Forensic Science International: Digital Investigation 44 (2023) 301513

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS 2023 EU - Selected papers of the Tenth Annual DFRWS Europe Conference

Towards generic memory forensic framework for programmable logic controllers☆,☆☆

Rima Asmar Awad [a, *], Muhammad Haris Rais [c], Michael Rogers [b], Irfan Ahmed [c], Vincent Paquit [a]

[a] Oak Ridge National Laboratory, Oak Ridge, TN, 37830, USA
[b] Tennessee Technological University, Cookeville, TN, 38505, USA
[c] Virginia Commonwealth University, Richmond, VA, 23284, USA

ARTICLE INFO

Article history:

Keywords:
CPS forensics
Memory forensics
JTAG
SCADA
PLC
ICS
Embedded devices

ABSTRACT

A Programmable Logic Controller (PLC) is a microprocessor-based controller that is used to automate physical processes in critical infrastructure and various other industries and manufacturing sectors. Initially, PLCs were completely isolated from the Internet, and cyber security was not incorporated at the time of development. The introduction of industry 4.0 and the evolution of ICS systems to communicate over public IP addresses from the Internet enhanced productivity and efficiency, but Internet connectivity exposed the systems and their vulnerabilities, which led to an increase in cyber attacks. When a system is sabotaged/compromised, security analysts need to get to the root cause of the attack as quickly as possible to recover the system. To do so, memory forensic analysis is critical to provide a unique insight into the run-time memory activities and extract a reliable source of evidence. In this paper, we analyze the memory structure of the Schneider Electric Modicon M221 PLC. To build a memory profile, we reverse engineer the communication protocol and conduct differential analysis to gain knowledge about the structure of the memory and the low-level representation of control logic instructions. We then identify dynamic and static memory regions by modifying different project fields and conducting differential analysis, which allows us to identify boundaries of critical memory structures and extract important forensic artifacts that can be found in the memory. The Python implementation of the memory profile can help reduce the time and effort required for manual analysis in case of cyber incident or system failure.

1. Introduction

Programmable Logic Controllers (PLC) play a vital role in industrial control systems and many other manufacturing sectors. PLCs are used to automate industrial processes by collecting input data from field devices such as sensors, processing the data, and then sending commands to actuator devices such as motors (Ahmed et al., 2016). The PLCs memory code (control logic) executes in cycles, in which each cycle reads input from multiple resources, executes the control logic instructions to calculate the output, and then sends that output to the physical process in the form of analog or digital signals. PLC's have been in use since the early 1970s as the means for monitoring, and remotely controlling geographically distributed physical processes such as water treatment and distribution, oil and gas pipelines, and electrical power transmission and distribution. As microprocessors, personal computers and networking technology evolved over time, industrial control systems incorporated these advances in technology.

However, PLC operations are still typically sent via insecure proprietary network protocols that can be easily reverse-

☆ This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes.

☆☆ DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

* Corresponding author.
E-mail address: awadrl@ornl.gov (R.A. Awad).

https://doi.org/10.1016/j.fsidi.2023.301513

engineered by attackers, which allows them to create open-source APIs and develop tools that can be used for malicious purposes (Senthivel et al., 2017, 2018). Due to their critical role and lack of security mechanisms, PLCs are often an easy target for cyber-attacks that leads to sabotaging physical processes and the destruction of critical infrastructure. Despite increasing attention to PLC security, integrating security into PLCs remains difficult as they are intended to be simple input/output machines with deterministic behavior and limited resources (Ahmed et al., 2012). For example, although a layer of encryption has been added to new communication protocols, those protocols can provide secure communication to new products and are not supported by legacy devices, leaving them vulnerable to attacks.

The lack of security in PLCs also allows attackers to maliciously modify both the firmware and control logic programs (Qasim et al., 2022; Ayub et al., 2021; Zubair et al., 2022a). Once an intruder reverse-engineers and modifies the device's firmware, with proper access, the intruder can download it to the device. Similarly, an attacker can access control logic on the device and modify it. Even worse, a sophisticated attacker with minimal knowledge about the physical process, can craft malicious control logic, and download it to the PLC after bypassing authentication (Kalle et al., 2019). In another scenario, memory code injection attacks can bypass network intrusion detection systems and deliver malicious control logic to the target device's memory. To avoid detection, the attacker can write the malicious instructions to a memory segment other than the code segment, which is possible due to the lack of write/execution protection in PLC memory (Yoo et al., 2019a). Conducting a firmware modification attack is not easy because it requires the attacker to have extensive knowledge about the target PLC's hardware components and bypass software whitelisting. Therefore, control logic injection attacks are more likely to launch because they are easier to achieve and harder to detect.

Memory forensic analysis is crucial to help answer forensic questions following an attack or a system failure (Davis et al., 2022; Ahmed et al., 2015; Bhatt and Ahmed, 2018). However, in industrial control systems, conducting forensic analysis is a tedious and time-consuming process due to heterogeneous hardware architectures and proprietary firmware/communication protocols, which make it challenging to employ a unified framework for all PLC memory forensics (Awad et al., 2018). A methodology that describes the process of analyzing the memory of PLCs that share similar hardware components will provide valuable information to the forensic analyst, and reduce the effort and time needed to accomplish this task manually. Reduced analysis time will result in a faster response to a cyber-attack or a system failure.

In this paper, we use differential analysis to understand the layout of the Modicon M221 memory and use the acquired knowledge to build a memory profile for the M221 PLC. We use the memory profile to extract important forensic artifacts including the control logic, the metadata file, and the data blocks. We then evaluate the effectiveness of our methodology by simulating a traffic light control logic manipulation and extracting memory forensic artifacts that present evidence about the incident.

The rest of the paper is organized as follows. Section 2 covers background and related works, section 3 presents the memory analysis methodology, section 4 presents an evaluation of the memory analysis followed by a discussion and future work in section 5, and finally section 6 concludes the paper.

2. Background & related work

In this section, we present an overview of PLCs, their basic components, and their protocols. In addition, we go over the current state of the art in the realm of PLC memory forensics.

2.1. Background

The PLC has revolutionized the automation and manufacturing processes. Today, PLCs maintain the same core functionality and simplicity that originally made them so popular with manufacturers. However, because of the continued advancement in processor and memory technology, PLCs continue to shrink in size and increase in power and speed. These technological advancements have led to new capabilities, including support for multiple communication protocols. Modern PLCs also seamlessly integrate with powerful machine monitoring software and supervisory control and data acquisition (SCADA) systems and thus provide new ways for manufacturers to drive improvements to their operations' efficiency and performance through machine data analysis. To accomplish monitoring and automation tasks, the PLC's memory code (control logic) executes in cycles, where it reads input from multiple resources, then executes the control logic instructions to calculate the output that is, then, sent to the physical process in the form of signals. PLCs have been in use since the early 1970s as the means for monitoring, and remotely controlling, geographically widely distributed processes such as water treatment and distribution, oil and gas pipelines, and electrical power transmission and distribution.

2.1.1. PLC components

A typical PLC consists of basic interconnected components that work together to allow the controller to accept inputs from various sensors, make logical decisions as programmed, and control outputs such as starters of motors, solenoids, drives, and valves. As shown in Fig. 1 a typical PLC is made up of the following:

Power Supply: This unit supplies the required power to the components in the control panel by switching from 120VAC to 24VDC.

Processor: The processor module contains the microprocessor that performs control functions and computations, as well as the memory required to store the program.

Input/Output (I/O): I/O modules provide the means of connecting the processor to the field devices and can be either digital or analog.

