2017

Leveraging the SRTP Protocol for Over-the-Network Memory Acquisition of a GE Fanuc Series 90-30

George Denton
University of New Haven

Filip Karpisek
Brno University of Technology

Frank Breitinger
University of New Haven, frank.breitinger@unil.ch

Ibrahim Baggili
University of New Haven, ibaggili@newhaven.edu

DFRWS 2017 USA — Proceedings of the Seventeenth Annual DFRWS USA

Leveraging the SRTP protocol for over-the-network memory acquisition of a GE Fanuc Series 90-30

George Denton [a], Filip Karpisek [b], Frank Breitinger [a, *], Ibrahim Baggili [a]

[a] Cyber Forensics Research and Education Group (UNHcFREG), Tagliatela College of Engineering, ECECS, University of New Haven, 300 Boston Post Rd., West Haven, CT, 06516, USA
[b] Faculty of Information Technology, Brno University of Technology, Czech Republic

A B S T R A C T

Keywords:
GE Fanuc Series 90-30
Live memory acquisition
GE-SRTP protocol
SCADA
PLC

Programmable Logic Controllers (PLCs) are common components implemented across many industries such as manufacturing, water management, travel, aerospace and hospitals to name a few. Given their broad deployment in critical systems, they became and still are a common target for cyber attacks; the most prominent one being Stuxnet. Often PLCs (especially older ones) are only protected by an outer line of defense (e.g., a firewall) but once an attacker gains access to the system or the network, there might not be any other defense layers. In this scenario, a forensic investigator should not rely on the existing software as it might have been compromised. Therefore, we reverse engineered the GE-SRTP network protocol using a GE Fanuc Series 90-30 PLC and provide two major contributions: We first describe the Service Request Transport protocol (GE-SRTP) which was invented by General Electric (GE) and is used by many of their Ethernet connected controllers. Note, to the best of our knowledge, prior to this work, no publicly available documentation on the protocol was available affording users' security by obscurity. Second, based on our understanding of the protocol, we implemented a software application that allows direct network-based communication with the PLC (no intermediate server is needed). While the tool's forensic mode is harmless and only allows for reading registers, we discovered that one can manipulate/write to the registers in its default configuration, e.g., turn off the PLC, or manipulate the items/processes it controls.

© 2017 The Author(s). Published by Elsevier Ltd. on behalf of DFRWS. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Introduction

A Supervisory Control And Data Acquisition (SCADA) system is a remote monitoring and control unit that operates with coded signals over a communication channel used in a variety of applications, e.g., in prisons to operate cell doors, in dams to open or close gates, or in gas transmission for pressure regulation. They have become systems intertwined with our daily lives and are used in domains like power generation, water treatment and civil defense to name a few. One important component of SCADA systems is a Programmable Logic Controller (PLC). PLCs are digital devices usually used for automation of industrial/mechanical/electrical processes. Typical applications include control of machines in factories and amusement park rides.

Given their widespread, PLCs have become a common target for attackers. For instance, Zetter (2011) reported in 2011 that "34 exploits were published by a researcher on a computer security mailing list and targeted seven vulnerabilities in SCADA systems produced by Siemens, Iconics, 7-Technologies and DATAC". These exploits can be employed by worms and were demonstrated by Spenneberg et al. (2016) at Black Hat Asia. "The worm scans the network for new targets (PLCs), attacks these targets and replicates itself onto the found targets. The original main program running on the target is not modified." Another prominent example was Stuxnet (Langner, 2011).

In our paper we analyzed one specific PLC, the GE Fanuc Series 90-30 from General Electric (GE) which was introduced to the automation market almost 30 years ago.[1] It has been installed in a number of systems worldwide, and will at some point be discontinued and replaced with a new line with enhanced security features. Of note, however, is that it would be quite a feat for

---

[1] Manual's date is 1990 and can be found here: http://plcproducts.com/sites/pl/files/manuals/ge_fanuc/ic610chs110a_ge_fanuc_series_one_user_manual.pdf.

organizations worldwide to transition to a new system for both financial and technical reasons.

As most PLC software is proprietary, our research focused on understanding the communication protocol called GE-SRTP and its security mechanisms. The protocol is used to interface the controllers to Human Machine Interface (HMI) computers. While our testing involved the GE Fanuc Series 90-30, our findings apply to all PLCs that employ GE's proprietary communication protocol known as GE-SRTP.

To perform our assessment, we set up an environment which consists of the GE Fanuc Series 90-30 PLC and a machine to sniff/analyze the network traffic. This allowed us to reverse engineer the GE-SRTP protocol and to understand the request and response bit type messages to and from the GE Fanuc Series 90-30. More specifically, the research focused on acquiring live memory from the PLC with the mindset of minimizing the effect on the process being operated whilst bypassing the HMI (valid commands are directly sent to the PLC). While our tests did not show any impact, there may be a chance that our tool negatively impacts the process and the performance in a production environment. Our main goal was to construct a method/forensic tool which will allow an end user to analyze the memory in case the PLC or HMI software is compromised. Thus, in case of a cyber attack, an incidence response team would not have to rely on the HMI software. More significantly, during our experimentation we found that commands such as opening/closing a motorized valve, opening/closing a circuit breaker, or suddenly starting/stopping a turbine can be executed by an attacker if the device is in its default configuration. These actions can lead to equipment damage with repair cost ranging from thousands to millions of dollars or fatality of workers.

The GE Fanuc Series 90-30 platform was selected for evaluation because of multiple years of experience maintaining, troubleshooting, and writing ladder logic by the research team as well as its wide adoption. Our intention is not to cause any harm to the manufacturer of the product but to create the much needed awareness of the security issues, as well as practical methods for the PLC's incidence response.

The remainder of this paper is structured as follows: Sec. GE Fanuc Series 90-30 and its memory describes the various memory types in the GE Fanuc Series 90-30. In Sec. Testing environment setup we briefly present our testing environment followed by the methodology in Sec. Methodology. The core of this paper is Sec. Understanding the GE-SRTP protocol where we describe the GE-SRTP protocol. Our tool which is capable of reading and writing to the PLC memory is described in Sec. Tool. Sec. Related work summarizes the related work. The last three sections start with a discussion of our findings (Sec. Discussion), followed by the future work (Sec. Future work) and lastly the conclusion (Sec. Conclusion).

GE Fanuc Series 90-30 and its memory

A PLC consists of several registers which can be programmed as well as accessed using a HMI. A HMI is commonly a piece of software running on a PC. In the following we summarize the memory prefix types supported by the GE Fanuc Series 90-30 which are used in our experiments. They are necessary to understand our application and will be referenced throughout the paper. The memory register information was pulled from GE's reference manual GFK-0467M (GE Fanuc Automation North America, Inc, 2002). Attached to the memory prefix is usually a number to indicate the exact address location of the memory, e.g., %R581.

1. Register Memory: The prefix %R is used to assign system register references used to store data in the program such as set points.

2. Analog Input Memory: The prefix %AI is used to represent an analog input references that stores an analog input data from a field device.
3. Analog Output Memory: The prefix %AQ is used to represent an analog output references that holds an analog output value.
4. Discrete Input Memory: The prefix %I represents input references. The references are located in the input status data table that stores the state of all input modules received during the input scan.
5. Discrete Output Memory: The prefix %Q represents physical output references that are stored in the output status table. The values from the table are sent to the output modules during the output scan.
6. Discrete Temporary Memory: The prefix %T represents temporary references. They can be used in the program many times. These are temporary values which mean it can be lost during power failure or transition between the PLC run mode and stop mode.
7. Discrete Momentary Memory: The prefix %M represents internal references.
8. System Memory: The prefix %S represent system status references used to access timers, scan information, fault information in the PLC. The system references include %SA, %SB, and %SC.
9. Discrete Global Memory: The prefix %G represents global data references. These references are used to access contact and coil statuses shared by multiple PLCs.

