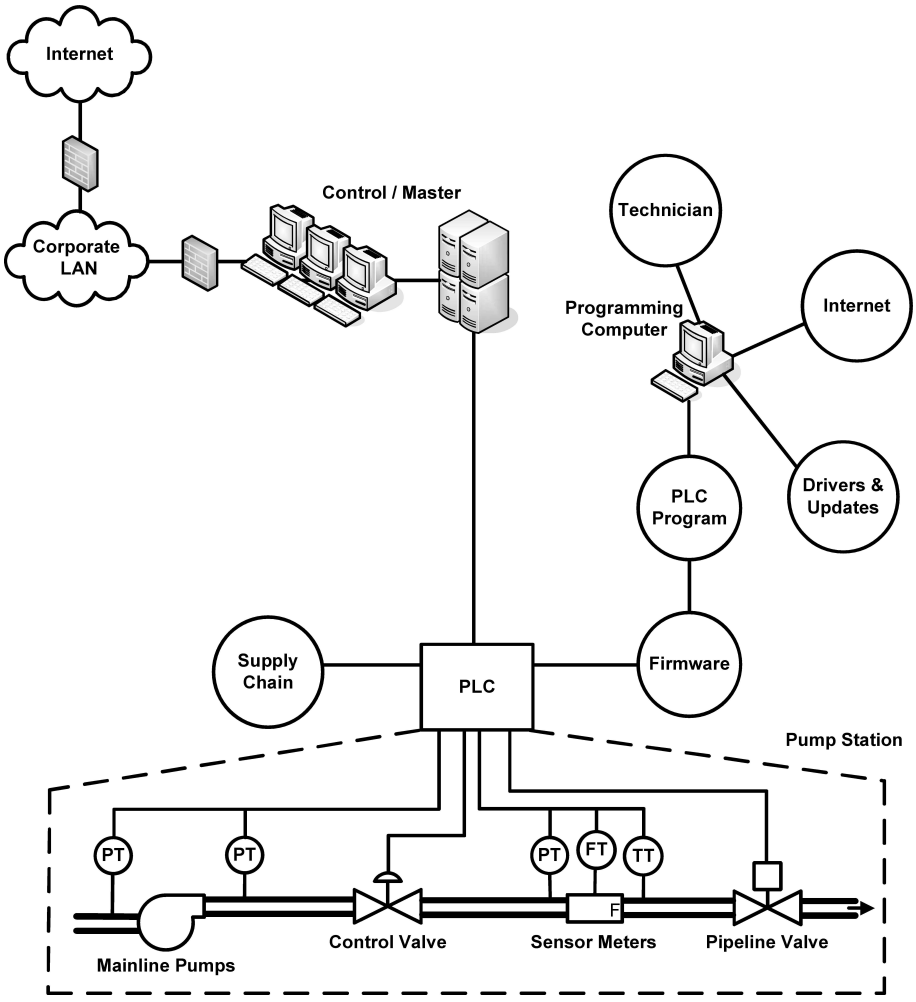

Figure 1. Example non-traditional SCADA system inputs.

3.1 Evaluation Environment

The environment for evaluating the verification tool used a standard Windows XP personal computer and an Allen Bradley FlexLogix 5434 PLC. The personal computer emulated a programming or maintenance system designed to upload the RSLogix firmware. The upload computer had the Rockwell Software RSLogix 5000 suite installed as well as ControlFLASH 9.00.015, the firmware loading program. For the initial baseline capture, the verification tool was connected to the primary communication line via a passive serial adapter tap, enabling the interception of communications data while preserving communications between the uploading computer and PLC. The serial port was configured

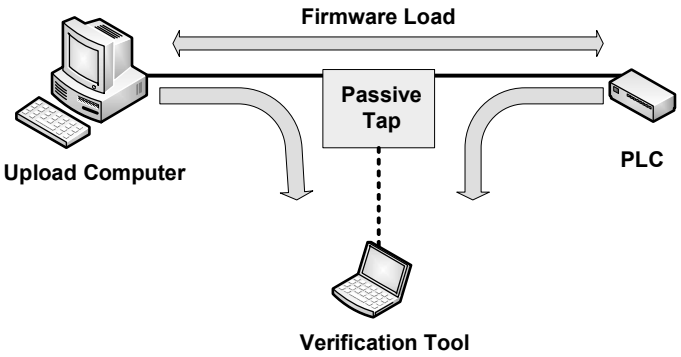

Figure 2. Initial baseline capture configuration.

for the serial data capabilities of the PLC, specifically a baud rate of 19200, eight data bits, no parity and one stop bit. Figure 2 shows the environment used for the initial baseline capture.

