Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS 2022 EU - Selected Papers of the Ninth Annual DFRWS Europe Conference

Memory forensic analysis of a programmable logic controller in industrial control systems

Muhammad Haris Rais [a, *], Rima Asmar Awad [b], Juan Lopez Jr. [b], Irfan Ahmed [a]

[a] *Virginia Commonwealth University, Richmond, VA, 23284, USA*
[b] *Oak Ridge National Laboratory, Oak Ridge, TN, 37830, USA*

ARTICLE INFO

Article history:

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

ABSTRACT

In industrial control systems (ICS), programmable logic controllers (PLCs) are used to automate physical processes such as nuclear plants and power grid stations, and are often subject to cyber attacks. As in conventional IT domain, the memory analysis of the PLCs can help answer important forensic questions about the attack, such as the presence of malicious firmware, injection of modified control logic (the program running on the PLC), and manipulation of I/O devices (e.g., sensors and actuators). Unlike conventional IT domain, PLCs have heterogeneous hardware architecture, proprietary firmware and control software, making it challenging to employ a unified framework for their memory forensics. For merely extracting artifacts of forensic importance, reverse-engineering the firmware is a tedious task, and the effort needs to be repeated for every PLC model. As a community, a step-wise approach to tackle this challenge is to analyze the memory of specific PLCs, and subsequently find a generic framework applicable to all PLCs. Our work is a step forward in this direction. By following a methodology that focuses on the functional layer of PLCs instead of reverse engineering the firmware, we analyze the digital forensic artifacts available in a common PLC, Allen-Bradley ControlLogix 1756-L61. Before diving into the memory dump, we analyze the PLC control software to create a list of important artifacts that are sure to exist in the PLC memory dump. The approach employs a setup where PLC control software RSLogix-5000 is connected to the PLC, and the memory dump can be obtained as and when needed. We create test cases that sequentially highlight each category of artifacts, followed by an examination of the resultant impact on memory. After attaining the listed artifacts, we employ conventional string and known data searches to extract interesting information present in this PLC's memory. The memory analysis profile, presented as a Python library and shared with the community, can help a forensic investigator to readily extract forensic artifacts from the same model's controller. The adopted approach may help researchers in creating memory profile of other PLCs, and ultimately formulating a generic PLC memory analysis framework.

1. Introduction

In industrial control systems (ICS), programmable logic controllers (PLCs) are intelligent and rugged embedded systems running control applications to monitor and control a physical processes such as gas pipelines, nuclear plants, and wastewater treatment facilities. Attackers target PLCs to sabotage physical processes (Garcia et al., 2017; Kalle et al., 2019; Senthivel et al., 2018; Yoo et al., 2019; Ayub et al., 2021; Qasim et al., 2021). For example, an attacker can change the condition to operate the safety release valve causing the internal pressure of a boiler to exceed beyond safety limits resulting in an explosion. A sophisticated attacker can also engineer the feedback sent by the PLC to avoid early warning to the operator. Although researchers have recommended independent monitoring techniques to attain authentic state of an industrial process (Rais et al., 2021a, 2021b; Qasim et al., 2019), attacks that manipulate inline feedback have higher chances to succeed. Conventional IT forensic analysis of the control software may not provide the required details. However, the forensics of the PLC's memory can provide important information about the attack, such as the attacker's details, modifications in the firmware and the application, and the actual state of the controlling Input/Output (IO) devices (Ahmed et al., 2012, 2017).

* Corresponding author.
E-mail address: raismh@vcu.edu (M.H. Rais).

https://doi.org/10.1016/j.fsidi.2022.301339

PLC memory forensics is challenging since PLCs have proprietary, legacy hardware with heterogeneous architectures, firmware, and programming applications. Currently, there is no memory forensics methodology that provides comprehensive coverage of the memory contents in a PLC. The existing forensic efforts are mostly limited to extracting and analyzing firmware from a PLC such as Basnight (2013), Avatar (Zaddach et al., 2014), Zhu et al. (2017), and Mulder et al. (2012).

Analyzing PLC's memory by reverse-engineering the proprietary firmware is not a simple and replicable task. Moreover, a forensic investigator is not interested in the firmware detailed working. A methodology is required to cut down the time and efforts by focusing on extracting the forensically interesting information. In this paper, we employ a differential analysis based methodology to extract and analyze digital forensic artifacts in Allen-Bradley ControlLogix 1756-L61 controller's memory dump. We successfully extract the running firmware, the control application (commonly called Control-logic), the physical process state information measured through the input from sensors and output to actuators, the PLC operational configuration data, and important event logs. By employing a sequence of planned test cases and analyzing the resultant memory variations, we generate a memory profile comprising a set of rules to extract artifacts from a memory dump. The profile generation task includes identifying data structure definitions, searching data structure instances in a memory dump, and formulating the rules. The resultant profile embedded in a Python library can be used by the community to instantly acquire forensic artifacts from an unseen memory dump.

The rest of the paper is organized as follows. Section 2 covers the background and the related work, followed by the experiment's setup presented in section 3. Section 4 describes the proposed methodology and its implementation. In section 5, we present the extracted forensic artifacts, followed by the discussion, and the conclusion.

2. Background and related work

2.1. Background

Fig. 1 presents a simplified architectural view of a PLC comprising three layers: the hardware, the firmware, and the application layer. The rugged hardware is augmented with a variety of Input/Output modules connecting the PLC to the corresponding physical process. Typically, a PLC runs the control application in an infinite loop. During each cycle, the PLC firmware acquires the physical process state through connected sensors, employ the control logic to find the new output states, and push them to the actuators through the output module.

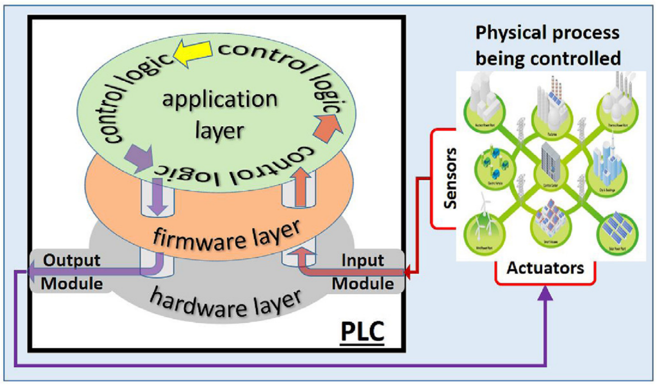


Fig. 1. PLC architecture.