Testing environment setup

Our testing environment was composed of the following hardware and software components:

1. GE Fanuc Series 90-30 (5-Slot Base IC693CHS397C, CPU 331, 120/240VAC Power Supply IC693PWR321X (includes Serial port), and CMM321 Ethernet Interface).
2. Netgear Prosafe 16 port 10/100 Switch including Category 5 cables.
3. HP Compaq NC6400 laptop.
4. Software packages: MS Excel, Proficy Machine Edition 6.0, Wonderware Intouch v9.5, Wireshark (including a plugin that we created), Wonderware IO servers for Host Communications version 8.1.101.0. All software was running on a single laptop.

Connecting the mentioned devices leaded to the topology shown in Fig. 1. In the following we describe these components in some more detail.

HMI computer. A laptop running Windows XP is used by system operators to communicate with the PLC.[2] In our experiment, a simulated HMI was created with Wonderware Intouch v9.5. The communication between the GE Fanuc Series 90-30 and the workstation requires Wonderware IO server which can have different input sources. For our experiments, we used the Wonderware Intouch software as well as Microsoft Excel and the Dynamic Data Exchange (DDE[3]) protocol. On the network layer (Wonderware IO server to the GE Fanuc Series 90-30), the GE-SRTP protocol is utilized to transport the data. Fig. 2 provides an overview of the communication flow.

GE Fanuc Series 90-30. The PLC system is made up of a CPU controller, Input/Output (I/O) modules, power supply module,

---

[2] An older operating system was used because Proficy Machine Edition and Wonderware Intouch v9.5 is only compatible with this system.

[3] "DDE allows one program to subscribe to items made available by another program, for example a cell in a Microsoft Excel spreadsheet, and be notified whenever that item changes". More information can be found in Microsoft's specification Microsoft (N/A) or on https://en.wikipedia.org/wiki/Dynamic_Data_Exchange.
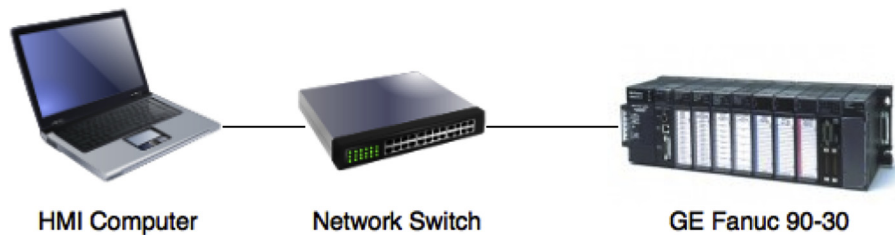
Fig. 1. The network topology used for setting up the simulated SCADA system.
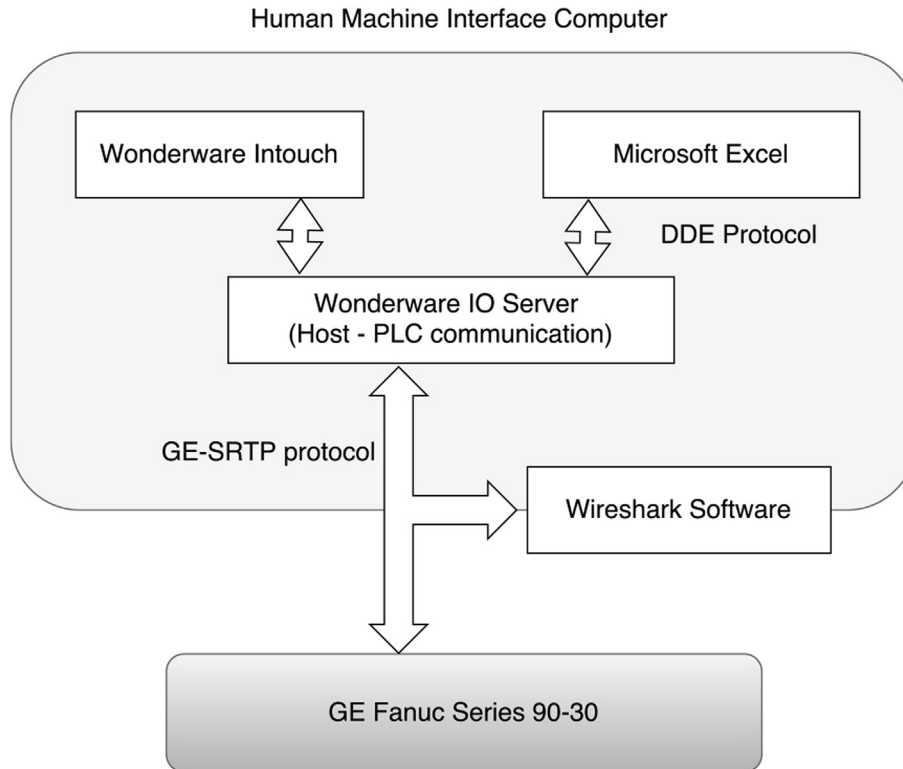

Fig. 2. Diagram of the signal flow between the HMI computer programs and the PLC.

communication module, and a baseplate. A type 331 CPU controller was used for this research. It has 6 KB of RAM, a 10 MHz processor, and a 80188 microprocessor. Next, the ladder logic as depicted in Fig. A.6 (Appendix A) was created in Proficy Machine Edition software and placed on the PLC via a serial port (built into the power supply module). The Ethernet port on the communication module is used to connect the system to a TCP/IP network. All the power required for the PLC components mounted on the baseplate is supplied by the power supply module wired to a 120 VAC external source.

Starting point. For initial tests, we utilized an Excel macro as it allowed us to read and write to PLC memory efficiently – we were able to write to various memory types in the PLC in less than a minute.[4] Configuring the Wonderware IO server to communicate with the PLC involved defining an application name, topic name, and item name. These names were also used in the Excel macro to initiate a DDE conversation with the Wonderware IO server. Once the conversation was established, the IO server responded to the Excel macro client with the PLC memory data requested (note,

there is no direct connection to the PLC). The client can either read or write to memory.

Based on our understanding of the protocol, the initialize bit streams of all zeros were exchanged between the master device (HMI) and the PLC slave to start communication between the devices. After the connection was verified, we immediately started to request different PLC memory types with an Excel macro running on the HMI computer.

Privilege level. The GE Fanuc Series 90-30 has four password protected privilege levels ranging from 1 to 4 which are eight digits max length ASCII passwords. By default a privilege level is not assigned but must be set by the customer. Each privilege level restricts the master to certain tasks. For instance, with a privilege level 2, the master can write to system memory, toggle force system memory, clear fault table, set PLC time and change PLC state. Privilege level 1 allows tasks such as but not limited to reading system memory, reading task memory, returning fault table, program store, and returning control program name. At privilege level 3, the controller ID can be set and a program may be uploaded. Depending on the PLC firmware version, privilege level 4 is required to set controller ID instead of level 3. Our experiments are carried out using the default configuration with no password. We argue that if a system has been running over several years,

---

[4] To conduct a similar task with the HMI, we would have to build multiple graphics on the HMI page to read PLC memory and add push buttons to write to memory. The HMI can only modify/read some of the memory types.

administrators have not changed the configuration and the communication still runs with no authentication. However, we hope to analyze the impact of passwords in the future.