During the baseline capture, the uploading computer was connected directly to the PLC with the corresponding DF1 driver controlling communications exchange on the uploading computer side. Firmware version 15.06.01 was loaded on the PLC with the baseline firmware file and the corresponding binary files stored in the ControlFLASH program directory. Each individual ControlFLASH upload was executed in an identical manner using the same version and the same selection method to eliminate variations in the serial data transfer. Data capture started when the ControlFLASH program was opened and ended when the firmware upload was announced as having been completed.

The verification tool was initialized using two captures to create a known good baseline. During this initial capture phase, a firmware load from the uploading computer to the PLC was captured using the serial line. Thus, the verification tool received all the transferred data in a passive manner. The tool used a multithreaded environment to capture serial port data from each line as soon as it became available and stored the data in a binary format. The captured data was then separated into data sent from the firmware uploading computer and reply data sent from the PLC. Note that the data from the uploading computer contained the uploaded firmware bytes.

The entire firmware loading process was executed and captured twice. This allowed the capture program to identify variable protocol packet fields. The two captures were then used to create a baseline for communications. The communications were parsed and the variable bytes were checked against typical protocol field patterns [1, 11, 12, 14]. After all the differences were accounted for, a protocol profile was created, which contained the pattern for communications. The pattern was then applied to the received data in order to emulate future communications.

After the firmware has been loaded on the PLC, the PLC can be taken out of the communications setup and the emulation environment can be implemented

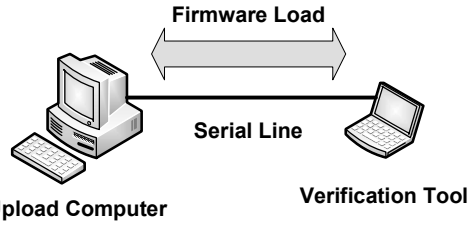

Figure 3. Emulator environment configuration.

by directly connecting the uploading computer to the verification tool (Figure 3). The emulator may then be used to capture and verify subsequent firmware uploads. Note that the emulation environment enables an independent system to evaluate the firmware without the need for the PLC. Indeed, this configuration allows multiple PLC platforms and firmware versions to be validated on a single system without impacting system operations.

3.2 Design

The verification tool was designed in Microsoft Visual Studio to execute on the Windows 7 operating system. The program utilizes two separate serial ports for data capture, where each port is monitored by a thread. The protocol packet field format is not hard coded; instead, it is adapted through analysis of the captured communications data.

The captured data is first grouped into similar packet-like segments. The groupings are based on a start field block consisting of the maximum possible length that occurs a maximal number of times throughout the captured data – both these traits are indicative of a start field block.

A similar method is used to find end field blocks. Note that start/end blocks for the PLC and uploading computer are identified independently. This is because the start/end field blocks may not be the same between the two devices.

After the packet blocks are formed, field mismatches are identified between different communications captures and are accounted for by stateful packet fields. The analysis process checks for typical stateful fields including a BCC error correcting code and a packet identification byte that is sent by the computer and echoed back by the PLC. Escape bytes are also checked because escaping bytes may not be included in the checksum.

The program uses a brute-force parsing method to find optimal field matches. During the evaluation, data captures averaged 2 MB with parsing taking an average of ten minutes (the parsing time would increase for larger data captures). After the initial parsing, the protocol profile that is created is saved and reloaded for future execution to reduce the subsequent run time. Note that the duration of a firmware upload does not change because this is determined by the serial baud rate and the firmware data size.

Given a known good baseline configuration, the emulator is connected to the firmware upload computer directly, replaying PLC responses in accordance with any stateful protocol modifications. The capture program sends responses after each packet is received and can independently induce a firmware upload without the presence of a PLC.

After the uploading is complete, a verification check is performed on the captured communications, validating the received firmware bytes. If all the stateless bytes are the same and the communication matches the baseline communication profile, then it can be concluded that the firmware upload is consistent with the baseline and that the firmware is unmodified.

More thorough checking of the uploaded capture can be performed if the protocol standards are known. All the packets that contain the command to upload firmware packets are parsed to reproduce the loaded firmware file byte-for-byte. This is a more thorough check because it uses prior protocol knowledge to isolate and reproduce the uploaded firmware. The tool has this capability on full-duplex DF1 with the embedded Common Industrial Protocol [1, 11].