Memory: The part of the PLC that stores data, programs, and information. The process of adding new information into memory is called *writing*, and the process of retrieving information from a memory location is called *reading*. PLCs usually have both volatile and non-volatile memory to store various types of data.

2.1.2. Communication protocols

A protocol is a set of rules that regulates communication among networked devices. In PLCs, communication protocols are needed to communicate with remote I/O devices, remote control devices,

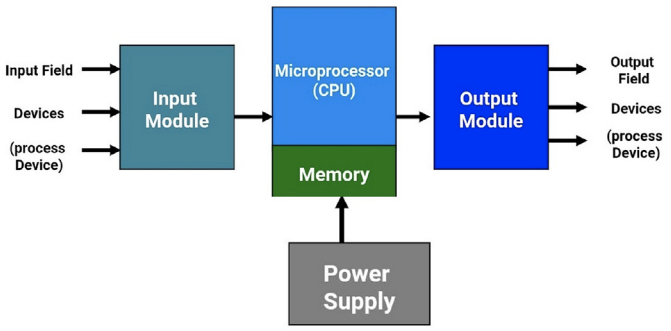

Fig. 1. PLC components.

and engineering software. In their infancy, PLC protocols used to be proprietary, with each manufacturer having its own closed protocols. However, modern, open protocols, such as Modbus, establish the rules for data transmission, although proprietary communication methods are still used by many vendors. Because proprietary protocols support programming software to get access to the physical memory of PLCs, they have been studied and reverse-engineered by both academia and industry. Consequently, knowledge acquired from analyzing a protocol can be used as an important tool for memory forensic artifact acquisition.

2.2. Related work

Awad et al. (2018) present a survey on the published forensics methodologies and frameworks applied to SCADA systems. The authors demonstrate a gap in the forensics of end-point devices in CPS, especially for the live forensics and acquiring root cause evidence at run time.

Ahmed et al. (2017) presented the opportunities for network-level and device-level attacks on PLCs and the forensic tools and methodologies that can help investigate those attacks. The paper also highlighted the forensic challenges that include proprietary and closed-source firmware and inadequate PLC network forensic tools.

Chow et al. (Chow and Chan) also discuss various possible attack scenarios, and identified important logging information available in the Siemens TIA portal, PLC diagnostic buffer, and Human Machine Interface (HMI) module. The information is useful during forensic investigations.

Many research efforts are focused on extracting and analyzing PLC firmware (Zhu et al., 2017; Basnight et al., 2013; Mulder et al., 2012). Although the firmware analysis can provide valuable forensic artifacts, the current state of the PLC including the control logic and input/output data states cannot be found through firmware analysis.

A few recent works focused on control logic modeling/verification that acts as a cyber-physical security intrusion detection engine to detect malicious code injection attempts or the modeling of runtime PLC control logic behavior, to detect abnormal behaviors (Zonouz et al., 2014; Garcia et al., 2016; Xie et al., 2022).

Wu et al. (Wu and Nurse, 2015) uses PLC debugging tools such as PLC Logger to acquire and utilize PLC memory data. However, this approach is restricted to specific PLC models from Siemens. Denton et al. (2017) investigate General Electric's proprietary protocol, GE-SRTP, to enable direct interaction with G.E PLC. The authors demonstrated the capability to modify the control logic through their tool using access memory registers.

Sushma et al. (Kalle et al., 2019) and Qasim (Qasim et al., 2019, 2020) extracted the PLC control logic by analyzing the network traffic between the PLC and the engineering software.

Awad et al. (2019) discusses memory acquisition techniques available in IT field and the possibility of applying them to SCADA end-point devices. The authors attempted to obtain the entire memory of M221 PLC through the communication protocols and via JTAG port. The memory acquisition via JTAG was not successful because the JTAG pins were disabled by the manufacturer. Zubair et al. (2022b) proposed a framework to remotely acquire a PLC memory. The authors append a small code to the control logic that would execute and read the memory contents from areas that are not authorized for reading through a network protocol. While this work analysis portions of acquired memory, it focuses on memory extraction and not on actual memory analysis.

Yoo et al. (2019b) proposed a technique to identify control logic attacks by creating and updating a local copy of the current memory state using the network communication. They employ 42 unique features of the control logic and employed a classification algorithm to identify malicious control logic injections or modifications.

Rais et al., 2021, 2022 acquired the memory of Allen−Bradley Control Logix 1756 PLC through the JTAG interface available on the PLC, and proposed a differential analysis-based framework to extract forensic artifacts from the memory dump. The extracted artifacts include PLC firmware, control logic, input, and output pin states, and PLC operational parameters, however, the approach require a PLC for the profile generation. Although this work is the most relevant to the work presented in this paper, our approach does not require a PLC to identify data structures in memory dump, and it can be used on any device of the same model regardless of the project or environment it is used in.

The existing literature focuses on identifying control logic attacks, firmware reverse engineering, and intrusion detection. Forensic artifacts are necessary to identify the root cause of abnormal behavior and answer important questions about the attack, such as how and when the attack happened, who was involved, and what the attack targeted. Our study focuses on extracting forensic artifacts, to help answer those questions.

3. Memory forensic analysis of M221 PLC

In this section, we acquire the memory of the M221 PLC after reverse engineering the communication protocol. To build a memory profile, we use pointers found in configuration blocks to identify the essential memory and structures that contain important forensic artifacts and characterize the state of PLC.

3.1. Modicon M221 PLC

The Schneider Electric Modicon M221 Logic Controller, designed to control basic automation for machines, has 9 digital inputs, 2 analog inputs, and 7 relay-type digital outputs. The M221 PLC has a 32-bit micro-controller (Renesas RX630) that operates at 100 MHz and supports both serial and Ethernet communication. The PLC uses a Modbus encapsulated proprietary protocol and is configured/programmed with the EcoStruxure Machine Expert - Basic software which supports the following IEC 61131−3 programming languages:

- Instruction List: Low-level textual language that resembles assembly
- Ladder Diagram: Graphical programming language that mimics physical circuits, and consists of vertical rails (supply power) and as many “rungs” (horizontal lines) as needed to represent control circuits.
- Grafcet ListSFC: A standardized graphical specification language for the design of a controller's dynamic behavior.

The memory of M221 PLC consists of two types:

- Volatile: The Random-access memory (RAM) has a size of 512 kbytes of which 256 kbytes are used for internal variables and 256 kbytes for application and data. The RAM memory is used to execute the application and contains data needed at runtime.
- Non-volatile: This type of memory has a size of 1.5 Mbytes, of which 256 Kbytes is used to back up the application and data in case of a power outage. Its main use is saving application backup.

3.2. Memory acquisition

In digital forensics, forensic artifacts can either be obtained from the non-volatile memory image or RAM image. However, to make the attack concealed, most intruders try to hide the footprints of the attack by injecting malicious code into RAM without leaving any traces in the non-volatile memory. The forensic analyst can take the device offline to investigate the root cause of the attack, but the volatile memory will be lost at that point. consequently, the RAM image provides better results in identifying the malicious behavior, as all the running instructions must be executed from the RAM to perform its intended task. ICS protocols have documented function codes to read specific data types, such as input, output, counters, and timers, from a live PLC. However, function codes to acquire the RAM and non-volatile contents are not documented, and in some cases are not implemented (Ahmed et al., 2017). Alternatively, a Joint Test Action Group (JTAG) port can be used to access the entire memory space (Rais et al., 2021; Awad et al., 2019). However, JTAG memory acquisition is slow and requires physical access to the device, and manufacturers hide or disable the JTAG port. Hence, JTAG memory acquisition is typically not practical in a deployed system. To acquire the full memory of an M221 PLC, we choose to use protocol function codes, as the entire memory addresses are protocol mapped (Zubair et al., 2022b).

