Summer 8-9-2017

Automatic Forensic Analysis of PCCC Network Traffic Log

Saranyan Senthivel
Saranyan Senthivel, ssenthiv@uno.edu

Automatic Forensic Analysis of PCCC Network Traffic Log

A Thesis

Submitted to Graduate Faculty of the
University Of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science
Concentration in Information Assurance

by

Saranyan Senthivel

B.Tech., Anna University, 2012

August 2017

Acknowledgment

I would like to thank my supervisor Dr. Irfan Ahmed, especially for his constant support throughout my study.

I would like to thank Dr. Vassil Roussev and Dr. Minhaz Zibran for serving in my thesis Defense Committee.

My special thanks to my family and wonderful friends who were with me when things were tough, who never left my side and made me cheerful when I didn't have anything to cheer about.

This is a milestone in my journey.

Dedicated to my Parents, Friends & Teachers.

Contents

List of Figures

List of Tables

Listings

Abstract

Most SCADA devices have a 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 thesis 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 shows that more than 30 files are transferred to/from the PLC when downloading/uplloading 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 lo-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.

KEY WORDS

SCADA forensics, SCADA protocol, PCCC, network traffic analysis, programmable logic controller

Chapter 1

Introduction

Supervisor Control And Data Acquisition (SCADA) systems are used to automate industrial physical 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 last two decades, with the increased convergence of data networks SCADA systems are ever more tightly integrated with the TCP/IP infrastructure [2]. Although this standardization of all communication brings substantial economic advantages, it also brings the potential of remote attackers gaining access to inherently insecure devices, and to execute attacks on the physical infrastructure with potentially catastrophic consequences [3,4]. Stuxnet, for instance, is a malware that specifically targets industrial automation systems [5].

SCADA systems generally consist of sensors, actuators, Programmable Logic Controllers (PLCs), and a Human Machine Interface (HMI) [6,7]. A PLC is located at a remote field site to monitor and control a physical process locally. HMI and other SCADA services (such as engineering workstation and historian) run at a control center for remote monitoring and control of physical process.

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 a 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 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.

In this thesis work, we have presented a comprehensive analysis of PCCC (Programmable Controller Communication Commands) protocol for transferring control logic to a PLC. We use Allen-Bradley's RSLogix 500 programming software [8] and Micrologix 1400 PLC [9] 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 performed a detailed analysis of the network traffic of PCCC protocol and discover the entire process of transferring a control logic program to a PLC.
- We exposed several unknown file types (in PCCC traffic) containing significant information of

forensic relevance such as SMTP client configuration, ladder logic program, and other system and network configurations. We have classified these file types according to their content.

- The techniques are realized in 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 mostly in human readable form.

The rest of the document is organized as follows: Chapter 2 presents a detailed analysis of transferring control logic via PCCC protocol. Chapter 3 presents a framework and implementation details of our prototype tool - `Cutter`, followed by chapter 4 that presents the evaluation results. Chapter 5 presents the related work followed by a conclusion in Chapter 6.

Chapter 2

Control Logic Transfer via PCCC

This chapter explores the transfer process of a control logic to a PLC using PCCC protocol. The goal is to identify digital artifacts of forensic relevance in PCCC network traffic log.

PCCC protocol The PCCC is a command/reply protocol and 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.

2.1 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) [1]. 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 [10] 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.

2.1.1 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 network traffic in promiscuous mode for analysis.

2.1.2 PCCC Message Fields

Table 2.1: Description of the fields of PCCC message

Field Name	Size (bytes)	Description
Requestor ID	1	Requestor ID
Vendor ID	2	Vendor ID
Serial Number	4	Serial Number
CMD	1	Command Code
STS	1	Status
TNSW	2	Transaction ID
FNC	1	Function code
PCCC Data	Variable	Data relevant to FNC

Table 2.1 lists the name and size of fields of a PCCC message over Ethernet. The first three fields represent requestor identification for `Execute PCCC` service used to process PCCC commands; the fields are Requestor ID, Vendor ID and Serial Number.

The rest of the fields are `CMD`, `STS`, `TNSW`, `FNC`, and PCCC data related to `FNC` that is analogous to operand and opcode in assembly languages, respectively. `CMD` contains code for command type, and `FNC` is a specific function under a command type. In some cases, `CMD` does not have `FNC` such as `0x01` for unprotected read, and `0x08` for unprotected write. `STS` (1-byte) is a status field. A request message always has `0x00 STS` value. `TNSW` is a (2-byte) transaction identifier. Request and corresponding reply messages share a same `TNSW` value. PCCC data is optional depending on `FNC` code. For instance, `FNC` code `0x03` request diagnostic status to the PLC and does not require any PCCC data. Table 2.2 lists `CMD` and `FNC` codes that are pertinent to our analysis.

Table 2.2: Command and Function Codes

Command Code	Function Code	Description
0x0F	0x80	Change Mode
0x0F	0xAA	Protected typed logical write with three address fields
0x0F	0xA2	Protected typed logical read with three address fields
0x0F	0x8F	Apply Port Configuration
0x06	0x03	Diagnostic Status
0x0F	0x52	Download Completed
0x06	0x00	Echo
0x0F	0x11	Get edit resource
0x0F	0x12	Return edit resource

2.1.3 Change of Operational Mode

A PLC supports different modes of operation such as `PROGRAM`, `RUN` and `TEST` [11]. When a PLC is operating in `RUN` mode, the physical input, output, and program logic are scanned continuously in a defined rate to control its respective physical process. In the `PROGRAM`, PLC stops executing the program logic and disables the scanning or modifying of the state of output ports. In the `TEST` mode, the program is executed but does not affect the output ports.