4. Evaluation Results

The tool was evaluated using two firmware versions of FlexLogix 5434. The two sets of captured data for protocol analysis were separated by an intermediate period of transferred data to account for any slow changing packet identification variables. All the communications data was successfully captured and parsed into packets using simple optimization algorithms for each variable field. Several versions of modified or incomplete firmware were uploaded, each of which was detected successfully. Additionally, all instances of unmodified firmware were uploaded without any false positive errors.

4.1 Analysis

Parsing, verifying and emulating serial communications between a computer and a PLC require no system modifications, but they provide a thorough security measure. Directly verifying serial data at the last point between the external system and the PLC provides increased assurance that any modifications from the firmware uploading device or any of its input vectors would be identified. Because the firmware uploading procedure follows a deterministic progression, modified firmware can be detected by comparing each new upload against the baseline, even with stateful protocol packet fields. In every test instance, the verification tool was able to identify all the possible outcomes associated with firmware uploads:

- If the firmware is modified to contain at least one more byte or one less byte, then the modified upload contains at least one more byte or one less byte than the baseline.

- If the firmware is modified but still contains the same number of bytes as the previous firmware, then it can be concluded that these bytes are either different from the bytes in the original firmware, or the bytes are in a different order, or both. Therefore, the corresponding packet bytes would be modified in the same manner and must be different from those in the baseline.
- If the firmware is unmodified but the firmware loading program is changed to send modified data, then any changes that correspond to either of the two previous situations are detected.
- If protocol fields are changed so that the load occurs in an identical manner as the baseline but the function codes are changed (e.g., a "store byte" command is changed so that the store is no longer executed, modifying the firmware bytes after they are stored on the PLC instead of prior to upload execution), then the modified function codes are detected as being modified from the baseline function codes.
- If there are no detected differences, then it can be concluded that the uploaded firmware is the same as the original firmware.

4.2 Discussion

The verification tool is currently limited to capturing serial data; however, a similar design can be implemented for other media. While there are many possible configurations for SCADA control devices, the majority of these devices use RS-232, which renders this solution feasible for use on a range of systems [3, 18].

The firmware is the lowest electronically-modifiable level of a SCADA control device. Indeed, firmware validation is the first logical step when considering electronic security. Beyond this, it is also necessary to ensure that the firmware cannot be uploaded remotely by any other means.

PLC memory does not conform with the typical von Neumann architecture. When firmware is loaded, a BIOS writes the uploaded firmware to ROM or flash memory, and the logic program is stored in volatile memory. Without a modified BIOS, the firmware cannot be self-modifying. Executable memory and data are stored separately, preventing a firmware level remote code injection from running PLC firmware.

4.3 Impact

The primary goal of validating firmware extends beyond ensuring that known good firmware is loaded on a PLC – it also helps create a closed system with respect to the PLC. PLCs have the highest level of local control over a SCADA system, so it is critical that they are verified at the basic hardware and software levels before additional security measures are applied at a higher level.

The verification tool offers a novel approach for SCADA security because it requires no system modifications or additions and does not affect the produc-

tion system. Additionally, the verification tool does not introduce new attack vectors to a PLC because the tool is not physically wired to exchange communications with the PLC. Moreover, the tool can operate independently of a PLC.

5. Conclusions

Creating security tools specific to SCADA systems is necessary to maintain and build trust in critical infrastructure systems. The verification tool described in this paper is a viable option for enhancing PLC firmware security. While serial data capture, data verification and emulation are by no means new concepts, the tool combines these concepts in a novel manner that is tailored to SCADA system security. Ideally, a system should be secured from the bottom up. From this point of view, the tool is significant because it helps verify the security of a PLC at the lowest electronically-modifiable level.

Other advantages of the verification tool are that it does not require modifications to the SCADA system, and that it can replay captured data and analyze firmware without the presence of a PLC. These advantages, along with the ability to isolate the tool from production systems and adapt it to various architectures, make the tool attractive for use in diverse SCADA environments.

Note that the views expressed in this article are those of the authors and do not reflect the official policy or position of the U.S. Air Force, Department of Defense or the U.S. Government.

Acknowledgements

This research was partially supported by ICS-CERT under Award HSHQDC-11-X-00089 from the U.S. Department of Homeland Security.

References

[1] Allen-Bradley, DF1 Protocol and Command Set: Reference Manual, Publication No. 1770-6.5.16, Milwaukee, Wisconsin, 1996.