2.2. Related work

Although no existing work focuses on the PLC memory forensic analysis, researchers have attempted to extract a specific artifact from the memory, or find forensic artifacts from the captured network traffic.

Zhu et al. (2017) proposed an automated method to determine the image base of firmware based on the jump tables data in the firmware. The authors collect ten firmware from vendors' websites to identify the firmware's jump table and calculate the image base to evaluate the algorithm. The experimental results show that the proposed algorithm is effective for the firmware that stores the absolute addresses in the jump table.

In (Basnight, 2013), Basnight presents a firmware modification attack scenario, where the PLC's firmware is extracted, analyzed, and reverse engineered to derive the firmware update validation method. The vulnerabilities of the validation algorithm are exploited, and a firmware modification attack was conducted. The author discusses that the firmware validation algorithm suffers form design weaknesses making firmware modification feasible for the attackers.

Mulder et al. (2012) analyzed the internals of control system field devices. They explained that the analysis of ICS field devices' firmware is more difficult because SOCs and custom components are potentially unique, poorly documented, and few analysts have experience with them. The authors analyzed the PLC hardware for a deeper understanding of the device's design. The study outlines few techniques and challenges for the firmware extraction. Due to excessive time required in firmware reverse-engineering, the authors also proposed to analyze PLC's backplane communication, instead.

Wu et al. (Wu and Nurse, 2015) investigates the possibility of using the PLC debugging tools to acquire memory data. The results of their experiment indicate that PLC Logger can be used to communicate and acquire data from the memory addresses of the PLC. However, the PLC logger authors indicate that the approach is limited to the use of a specific brand of Siemens PLC and one debugging tool.

Denton et al. (2017) examines the GE-SRTP network protocol, a proprietary protocol developed and used by General Electric. The protocol is reverse-engineered and then analyzed in relation to the PLC requirements. The authors are able to change the logic of the program running on the PLC. The developed tool is also able to communicate directly with the PLC, and access memory registers.

Findrik et al. (Findrik et al., 2018) introduces PLCBlockMon, a functional block for Siemens S7 controllers implemented in ladder diagram to provide data logging and memory acquisition capability. Since PLCBlockMon runs in the device's memory, it is only as secure as the load memory of the PLC, and further memory verification is required.

Sushma et al. (Kalle et al., 2019) utilizes the network traffic to extract and manipulate the control logic transferred from the engineering software to the PLC. The approach is used for control logic extraction, while other forensic artifacts, such as a compromised firmware, are not covered. Even for the control logic extraction from the network traffic or the memory dump, a fundamental difference is that the network traffic of transferring a program can be precisely pointed out and analysed. Searching and examining few KBs of control logic in a typically few hundred MBs of highly dynamic memory dump is not simple. The challenge is further augmented as firmware may organize the control logic in sections distributed across the memory. Qasim et al. (Qasim et al., 2020) extract the control logic from the captured network traffic of an attack instance by utilizing the built-in decompiler available with the Engineering software.

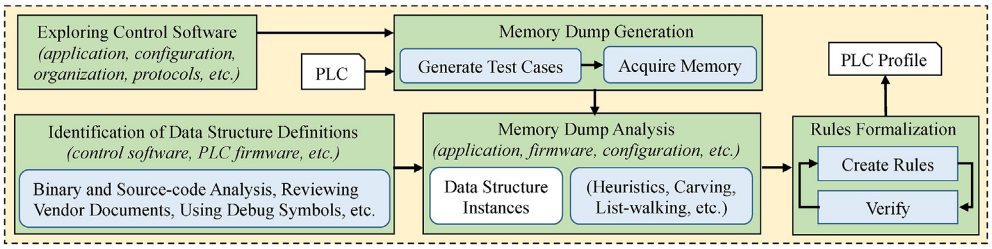

Exploring Control Software
(application, configuration, organization, protocols, etc.)
PLC
Memory Dump Generation
Generate Test Cases
Acquire Memory
PLC Profile
Identification of Data Structure Definitions
(control software, PLC firmware, etc.)
Binary and Source-code Analysis, Reviewing Vendor Documents, Using Debug Symbols, etc.
Memory Dump Analysis
(application, firmware, configuration, etc.)
Data Structure Instances
(Heuristics, Carving, List-walking, etc.)
Rules Formalization
Create Rules
Verify

Fig. 2. PLC memory analysis methodology.

Most of the related work focuses the firmware extraction and analysis, and does not deal with other useful forensic information in the memory dump. While firmware is an important artifact, finding the control-logic running on the PLC, and IO data states are also important and highly relevant in an attack scenario. Other relevant work analyzes PLC data remotely using ICS protocols, which is limited to specific PLCs, and only covers control-logic extraction. As their approach is based on application layer, the authenticity of the acquired data from a compromised PLC may be questioned.

Our study is focused on the forensic analysis, and endeavors to make the maximum use of the available memory dump in answering the forensic questions, such as:

- Is the extracted firmware unmodified?
- Are there any changes in the control-logic? If yes, what are those changes?
- Are the sensors and actuators states captured in the memory dump as expected by the user?
- Are there any traces of the attacker?
- Has PLC's operational mode been changed recently? (The state changes from Run to Program mode to update the control-logic)

3. Experiment's Setup

The experiment's setup presented in Fig. 4 comprises the PLC, the engineering software, a memory acquisition setup and the proposed analysis software. As our analysis strategy works on iteratively programming the PLC with a series of programs and then analyzing the memory, a reliable memory acquisition setup is a requirement in this experiment. While the memory analysis is discussed in detail ahead, we give a short overview of the remaining components in this section. The list of the equipment and software used in the study is outlined in Table 1.

Table 1
List of equipment and software used in the case-study.