Methodology

As stated in Sec. Testing environment setup, the GE Fanuc Series 90-30 can be controlled on the application layer by the DDE protocol through Excel or using a HMI created with Wonderware Intouch. While our experiments started by having a Wonderware IO server and then sending commands using Excel/the DDE protocol, we realized that the server keeps record of all changes and immediately reports them back to the HMI software. Thus, we decided to focus on the network layer and the GE-SRTP protocol. Here, the challenge was that the GE-SRTP protocol is a proprietary protocol and therefore no documentation (how the protocol works or the structure of the protocol) is available to the public at the time of conducting this research.

In order to construct our software, we first reverse engineered the protocol which was mainly conducted by using Wireshark and capturing all network traffic; the setup network topology remained untouched as depicted in Fig. 1. To analyze the GE-SRTP protocol, four steps were followed:

1. We first wrote ladder logic to control a simple process and downloaded the code on the GE Fanuc Series 90-30 using the Proficy Machine Edition software.
2. We built a HMI project with the Wonderware Intouch software and ran it on a laptop to interact with the PLC.
3. We read and wrote to different memory register types in the PLC using the HMI computer and simultaneously captured the traffic for each read and write data request.
4. We analyzed the captures, examined sections that changed and formed a comprehensive understanding of the GE-SRTP Protocol.

Note, for the last item we additionally utilized a description of the Serial Network Protocol (SNP). More details are provided in the following section.

Understanding the GE-SRTP protocol

As outlined in the methodology, we started by sending and capturing single packets between the HMI and the GE Fanuc Series 90-30 in order to examine which bits/bytes change when specific commands were sent. While this revealed some initial hints on the protocol, it allowed us to compare it to the SNP protocol specification (General Electric, 1998) which is the predecessor of the GE-SRTP protocol. Particularly, the SNP protocol fulfills a similar purpose as it allows control of a PLC over serial network ports but is no longer supported by GE process controllers. For the remainder of this paper, whenever we talk about the SNP protocol description, we refer to the description provided by General Electric (1998).

The comparison of our initial findings of the GE-SRTP protocol and the documented SNP protocol revealed that both protocols have similarities but differ in the offsets of the bytes. With the help of the documentation, we developed a Wireshark plugin to ease the analysis of the GE-SRTP packets captured on the network.

GE-SRTP plugin for Wireshark: In a first step, a dissector[5] for the GE-SRTP protocol was developed using Wireshark and its support for the Lua scripting language. Lua is a platform independent language and can be used. A dissector simply changes the representation of the data. Thus, instead of looking at a hexdump, it allowed

Table 1
Request message structure.

Byte offset	Field type	Common value
0	type	0x02
1	unknown/reserved	0x00
2	sequence number	
3	unknown/reserved	0x00
4	text length	0x00
5–8	unknown/reserved	0x00
9	unknown/reserved	0x01
10–16	unknown/reserved	0x00
17	unknown/reserved	0x01
18–25	unknown/reserved	0x00
26	time (seconds)	0x00
27	time (minutes)	0x00
28	time (hours)	0x00
29	unknown/reserved	0x00
30	sequence number	
31	message type	0xc0
32–35	mailbox source	0x00 00 00 00
36–39	mailbox destination	0x10 0e 00 00
40	packet number	0x01
41	total packet number	0x01
42	service request code	
43–47	request type dependent	
48–55	unknown/reserved	0x00

us to extract specific bytes, change their representation into decimal and add labels.

Request network packet analysis results

An overview of the request packets is shown in Table 1 where the payload has a total length of 55 bytes. Most of the fields in the packets remain fixed throughout our testing and are classified as *unknown/reserved*. We will discuss the variable fields (empty value in the table).

The *type field* is commonly 0x02 for a request packet and will change for the response packet to 0x03. Throughout testing, we did not observe any other values in the *type field*.

We believe that the *sequence number* is repeated twice in the message structure (byte offset 2 and byte offset 30) and is used to identify the request and response message pair. The master includes a byte in the message and the slave copies the byte in its acknowledgment completion message or error message.

The more interesting bytes are clearly at the end of the payload. The *service request code* (byte 42) varies on the type of memory that is being requested where an overview is shown in Table 2.

Bytes 43–47 are used to access the different memory types. The first byte (byte 43) is called 'segment selector' (according to the SNP specification). The selector is a hex-value that indicates which and how a memory register is accessed. Most memory types can be accessed as either bit or byte, and some only allow 'word'. Particularly, discrete memory types (%Q, %I, %M, etc.) can be accessed as bit data or byte data, and word memory types (%R, %AI, and %AQ) can only be accessed as word data. An overview is shown in Table 3.

Bytes 44 and 45 indicate the memory offset which will be accessed starting with zero. Bytes 46 and 47 specify the data length for the memory type to be accessed. Offset and length both have the least significant byte first followed by most significant byte. The segment selector dictates whether the address and data count are in bits, bytes, or words. Here are some examples for values of the key fields when reading PLC system memory[6] after both master and slave have established a connection:

---

[5] https://wiki.wireshark.org/Lua/Dissectors (last accessed 2017-Jan-21).

[6] Examples are partially taken from SNP page 6–10, General Electric (1998).

```
%Q497 to %Q512 in bit mode:
Service Request code = 04 (read memory)
Segment Selector = 48 (%Q memory in bit mode)
Data Offset = 3E 00
                    (= 0x003e = 62 = byte index to %Q497)
Date length = 10 00 (= 0x0010h = 16 bits)

%Q497 to %Q512 in byte mode:
Service Request code = 04 (read memory)
Segment Selector = 12 (%Q memory in byte mode)
Data Offset = F0 01
                    (= 0x01F0 = 496 = bit index to %Q497)
Data length = 02 00 (= 0x0002 = 2 bytes)
```

In the following example, we write a value of 57_{dec} to register memory %R39. A representation of the request packet is shown in Fig. 3 where the red (in the web version) highlights the key values — the service request code, segment selector, data offset, data length, and the value to be written to the register memory.

PLC will interpret this packet as follows:

```
%R39 in word mode:
Service Request code = 07 (write system memory)
Segment Selector = 08 (%R memory in word mode)
Data Offset = 26 00
                    (= 0x0026 = 38 = bit index to %R39)
Data length = 01 00 (= 0x0001 = 1 word)
Write Data = 39 00 (= 0x0039 = 57)
```

PLC response network packet analysis results

The response messages from the PLC are similar to the request message structure and are summarized in Table 4. While most of the bytes remain fixed, the following are variable:

The *sequence number* (byte offset 2) serves the same purpose as the sequence number used in the request packet structure.

The *status code* (byte 42) is usually zero which indicates that no error has occurred. One of the most common errors during a service request is insufficient privilege level. Other errors are dependent on the service request such as an invalid parameter in a request message (e.g., specifying the wrong segment selector (0x0a) to write to register memory type).

A value of 0xd1 in the *message type* field (byte 31) of the reply message is an indicator that the master request was rejected by the PLC. This type of message is called Error Nack Mailbox message according to the SNP specification. All Error Nack messages have a major status code (byte 42) and a minor status code (byte 43). Some of the error status codes are insufficient privilege level (0x02) for the requested task, a full PLC service request queue (0x07), illegal

Table 2
Types of service request codes.