Our next observation is that, in order to transfer a control logic to/from the PLC, the programming software changes the mode of the PLC from `RUN` to `PROGRAM` mode. When the transfer is completed, the mode is switched back to `RUN`. `FNC` code `0x80` is used to change the mode of PLC to `PROGRAM`, `RUN`, or `TEST`. It only requires one field in PCCC data to mention the code of the mode to change. We find that `0x01` and `0x06` are used for `PROGRAM` and `RUN` modes respectively. The mode-change is particularly useful to delimit the start and end of a logic transfer. Clearly, it could also be used as an indication of logic transfer, however, more scrutiny is required for a forensic evidence since it is possible to switch the modes without transferring any logic.

2.1.4 Control Logic Program

PLC logic programs are written using the programming languages defined in `IEC 61131-3` [12], such as *Ladder Logic*, and *Instruction List*. RSLogix supports only Ladder Logic programming. To download a control logic to a PLC, RSLogix writes to the PLC. Similarly, it reads from the PLC to upload a logic. In PCCC protocol, `FNC` code `0xA2` and `0xAA` are used for reading from and writing to a PLC respectively. These `FNC` codes require multiple fields in PCCC data to be properly set, file type and file number (Table 2.3). Both downloading and uploading

Table 2.3: Sub-fields of PCCC data field for `FNC` code `0xA2` and `0xAA` to read from and write to a PLC

Field Name	**Size (bytes)**	**Description**
Byte Size	1	Number of bytes to read/write
File Number	1	File ID
File Type	1	Represent the file content
Element No.	1	elements within a file
Sub-element No.	1	sub-elements within an element

processes involves transfer of multiple files of different types, such as low-level representation of ladder logic, counter, timer, and configuration files. As already mentioned, the compilation of

ladder logic program does not produce local output on the engineering workstation. However, when the program is downloaded/uploaded to/from the PLC, we analyze the file type field in the messages and find that almost 30 types of files are transferred to the PLC.

Unfortunately, most of these file types (including ladder logic) are *not publicly documented* (Table 2.4). The known file types are described in [1], and contain the data on input/output physical ports, and the data in-use by the program logic such as timer, and counter.

Table 2.4: Association of file-types and their respective codes mentioned in the vendor's manual [1].

File Type	Description
0x03 0x22 0x24 0x47 0x49 0x4C 0x4D 0x60 0x69 0x91 0x92 0x93 0x94 0x95 0x96 0xA1 0xA2 0xE0 0xED	Unknown Type
0x82	Output
0x83	Input
0x84	Status
0x85	Binary bit
0x86	Timer
0x87	Counter
0x88	Control bit
0x89	Integer
0x8A	Floating point
0x8E	ASCII
0x8D	String

2.2 Analysis of Unknown File Types

The goal of this section is to analyze and document the files of unknown type (Table 2.4) based on their contents.

2.2.1 Differential Analysis

The approach we took to classify the files of unknown types is differential analysis [13]. First, we create a baseline where the traffic of a program being transferred is captured and then, processed to extract files. In the next iteration, we make only one change either in ladder logic, configuration, or data in the RSLogix programming software and then, transfer the whole program to the PLC again while capturing the traffic. We extract the files again from the network traffic and compare them with the baseline files using `diff` utility from GNU Diffutils [14]. The file that has been changed should have been identified when comparing with its corresponding baseline file, and the rest of the files should be the same.

2.2.2 Test Cases

We apply our approach to several test cases composed of making changes in many different types of configuration options and data values in RSLogix such as configuration of IP address, enabling DHCP service, name of the processor, and the data values in input, output, timer, counter etc.

Table 2.5: Example test-cases for file type classification

Test Cases			
Data Path	**Original Data Value**	**Modified Data Value**	**Classified File-type**
Data Files/New/select Type:Binary	-	New file B9	0x85
Data Files/New/select Type:Integer	-	New file N10	0x89
Data Files/New/select Type:Long	-	New file L11	0x91
Data Files/New/select Type:Message	-	New file MSG12	0x92
Data Files/New/select Type:PID	-	New file PI13	0x93
Data Files/New/select Type:Programmable Limit Switch	-	New file PLS14	0x94
Data Files/New/select Type:Routing Information	-	New file RI	0x95
Data Files/New/select Type:Extended Routing Information	-	New file RIX	0x96
Controller/Channel Configuration/Channel 1 (tab)/DNP3 over IP Enable (Checkbox)	Unchecked	Checked	0x4D
Controller/Channel Configuration/Channel 0 (tab)/Driver(drop down menu)	DF1 Full Duplex	Shutdown	0x47
Controller/Channel Configuration/Channel 1 (tab)/SMTP Client Enable (Checkbox)/Chan. 1 SMTP	-	SMTP Configuration	0x4C
Controller/Channel Configuration/Channel 1 (tab)/Modbus TCP Enable (Checkbox)	Unchecked	Checked	0x49
Controller/Channel Configuration/Channel 1 - Modbus (tab)/Coils	0	3	0x49
Controller/Channel Configuration/Channel 1 (tab)/SNMP Server Enable (Checkbox)	Unchecked	Checked	0x49
Add New Rung in Ladder Logic (LAD)	I:0/0 and O:0/0	New Timer (T4)	0x03, 0x24, 0x22
Program Files/New/Create Program File	-	New File Number	0x22

Table 2.5 presents some examples of the test cases. It shows the complete path of a value that is modified along with the original and modified values. In some cases, the original values do not exist because they are generating new information such as creating new data file. Also, sometimes a single change may alter multiple files of different file types. For example, when a ladder rung is added to an existing program, we notice the change into the following three file types: `0x03`, `0x24`, and `0x22`.