3.2.1. Protocol reverse engineering

The Modicon M221 uses the UMAS (Unified Messaging Application Services) proprietary protocol for communication. UMAS is based on the well-known Modbus protocol and uses one of the reserved Function Codes specified in the Modbus Protocol Specification (Function Code 90 or $0 \times 5A$ in hexadecimal) for both request and reply packets. The UMAS protocol also has its own function codes that indicate to the PLC how it should handle the message. Function Codes are only sent in a request messages, while the responses will include a status indicating success or failure. Fig. 2 describes the communication between the client and the server via UMAS protocol.

To reverse engineer the UMAS protocol, we used Wireshark to collect the UMAS traffic while conducting various tasks from the engineering software. Most importantly, we downloaded and uploaded control logic to and from the connected PLC. To better understand the static and dynamic fields in the UMAS packets, we used the same control logic for download and upload requests. By conducting differential analysis, we noticed that for the most part, the data content stays the same, except for the session ID and the function codes. Fig. 3 shows the request and response packet format of the 0×28 function code used to read to memory addresses during a upload request. In a request packet, the starting address, data block size, and data to be read are specified, and in response packets, the only information provided is the status of a request.

Fig. 4 shows an example of an Download request message that uses function code (0×29) to write the project from a PC to PLC. In a request packet, the starting address and data block size of the obtained physical memory are specified, and a response packet provides information about the status, the data size, and the data payload.

After reverse-engineering the protocol, we determined various

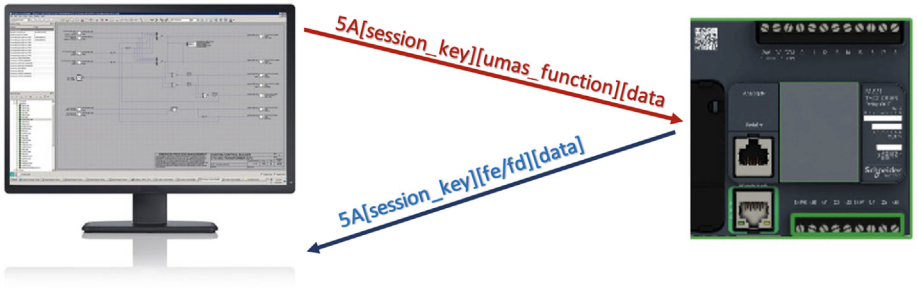

Fig. 2. Communication between the client and server.

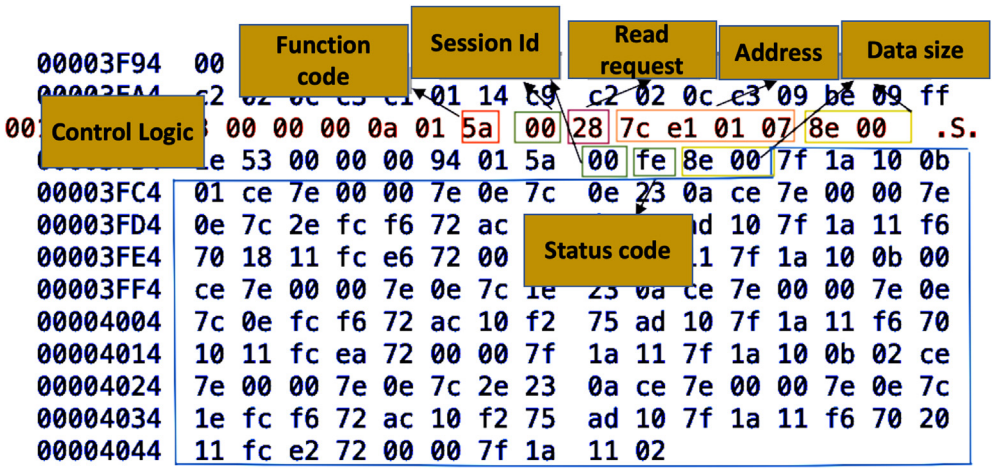

Fig. 3. UMAS Upload traffic.

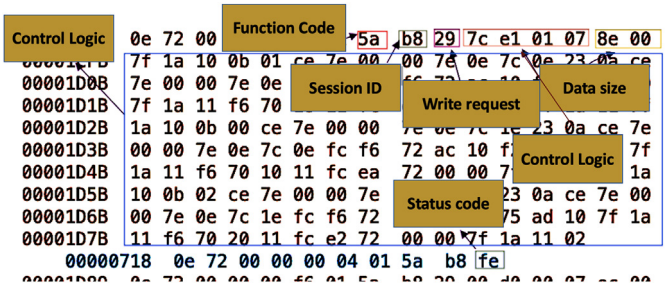

Fig. 4. UMAS Download traffic.

function codes used for different operations. Table 1 lists some of the function codes that are used for various purposes, such as starting the PLC and reading coils, and holding registers. Among these function codes, the function code (0×28) is what we used to read memory.

3.2.2. Memory layout analysis

After acquiring a full memory dump, we began our analysis process by running Binwalk (a tool for searching a given binary image for embedded files and executable code) on the memory dump. We found that Binwalk detects a zip formatted file at memory address $0 \times 00d00$. After extracting the zip file, we found that it was the project metadata file. Next, we modified various parts of the project using the engineering software, such as inserting a new rung, or mortifying an existing rung, and conducted a differential analysis to determine changes. We noticed that the configuration block is always present in download packets and written to the same address (0xD4FE0100). This block remains the same when the project does not change. After analyzing the configuration file, we found that it contains pointers to the start address and other information about other essential memory structures. Based on the deferential analysis results, we learned that the configuration block holds information about a second configuration file address and its size, the metadata file and its size, and the data block and its size. Fig. 5 shows the extracted configuration block and the pointers to other memory structures.

Using the information from the previously analyzed configuration block, we extracted the second configuration block and learned the address and the size of the control logic, the first configuration block, and the data block. Fig. 6 shows the second configuration file with the start address $0 \times 544F0407$.

Fig. 7 illustrates the basic memory structures found in the M221 memory dump and the location of pointers to the data structure's address. For the purpose of building the M221 memory profile, we took advantage of the static location of the first configuration block and used it to learn the location of other memory structures, so we can extract them and analyze them in a later step.

Table 1
UMAS function codes.