S/No	Equipment/ Software	Details
1	PLC	Allen-Bradley 1756 A10 with modules L61 ControlLogix 5561, HSC/A, IF8, 2x OB32/A, DNB B, ENBT A
2	PLC Control PC	core i7-8700/16 GB RAM/Windows 10 for PLC control software: connected to PLC over Ethernet
3	JTAG Debugger	Segger J-Link ARM V8.00
4	Power Supply	Tekpower TP-3005D-3 Digital Variable DC Power Supply (any 24V DC 3A will work)
5	PLC control software	RSLogix 5000 version 20.05.00 (CPR 9 SR 10), FactoryTalk Activation Manager v 4.05.01
6	Debugger software	SEGGER -J-Link V6.80d suite
7	Python library	PyLink Python library by Square Inc. for debugger

3.1. *Allen-Bradley ControlLogix 1756-A10 PLC*

Allen-Bradley is a reputed brand in industrial automation equipment with PLC market share of around 22% (Statista, 2017). Depending upon the requirements and scale of the underlying physical process, Allen-Bradley categorizes their programmable controllers as MicroLogix, CompactLogix, and ControlLogix. ControlLogix 1756 is one of the largest control systems manufactured by Allen-Bradley for high availability and high performance applications. We use a 10 slot modular chassis (ControlLogix 1756-A10) for our study as shown in Fig. 3, running the latest firmware version 20.019. The chassis is populated with a power supply and a controller module, two digital and one analog I/O modules, one Ethernet and one DeviceNet communication module, and one counter module. The controller module (1756-L61) is the brain of the system, and runs the control application programmed through the engineering software- RSLogix 5000.

3.2. *Control software- RSLogix 5000*

The engineering software provides a programming environment for the PLCs through a simple GUI-based programming language, called Ladder logic. RSLogix 5000 also communicates with the PLC over a serial or IP channel to upload or download the control logic, and monitor the PLC status.

3.3. *JTAG based memory acquisition setup*

PLC's memory acquisition is not a well researched subject with very few options available. We selected Allen-Bradley 1756-A10 PLC due to the availability of a reliable hardware based memory acquisition system (Rais et al., 2021c) using the Joint Test Action Group (JTAG) port. Later standardized by IEEE 1149.1, JTAG is an industry initiative to simplify the testing of complex circuit boards by adding a small functionality in the integrated circuits (IC). JTAG interface is also used for debugging and programming tasks, and can read and write to the processor's controlled memory. To communicate between a PC and a PLC's processor through JTAG interface, an intermediate device is required to convert a standard PC interface (Serial, Ethernet, or IP) signals to JTAG compliant signals. In our setup, we employ Segger Microcontroller System's J-Link debugger to read the memory contents of the PLC.

4. PLC memory analysis methodology

Fig. 2 presents the proposed methodology to analyze the memory dump of the PLC.

4.1. *Exploring the engineering software*

The PLC engineering (programming) software is typically a vendor's proprietary software that provides an interface for

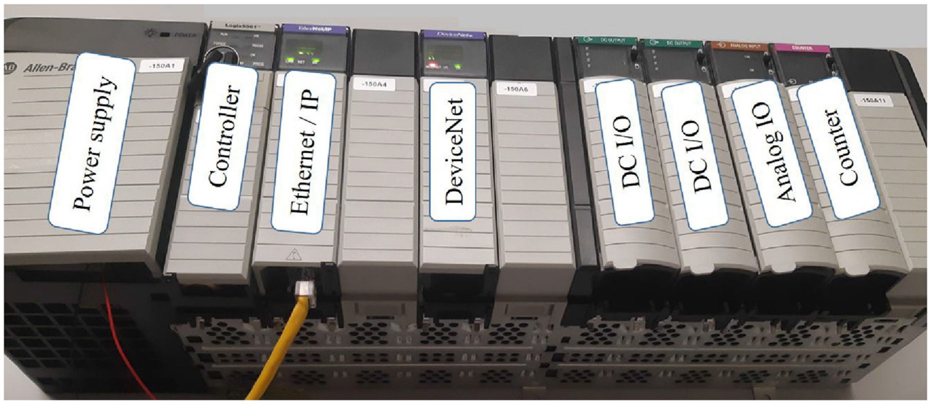


Fig. 3. Allen-Bradley ControlLogix 1756-A10 modular controller.

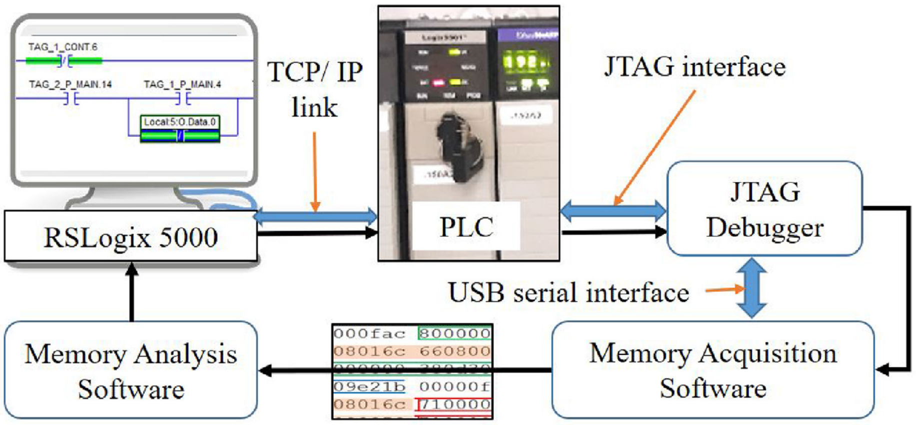


Fig. 4. Experiment setup for memory acquisition and analysis.

configuring, monitoring, and upgrading a PLC. Acquaintance with the engineering software serves two purposes: 1) Recognizing forensically essential data to search in a memory-dump, and 2) understanding the PLC configuration process required for generating test programs. Both of these aspects are required for the differential analysis. An efficient way to familiarize with the software is to connect it to a test PLC and explore the features. Following aspects should be given consideration during control software study.