2.2.3 Results

Table 2.6 presents the results of our findings; file type `0x22` contains low-level representation of ladder logic, while the `0x47`, `0x49`, `0x4C` and `0x4D` contain system configurations. For instance, `0x4C` stores email server name/IP, and user authentication details (i.e. username and password). Our further analysis shows that these details are transferred as plain text over the network, and thus, are prone to eavesdropping.

Table 2.6: Classification of unknown file types

File Type	Classification (based-on content)
0x22	Ladder Logic - Control Logic Program
0x47	DF1 (channel 0) Configuration
0x49	Ethernet Configuration
0x4D	DNP3 Configuration
0x4C	SMTP Configuration
0x92	Message
0x93	PID
0x94	Programmable Limit Switch
0x95	Routing Information
0x96	Extended Routing Information

2.3 Parsing of the Files

To create a parsing tool for the extracted files, we further use differential analysis to examine the file contents closely and identify how the contents are organized in the files. With respect to file size, data files vary significantly from 2 bytes to 512 bytes, while the configuration files always have a fixed size. Table 2.7 shows the observed average file sizes of different types.

Table 2.7: Average size of the files (in Bytes) captured during the control logic transfer

File Type	Description	Average Size in bytes
0x22	Ladder Logic	90
0x47	DF1 (channel 0) Configuration	180
0x49	Ethernet Configuration	532
0x4D	DNP3 Configuration	204
0x4C	SMTP Configuration	1800
Data Files	Input, Output, Timer, Counter, Integer, Status etc.	2 to 512 bytes

2.3.1 Main Configuration File

The file type `0x03` is the main configuration file containing information about the other files being transferred to a PLC. Figure 2.1 presents (in hexadecimal) the content of an example configuration file. The first two bytes provide the length of the configuration, followed by the PLC processor name (`UNTITLED` in this case) and the information about ladder file `0x22`, other configuration files such as `0x49, 0x4C, 0x4D, 0x47` and the data files.

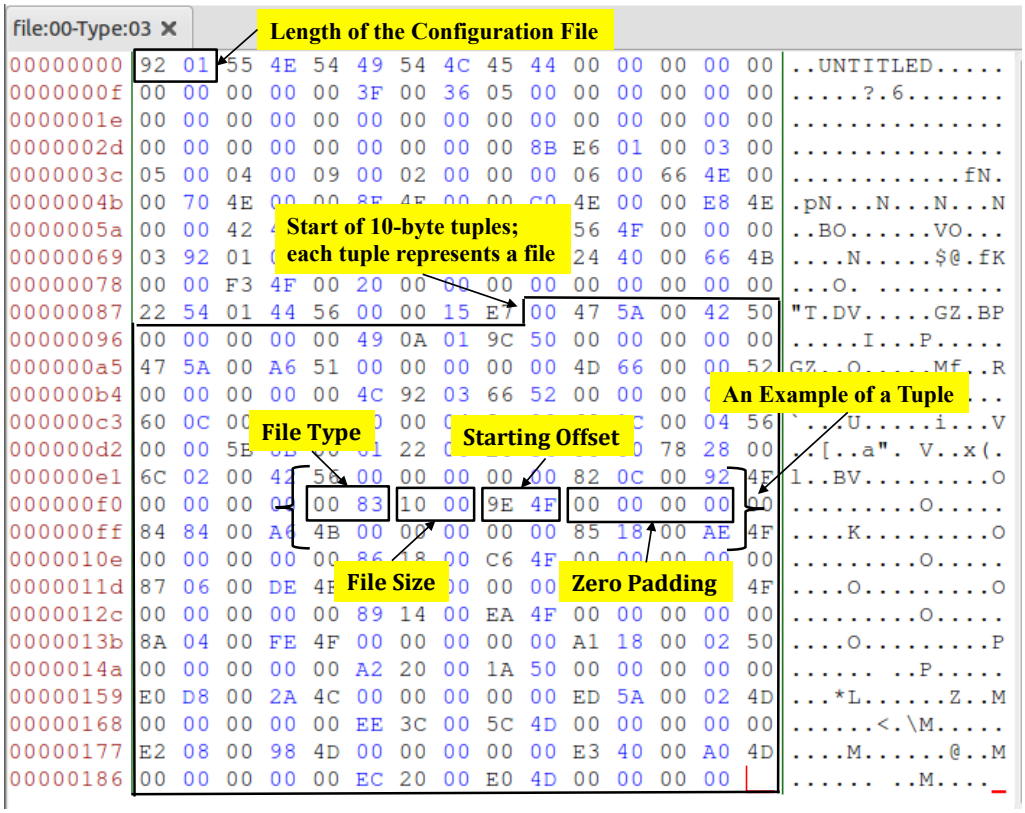


Figure 2.1: Configuration file field format

A 10-byte structure stores information about each file. The first 2-bytes identify the file type, such as `0x82, 0x83, 0x84, and 0x85`; the third and fourth bytes give the size of the file, followed by two bytes containing the starting offset of the file used with the ladder logic instructions in the file `0x22`. The remaining bytes 7-10 are filled with zeroes.