Function Code	Details
0×01	**INIT_COMM**: Initialize a UMAS communication
0×02	**READ_ID**: Request a PLC ID
0×03	**READ_PROJECT_INFO**: Read Project Information
0×04	**READ_PLC_INFO**: Get internal PLC Info
0×06	**READ_CARD_INFO**: Get internal PLC SD-Card Info
0×24	**READ_COIL_REGISTERS**: Read coils and holding registers from PLC
0×28	**READ_MEMORY**: Read memory blocks
0×29	**WRITE_MEMORY**: Write to memory blocks
0×40	**START_PLC**: Starts the PLC

3.2.3. Forensic artifacts extraction

Digital forensic artifacts are objects that contain data or evidence of something that occurred. For PLC memory forensics, memory forensic artifacts may include timestamps, control logic, network information, etc. A full memory dump is always necessary to understand the memory layout and obtain forensically sound data. However, after obtaining the memory layout in the previous step, we notice that the M221 memory has a multitude of padding data and unused memory spaces. For our purpose, we filter out the padding data, such as 0X00 and 0xFF, and focus on the essential memory blocks that contain the forensic artifacts. For the M221, we found the meaningful addresses to be the following:

- 0×00 - 0x1ffff: On chip RAM contains first configuration block and PLC network information
- 0×7000000 - $0 \times 707ffff$: External RAM contains second configuration block, metadata file, control logic and data file
- 0xfff00000 - 0xfff7ffff: On chip ROM contains firmware

3.2.3.1. Metadata file. We initially found the address of the metadata file using Binwalk, but we also found pointers to its address in the first configuration block. The compressed metadata file, named

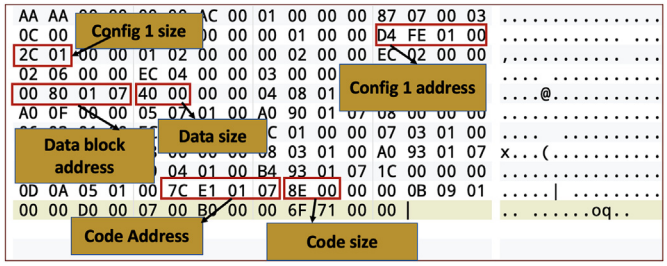

Fig. 6. Pointers in Configuration block to other critical memory blocks.

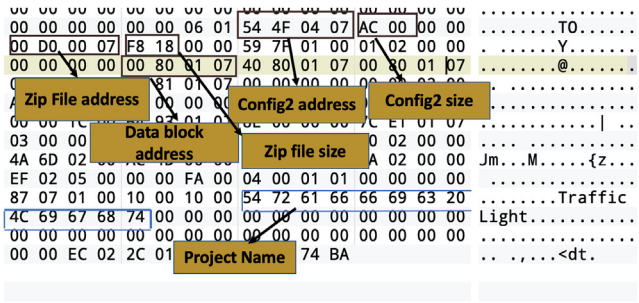

Fig. 5. Pointers in Configuration block to other critical memory blocks.

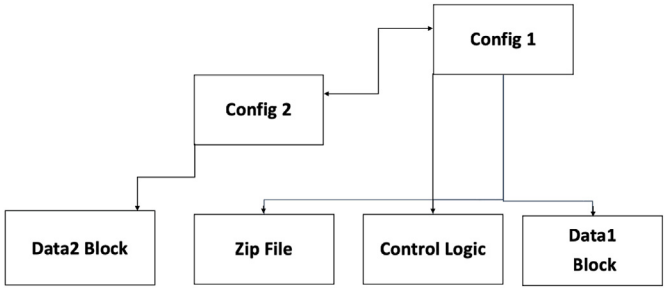

Fig. 7. Pointers in Configuration block to other critical memory blocks.

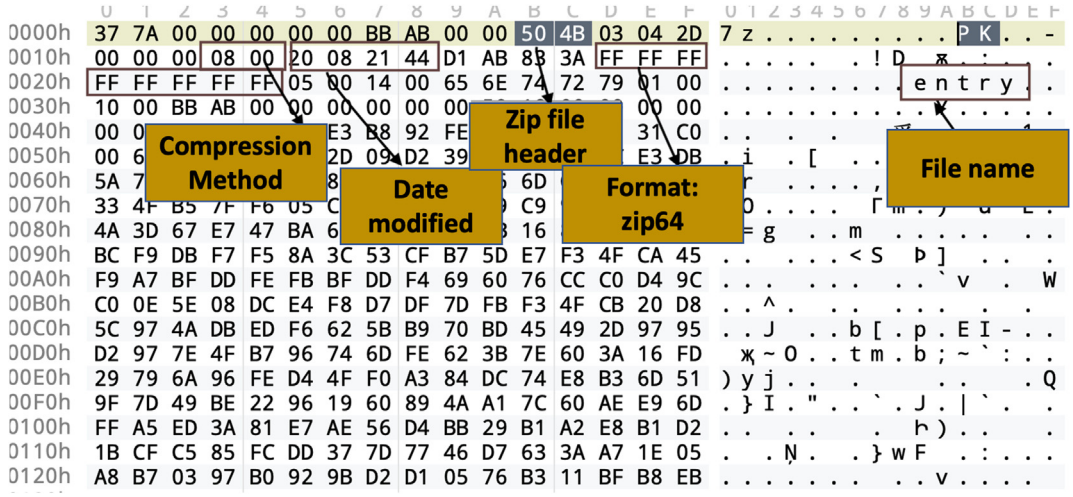

Fig. 8. Extracted metadata file in zip format.

"entry", is usually located in the external RAM area and begins with zip file signature bytes:/x50/x4b/x03/x04 as shown in Fig. 8. After decompressing it, we noticed that it contains information about the semantics of data objects used in the control logic and project information as shown in Fig. 9. The metadata file is not necessary for the control execution, but an attacker can use it to gain knowledge

```
MetaDataEntity xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
001/XMLSchema">
 <Devices>
   <DeviceMetadata>
     <Name>MyController</Name>
   </DeviceMetadata>
 </Devices>
 <ModemReference>No Modem</ModemReference>
 <T>
   <Index>0</Index>
   <Type>TP</Type>
   <Preset>5</Preset>
   <Base>OneSecond</Base>
 </T>
 <T>
   <Index>1</Index>
   <Type>TP</Type>
   <Preset>10</Preset>
   <Base>OneSecond</Base>
 </T>
 <T>
   <Index>2</Index>
   <Type>TP</Type>
   <Preset>10</Preset>
   <Base>OneSecond</Base>
 </T>
 <Pids />
 <Pous>
   <PouMetadata>
     <Name>New POU</Name>
     <Rungs>
       <RungMetadata>
         <MainComment />
         <Comments />
         <Name>Green </Name>
         <IsLadderSelected>false</IsLadderSelected>
       </RungMetadata>
```

Fig. 9. Decompressed metadata file.

about the physical process controlled by the PLC. Furthermore, an intruder can overwrite the metadata file with malicious code, so its absence in a memory dump might indicate tampering.

3.2.3.2. Control logic. Control logic is the PLC program that is usually written using engineering software and then downloaded to the PLC. Although engineering software offers more than one programming language, ladder logic is typical. It mimics circuit diagrams with "rungs" of logic read left to right. Each rung represents a specific action controlled by the PLC, starting with an input or a series of inputs (contacts) that result in an output (coil). Control logic runs in cycles and processes input signals to produce output signals. Our ladder logic program consists of three rungs, each with two input memory bits, a timer, and an output memory bit. The program is a simulation of a traffic light such that the green light stays on for 10s and then turns off, an orange light turns on for 5s, and then a red light turns on for 10s. Fig. 10 and Fig. 11 shows an example of PLC control logic in both ladder logic language and instruction language.

Control logic can be considered one of the most important forensic artifact that a forensic investigator should acquire and analyze at the time of a cyber incident or system failure. To find the address of a control logic structure in our memory dump, we use the pointer reference that we previously found in the second configuration block. In our project, the control logic structure can be found at address 0x7CE10107 and its size is 142 bytes. To better understand the logic and compare it to the code in our engineering software, we tried decompiling it using the Eupheus decompiler (Kalle et al., 2019). For the instructions that are not recognized by the decompiler, we conducted differential analysis, after adjusting instructions in rungs, to find the low-level representation of the ladder logic. Fig. 12 shows the ladder logic that we extracted using our tool and the translation from RX630 instructions to the instruction-set language.