Hex value	Service request code
0x00	PLC short status request
0x03	return control program names
0x04	read system memory
0x05	read task memory
0x06	read program memory
0x07	write system memory
0x08	write task memory
0x09	write program block memory
0x20	programmer logon
0x21	change PLC CPU Privilege Level
0x22	set control ID(CPU ID)
0x23	set PLC (run vs stop)
0x24	set PLC time/date
0x25	return PLC time/date
0x38	return fault table
0x39	clear fault table
0x3f	program store (upload from PLC)
0x40	program load (download to PLC)
0x43	return controller type and id information
0x44	toggle force system memory

Table 3
Segment selectors' overview.

Memory type	Bit-selector	Byte-selector	Word-selector
Discrete Inputs (%I)	0x46	0x10	
Discrete Outputs (%Q)	0x48	0x12	
Discrete Internals (%M)	0x4c	0x16	
Discrete Temporaries (%T)	0x4a	0x14	
%SA Discrete	0x4e	0x18	
%SB Discrete	0x50	0x1a	
%SC Discrete	0x52	0x1c	
%S Discrete	0x54	0x1e	
Genius Global Data (%G)	0x56	0x38	
Analog Inputs (%AI)			0x0a
Analog Outputs (%AQ)			0x0c
Registers (%R)			0x08

```
                    02 00    05 00 00 00 00 00 00 01
00 00 00 00 00 00 00 01    00 00 00 00 00 00 00 00
00 00 00 00 03 c0 00 00    00 00 10 0e 00 00 01 01
07 08 26 00 01 00 39 00    00 ff ff 00 ff ff
```

Fig. 3. Request packet for writing to register memory.

Table 4
ACK reply message structure.

Byte offset	Field type	Common value
0	type	0x03
1	unknown/reserved	0x00
2	sequence number	
3	unknown/reserved	0x00
4	text length	0x00
5–16	unknown/reserved	0x00
17	unknown/reserved	0x01
18–25	unknown/reserved	0x00
26	time (seconds)	
27	time (minutes)	
28	time (hours)	
29	unknown/reserved	0x00
30	unknown/reserved	value varies
31	message type	0xd4
32–35	mailbox source	0x10 0e 00 00
36–39	mailbox destination	0x20 5a 00 00
40	packet number	0x01
41	total packet number	0x01
42	status code	
43	minor status code	
44–49	return data	
50–55	PLC status	

mailbox type (0x06) defined in the request packet, illegal service request (0x01) in the request task, or protocol sequence error (0x04) which means the PLC receives a message out of order. More details are provided in the specification (General Electric, 1998, pp. 3–12++). We will not focus on the error codes in our work.

The PLC status or 'piggy-back' status is found on the tail end of all ACK reply messages (bytes 50–55) and consists of:

- The control program number (byte 50) indicates whether or not the master is logged into the program task. In our case the program task is the ladder logic created and downloaded to the PLC to operate the head gate. If a 0xff is returned, the master is not logged in to the program task and 0x00 returned value signifies that the master is logged into the program task.
- The current privilege level (byte 51) as discussed in Sec. Testing environment setup.
- The last sweep time (bytes 52–53) is the last elapsed time to fully execute program task.
- The PLC status word (bytes 54–55) is a bit more complex. Each bit in PLC status word either gives a status or fault in the PLC. For example, bit 2 reports if the I/O fault table has changed since it was last read. The value 1 means it changed and 0 signifies no change. A description for the remaining bits can be found on page 3–12 in the SNP documentation.

The data requested by the request message can be found in bytes 44–49. If more data is requested, this section will expand in order to hold all the data requested. In case the requested data is more than 6 bytes (44–49), this section gets extended. The last six bytes (commonly 50–55) always contain the PLC status.

Example for a response of a byte type memory request. Let us assume there was a request for %M1 as byte data from the PLC, then the response would look like Fig. 4. A successful request acknowledgment is indicated by a 0xd4 message type (byte offset 31). The

returned data byte (byte 44) in red (in the web version) contains the values of %M1 to %M8 starting from the least significant bit to the most significant bit. The remaining bytes (bytes 45 to 49) that are not requested in the request message are returned as zeros regardless of the true values of the internal memory registers.

Tool

We condensed our knowledge of GE-SRTP protocol into a tool that is capable of communicating with the PLC directly at the TCP/IP layer (i.e., there is no need for the Wonderware IO server). While our main focus was the forensic aspect to read memory and identify attacks, our tool is also capable of writing to the different memory types. Due to the critical nature of the tool (should it be used in cyber attacks), we removed the writing functionality before releasing the tool which can be found on our website: https://www.unhcfreg.com/datasetsandtools. Given that the released tool does not have a writing capability, it is more forensically sound.

Our application was created using the multi-platform Qt Framework which can run on all major operating systems (Windows, Linux and Mac OS). However, in order to run the source code directly, the Qt creator software is required to open the project. Once installed, the IP address of the PLC must be entered in the software tool to take advantage of the available features. Two screenshots are provided in Appendix B.

Application capabilities

While our original implementation had read and write capabilities, we removed the writing capabilities in the released version. In a nutshell, our tool has following features:

- Reading the name of the program task currently running on the PLC.
- Reading & writing values of all registers on the PLC device.
- Reading PLC fault tables, I/O fault tables & CPU controller ID. The fault tables log all PLC and I/O modules abnormal operations such as low battery in PLC CPU or constant sweep exceeded.
- Master logging into and out of the program task.
- Changing the non-password protected privilege level of the master prior to a PLC service request.
- Enabling/disabling I/O modules operation. I/O modules are used by the PLC to interface with a field devices or instruments. They are inserted in the PLC backplane slots and wired to instruments using manufacturer wiring diagram. We did not use an I/O module in our experiment because the headgate position movements were simulated using scripts built into the Wonderware application. If we had a 'real' headgate for experimenting, a position transmitter (instrument) would be physically connected to the gate. Once the gate starts to move either upward or downward, a proportional 4–20 milliamp (ma) signal would be transmitted on wires to an analog input I/O module. The signal would then be scanned by an analog input register memory (%AI03) in the PLC and then converted (ma to feet) to engineering units for displaying the gate position on the HMI and/or in the PLC program task.
- Changing the PLC state (RUN/STOP).

```
                    03 00    05 00 00 00 00 00 00 00
00 00 00 00 00 00 01 00    00 00 00 00 00 00 00 00
21 15 06 00 03 d4 10 0e    00 00 20 5a 00 00 01 01
00 00 08 00 00 00 00 00    ff 02 27 00 3c 00
```

Fig. 4. Response message structure.

Application architecture and main classes

The tool is written in C++ using the Qt framework and is designed in a multi-thread manner. One thread controls the GUI and the second thread communicates with the PLC device continuously. We chose this architecture because the communication with the PLC device can take quite some time and a single-threaded application (where both GUI and communication execute on the same thread) would cause the GUI to lock (freeze).

Our application consists of three major classes:

- `MainWindow` runs in the first thread and governs everything regarding Graphical User Interface (GUI).
- `PLC_comm_bridge` runs in a second thread and provides the interface to send and receive commands to/from the PLC.
- `PLC_record` object that contains information about one request and its response.

`MainWindow` receives a request from a user via the GUI (buttons, input boxes, etc.), initializes a `PLC_record` object and creates the data for the request, typically:

- request type (read memory, write memory, programmer logon, …)
- memory location (system, task, program block)
- memory type (discrete input, discrete output, …)
- memory address and length

After the request is filled by `MainWindow`, it is passed to `PLC_comm_bridge` that creates a corresponding data packet and sends it to the PLC. After `PLC_comm_bridge` receives a response packet from the PLC, it parses the packet and hands the response values to a PLC_record object that it previously received from `MainWindow`. Lastly, it is passed back to `MainWindow`. An example of reading memory from the PLC is depicted in Fig. 5.