2.3.2 SMTP File

The file type `0x4C` is the SMTP configuration file; Figure 2.2 shows an example SMTP file, which has a fixed size of 1800 bytes. The first 16 bytes contain the signature bits followed by fourteen 64-byte fields. Each field is organized into two sub-fields: length and data. The length field consists of two bytes containing the size of the data in the data field. Since the data in the data field may vary, the record is padded with zeros. The SMTP fields appear in the following sequence in the file: Username, Password, SMTP Server, `From` Address, and 10 `To` Address fields.

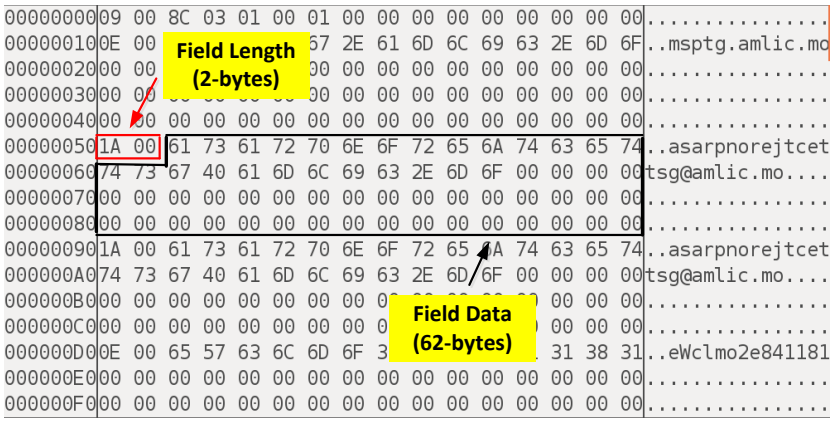


Figure 2.2: SMTP field format

2.3.3 Data Files

Several data files are transferred while uploading/downloading a logic program. Figure 2.3 shows the content of an example *Binary* file with a type of `0x85`; it has 12 elements from (B3:0/1 to B3:0/11). Similarly, Figure 2.4 shows the content of an *Integer* file: its type is `0x89`, and has ten elements from N7:0/1 to N7:0/9.

Figure 2.5 shows an example Timer file, type `0x86`. The file contains 4 timers; each timer is configured with the parameters, Base, Preset (Pre), and Accumulator (ACC).

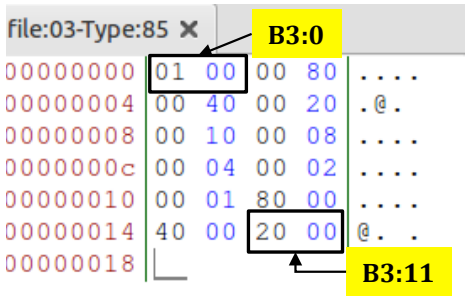


Figure 2.3: Binary File - Hex Format

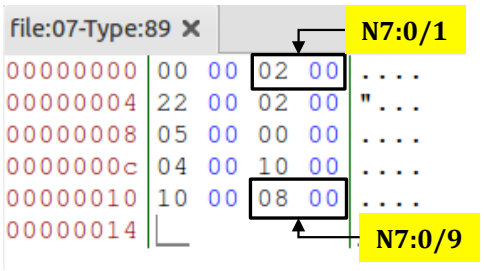


Figure 2.4: Integer File - Hex Format

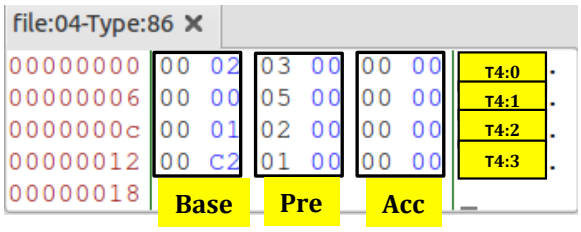


Figure 2.5: Timer File - Hex Format

Chapter 3

Cutter - A Prototype Network Forensic Tool

Based on our findings in the last chapter, we built a prototype tool `Cutter` to extract digital artifacts from a PCCC network traffic log. The tool is implemented in Python using the PyShark package, which allows the use of Wireshark dissectors for decoding packet content. The tool consists of five functional modules: parsing of PCCC messages, identification of the boundary of a logic transfer, message filtering, reassembling of the messages into files, and analyzing/parsing files to extract information.Figure 3.1 shows the working mechanism of `Cutter`. The tool will be available at [15].

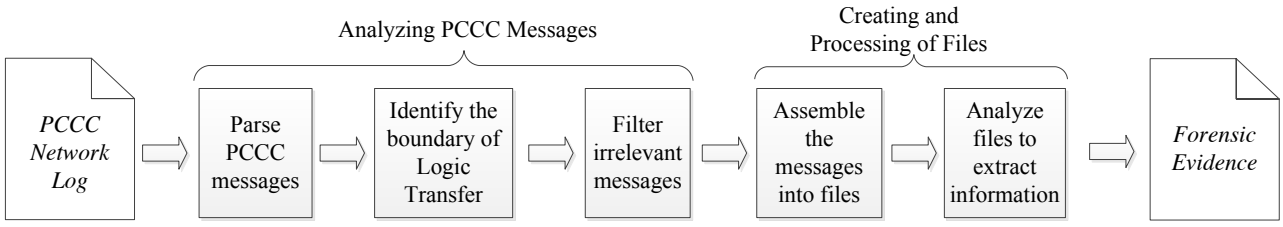


Figure 3.1: The working mechanism of `Cutter`.