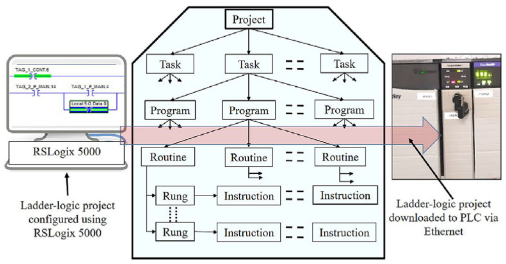
Project Organization. The vendors organize the project running on the PLC (referred to as "control logic" (Kalle et al., 2019)) in various ways. The understanding of the organization helps in assessing data structures instances, child–parent relationships, and their links with other structures. For instance, if the control-logic exists in a known hierarchical model (an example discussed in the case study), one can estimate the types and the count of data structures to be seen in the dump against a particular program.

Allen-Bradley uses a hierarchy of tasks, programs, and routines to configure a project that runs on the PLC. Fig. 5a presents the project organizational hierarchy. Every project (also called control application) has at least one task. A task may consist of one or more programs with multiple routines comprising multiple rungs and instructions. An instruction has a type and one or more operands. These operands, described as "tags", can be defined under various scopes. The physical IO tags representing the IO modules' actual pins connected to the physical sensors and actuators are defined under controller scope accessible to all programs. In contrast, logical tags (or variables) can be defined within a restricted scope applicable to a single program. The 1756-L61 controller supports 32 tasks with up to 100 programs per task.

Named Structures. A PLC project consists of few structures that can be named. For example, the physical IO pin groups, the logical variables, routines, programs, and tasks usually contain a name associated with each instance of the data structure. If a name can be configured in the program, it is likely to be seen in the memory dump. Rungs are numbered sequentially, and are not named. While doing the control software analysis, all named structures are marked for further analysis.

Unique Data Patterns. An essential part of every PLC is the Input and Output pins groups. A user programs the PLC to ensure particular status of output pins under certain logical conditions. To facilitate complex control flow of the program, logical variables are also supported in RSLogix 5000. If these groups of variables are set in a planned manner, corresponding locations in the memory dump can be identified.

PLC Configuration Data. Engineering software's study also helps in identifying different services (web/SMTP), open ports, network addresses, authentication schemes and databases, and logging information that should be searched in the memory dump. ControlLogix 1756-L61 can be configured to keep a backup project in the available SD-card and use it as per user-configured options. Fig. 5b presents a snapshot of RSLogix controller settings, showing the firmware image in the SD-card and the controller, alongside the backup activation settings. Through the license manager installed



(a) Project Organization in RSLogix 5000

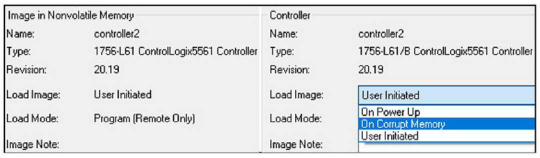
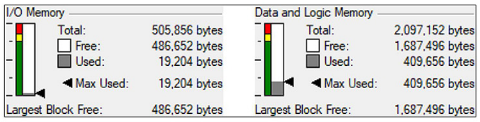


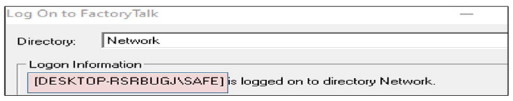
Image in Nonvolatile Memory		Controller	
Name:	controller2	Name:	controller2
Type:	1756-L61 ControlLogix5561 Controller	Type:	1756-L61/B ControlLogix5561 Controller
Revision:	20.19	Revision:	20.19
Load Image:	User Initiated	Load Image:	User Initiated
Load Mode:	Program (Remote Only)	Load Mode:	On Power Up / On Corrupt Memory / User Initiated
Image Note:		Image Note:	

(b) PLC setting for loading backup project in SD-card



I/O Memory		Data and Logic Memory	
Total:	505,856 bytes	Total:	2,097,152 bytes
Free:	486,652 bytes	Free:	1,687,496 bytes
Used:	19,204 bytes	Used:	409,656 bytes
Max Used:	19,204 bytes	Max Used:	409,656 bytes
Largest Block Free:	486,652 bytes	Largest Block Free:	1,687,496 bytes

(c) utilization of Logic and I/O Memory from RSLogix 5000



Log On to FactoryTalk

Directory: Network

Logon Information

[DESKTOP-RSRBUGJ\SAFE] is logged on to directory Network.

(d) Logon details from Rockwell Licensing Application

Fig. 5. Information found through exploring RXLogix 5000.

with RSLogix, we can find the login information of the RSLogix user as shown in Fig. 5d. If found in the memory dump, the information can have forensic significance.

Logs and Other Important Data. Other interesting information provided by RSLogix 5000 includes the PLC's operational mode, connected user's details, session duration, error logs etc. During the engineering software study, we explore the possible logs that PLC is providing to the control software. After seeing the logs structure in the engineering software, locating them in the memory dump gets easier.

4.2. *Memory dump generation*

Generate Test Cases. After understanding the organizational hierarchy in Fig. 5a, a set of test cases is created containing a varying number of instances of the identified parameters. The test cases cover the interesting information spectrum from the control-logic to the logging data.

Acquire the Memory. The PLC is configured using RSLogix 5000 software installed on the control PC connected over Ethernet to the PLC. The experiment uses a Joint Test Action Group (JTAG) interface based memory-acquisition system for the test PLC. After identifying the memory regions in the initially acquired complete memory dump, we focus on partial acquisition of the interesting zone for each iteration, to cater for JTAG's slow acquisition speed. For instance, memory-region hosting the firmware is not acquired repeatedly during the control-logic recovery exercise.

4.3. *Identification of data structure definitions*

Data structure definitions may be acquired through vendor's documentation. We could not find any documentation at Allen-Bradley official website specifying the data structure definitions. The other option of PLC's firmware reverse-engineering has been partially conducted in the past (Basnight et al., 2013; Garcia et al., 2017). This experiment focuses on extracting the data structure definitions through dynamic differential analysis without reverse-engineering the firmware or control software binary files.

Dynamic Differential Analysis. The primary method used for finding relevant data structures definitions is the dynamic differential analysis of the control software. The approach involves setting a unique pattern, searching it in the memory, and applying manual analysis to figure out the data structure boundary and definition. For large data structures, the effort is focused on finding only the forensics-related fields aligned with the study's goals.