3.2.3.3. Project information. We found that the chip RAM has a "Communication" object that contains information about the device model, project name and some network information. Fig. 13 shows the information found about the device name, device IP address, device MAC address, the subnet mask, and the version of the firmware installed on the device. **Project Name**: Traffic Light **Device model**: M221 TM221CE16R **Device IP address**: 192.168.1.221 **subnet mask**: 255.255.255.0 **firmware version**: V1.4.0.3.

3.2.3.4. IO data. To communicate with field devices, a PLC uses I/O modules to process digital/analog input from input devices and outputs digital/analog signals to output devices. Because I/O data controls the behavior of the PLC, an attacker can tamper with the values to manipulate the physical process. Hence, acquiring the I/O data is important for forensic analysis. In a previous step, we found that the start address for the data block is 0×00800107. We used the start address and the size specified in the configuration block to extract the data block. Because the data block does not reflect any comments or tags from the engineering software, we again use differential analysis to figure out the I/O data values in our memory dump. To accomplish that, we extract the memory of the same program while the program is running at different points in time. As seen in Fig. 14, the dynamic values in red represent the I/O values.

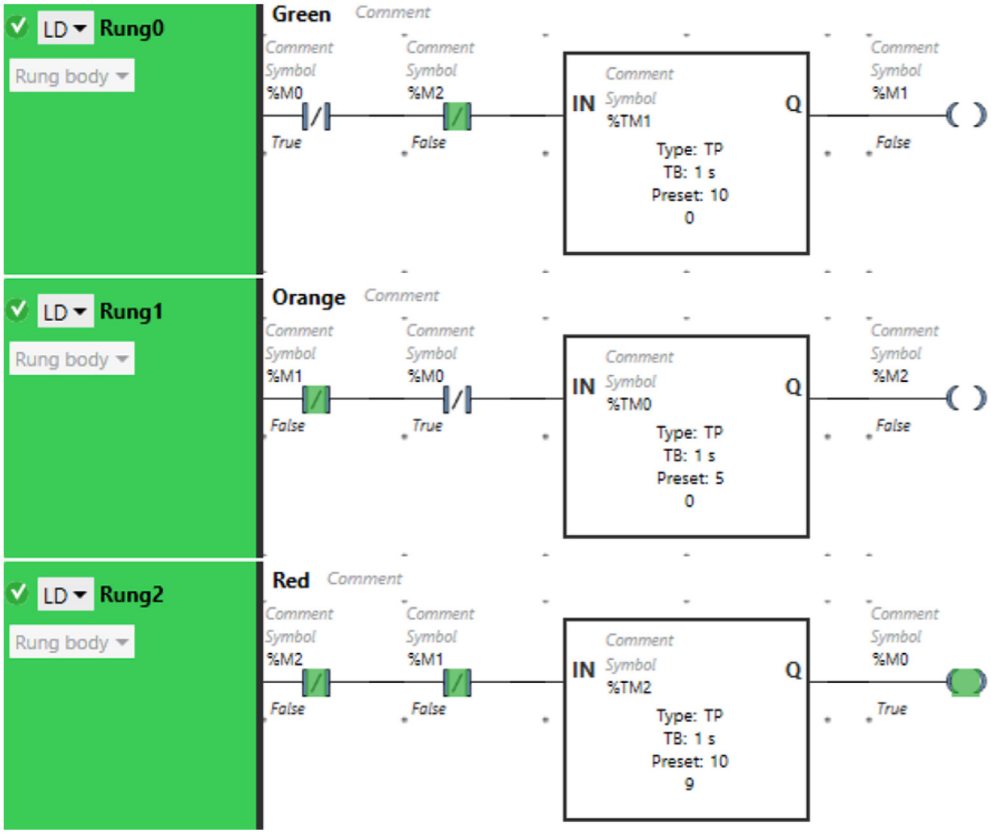

Fig. 10. Control logic in ladder logic language.

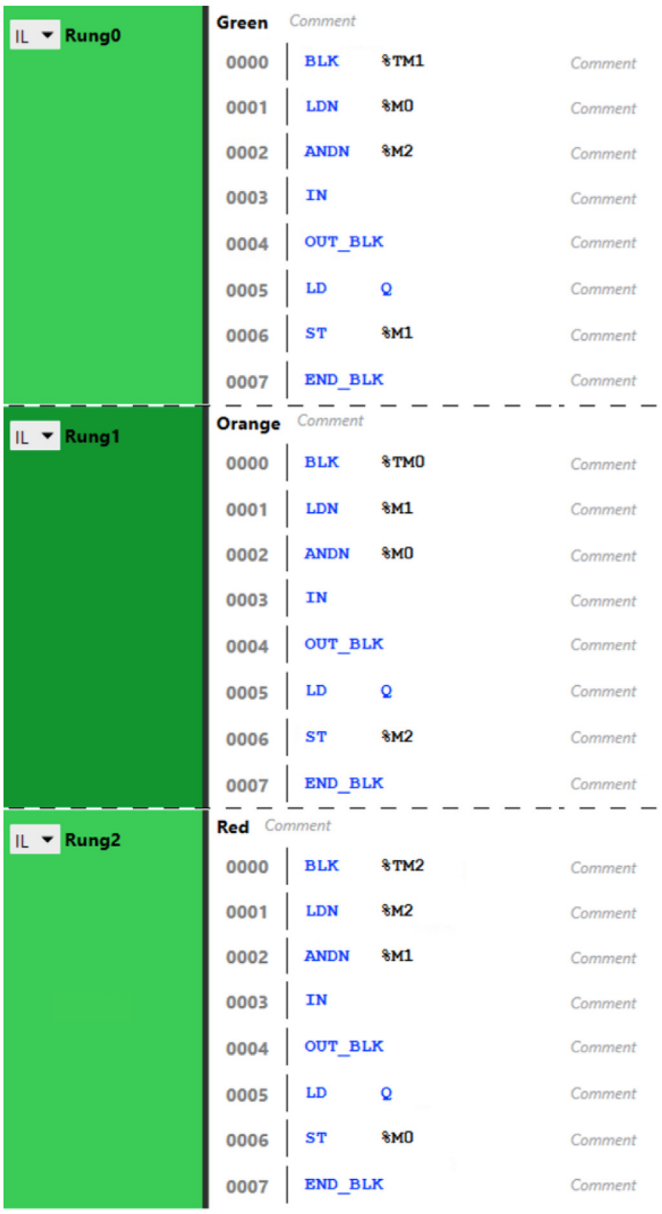

Fig. 11. Control logic in Instruction Language.

4. Evaluation: traffic light scenario

To evaluate our memory analysis tool, we use engineering software to download ladder logic that simulates the traffic light. Then, we manipulated the ladder logic to light the red and green lights simultaneously. Though we used the engineering software to manipulate the code, in a real-life scenario, an attacker can manipulate the code in the memory without using the engineering software as demonstrated in (Yoo et al., 2019a) and (Kalle et al., 2019). We then extracted the control logic memory code, based on the steps described in section 3.

The ladder logic program consists of three rungs, each with a timer that controls the time the light will be on. In our scenario, the source of input signals is memory bits with values that change after each rung executed as shown in Fig. 15. In the first rung, the green light (%Q.0.0) is turned on for 10 s when memory bits %M0 and %M2 are false. In rung two, the orange light (%Q.0.1) is activated for 5 s if %M2 and %M1 are false. Finally, rung three energized the red light (%Q.0.2) if %M1 and %M0.

To have both the green and red lights turned on at the same time, we add an "OR" "Normally Open Contact" instruction to the "Green" rung, and we set the input to be (%M0). By doing so, the green light turns on in two cases: when %M0 and %M2 are false **OR** when %M0 is true. Fig. 16 shows the manipulated control logic.