3.1 Goal and Assumptions

Given a network packet capture, our goal is to identify the traces of logic transfer to PLC, and further extract the whole logic program and relevant data to files on disk, if the traces are found. To achieve my goal, we made the following assumptions about the network capture, We analyze.

- *PCCC protocol is used for transferring the logic program to PLC.* `Cutter` is a prototype tool to demonstrate network forensic capability for a SCADA protocol. For this work, we use PCCC protocol but the tool can further be extended to support other protocols in similar

fashion.

- *PLCs change their mode of operation when downloading logic program.* This assumption is often valid because PLCs maintain separate mode of operations for running normally, debugging, and updating firmware.

- *PLCs implement the PCCC message format correctly.* This assumption must generally hold true to avoid any confusions in communication among PLCs and SCADA services at control center. If variants of a protocol exist, a fingerprinting technique may be used to identify the version of the protocol to parse the network traffic correctly. However, it is not in the scope of our work to develop a fingerprinting technique for PCCC.

3.2 Implementation

3.2.1 PCCC message parsing

The PCCC message is located at the application layer in TCP/IP stack along with EtherNet/IP and CIP headers. In order to reach to the PCCC message content, the tool skips the packet headers of lower layers and the EtherNet/IP and CIP headers in the application layer. Since Wireshark lacks the dissector for PCCC, the tool implements its own parser to process the PCCC message contents.

3.2.2 Identifying the boundary of control logic transfer

The tool starts from the first packet in the network traffic log, and searches for specific PCCC messages used for changing the mode of PLC from `PROGRAM` to `RUN`, and vice versa. Specifically, the PCCC uses `CMD` code `0x0F`, and `FNC` code `0x80` for changing the mode. The first message during the search represents start of the transfer, and the occurrence of the second message depicts end of the transfer. Listing 3.1 presents the pseudocode for identifying the boundary of logic transfer.

3.2.3 Message filtering

Within the boundary, a number of PCCC messages exists that are irrelevant to the recovery of files. These are mostly *read* and *echo* commands for retrieving updated data from the PLC. The

```
for j = 0 to req_pktcount do
    if req_pkts[j][5] == "0x80" then
      chng_mode_detect <-- req_pkts[j][0]
    end if
end for
```

Listing 3.1: Pseudocode of identification of boundary of logic transfer

tool filters out these messages, and only focuses on the messages that are writing to the PLC. Listing 3.2 presents the pseudo-code of the filtering process. It shows that the packets starting with the command code `0x0F`, *request* message, are discarded, as are the corresponding *response* messages (`0x4F`). `0x06` and `0x46` are *echo* and *echo response* packets, respectively, and are also dismissed.

```
for i = 0 to pktcount do
  if allpkts[i][0] == '0x0F' then
    req_pkts <-- allpkts[i]
  else if allpkts[i][0] == '0x4F' then
    res_pkts <-- allpkts[i]
  else if allpkts[i][0] == '0x06' then
    echo_pkts <-- allpkts[i]
  else allpkts[i][0] == '0x46' then
    echo_res_pkts <-- allpkts[i]
  end if
end for
```

Listing 3.2: Pseudocode of packet filtering

3.2.4 Assembling of packets into files

`Cutter` considers the packets for further processing that has `CMD` code `0x0F`, and `FNC` code `0xAA` and are used for (protected-typed logical) write operations. The file type and file number fields are used to represent a unique file for writing. `Cutter` uses these fields to assemble the data into their respective files. While processing the packets, when `Cutter` finds a new combination, it creates a new file on disk with the name containing file type and number. If the tool encounters a packet for a file already existed, it appends the packet contents in the relevant file on disk. Listing 3.3 presents the pseudocode of the assembling process.

```
void print_details(req_pkt,res_pkt,pkt_boundary,filepath){
  if req_pkt[5] == "0xAA" then
   filename = filepath+"/download-"+
              str(pkt_boundary)+
              str(req_pkt[7])+"-Type:"+
              str(req_pkt[8])
   if not path_exists(filename) then
     makedirectory(filename)
   end if
  end if

  with open(filename,'append')
   for buffer in req_pkt[11:]
     filename.write(buffer.decode('hex'))
   end for
}
```

Listing 3.3: Pseudocode of assembling of packets into files

3.2.5 Analyzing files to extract information

`Cutter` parses each file and extract any useful information based on the analysis discussed in the prior chapter.

Chapter 4

Evaluation

4.1 Experimental Settings

Figure 4.1 illustrates the experimental environment. It consists of an Allen Bradley's PLC, Micrologix 1400 series B, and four virtual machines (VMs). The PLC has input and output physical ports. The input ports connect with two push buttons and two toggle switches to provide digital inputs to the PLC. The output ports are connected with four different color LEDs: red, orange, green, and blue.