Control-logic related Structures. To find the control-logic associated data structures, a set of programs (called projects in RSLogix 5000) is created. Starting with a single input and output instruction project, the projects gradually get more complex to include all boolean instructions, multiple rungs, routines, and programs. To utilize string searches, unique and conspicuous names are used for all the projects' configurable named fields. We successfully identified the definitions of all boolean instructions, rung, branching, routine, and program structure.

IO Data related Structures. To identify IO data-related structures, the test program's logic is set to produce conspicuous ASCII characters for the tags. For example, a 4 bytes (32 pins) tag configured to produce the hex value “41 42 41 42″ is shown as “ABAB” in the dump. We identify the definitions of various types of physical and logical IO structures.

Configuration-related Structures. We can identify configuration-related data structures, including the time, time-zone settings, IP address, controller name, project's name, backup settings, backup files, etc.

4.3.1. *Extracting the data structure definitions for the named structures*

The named structures identified in section 4.1 are string searched in the memory dump to identify their location. The name or description may exhibit one or more of the following features.

- Name describes the complete structure
- Name is part of a bigger contiguous data structure
- Name is part of a spatially staggered, linked data structure

For instance, the name assigned to the controller is a complete structure. The string “time zone” is followed by the time zone data. In case of staggered data structure, the data fields or groups at different locations are linked together. To extract the complete data structure definition, we perform the following analysis on the “string” command's output.

- Analyze the preceding and succeeding data to identify the boundary markers of the structure
- Analyze the potential forward links available in the vicinity of the description data

- Identify and analyze the reverse links in the memory pointing to the description data being analyzed.

From software perspective, the functional task of a PLC is not very complex, and we do not expect deeply nested chained data structures in the memory dump. Therefore, the above analysis can be performed conveniently in a semi-automatic manner.

4.3.2. Utilizing unique patterns for IO data structure identification

The states of IO data pins or the logical tags are set in groups of bits. For example, the physical pins groups representing the digital IO blocks are usually defined as one byte, two bytes, four bytes structure etc., where each bit is representing a unique pin. The logical variables can also be defined in a similar fashion. A unique pattern (such as ABABABAB) assigned to a tag group can be easily searched in the memory dump. In few iterations of test programs, the tag locations can be confirmed. The analysis of the vicinity, forward and reverse links should be conducted to identify the starting and ending patterns (if any), and other fields associated with the tag group data structure.

4.4. Identifying data structure instances through memory dump analysis

Once the data structure definitions are finalized, we apply them to identify their instances in a memory dump. In this section, we discuss some useful forensic artifacts created by finding connectivity among the constituent smaller structures. The detailed artifacts are presented in section 5.

4.4.1. Identification and recovery of control logic

Technically, the control logic can be termed as a comprehensive data structure that represents the project running on the PLC. It contains a set of pairs of instructions and applied operands whose sequence is defined by the user in the control application.

The first step in the control logic extraction is identifying its location in the memory. As control logic uses references to the instructions and operands, an effective way is to generate a sequence of programs by repeating and atomically modifying the instructions and the operand groups, and conducting differential analysis to identify the locations of expected changes in the memory.

Once the location is identified, the next step is to correlate the project structure defined in the control software with the identified zone. For instance, we configure two routines in the control logic, one with two rungs, and one with three rungs. we expect to see two structures corresponding to routine, one comprising of two and other three structures corresponding to rungs. This type of analysis provides us the information of the project structure elements. Last step is to verify the identifiers used for each type of instruction.

4.4.2. Forensically important Logs Recovery

The control software study suggests the types of logs generated by the PLC. Manually triggering the log generating events, and searching them in the memory identifies the location and semantics of the logs. We observed that specific structures, when amended, do not overwrite the previous instances. Information displacement analysis over multiple calculated iterations helps identify the past activity information pattern (or log) that can be of great value in the forensic investigation. Carved data instances that are not linked anymore are retained for investigation.

4.4.3. Extracting the firmware

In the PLC's memory dump, we can expect more than one copy of the firmware, in the non-volatile memory and in the RAM. There can be situations when an attacker is able to modify only the RAM contents, thus both the copies are distinct forensic artifacts. Using the reference firmware from the vendor and applying the longest pattern matching techniques, we can identify the firmware locations, and extract it. Another way is to identify the image base addresses through PLC documentation or existing techniques in the literature, and fetch the file.

4.5. Rules formalization

The information extraction process is formalized into a set of rules applied to an unknown dump to extract the forensic information. The process involves rules creation and verification.

Creating Rules. The analysis phase is carried out based on the prior knowledge of the test cases. On the other hand, the rules generation process addresses how to search a new memory-dump for user-required information in a reliable manner. If multiple ways lead to the same information in the test cases, which one/ones should be considered in the rule creation? List-walking is preferred over data-carving in rule selection. To break a tie among the walks to the same destination, a heuristic search may be conducted to find a reliable beginning to start the walk towards each type of desired information.

Another critical aspect in rules creation is to extract and distinguish between the current and obsolete information. Even if required information is wholly recovered through list-walking, data-carving-based rules should be incorporated to extract forensically useful (de-linked) past information.

Verifying Rules. Representative and unused test cases are generated to verify the rules created above. The verification is carried out for all information categories and their constituents. If certain rules occasionally failed the verification standard and could not be rectified, they are still incorporated with a reduced confidence metric. We tested the offline and online modules with fifty different test cases to verify the performance for the recovered firmware, control-logic, physical and logical IO pins state, configuration data, and the mentioned logs. The finalized memory analysis profile passed all the test cases. For the interested researchers, we have shared a variety of memory dumps (and adding more), the corresponding engineering software project files, the extracted artifacts files along with the complete source code for the memory analysis.

The outcome of the rules formalization step is a set of rules that specify the “Memory Analysis Profile” for the particular PLC. A useful and efficient implementation of the profile is to incorporate the rules in a software tool. The profile can be readily used for the same PLC specifications for future requirements and further research.

5. Digital artifacts in ControlLogix 1756-L61

Table 2 presents ten important data structures out of the twenty two identified in the PLC memory. We have named them as per their purpose. Due to space constraint, we explain “Asg_DT” shown in Fig. 6, “Tag_desc_DT” shown in Fig. 7, “Asg_DT_LinkedList”, and “Mem_block_DT” as few interesting examples.