[H] To verify that we can detect control logic manipulation with our tool, we acquire a memory dump before and after manipulation. We then extract the memory code of the control logic block from both dumps and compare the two extracted blocks. As seen in Fig. 17, the instruction injected at the time of manipulation can be easily noticed (highlighted in yellow): **2206F6700000**. After decompiling the RX630 code, we find out that it is equivalent to the IL instruction that we added using the engineering software: **OR %M0**.In addition to analyzing the raw memory bytes, the forensic analyst can also take notes of the size of a code block. Under normal circumstances, the size of the code block should not change as long as the PLC is running the same project. When code is injected into memory, the size of the extracted control logic will most likely change.

5. Discussion and future work

PLCs were not developed with security in mind, hence, they are an easy target for cyber attacks. Following a cyber incident, the forensic analyst needs to get to the root cause of the incident to recover the system as quickly as possible. Memory forensic artifacts

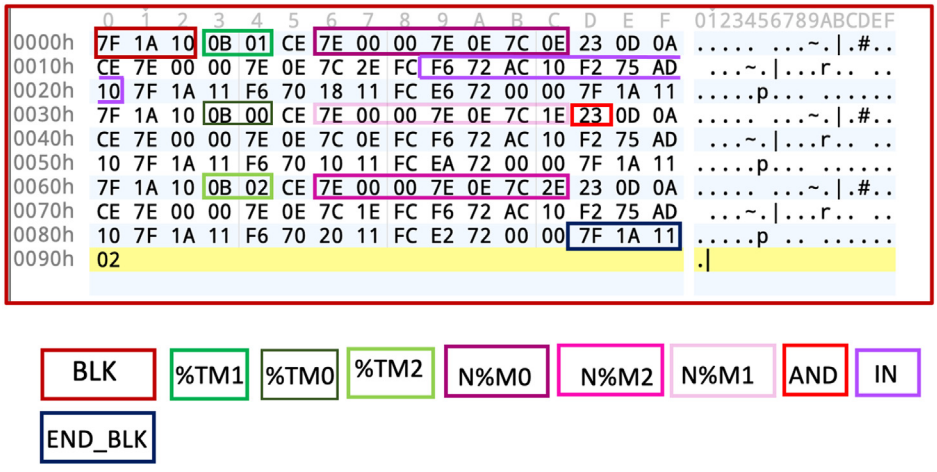

Fig. 12. Control Logic extracted from memory dump.

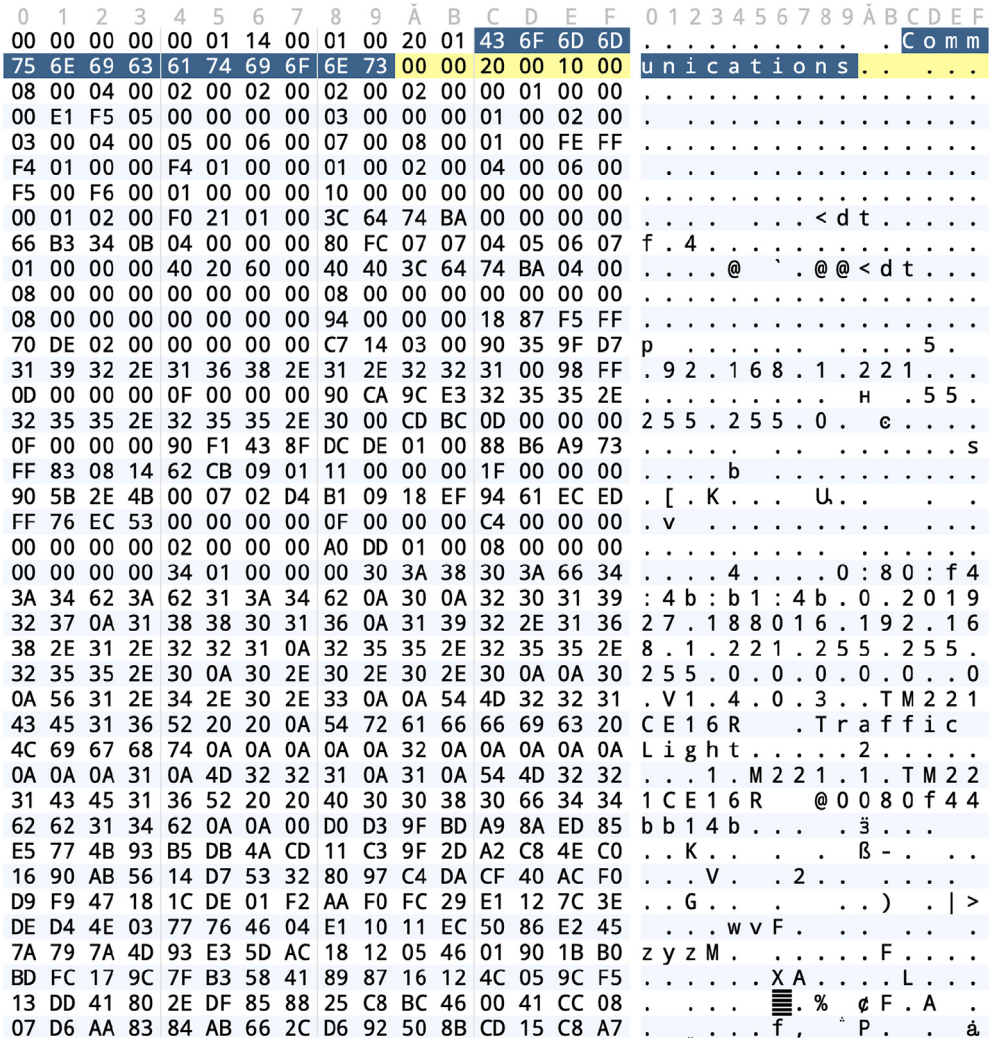

Fig. 13. Communication object from on-chip RAM.

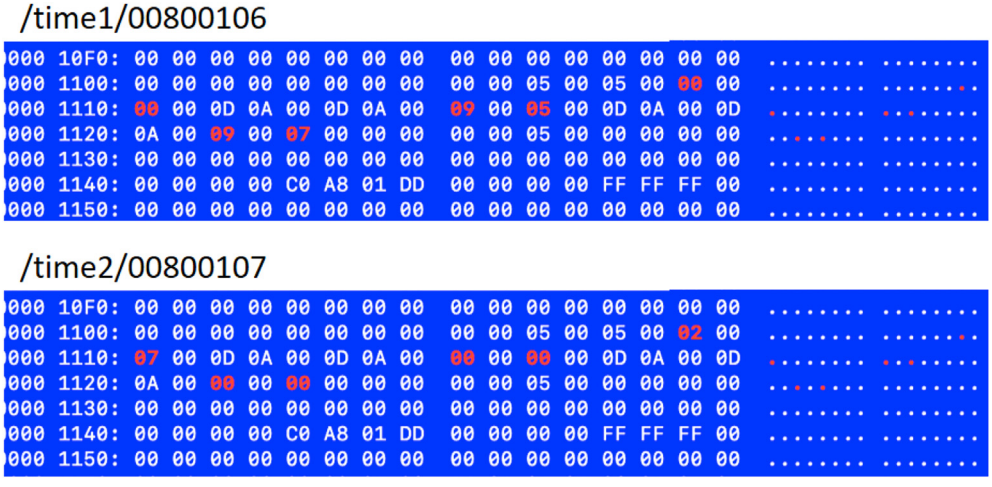

Fig. 14. Data Block with different values at different points of time.