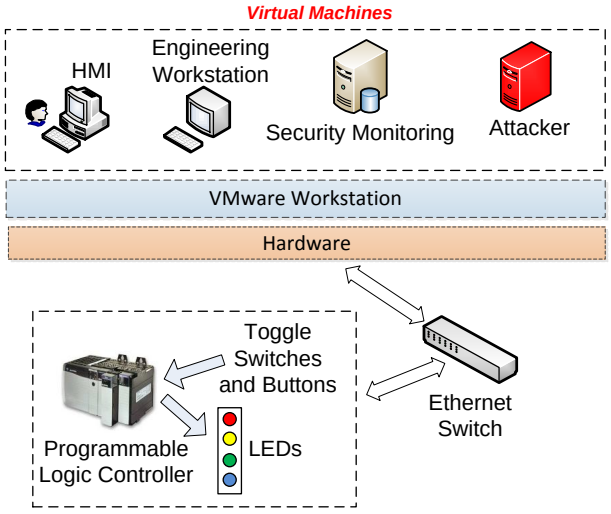


Figure 4.1: The experimental setup for the evaluation of `Cutter`.

The PLC and the physical computer are connected via an Ethernet switch. Two VMs are running SCADA services – human machine interface software, and engineering workstation running RSLogix. One of the VMs is for security monitoring and is running Wireshark to capture all network traffic (in promiscuous mode); the last VM is a simulated attacker's machine that can communicate with the PLC and send messages to transfer logic program and alter physical process state (LEDs

in this case).

Table 4.1 shows the system configuration of VMs and host machine used in the evaluation

Table 4.1: System Configuration of virtual machines (VM) and host physical machine used in evaluation

System	**OS**	**Machine Type**	**Bits/Cores/ RAM/HDD**
Host Machine	Win 10	Physical Machine	64Bit/4/8GB/420GB
Engineering Workstation	Lubuntu	VM for Cutter and Wireshark	64Bit/1/4GB/50GB
Security Monitoring	Win 7	VM for RSLogix	64Bit/4/2GB/40GB

4.2 Comparing two Ladder-logic files

SCADA owners/operators can use `Cutter` to maintain baseline original files, which can later be used to facilitate a forensic investigation. For instance, if a an engineering workstation on control network is compromised, and the PLC programming software installed on it is used to modify the control-logic of a remote PLC, the captured network traffic can be analyzed with `Cutter` to extract files, and compare them with the baseline files. Any deviation can be used as potential indicator of compromise.

To evaluate this scenario, we create a legitimate control logic program in RSLogix and transfer it to the PLC from the engineering workstation while capturing the packets. `Cutter` takes the network log as an input and extracts the original files. Later, we transfer a completely different control logic from an attacker's machine to PLC, and capture the network traffic.

`Cutter` analyzed the network traffic, extracted the files, and then compared them with the baseline files obtained initially from the normal network traffic. It correctly identified that files of types `0x03, 0x24, 0x02, 0x49, 0x83, 0x22, 0x84`, and `0x86` has been modified.

In other words, `Cutter` is able to detect the attack effectively by: a) identifying the transfer of control logic in the network traffic, and b) showing the file differences with respect to the baseline capture.

4.3 Comparing two SMTP Files

We evaluate the `Cutter`'s parsing ability for an SMTP file. We enable the SMTP option and transfer the program to the PLC and capture the network traffic. The evaluation results are tabulated in Table 4.2. It shows that `Cutter` is able to parse the SMTP file accurately.

Field Name	**Given Value**	**Parsed Correctly?**
Email Server	smtp.gmail.com	Yes
From Address	saranprojecttest@gmail.com	Yes
Username	saranprojecttest@gmail.com	Yes
Password	**************************	Yes
To Address[0]	xyztest@gmail.com	Yes
To Address[1]	somebigemailaddress@gmail.com	Yes
To Address[2]	someducationinstitueaddress@uno.edu	Yes
To Address[3]	testuno@uno.edu	Yes
To Address[4]	testsyahoo@yahoo.co.us	Yes
To Address[5]	thesmallemail@hotmail.com	Yes
To Address[6]	test1@aol.com	Yes
To Address[7]	test2@drmcet.ac.in	Yes
To Address[8]	test2scada@gmail.com	Yes
To Address[9]	tester@outlook.com	Yes

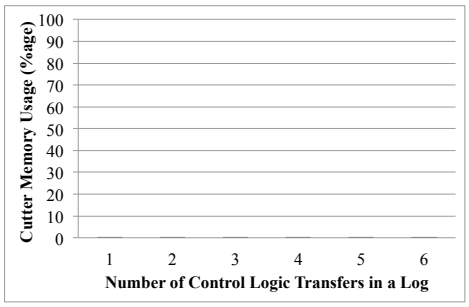
Table 4.2: Accuracy of `Cutter` for parsing an SMTP file

We further use the `Cutter` to compare two similar SMTP files in a scenario where an attacker adds his email address in the SMTP configuration and download it to the PLC. As a result, the PLC starts sending email notifications to the attacker.

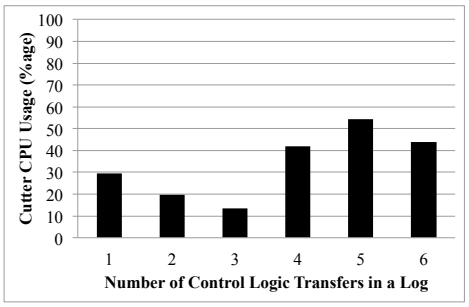
To create the scenario, we modify the SMTP configuration, add a different email address, and transfer the program to the PLC. While transferring the configuration to the PLC, the network packets are captured and processed by `Cutter`. By comparing the SMTP entries with original baseline entries, we are able to identify the different (suspicious) email entry in SMTP configuration file.