Asg_DT. This 40-bytes data structure is a generic assignment data structure used for the tasks, programs, routines, and physical and logical IO tags. It always has starting and ending dwords as”80 00 00 0A″, where 0A represents its size in dwords. We identify the following important fields in Asg_DT:

- pointer to a firmware routine
- pointer to the next instance of Asg_DT
- memory location that the tag is representing

Table 2
Important data structures found in the PLC memory.

Sr.	Name (Given)	Purpose of data structure	Starting Marker	Ending Marker	size (dwords)	Important Fields/Remarks
1	Asg_DT	Main DT defining physical/logical tags, programs, routines etc.	80 00 00 0A	80 00 00 0A	10	Pointers to: Description Definition, Next Asg_DT instance pointer (as liked-list)
2	Tag_desc_DT	Defines the name of the data structure instances	80 00 00 xx yy	80 00 00 xx	variable	xx defines dt's length in dwords yy defines name's length (bytes) followed by name (in ASCII)
3	Program_DT	Defines program's structure One occurrence per program	80 00 00 58	80 00 00 58	88	Specifies the routines and the tags associated with this program
4	Routine_DT	Defines routine's structural composition One occurrence per routine	80 00 00 32	80 00 00 32	50	Pointer of the routine's definition in the control-logic
5	Control_ Logic	Represents the configured project running on the PLC	80 00 0F AC *	—	variable	A group of DTs containing the entire project defined in RSLogix 5000. * Starts after this marker
6	Routine_Logic	Main data structure of control logic; shows the routine's configuration	80 00 00 xx	80 00 00 xx	variable	Contains a series of rungs of the routine as configured by user xx is the number of dwords
7	Rung	Child data structure of Routine_Logic; Shows rung's configuration	78 09	7A27	variable	Contains instructions and branching as configured by the user
8	Branching	Child data structure of Rung	70 00 00 00	73 00 00 00	variable	For parallel conditions in control-logic 71 00 00 00 represents next branch
9	Instruction	Represent instruction and operand	Nil	Nil	variable	For most boolean instructions, one dword for instruction & tag;
10	Mem_block_DT	Specify in-use memory start and end	80 00 00 01	80 00 00 01	variable	2 blocks in code & data memory region and 2 blocks in IO memory region

• pointer to the tag description (name)

Asg_DT Linked-List. Multiple instances of Asg_DT data structures are connected in a singly linked-list. Fig. 6 presents a list showing all the tags assignments instances for one program (only selected fields are shown). The last assignment has a next pointer field set to “Null”.

Tag_desc_DT. This data structure is used for setting the names of different data structure instances. For example, the names given to routines, programs, tags in the engineering software are assigned using this data structure. We name it as “Tag_desc_DT”. The length of the structure depends upon the characters in the configured name. The last byte of the first and the last dword (in little-endian format) represents the number of dwords in the data structure, while the first three bytes are always “80 00 00”. The first byte of the second dword represents the character-length of the name field. If the last character does not coincide with the dword boundary, we can expect up to three unused bytes. Fig. 7 shows one instance of “Tag_desc_DT” with the configured name as “TAG1”,

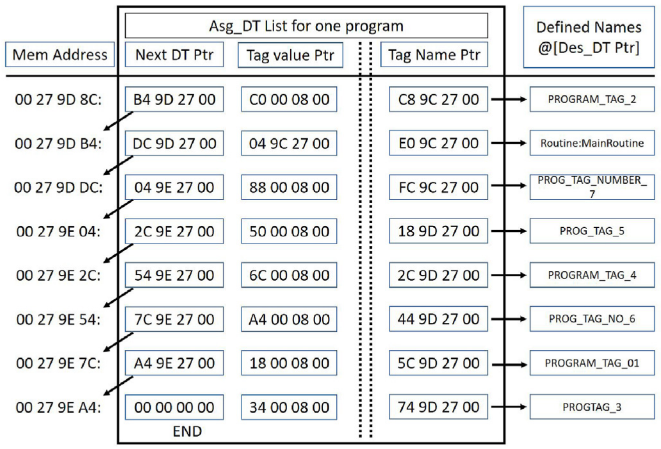


Fig. 6. Linked List of Asg_DT data structure for one program.

and having three unused bytes. It is a simple example where carving using boundary markers may not provide the correct result.

Mem_block_DT. As shown in Fig. 5c, ControlLogix-5561 uses distinct memory zones for Data, logic memory hosting for the control logic, and IO memory for the physical IO data. We find that each of the memory types is split into two parts. The first part starts at the beginning of the zone, while the second part ends at the memory zone's last byte. The first part's ending and the second part's starting addresses are floating to accommodate variation in the memory usage. The beginning and the end of each part have a fixed marker -“80 00 00 01″ in little-endian format. We name each part as “Mem_block_DT”. As mentioned, this data structure's size is variable, but a markers-based signature is reliable and consistent throughout the experiment.

Connecting Graphs. The main data structure used for assignment of any data structure instance in “Logic and Data memory” is “Asg_DT”. Every instance of Asg_DT has links to the description of the instance, its definition and the value. For example, for every program, routine or IO tag in the project, there is one instance of Asg_DT. As we traverse, the graphs differ for different types of data. Fig. 8a and b represent graphs for data structures of type “Routine_DT” and “Logical IO tag” respectively, showing one extra level for “Routine_DT”.

In the succeeding paragraphs, we mention some of the information artifacts found in the memory.

Firmware base-address. The firmware is identified at two locations in the memory dump. First instance occurs at address 0x00D00000 and ends at 0x00FAD1AF (in the volatile memory), and the second instance is based at address 0x0A1A0000 up to 0x0A44D1AF (in the non-volatile memory). Both instances match with the downloaded firmware. Zhu et al. (2017) also found same base-address for another controller from the same vendor.