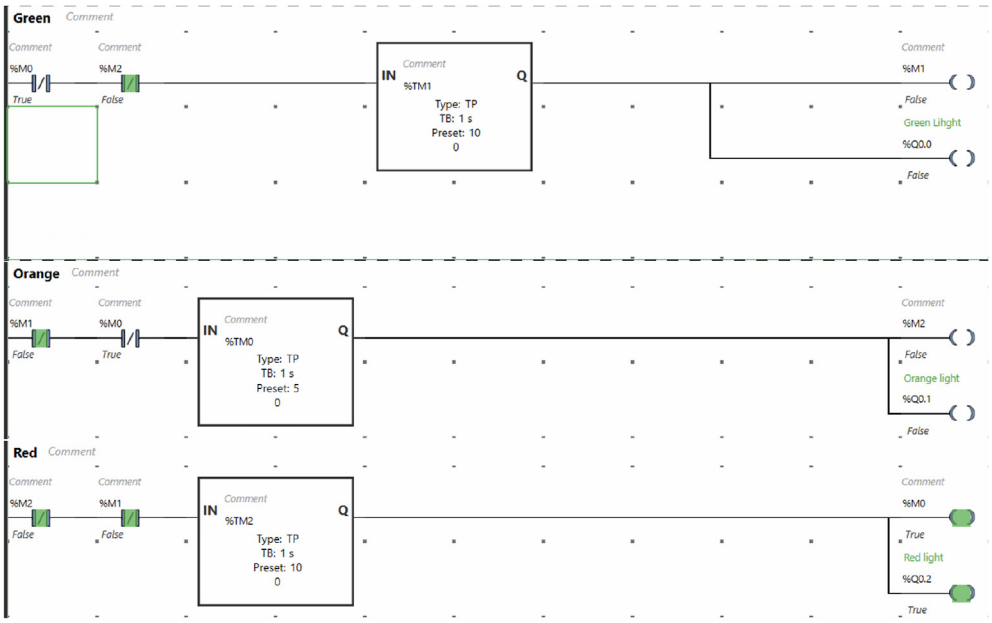

Fig. 15. Control Logic for traffic light.

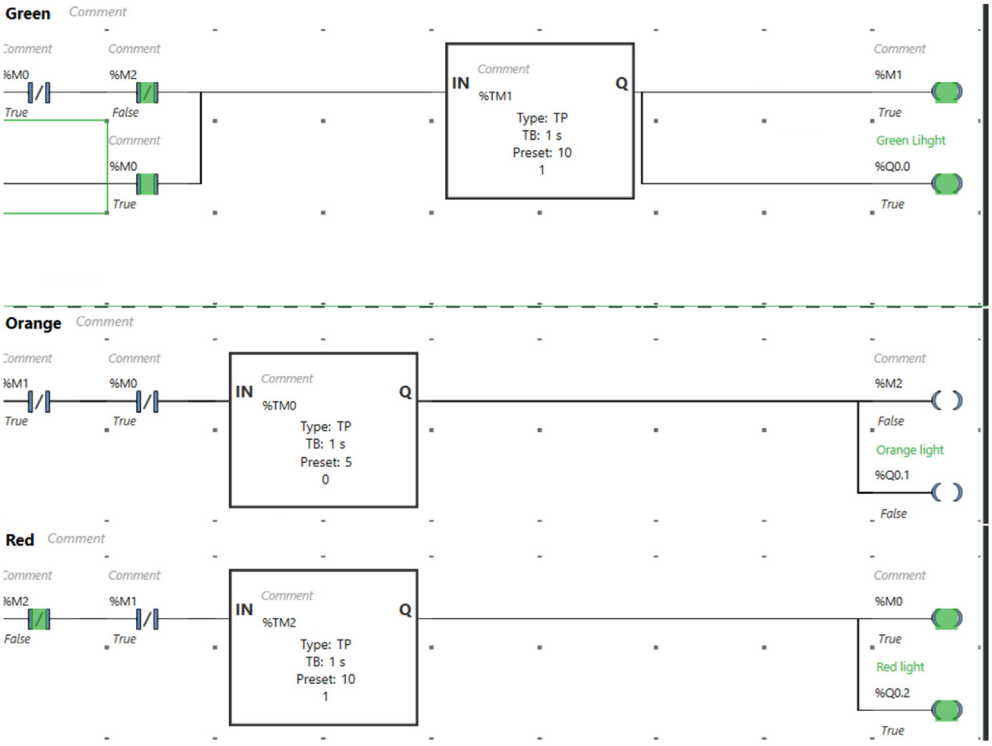

Fig. 16. Manipulated traffic light control logic: Green turns on while red only is supposed to be on.

provide important information to represent the system state. However, the diverse architecture and protocols used in PLCs make it challenging to acquire and analyze PLC memory images. In our memory analysis methodology, we rely on knowledge acquired by reverse engineering the communication protocol, and applying differential analysis to learn about the layout of the memory. Since the configuration file is always written to the same address, our approach can be used to analyze the memory of other devices of the same model. For future work, we aim to automate the detection of memory blocks and analyze the memory of PLCs from other vendors.

6. Conclusion

Memory forensic artifacts provide important information to represent the system state. However, the diverse architecture and

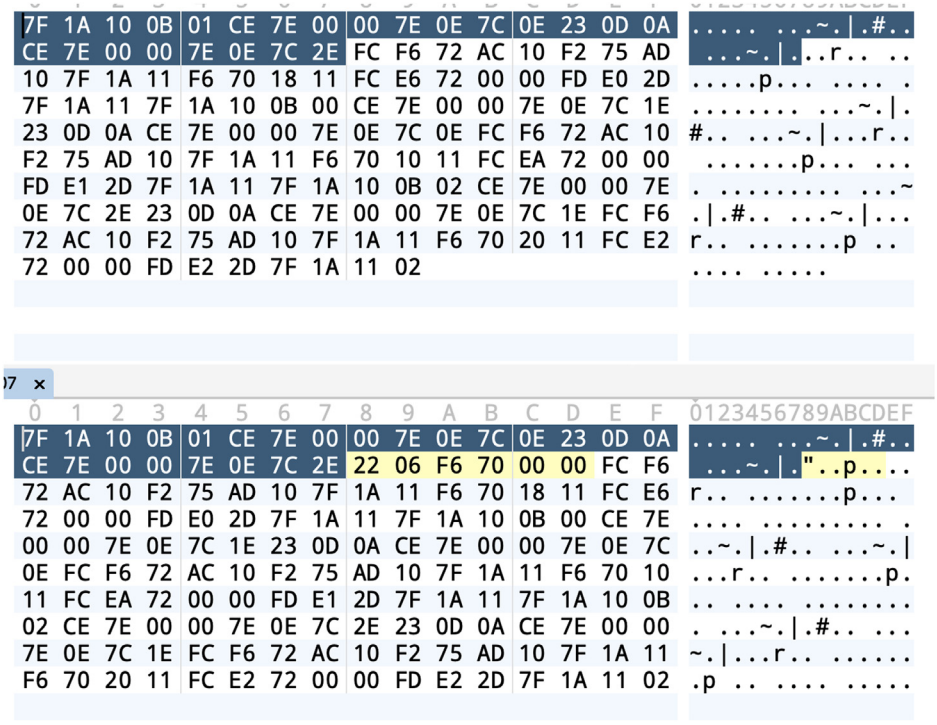

Fig. 17. Extracted control logic shows the difference between original logic and manipulated logic.

protocols used in PLCs make it challenging to acquire and analyze PLC memory images. In this paper, we take a step forward in this direction and present the memory forensic analysis of Schneider Modicon M221 PLC. We create a memory forensic profile by identifying the static configuration block on the device and tracing the pointers that reference other memory blocks and data structures. We identify essential memory regions and extract important forensic artifacts including the control logic and the I/O data. We evaluate our analysis methodology by simulating a traffic light attack.