4.4 Performance Evaluation

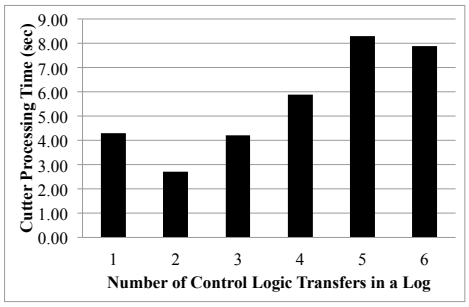
This section discusses the processing speed, CPU and memory usage of `Cutter`. Figures 4.2a, 4.2b, 4.2c, and 4.2d present the evaluation results. The packets are captured while transferring a logic program to the PLC. Multiple network dumps are created with increasing number of control logic programs to be transferred to the PLC.



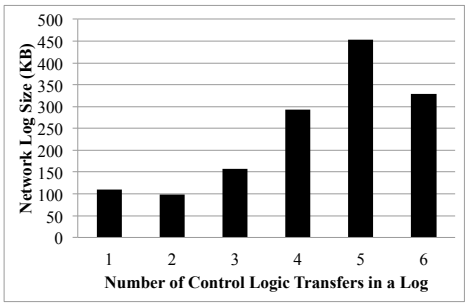
(a) Memory usage of `Cutter` when processing network packet capture files containing different number of logic-program transfers.



(b) CPU usage of `Cutter` when processing network packet capture files containing different number of logic-program transfers.



(c) Processing time of packet network capture containing different number of logic-program transfers.



(d) Size of network packet capture files containing different number of logic-program transfers.

Figure 4.2: Performance Evaluation of `Cutter`

`Cutter` takes around three to eight seconds to process a network capture of size around 100 to 450 kilobytes. Also, `Cutter` is not a resource-intensive tool, which has a small memory footprint and consumes around 15 to 60% CPU. It is worth mentioning that the current implementation of `Cutter` does not support multi-threading, and thus, the performance of `Cutter` can further be improved.

Chapter 5

Related Work

As early as 2006, Igure *et al.* [16] analyzed the emergent landscape of security challenges for SCADA systems in the face of accelerating integration with TCP/IP networks: a) *access control*–it is difficult to enforce define and enforce access control policies for resource-constrained devices; b) *firewalls and IDS*–developing protocol-aware firewall and IDS rules requires detailed knowledge of the operation and vulnerabilities of the protocol; c) *protocol vulnerability assessment* requires scarce domain knowledge and judgement; d) *cryptography and key management*–it is a challenge to reconcile the use of strong cryptographic mechanisms with the overriding safety priority of SCADA devices; e) *device and OS security*–the limited capabilities of the employed hardware make it inherently less capable of handling denial-of-service attacks that can have catastrophic consequences; f) *security management*–SCADA systems tend to have a much longer (15-20 year) life cycle, which makes it challenging to maintain up-to-date firmware, especially for devices no longer in production.

The Distributed Network Protocol (DNP3) is the predominant SCADA protocol in the North American energy sector and is in used by more than 75% of utilities. East *et al.* [17] provide a detailed analysis of the DNP3 protocol layers with respect to threats and targets, and identifies 28 attacks and 91 attack instances. The effects of the attacks range from obtaining network or device configuration data to corrupting outstation devices and seizing control of the master unit. The developed taxonomy considers attacks that are common to the three layers common to all implementations–the data link, pseudo-transport, and application. The impact of the attacks can be loss of confidentiality, loss of awareness, and loss of control.

The Modbus family of protocols is widely used in industrial control applications, especially for pipeline operations in the oil and gas sector. Modbus defines the message structure and communication rules used by process control systems to exchange SCADA information for operating and controlling industrial processes. [18] built an attack taxonomy and, similar to [17], classify the impact into loss of confidentiality/awareness/control. In particular, the authors developed 20

distinct attacks against the Modbus serial variant of the protocol, and 28 distinct attacks against the Modbus TCP version.

Kleinmann and Wool [19] present a DFA (Deterministic Finite Automaton) based intrusion detection system for the network traffic of S7comm (S7 Communication). S7comm [20] is a proprietary protocol for Siemens S7-300/400 family. The IDS is designed based on the observation that S7 traffic that is coming to/from a PLC is highly periodic. It acheives the accuracy of 99.26%.

Wireshark [21] is the leading tools for interactive network packet analysis. It can parse packets from numerous network protocols and can reconstruct protocol conversations, such as TCP streams. The data can be viewed in variety of formats like ASCII, EBCDIC, HEX Dump, C Arrays and Raw.

For industrial networks, *NetDecoder* [10] is among the most popular analytical tools. It is designed to diagnose and troubleshoot communication problems in industrial Networks. Some of the Ethernet protocols supported by NetDecoder are Modbus/TCP, EtherNet/IP (CIP and PCCC) [22], Allen-Bradley's CSP/PCCC, DNP3 over Ethernet [23], IEC 60870-5-104 [12], PROFINET [24], CC-Link IE.

Chapter 6

Conclusion and Future work

In this work, we presented a detailed analysis of the PCCC protocol employed by SCADA networks. Prior to this effort, only partial information was made available by the vendors, which was insufficient to build meaningful security and forensics applications. Starting with incomplete information, we systematically applied a differential analysis technique to reverse engineer the structure and format of the protocol messages to the point where useful information can be extracted from the network capture. Specifically, our proof-of-concept tool, `Cutter`, can parse the content of PCCC messages, extract digital artifacts and present them in human-readable form such as SMTP configuration. The evaluation results show that `Cutter` is useful in identifying any transfer of control logic to the PLCs, extract and store digital artifacts into files on disk and compare them from previously-stored normal files. `Cutter` is lightweight that does not require significant memory and CPU to work effectively.