Test procedure

To validate our tool, we designed a head gate control application which is a motorized gate that opens vertically to supply water from an impoundment to spin a hydro turbine. The water flow from the impoundment stops when a close command is initiated by the operator. A generator is coupled to the turbine to convert mechanical energy to electrical energy where it can be used to power up households and businesses. There is no gate position transmitter connected to the controller.

Alarms shown on the HMI are PLC communication failure (PLC COMMS), control power lost, and head gate fault which is triggered by the gate timing out during operation. A PLC COMMS alarm is raised when communication is lost between HMI and PLC. The gate close push button, gate moving up push button, and gate stop push button are used to operate the gate. The ladder logic downloaded to the PLC was created in Proficy Machine Edition 6.0.

The experiment entails operating the head gate from the HMI, and acquiring live memory from the GE Fanuc Series 90-30 using our application in forensic mode. We expected that the values coincide. For the second part of the experiment we used the software tool in writing mode to manipulate the PLC memory by sending a gate open and close commands directly to the PLC—bypassing the HMI software and the Wonderware server. In short, we did not encounter any errors during both experiments. The simulated headgate elevation reading displayed on the HMI was overridden in the experiment by writing a value to the PLC memory address for the gate elevation reading.

Related work

Before commencing any forensic investigation, the possible types of attacks on the SCADA system must be understood. These attacks are categorized into three groups: the communication stack, hardware, and software. Communication stack attacks occur on the network and application layers. Examples of these type of attacks are SYN flooding and packet replay. A hardware attack occurs when unauthenticated remote access is gained into the device, and data set points are changed causing the SCADA system to fail. Example of software attacks are a Buffer Overflow and a SQL Injection. We present here some work related to SCADA security.

SCADA security

Miller (2005) discussed several important areas that are ripe for research such as studying the reliability and security of independent SCADA systems. Process Control Systems (PCS) such as PLCs and SCADA are critical, and downtime can cost millions of dollars in the energy sector. Other topics for research are identifying and assessing vulnerabilities of different SCADA systems, and techniques for handling cyber attacks. Chandia et al. (2008) focused on two strategies for securing SCADA networks. The first strategy proposed was the security services suite. This strategy provides security at five different levels of the SCADA network architecture
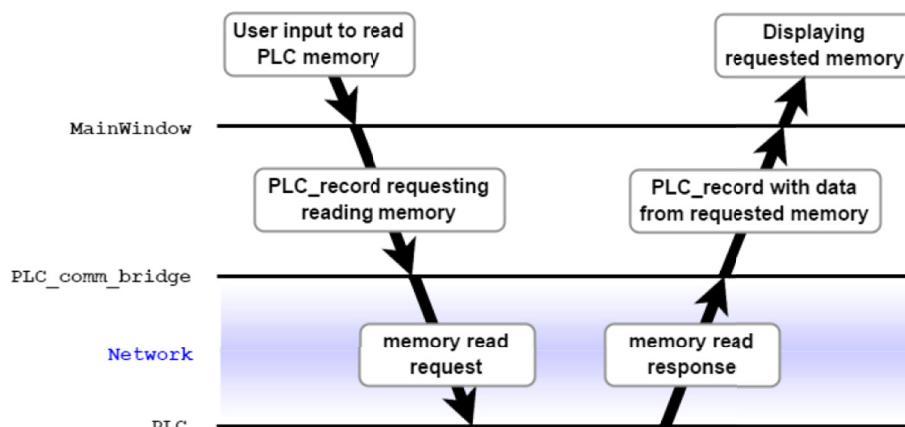

Fig. 5. Tool operation example with reading and displaying data from PLC memory.

and adheres to industry and government standards. The second one will be explained in Sec. SCADA Forensic and Incident Response.

Queiroz et al. (2009) built a security test bed to simulate a waster water plant's SCADA system just to study the effect of cyber attacks. Their research successfully disturbed normal operation of the water plant by using a TCP SYN flooding attack. The modbus SCADA protocol was used for the simulation exercise. Zigbee and DNP3 protocols will be used for their future work. Another means of securing SCADA systems was discovered by Valli (2009). Their plan was to use a SNORT Intrusion Detection System (IDS) rules and a honeypot to create multiple layers of defense for SCADA networks from known vulnerability or exploits by hackers. An extension to the SNORT IDS software called the *Quickdraw* was created by Peterson (2009) to monitor SCADA network traffic, to detect events to be logged in the controller, to create security events, and to forward events to a historian for analysis. The software can be used by ten different controllers, and its purpose is to generate security event logs for SCADA controllers that lack this functionality. This research allowed legacy SCADA system to capture security events without changing out hardware and disturbing operation.

An attack on a Siemen's PLC was demonstrated by German researches at Asia's Black Hat conference. Spenneberg et al. (2016) proved the concept that a computer worm can be installed on the SIMATIC S7-1200 and spread to other S7 PLCs within the subnet. Protection features offered by PLC S7-1200 to safeguard customers from the worm virus were discussed. Unlike the earlier version of computer worms, the virus only targets S7 that uses a special port for communication. Secondly if the target is already infected by the virus, the PLC will detect it and skip the target. Also, no computer is required to spread and to attack the PLC targets. The adverse effects of the virus running on the PLC are output manipulation or changing of the PLC output signals to field devices and denial of service caused by the worm implementing a continuous loop. The continuous loop will stop PLC execution due to the watch dog timer exceedance.

A model of a potential cyber attack on SCADA systems was discovered by Li et al. (2016). The model was called False Sequential Logic attack and its purpose was to disturb the safe order of operation of a process causing serious damage to the process and/or equipment. The authors modeled their attack on a batch neutralization process that consists of two holding tanks with different ingredients used to make a product in a third holding tank. Each ingredient tank had a transfer pump to supply their product to the third holding tank. The correct order to safely operate the process was discussed and then several cases of false sequential attacks were modeled and implemented on the process. After the attack implementation, the impact of each case was analyzed.

Another idea that focused on authentication, secure communication and integrity to improve SCADA systems was presented by Vegh and Miclea (2015). By securing the communication channels of cyber-physical systems, the information that is transmitted between the field devices and the controllers will always be authenticated, complete, unchanged, and available.

Due to the escalating cyber threats on SCADA, Cherdantseva et al. (2016) reviewed and compared effective risk assessment methods for SCADA systems where the authors explored the ideas of what can go wrong, likelihood that it would go wrong, and the consequences. The authors stress that a number of risk assessment methodologies exist for IT systems but it does not apply to SCADA systems without adjustment. To carry out the research, a string search was conducted from keywords "SCADA" and "risk assessment" on papers from IEEE Xplore, ACM, SCOPUS, and Web of Science between the year 2004 and 2014. Papers that suggested a new risk assessment method for SCADA systems were the only ones chosen. In the end, 24 papers were found describing risk assessment methods on SCADA. Each method was examined based on criteria such as aim, application domain, stages of risk management addressed, key concept of risk management covered, impact measurement, sources of data for deriving probabilities, evaluation method, and tool support.

SCADA forensic and incident response

A first toolkit to support forensic analysis of SCADA systems was proposed by Stirland et al. (2014) which was based on the previously introduced forensic methodology from Wu et al. (2013). The toolkit was developed to conduct a complete digital forensic investigation using a full collection of forensic software and hardware. The toolkit required for SCADA forensics differs from the software and hardware required for computer forensics.

