Digital Investigation 22 (2017) S57–S65

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2017 USA — Proceedings of the Seventeenth Annual DFRWS USA

SCADA network forensics of the PCCC protocol

Saranyan Senthivel, Irfan Ahmed*, Vassil Roussev

GNOCIA, Department of Computer Science, University of New Orleans, 2000 Lakeshore Dr, New Orleans LA, 70122, USA

Keywords:
SCADA forensics
SCADA protocol
PCCC
Network traffic analysis
Programmable logic controller

A B S T R A C T

Most SCADA devices have few built-in self-defence mechanisms, and tend to implicitly trust communications received over the network. Therefore, monitoring and forensic analysis of network traffic is a critical prerequisite for building an effective defense around SCADA units. In this work, we provide a comprehensive forensic analysis of network traffic generated by the PCCC(Programmable Controller Communication Commands) protocol and present a prototype tool capable of extracting both updates to programmable logic and crucial configuration information. The results of our analysis show that more than 30 files are transferred to/from the PLC when downloading/uploading a ladder logic program using RSLogix programming software including configuration and data files. Interestingly, when RSLogix compiles a ladder-logic program, it does not create any low-level representation of a ladder-logic file. However, the low-level ladder logic is present and can be extracted from the network traffic log using our prototype tool. The tool extracts SMTP configuration from the network log and parses it to obtain email addresses, username and password. The network log contains password in plain text.

Introduction

Supervisor Control And Data Acquisition (SCADA) systems are used to automate industrial processes, such as power generation and distribution, gas and oil pipelines, and water and waste management. Their primary design requirement is *safety*, which typically requires real-time response to changes in the monitored processes, and an ability to handle harsh working environment; they were never designed to withstand cyber attacks of any kind. Early SCADA systems were deployed in specialized isolated networks, which are not connected with corporate networks, or the Internet. Thus, they were protected from remote attacks by virtue of not being accessible over the network.

Over the past two decades, with the increased convergence of data networks, SCADA systems are ever more tightly integrated with the TCP/IP infrastructure (Ahmed et al., 2012). Although the standardization of all communication brings substantial economic advantages, it also brings the potential of remote attackers gaining access to inherently insecure devices, and executing attacks on the physical infrastructure with potentially catastrophic consequences (McLaughlin et al., 2016; Robinson, 2013). Stuxnet for instance, is a malware that specifically targeted industrial automation systems (Langne, 2013).

SCADA systems generally consist of sensors, actuators, programmable logic controllers (PLCs), and a human machine interface (HMI) (Stouffer et al., 2011; Macaulay, 2012). A PLC is deployed at a remote field site to provide immediate monitoring and control of a physical process. HMI and other SCADA services (such as engineering workstation and historian) run at a control center and provide the means for operators to remotely observe and control the processes.

A PLC communicates with its respective control center to send the current state of physical process, which is then displayed by HMI graphically for control operators. It uses sensors to obtain the current state of physical process (such as pressure of the gas in pipeline), and actuators (such as solenoid valve) to alter the current state depending on the logic in the PLC. For example, a PLC may be programmed to maintain pressure in a gas pipeline between 40 and 50 PSI. Based on readings from the pressure sensor, if the gas pressure is more than 50 PSI, the PLC opens the solenoid valve to release some gas until the pressure is reduced to 40 PSI.

An engineering workstation at the control center runs PLC programming software, which is used by control engineers to program and transfer the control logic to a PLC over the network. Unfortunately, an attacker can also acquire and utilize the software to create a malicious control logic program, and download it to a PLC after establishing a communication with the PLC. At worst, an

* Corresponding author.
E-mail addresses: ssenthiv@my.uno.edu (S. Senthivel), irfan@cs.uno.edu (I. Ahmed), vassil@cs.uno.edu (V. Roussev).

http://dx.doi.org/10.1016/j.diin.2017.06.012

attacker can compromise an engineering workstation and utilize its programming software to re-program the PLCs, or to modifying the current logic in the PLCs. The Stuxnet malware is a pertinent example that mainly targets engineering workstation running Windows operating system, and compromises Siemens STEP7 programming software to further infect the Siemens PLCs.

The most direct approach to investigating a potential breach is to attempt to acquire the current logic from PLCs using the programming software for further analysis. However, this method is not viable if the communication between the PLC and control center is disrupted. Also, the communication with the PLCs may not be reliable if the system is under a cyber attack and the attacker may manipulate the communication such as through man-in the middle attack.

Therefore, to *reliably* investigate these kind of attacks, SCADA network traffic log must be kept and analyzed to identify unauthorized transfer of control logic to PLCs including extracting relevant forensic artifacts. A first step in this direction is to understand how a programming software transfers the PLC control logic over the network using a SCADA protocol.