References

Ahmed, I., Obermeier, S., Naedele, M., Richard III, G.G., 2012. Scada systems: challenges for forensic investigators. Computer 45 (12), 44–51.

Ahmed, I., Roussev, V., Ali Gombe, A., 2015. Robust fingerprinting for relocatable code. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, CODASPY '15. Association for Computing Machinery, New York, NY, USA, pp. 219–229. https://doi.org/10.1145/2699026.2699104.

Ahmed, I., Roussev, V., Johnson, W., Senthivel, S., Sudhakaran, S., 2016. A scada system testbed for cybersecurity and forensic research and pedagogy. In: Proceedings of the 2nd Annual Industrial Control System Security Workshop. ICSS '16, Association for Computing Machinery, New York, NY, USA, pp. 1–9. https://doi.org/10.1145/3018981.3018984.

Ahmed, I., Obermeier, S., Sudhakaran, S., Roussev, V., 2017. Programmable logic controller forensics. IEEE Security & Privacy 15 (6), 18–24.

Awad, R.A., Beztchi, S., Smith, J.M., Lyles, B., Prowell, S., 2018. Tools, techniques, and methodologies: a survey of digital forensics for scada systems. In: Proceedings of the 4th Annual Industrial Control System Security Workshop, pp. 1–8.

Awad, R.A., Lopez, J., Rogers, M., 2019. Volatile memory extraction-based approach for level 0-1 cps forensics. In: 2019 IEEE International Symposium on Technologies for Homeland Security (HST). IEEE, pp. 1–6.

Ayub, A., Yoo, H., Ahmed, I., 2021. Empirical study of plc authentication protocols in industrial control systems. In: 2021 IEEE Security and Privacy Workshops. SPW), pp. 383–397. https://doi.org/10.1109/SPW53761.2021.00058.

Basnight, Z., Butts, J., Lopez Jr., J., Dube, T., 2013. Firmware modification attacks on programmable logic controllers. Int. J. Critical Infrastruct. Protect. 6 (2), 76–84.

Bhatt, M., Ahmed, I., 2018. Leveraging relocations in elf-binaries for linux kernel version identification. Digit. Invest. 26, S12–S20. https://doi.org/10.1016/j.diin.2018.04.022.

K. Chow, R. Chan, Plc Forensic Analysis: Forensic Traces of Industrial Control System Attacks.

Davis, M., McInnes, B., Ahmed, I., 2022. Forensic investigation of instant messaging services on linux os: discord and slack as case studies. Forensic Sci. Int.: Digit. Invest. 42, 301401. https://doi.org/10.1016/j.fsidi.2022.301401 proceedings of the Twenty-Second Annual DFRWS USA.

Denton, G., Karpisek, F., Breitinger, F., Baggili, I., 2017. Leveraging the srtp protocol for over-the-network memory acquisition of a ge fanuc series 90-30. Digit. Invest. 22, S26–S38.

Garcia, L., Zonouz, S., Wei, D., De Aguiar, L.P., 2016. Detecting plc control corruption via on-device runtime verification. In: 2016 Resilience Week (Rws).

Kalle, S., Ameen, N., Yoo, H., Ahmed, I., 2019. CLIK on PLCs! Attacking control logic with decompilation and virtual PLC. In: Proceeding of the 2019 NDSS Workshop on Binary Analysis Research. BAR).

Mulder, J., Schwartz, M., Berg, M., Houten, J.V., Urrea, J.M., Pease, A., 2012. Analysis of field devices used in industrial control systems. In: International Conference on Critical Infrastructure Protection. Springer, pp. 45–57.

Qasim, S.A., Lopez, J., Ahmed, I., 2019. Automated reconstruction of control logic for programmable logic controller forensics. In: Lin, Z., Papamanthou, C., Polychronakis, M. (Eds.), Information Security. Springer International Publishing, Cham, pp. 402–422.

Qasim, S.A., Smith, J.M., Ahmed, I., 2020. Control logic forensics framework using built-in decompiler of engineering software in industrial control systems. Forensic Sci. Int.: Digit. Invest. 33, 301013.

Qasim, S.A., Ayub, A., Johnson, J., Ahmed, I., 2022. Attacking the iec 61131 logic engine in programmable logic controllers. In: Staggs, J., Shenoi, S. (Eds.), Critical Infrastructure Protection XV. Springer International Publishing, Cham, pp. 73–95.

Rais, M.H., Awad, R.A., Lopez Jr., J., Ahmed, I., 2021. Jtag-based plc memory acquisition framework for industrial control systems. Forensic Sci. Int.: Digit. Invest. 37, 301196.

Rais, M.H., Awad, R.A., Lopez Jr., J., Ahmed, I., 2022. Memory forensic analysis of a programmable logic controller in industrial control systems. Forensic Sci. Int.: Digit. Invest. 40, 301339.

Senthivel, S., Ahmed, I., Roussev, V., 2017. Scada network forensics of the pccc protocol. Digit. Invest. 22, S57–S65.

Senthivel, S., Dhungana, S., Yoo, H., Ahmed, I., Roussev, V., 2018. Denial of engineering operations attacks in industrial control systems. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY '18. ACM, New York, NY, USA, pp. 319–329.

Wu, T., Nurse, J.R., 2015. Exploring the use of plc debugging tools for digital forensic investigations on scada systems. J. Digit. Forensics Security Law 10 (4), 7.

Xie, Y., Chang, R., Jiang, L., 2022. A malware detection method using satisfiability modulo theory model checking for the programmable logic controller system. Concurrency Comput. Pract. Ex. 34 (16), e5724.

Yoo, H., Ahmed, I., 2019a. Control logic injection attacks on industrial control systems. In: Dhillon, G., Karlsson, F., Hedström, K., Zúquete, A. (Eds.), ICT Systems Security and Privacy Protection. Springer International Publishing, Cham,

pp. 33–48.
Yoo, H., Kalle, S., Smith, J., Ahmed, I., 2019b. Overshadow plc to detect remote control-logic injection attacks. In: Perdisci, R., Maurice, C., Giacinto, G., Almgren, M. (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment. Springer International Publishing, Cham, pp. 109–132.
Zhu, R., Zhang, B., Mao, J., Zhang, Q., Tan, Y.-a., 2017. A methodology for determining the image base of arm-based industrial control system firmware. Int. J. Critical Infrastruct. Protect. 16, 26–35.
Zonouz, S., Rrushi, J., McLaughlin, S., 2014. Detecting industrial control malware using automated plc code analytics. IEEE Security & Privacy 12 (6), 40–47.
Zubair, N., Ayub, A., Yoo, H., Ahmed, I., 2022a. Control logic obfuscation attack in industrial control systems. In: 2022 IEEE International Conference on Cyber Security and Resilience. CSR), pp. 227–232. https://doi.org/10.1109/CSR54599.2022.9850326.
Zubair, N., Ayub, A., Yoo, H., Ahmed, I., 2022b. Pem: remote forensic acquisition of plc memory in industrial control systems. Forensic Sci. Int.: Digit. Invest. 40, 301336.