Van Der Knijff (2014) compared control systems forensics to SCADA IT forensics. The research focused on the steps to perform a forensic examination on a control system. An investigation strategy is critical at the start of the investigation. Also, of importance is to preserve the original state of the evidence, data acquisition, and data analysis. The author stressed the importance of seeking assistance from an experienced field engineer during a forensic acquisition to guarantee process safety, business continuity, and examination efficiency. Furthermore, the author stated that capturing live data from a SCADA system without disturbing the controlled process remains a challenge for investigators, but it can be done by switching to the backup SCADA controller and performing data acquisition on the attacked main controller in a redundant system. Other forensic challenges were discussed such as customized operating system kernels, resource-constrained devices, inadequate logging of events, and lightweight data acquisition.

In Hay et al. (2009)'s research, challenges and progress with live analysis of SCADA systems were discussed. Live analysis tools give the investigator a more complete picture of SCADA past and current states. It collects data from SCADA systems to reconstruct and analyze past events. The remainder of this section will focus on different types of models for acquiring and analyzing live data from SCADA systems:

Related work, by Miller (2005) focused on the forensic analysis of a breached SCADA system. It consisted of a network system that captures and stores data to aid in the investigation of a cyber attack. The forensic architecture consisted of multiple forensic agents that captured SCADA network traffic data and stored the data in a Warehouse to be accessed when needed. Our approach differs in collecting evidence since data is only collected from one computer in real time. Once the data is stored and it can then be harshed for future analysis.

Another model presented by Taveras (2013) captures forensic data without disturbing the operation of a live SCADA system using forensic watch dogs that constantly listen to SCADA events. Evidence is only collected if any of the events violate a set of predefined rules. The model switches back to a monitor model when the events become normal. Data can only be collected by their software tool after a security breach is detected and someone starts collecting forensic data. There are no watch dogs available to automatically collect forensic evidence. With that said, their tool may be employed as a watch dog monitor when continuously pinging a PLC.

Kilpatrick et al. (2006) placed the forensic agents in strategic locations inside the SCADA network to capture relevant traffic. Each agent transfers a synopsis of each packet to a data warehouse housed in a secure location for storage and retrieval. The data warehouse analyzes each packet synopsis, creates a data signature, and stores both data signature and synopsis in an area reserved for each forensic agent. The architecture focused on SCADA protocols such as Modbus and DNP3 that can be encapsulated inside an Ethernet frame and transported within the network. Future research plans included support for serial communication protocols such as RS 232 and RS 485.

Another model was created by Ahmed et al. (2012) to investigate SCADA systems using seven phases. It provided more details when performing a full forensic investigation compared to the traditional IT forensic processes not suitable for SCADA systems. Currently, there are little to no data acquisition tools available to extract data from PLCs because of the lack of demand for tool makers to produce software for SCADA forensic data collection. The existing tools that support SCADA forensics are Hex dump analysis tools, network forensic software used to monitor network traffic, and Digital Bond software used to log security events for the Rockwell Automation ControlLogix PLC. An experiment was conducted using the Siemens S7 PLC and Siemens STEP7 to create a program for a traffic light system. The traffic light system was hacked using a packet to turn all traffic lights green. The experiment showed that the proposed forensic capability architecture will provide forensic artifacts for collection after an incident.

Wu et al. (2013) improved on the forensic model described in the previous paragraph to carry out a more detailed and full forensic investigation of SCADA systems. Analyzing SCADA systems on Layer 0–Layer 2 of the Open Systems Interconnect (OSI) model. Layer 0 consists of field devices such as control valves, pressure transmitters, and flow meters connected to the network. At Layer 1, SCADA controllers are connected to field devices. Layer 2 is the Demilitarized Zone (DMZ) that consists of historians, domain controllers, and application servers.

Discussion

Nowadays, attackers are not only focusing on computers anymore but on all connected devices like smart phones, Internet of Things and PLCs. While when creating newer devices and protocols, developers may have cyber security in mind, this may not have been of priority one or two decades ago when PLCs and their protocols were developed. As highlighted in the related work section, especially recent years showed that PLCs are becoming more and more attractive for attackers. Our research showed that once an attacker gains access to the network, it is possible to start/stop the ladder program execution on a PLC with default configuration, downloading/uploading software codes or send arbitrary commands. Thus, we want to stress the importance of security in this area. We want to also stress the importance for the creation of forensic tools that help in the acquisition and analysis of PLC memory.

One possibility, in order to make it harder for an active adversary, is to configure authentication methods. For instance, the GE Fanuc Series 90-30 supports privilege levels which must be enabled and allow password protection. Note, by default the privilege levels are turned off and thus it requires the administrator to change the configuration. Given the long history of these devices (some of them might have been running for almost 30 years), privilege levels are rarely activated (based on our experience and contacting administrators at various organizations). On the other hand, vendors should update protocols and firmware in order to enforce authentication and allow passwords longer than 8 digits.

In our opinion, one of the most important challenges is to raise the awareness with administrators that PLCs are currently *not* well protected (can easily be manipulated) and that they need to become active in order to secure their systems. If a breach happens, we need reliable tools to acquire a PLC's memory, which can help in the forensic analysis of these devices.

Future work

There are several points we would like to focus on in future work. First, we would like to add new features to the forensic tool such as pulling the description assigned to the memory types (%I, %M, %R, etc.) over the network. This information can be useful to determine the function of the memory type. Additionally, we also want to determine the unknown field types of the GE-SRTP protocol. We will spend more time studying the privilege levels to evaluate how secure they are, e.g., can we circumvent the authentication and/or brute force it.

Furthermore, to acquire system memory from the PLC, our application had to connect to the PLC by sending two initialized packets prior to sending request messages. More research needs to be conducted using different PLC models that support the GE-SRTP protocol. The goal is to confirm if the structure of the initialized packets are standardized for different models.

Lastly, we would also like to put a different angle on our future research. While during this research the tool was being used as a master device to request data from a slave PLC on the network, we want to analyze if it is possible to use the tool as a slave device to steal a connection from a slave PLC that exists on the Local Area Network (LAN). All requests (read or write) from the HMI IO server will be sent to the impersonator and response messages will be sent back to the master.

Conclusions

In this article we successfully reverse engineered the GE-SRTP protocol which is the communication protocol of Wonderware IO Server and the GE Fanuc Series 90-30 PLC. Our findings allowed us to create a tool that can bypass the traditional required software, e.g., Wonderware IO Server, and allows for direct communication with the device. Another benefit of the tool is to troubleshoot process related alarms or checking revising/new software code downloaded to the PLC.