PLC Backup in SD-Card. The SD-Card installed on the PLC for program backups also stores the firmware file. The firmware stored in the SD-card under “Executive.bin” filename contains a padding of “FF” until the last 4 hex digits of the address are all Fs making the size to 2,883,583 (or 0x2BFFFF) bytes. The two extracted firmware from the previously mentioned starting locations using the updated file-size results in perfect match with an exception of one

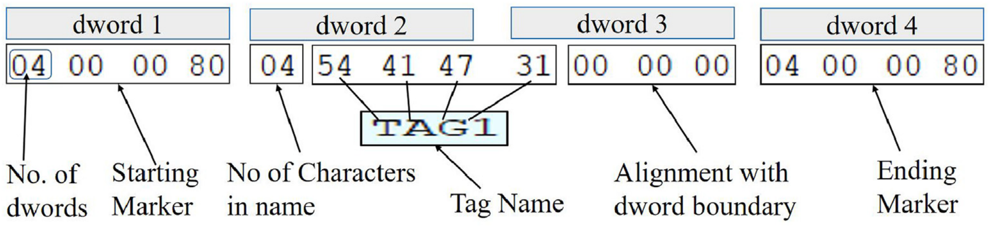


Fig. 7. Tag_desc_DT: data structure defining the tag's name.

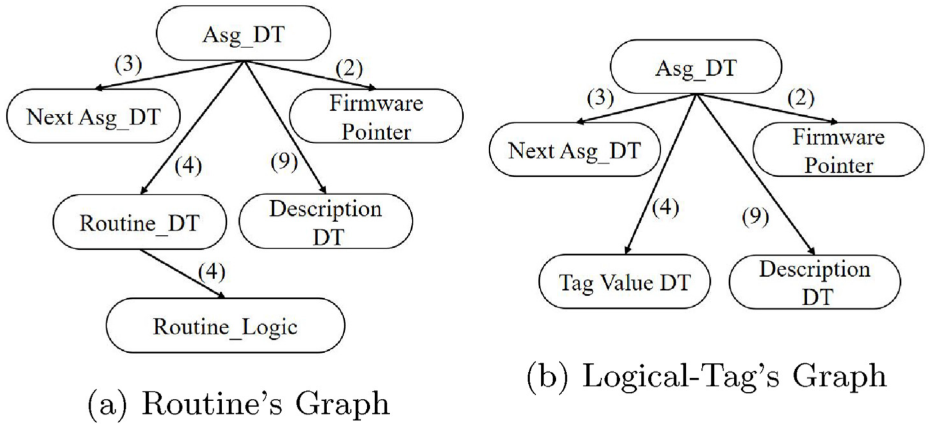


Fig. 8. Connecting graphs of data structures.

dword. The 7th dword from the beginning of the "Executive.bin" file is "3609″, while the memory instances and the vendor's website image contains "0000″ at this location. If the backup is restored, the memory still shows "0000″ at that location, indicating it to be a planned change for storing backup file on SDcard.

Memory Utilization. An unexpected change in memory utilization may point to some anomaly in the code. The sum of the sizes of both the Mem_block_DTs is exactly same as the memory utilization value mentioned in the engineering software. This finding also significantly reduces the search-zone for the control-logic and the IO data tags.

Control-logic Extraction. Recovery of the control-logic running on the PLC is an important goal of forensic investigation. Our code can identify the complete control logic in binary format. The code can successfully reverse-engineer multiple rungs, routines, programs and tasks. During project configuration, the routines are defined under programs. However, in the control-logic section in the memory-dump, the routines from all the programs are stacked together. Fig. 9a shows the RSLogix 5000 snapshot for a test configuration, while Fig. 9b shows the binary data of the same logic in the memory. The color-coded memory-dump of the control-logic perfectly correlates to the engineering software snapshots. Our analysis code's output for the same logic program is presented in Fig. 9c.

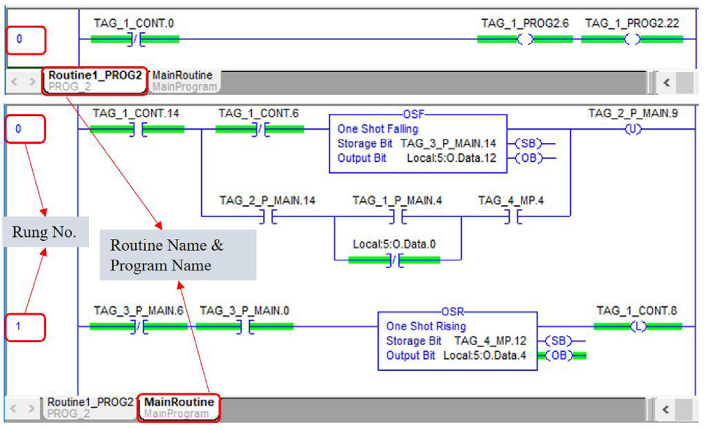
Decoding of Single-dword Instruction. Our experiment shows that most of the frequently used boolean instructions (such as Examine-if-closed XIC, Examine-if-open XIO etc) use single dword to jointly represent the instruction and the operand (say it as I&T dword). Algorithm 1 presents the decoding process for I&T dword. Consider the first I&T dword "4608016D″ in the "mainRoutine" in Fig. 9. Line 3 outputs 40 which is the instruction code for "XIC". The address "0008016C″ represents the tag "TAG_1_CONT". In line 7, the pin number is calculated as 14 implying the "tag_string" value to be "TAG_1_CONT.14". The joint I&T dword is decoded as XIC—TAG_1_CONT.14 as shown in the analysis code's output, and confirmed through the engineering software snapshot in Fig. 9a.

We have successfully decoded all instructions defined under "Bit" category in the engineering software. Some of these instructions have variable length that may extend up to 12 dwords. The maximum pins/bits of a tag supported in our code is 32.