6.1 Limitations

- `Cutter` can be used to do forensic analysis on PCCC only
- Parsing of ladder program is not programmed to support the relative addressing.

6.2 Future work

In addition to the parsing of relative addressing capabilities as a future work we are planning to take `Cutter` to support other industrial protocols like Modbus, DNP3 etc., This can be achieved by reverse engineering the protocols and educating `Cutter`

Bibliography

[1] Allen Bradley's DF1 protocol and command set, reference manual, http://literature.rockwellautomation.com/idc/groups/literature/documents/rm/1770-rm516_-en-p.pdf (2017).

[2] I. Ahmed, S. Obermeier, M. Naedele, G. G. Richard III, SCADA Systems: Challenges for Forensic Investigators, Computer 45 (2012) 44–51. doi:doi.ieeecomputersociety.org/10.1109/MC.2012.325.

[3] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A.-R. Sadeghi, M. Maniatakos, R. Karri, The Cybersecurity Landscape in Industrial Control Systems, Proceedings of the IEEE 104 (5) (2016) 1039–1057.
URL http://dblp.uni-trier.de/db/journals/pieee/pieee104.html#McLaughlinKWDSM16

[4] M. Robinson, The SCADA Threat Landscape, in: Proceedings of the 1st International Symposium on ICS & SCADA Cyber Security Research 2013, ICS-CSR 2013, BCS, UK, 2013, pp. 30–41.
URL http://dl.acm.org/citation.cfm?id=2735338.2735342

[5] R. Langne, To Kill a Centrifuge - A Technical Analysis of What Stuxnet's Creators Tried to Achieve, Tech. rep. (2013).

[6] K. A. Stouffer, J. A. Falco, K. A. Scarfone, SP 800-82. Guide to Industrial Control Systems (ICS) Security: Supervisory Control and Data Acquisition (SCADA) Systems, Distributed Control Systems (DCS), and Other Control System Configurations Such As Programmable Logic Controllers (PLC), Tech. rep., Gaithersburg, MD, United States (2011).

[7] T. Macaulay, Cybersecurity for Industrial Control Systems: SCADA, DCS, PLC, HMI, and SIS, 1st Edition, Auerbach Publications, Boston, MA, USA, 2012.

[8] RSLogix500, http://www.rockwellautomation.com/rockwellsoftware/products/rslogix500.page (2017).

[9] MicroLogix 1400 Series B, http://ab.rockwellautomation.com/Programmable-Controllers/MicroLogix-1400 (2017).

[10] NetDecoder, http://www.fte.com/products/NetDecoder.aspx.

[11] F. Swainston, A Systems Approach to Programmable Controllers, Butterworth-Heinemann, Newton, MA, USA, 1991.

[12] IEC 61131-3:2013, https://webstore.iec.ch/publication/4552.

[13] S. Garfinkel, A. J. Nelson, J. Young, A general strategy for differential forensic analysis, Elsevier (2012) S50 – S59doi:10.1016/j.diin.2012.05.003.

[14] GNU Diffutils, https://www.gnu.org/software/diffutils/ (2017).

[15] Cutter tool, https://github.com/ahmirf/cutter (2017).

[16] V. M. Igure, S. A. Laughter, R. D. Williams, Security issues in SCADA networks, Computers & Security 25 (7) (2006) 498 – 506. doi:http://dx.doi.org/10.1016/j.cose.2006.03.001.
URL http://www.sciencedirect.com/science/article/pii/S0167404806000514

[17] S. East, J. Butts, M. Papa, S. Shenoi, A Taxonomy of Attacks on the DNP3 Protocol, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 67–81. doi:10.1007/978-3-642-04798-5_5.
URL http://dx.doi.org/10.1007/978-3-642-04798-5_5

[18] P. Huitsing, R. Chandia, M. Papa, S. Shenoi, Attack taxonomies for the Modbus protocols, International Journal of Critical Infrastructure Protection 1 (2008) 37 – 44. doi:http://dx.doi.org/10.1016/j.ijcip.2008.08.003.
URL http://www.sciencedirect.com/science/article/pii/S187454820800005X

[19] A. Kleinmann, A. Wool, Accurate modeling of the siemens S7 SCADA protocol for intrusion detection and digital forensics, Journal of Digital Forensics, Security and Law (JDFSL) 9 (2) (2014) 37 – 50.
URL http://ojs.jdfsl.org/index.php/jdfsl/article/view/262

[20] S7comm wire-shark dissector plugin, (http://sourceforge.net/projects/s7commwireshark).

[21] Wireshark, https://www.wireshark.org/ (2017).

[22] EtherNet/IP, https://www.odva.org/Technology-Standards/EtherNet-IP/Overview (2017).

[23] DNP3, http://us.profinet.com/technology/profinet/.

[24] PROFINET, http://us.profinet.com/technology/profinet/.

Vita

Saranyan Senthivel received his Bachelor Degree in Information Technology from Anna University, India in 2012. He joined the University of New Orleans for Computer Science Master of Science program in Fall 2015. He started working as a Research Assistant in Cy-Phy Laboratory at University of New Orleans under the supervision of Dr. Irfan Ahmed. He have been doing active research in Cyber Physical Systems and analysing on the Network protocols used by these systems. His research interest includes SCADA Forensics, Network Forensics, Memory Analysis, and Reverse Engineering.