Our work can be helpful for cyber forensic investigations as we now do not have to rely on any proprietary software but can access the memory registers directly. While our intention was to build a forensic tool, we also discovered that it is possible to change register values by using our application in write mode if the device is in its default configuration. In other words, once an active adversary has access to the network, s/he is able to manipulate memory register values and may cause serious damage. Note, based on our lab tests, in forensic mode, network PLC can be conducted without disturbing process operation.
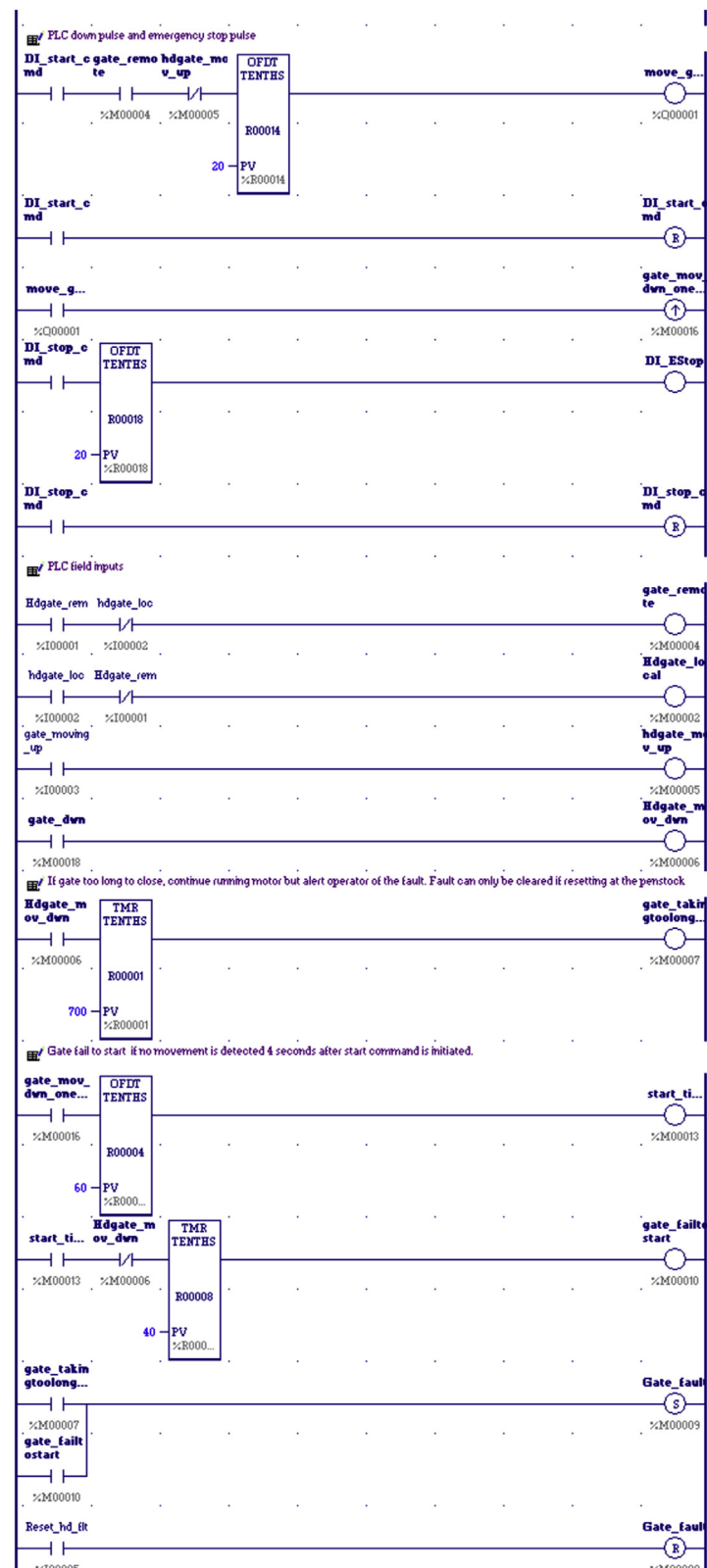
Appendix A. Ladder logic

Figure A.6. Ladder logic used to simulate the head gate process in a lab environment.

Appendix B. Tool

The GUI is separated into two main sections. Fig. B.7 shows the left half of our application which can be used to connect to a specific PLC and then will summarize the main attributes. The right side of the application is depicted in Fig. B.8 and equals a comprehensive 'table' containing the different registers in its head-columns and shows the values of the registers.
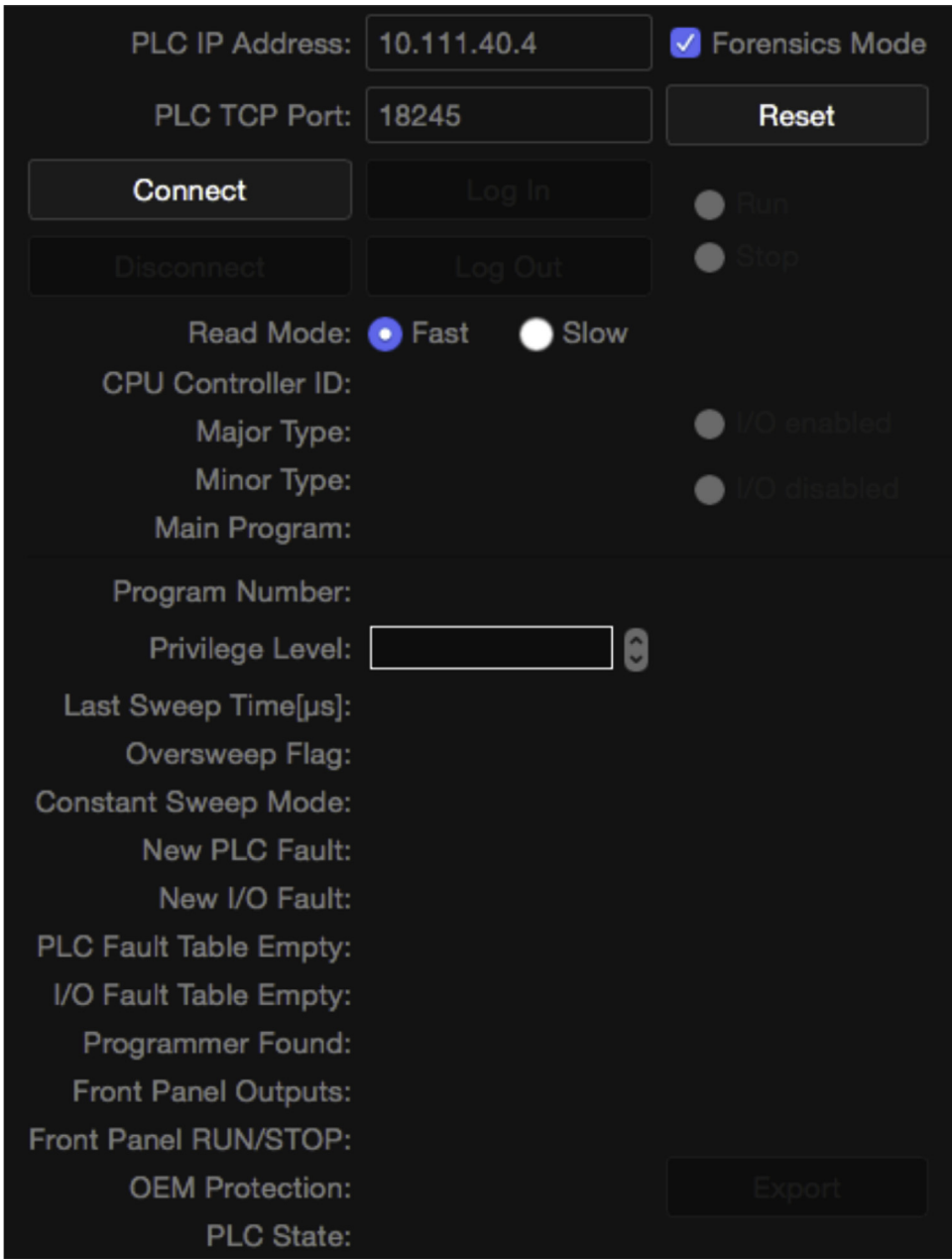