This paper presents a comprehensive analysis of PCCC protocol for transferring control logic to a PLC. We use Allen–Bradley's RSLogix 500 programming software (RSLogix500, 2017) and Micrologix 1400 PLC (MicroLogix 1400 Series B, 2017) for experiments. The analysis results show that when the programming software downloads or uploads a control logic program to and from the PLC, the network traffic not only contains the control logic but also system configuration and other data (such as counter, input, output, timer etc.). The PCCC message has file type and file number fields that we use to extract and store different type of information into files. Prior to this work, most of these file types had remained undocumented even in vendor specifications.

Using differential analysis, we performed a comprehensive set of experiments to understand the type of contents in the files and further classify unknown file types accordingly. One of the first observations is that, whenever RSLogix compiles the control logic, it does not create any output file on the workstation. In other words, there is no observable low-level representation of control logic, data or configuration file that is suppose to be transferred to and run by the PLC. This program, however, can be extracted from the network traffic; the first sign of logic transfer (in the log) is that the PLC is switched from `RUN` to `PROGRAM` mode, and back to `RUN` upon completion of the transfer.

Based on our findings, we developed a proof-of-concept prototype tool, called *Cutter*, to perform the forensic analysis of SCADA network traffic. `Cutter` is useful for identifying any transfer of logic program and configuration files to/from a PLC in a network packet capture, and further extracting them for forensic analysis. It parses the PCCC message format, identifies the boundary of the messages representing start and end of the transfer of logic program in a network traffic capture, filters out irrelevant messages within the boundary, and assembles the relevant messages (containing the program and other data files) in a correct sequence, and stores the assembled data in files on disk. It is also capable of parsing input, output and configuration files and presenting the content in a readable format for further analysis. The input and output files contain sensor readings and the state of other input devices (such as on or off in toggle switches), and actuator state respectively. The configuration files include SMTP client and network configurations such as username/password, email addresses, and IP/Subnet mask.

We evaluate the `Cutter` in two distinct scenarios. The first one simulates an attacker modifying the control logic of a PLC. When the logic is transferred to a PLC, it is captured in a network traffic log; `Cutter` analyzes the log and identifies the evidence of logic transfer successfully. It further extracts the transferred logic from the log and compares it with the original logic for integrity checking. In the second scenario, attackers modify the SMTP client configuration of a PLC by adding their email address to receive the copy of notifications. `Cutter` extracts the SMTP configuration from the log, compares it with the original, and identifies the attacker email address successfully.

In sum, this work makes the following contributions to the field:

- We perform a detailed analysis of the network traffic of PCCC protocol and reverse engineer the entire process of transferring a control logic program to a PLC.
- We identify several unknown file types in the PCCC traffic containing important information of forensic relevance, such as SMTP client configuration, ladder logic program, and other system and network configurations. We further classify these file types according to their content.
- We develop a network forensic tool, `Cutter`, that is able to extract forensic artifacts (or files of different types) from a PCCC network traffic log, and further parse them to extract information and present it in human readable form.
- We demonstrate the effectiveness of `Cutter` in two distinct scenarios: 1) detections of malicious control logic injection; and 2) detection of a compromised SMTP configuration.

The rest of the paper is organized as follows: Section Control logic transfer via PCCC presents a detailed analysis of the control logic transfer via the PCCC protocol. Section Implementation presents the implementation details of the `Cutter` prototype tool, followed by Section Evaluation with the evaluation results. Section Related work presents the related work followed by a conclusion in Section Conclusion.

Control logic transfer via PCCC

We first analyze the transfer process of a control logic to a PLC using PCCC protocol, with the goal of identifying the relevant forensic traces in the network traffic log.

PCCC protocol. The PCCC is a command/reply protocol that provides several operational functions, such as diagnostic status, change mode, and echo. It is supported by many popular PLCs including PLC-5, SLC500, and Logix family (such as Micrologix and Controllogix). The PCCC message is transported as an embedded object in EtherNet/IP (EIP) protocol, which is an adaption of common industrial protocol (CIP) over Ethernet.

Analysis of PCCC network traffic

Unfortunately, common network analysis tools, such as Wireshark, do not support PCCC protocol. There is a vendor document that describes the format of PCCC message; however, it is valid when the PCCC is used with DF1 link layer protocol (or for serial communication) (Allen Bradley's, 2017). As it turns out, the format is not completely aligned with the traffic observed over Ethernet. The focus of our research is to develop a forensic tool for Ethernet and IP infrastructure. Our lab has a licensed software, NetDecoder (NetDecoder) commonly used in the industry for debugging. NetDecoder supports PCCC and can parse its messages. We use it to understand the fields of a PCCC message and the messages involving in the transfer of control logic.

Data collection. We use the Allen Bradley `Micrologix 1400` PLC that supports PCCC protocol, and the RSLogix programming software to create a control logic program and transfer it to the PLC. The software is installed in a Windows 7 computer, which is directly connected to the PLC. We use NetDecoder to capture the