[2] C. Basile, S. Di Carlo and A. Scionti, FPGA-based remote-code integrity verification of programs in distributed embedded systems, *IEEE Transactions on Systems, Man and Cybernetics; Part C: Applications and Reviews*, vol. 42(2), pp. 187–200, 2011.

[3] W. Bolton, *Programmable Logic Controllers*, Elsevier Newnes, Oxford, United Kingdom, 2006.

[4] S. Boyer, *SCADA: Supervisory Control and Data Acquisition*, Instrumentation, Systems and Automation Society, Research Triangle Park, Durham, North Carolina, 2004.

[5] Department of Homeland Security, National Infrastructure Protection Plan, Washington, DC, 2009.

[6] N. Falliere, Exploring Stuxnet's PLC Infection Process, Symantec, Mountain View, California, 2010.

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

[8] W. Gao, T. Morris, B. Reaves and D. Richey, On SCADA control system command and response injection and intrusion detection, *Proceedings of the eCrime Researchers Summit*, 2010.

[9] G. Gilchrist, Secure authentication for DNP3, *Proceedings of the IEEE Power and Energy Society General Meeting on the Conversion and Delivery of Electrical Energy in the 21st Century*, 2008.

[10] S. Gorman, A. Cole and Y. Dreazen, Computer spies breach fighter-jet project, *Wall Street Journal*, April 21, 2009.

[11] D. Hristu-Varsakelis and W. Levine (Eds.), *Handbook of Networked and Embedded Control Systems*, Birkhauser, Boston, Massachusetts, 2008.

[12] Institute of Electrical and Electronics Engineers, 1815-2010 – IEEE Standard for Electric Power Systems Communications – Distributed Network Protocol (DNP3), Piscataway, New Jersey, 2010.

[13] M. Jakobsson and K. Johansson, Practical and secure software-based attestation, *Proceedings of the Workshop on Lightweight Security and Privacy: Devices, Protocols and Applications*, 2011.

[14] Modicon, Modicon Modbus Protocol Reference Guide, PI-MBUS-300 Revision J, North Andover, Massachusetts, 1996.

[15] T. Morris and K. Pavurapu, A retrofit network transaction data logger and intrusion detection system for transmission and distribution substations, *Proceedings of the IEEE International Conference on Power and Energy*, pp. 958–963, 2010.

[16] National Institute of Standards and Technology, Managing Information Security Risk: Organization, Mission and Information System View, NIST Special Publication 800-39, Gaithersburg, Maryland, 2011.

[17] O. Pal, S. Saiwan, P. Jain, Z. Saquib and D. Patel, Cryptographic key management for SCADA systems: An architectural framework, *Proceedings of the International Conference on Advances in Computing, Control and Telecommunication Technologies*, pp. 169–174, 2009.

[18] M. Schwartz, J. Mulder, J. Trent and W. Atkins, Control System Devices: Architectures and Supply Channels Overview, Sandia Report SAND2010-5183, Sandia National Laboratories, Albuquerque, New Mexico, 2010.

[19] W. Shaw, *Cybersecurity for SCADA Systems*, PennWell, Tulsa, Oklahoma, 2006.

[20] K. Song, D. Seo, H. Park, H. Lee and A. Perrig, OMAP: One-way memory attestation protocol for smart meters, *Proceedings of the Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops*, pp. 111–118, 2011.

[21] K. Stouffer, J. Falco and K. Kent, Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security, NIST Special Publication 800-82, National Institute of Standards and Technology, Gaithersburg, Maryland, 2006.

[22] J. Stradley and D. Karraker, The electronic part supply chain and risks of counterfeit parts in defense applications, *IEEE Transactions on Components and Packaging Technologies*, vol. 29(3), pp. 703–705, 2006.

[23] R. Turk, Cyber Incidents Involving Control Systems, Technical Report INL/EXT-05-00671, Idaho National Laboratory, Idaho Falls, Idaho, 2005.

[24] X. Wang, M. Tehranipoor and J. Plusquellic, Detecting malicious inclusions in secure hardware: Challenges and solutions, *Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust*, pp. 15–19, 2008.

[25] G. Wilshusen, Information Security: Cyber Threats and Vulnerabilities Place Federal Systems at Risk, GAO Report GAO-09-661T, Government Accountability Office, Washington, DC, 2009.

[26] G. Wilshusen, Cybersecurity: Continued Attention Needed to Protect Our Nation's Critical Infrastructure, GAO Report GAO-11-865T, Government Accountability Office, Washington, DC, 2011.