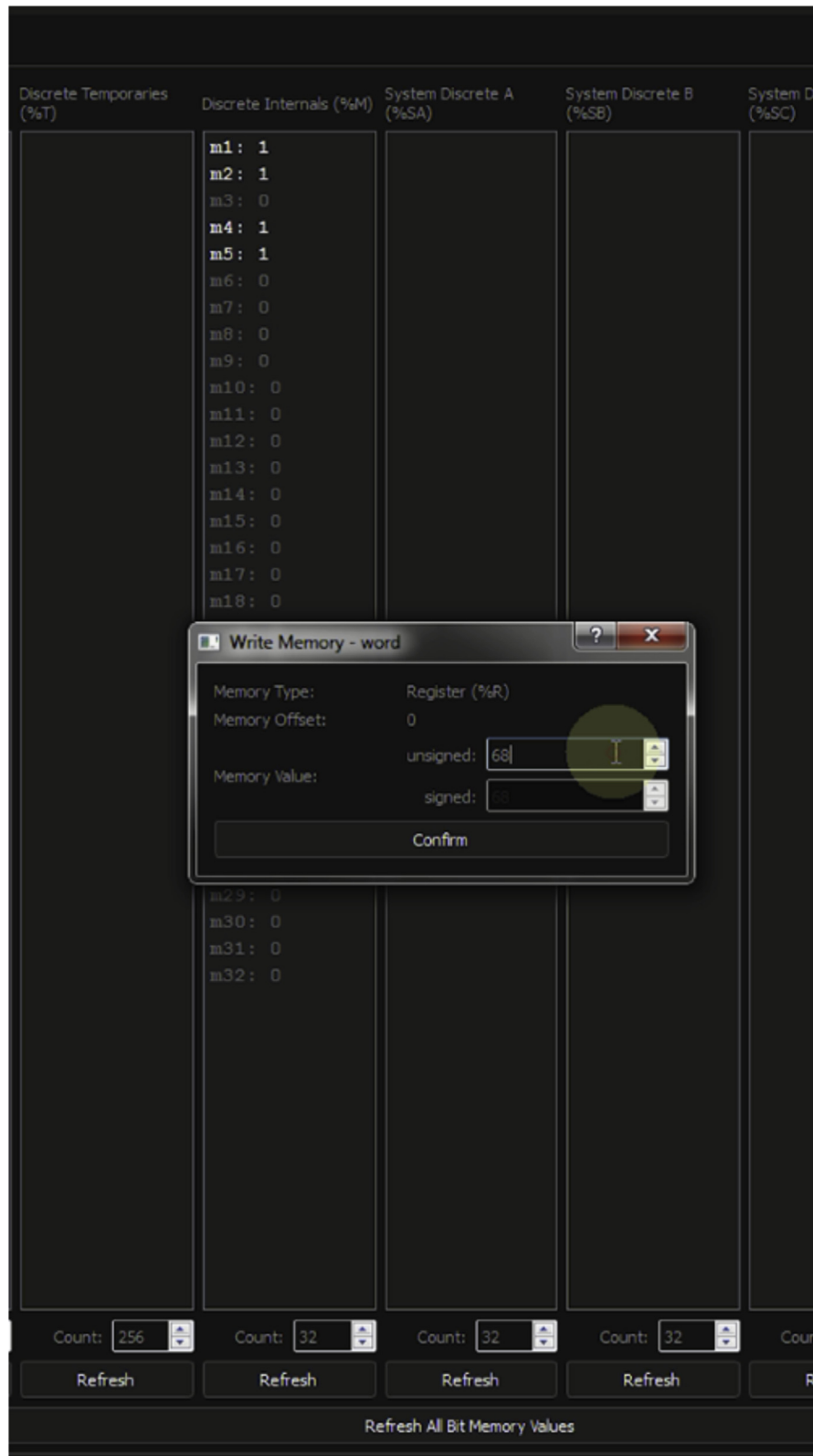**Figure B.7.** Left half of our application which allows a connection to be established to a PLC.

Figure B.8. Snapshot of the right half of our application which shows memory values.

References

Ahmed, I., Obermeier, S., Naedele, M., Richard III, G.G., 2012. SCADA systems: challenges for forensic investigators. Computer 44–51.

Chandia, R., Gonzalez, J., Kilpatrick, T., Papa, M., Shenoi, S., 2008. Security strategies for SCADA networks. In: Critical Infrastructure Protection. Springer, pp. 117–131.

Cherdantseva, Y., Burnap, P., Blyth, A., Eden, P., Jones, K., Soulsby, H., Stoddart, K., 2016. A review of cyber security risk assessment methods for scada systems. Comput. Secur. 56, 1–27.

GE Fanuc Automation North America, Inc, 2002. Series 90TM-30/20/Micro PLC CPU Instruction Set. General Electrics.

General Electric, 1998. Series 90 PLC SNP Communications (gfk-0529c ed.).

Hay, B., Bishop, M., Nance, K., 2009. Live analysis: progress and challenges. Secur. Priv. IEEE 7, 30–37.

Kilpatrick, T., Gonzalez, J., Chandia, R., Papa, M., Shenoi, S., 2006. An architecture for SCADA network forensics. In: Advances in Digital Forensics II. Springer, pp. 273–285.

Langner, R., 2011. Stuxnet: dissecting a cyberwarfare weapon. IEEE Secur. Priv. 9, 49–51.

Li, W., Xie, L., Deng, Z., Wang, Z., 2016. False sequential logic attack on SCADA system and its physical impact analysis. Comput. Secur. 58, 149–159.

Microsoft (N/A). About dynamic data exchange. https://msdn.microsoft.com/en-us/library/windows/desktop/ms648774(v=vs.85).aspx. Last accessed 2017-Jan-21.

Miller, A., 2005. Trends in process control systems security. Secur. Priv. IEEE 3, 57–60.

Peterson, D., 2009. Quickdraw: generating security log events for legacy SCADA and control system devices. In: Conference for Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology, pp. 227–229.

Queiroz, C., Mahmood, A., Hu, J., Tari, Z., Yu, X., 2009. Building a SCADA security testbed. In: Network and System Security, 2009. NSS'09. Third International Conference on. IEEE, pp. 357–364.

Spenneberg, R., Brüggemann, M., Schwartke, H., 2016. PLC-Blaster: A Worm Living Solely in the PLC. Black Hat Asia (p. N/A).

Stirland, J., Jones, K., Janicke, H., Wu, T., 2014. Developing cyber forensics for SCADA industrial control systems. In: The International Conference on Information Security and Cyber Forensics (InfoSec2014). The Society of Digital Information and Wireless Communication, pp. 98–111.

Taveras, P., 2013. SCADA live forensics: real time data acquisition process to detect, prevent or evaluate critical situations. Eur. Sci. J. 9.

Valli, C., 2009. SCADA forensics with snort ids. In: International Conference on Security & Management (p. N/A).

Van Der Knijff, R., 2014. Control systems/SCADA forensics, what's the difference? Digit. Investig. 11, 160–174.

Vegh, L., Miclea, L., 2015. Authenticity, integrity and secure communication in cyber-physical systems. J. Comput. Sci. Control Syst. 8, 33.

Wu, T., Disso, J.F.P., Jones, K., Campos, A., 2013. Towards a SCADA forensics architecture. In: Proceedings of the 1st International Symposium for ICS & SCADA Cyber Security Research, pp. 12–21.

Zetter, K., 2011. Attack Code for SCADA Vulnerabilities Released Online. https://www.wired.com/2011/03/scada-vulnerabilities/. Last accessed 2017-Jan-21.