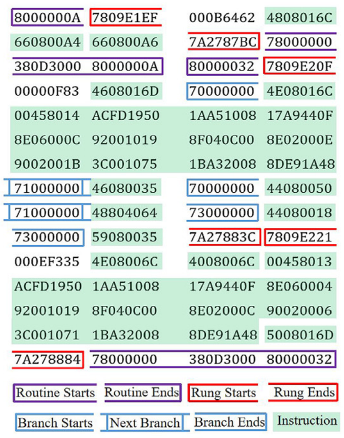
Algorithm 1. Decode single dword boolean instruction

Algorithm 1 Decode single dword boolean instruction

```
1: procedure INST_01_DECODE($I\&Tdword, core\_DT$)
2:     $[B_3 B_2 B_1 B_0] \leftarrow I\&Tdword$
3:     $inst\_code\ = B_3 / 8$
4:     $inst\_str = Inst\_Dict.get(inst\_code)$
5:     $bit\_no\ = B_3 \% 8$
6:     $byte\_no\ = B_0 \% 4$
7:     $tag\_pin\_no\ = byte\_no * 8 + bit\_no * 4$
8:     if ( then$B_2$ == 08)
9:         $tag\_addr = [00\ 08] + str(B_{1:0})$
10:    else
11:        $tag\_addr = [0C\ 00] + str(B_{1:0})$
12:    end if
13:    for x in core_DT: do
14:        if 0$\geq$ (x.addr - tag_addr) $\geq$ byte_no: then
15:            $tag\_str = x.tag\_name + "." + tag\_pin_no$
16:        end if
17:    end for
18:    return $(inst\_str +' -' + tag\_str)$
19: end procedure
```



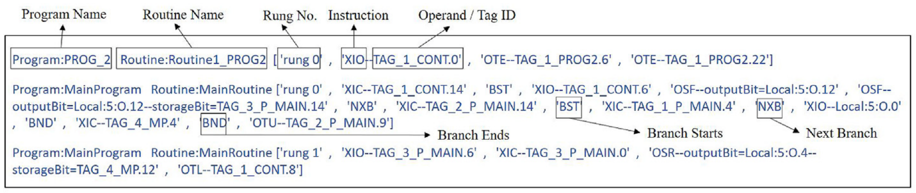
(a) Control-logic snapshot from RSLogix 5000



```
8000000A  7809E1EF    000B6462  4808016C
660800A4  660800A6    7A2787BC  78000000
380D3000  8000000A    80000032  7809E20F
00000F83  4608016D    70000000  4E08016C
00458014  ACFD1950    1AA51008  17A9440F
8E06000C  92001019    8F040C00  8E02000E
9002001B  3C001075    1BA32008  8DE91A48
71000000  46080035    70000000  44080050
71000000  48804064    73000000  44080018
73000000  59080035    7A27883C  7809E221
000EF335  4E08006C    4008006C  00458013
ACFD1950  1AA51008    17A9440F  8E060004
92001019  8F040C00    8E02000C  90020006
3C001071  1BA32008    8DE91A48  5008016D
7A278884  78000000    380D3000  80000032
```

Routine Starts | Routine Ends | Rung Starts | Rung Ends
Branch Starts | Next Branch | Branch Ends | Instruction

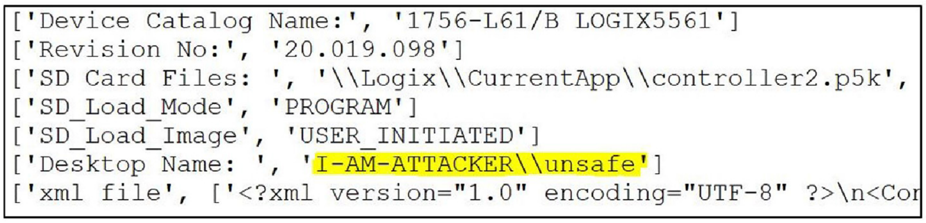
(b) Control-logic in the memory-dump



```
Program:PROG_2 Routine:Routine1_PROG2 ['rung 0' , 'XIO--TAG_1_CONT.0' , 'OTE--TAG_1_PROG2.6' , 'OTE--TAG_1_PROG2.22']
Program:MainProgram  Routine:MainRoutine ['rung 0' , 'XIC--TAG_1_CONT.14' , 'BST' , 'XIO--TAG_1_CONT.6' , 'OSF--outputBit=Local:5:O.12' , 'OSF--
outputBit=Local:5:O.12--storageBit=TAG_3_P_MAIN.14' , 'NXB' , 'XIC--TAG_2_P_MAIN.14' , 'BST' , 'XIC--TAG_1_P_MAIN.4' , 'NXB' , 'XIO--Local:5:O.0'
, 'BND' , 'XIC--TAG_4_MP.4' , 'BND' , 'OTU--TAG_2_P_MAIN.9']
Program:MainProgram  Routine:MainRoutine ['rung 1' , 'XIO--TAG_3_P_MAIN.6' , 'XIC--TAG_3_P_MAIN.0' , 'OSR--outputBit=Local:5:O.4--
storageBit=TAG_4_MP.12' , 'OTL--TAG_1_CONT.8']
```

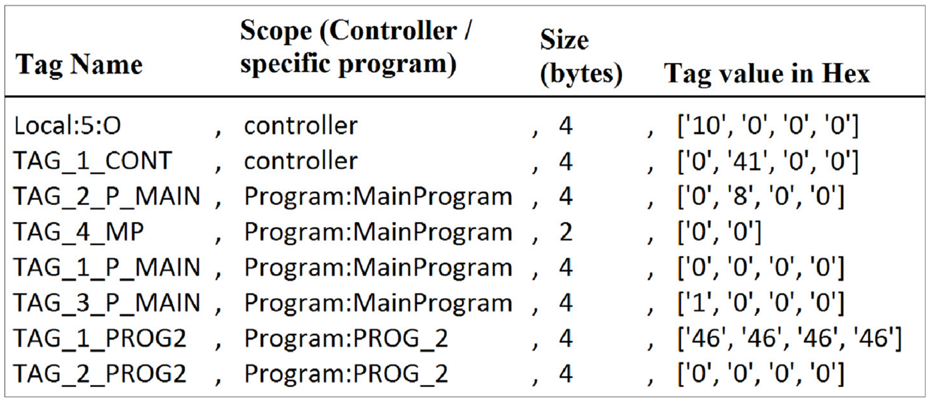
(c) A snapshot presenting recovered control-logic recovered from the memory matches the RSLogix 5000 view

Fig. 9. Programmed control-logic with corresponding memory-dump and analysis tool's output.



```
['Device Catalog Name:', '1756-L61/B LOGIX5561']
['Revision No:', '20.019.098']
['SD Card Files: ', '\\Logix\\CurrentApp\\controller2.p5k',
['SD_Load_Mode', 'PROGRAM']
['SD_Load_Image', 'USER_INITIATED']
['Desktop Name: ', 'I-AM-ATTACKER\\unsafe']
['xml file', ['<?xml version="1.0" encoding="UTF-8" ?>\n<Cor
```

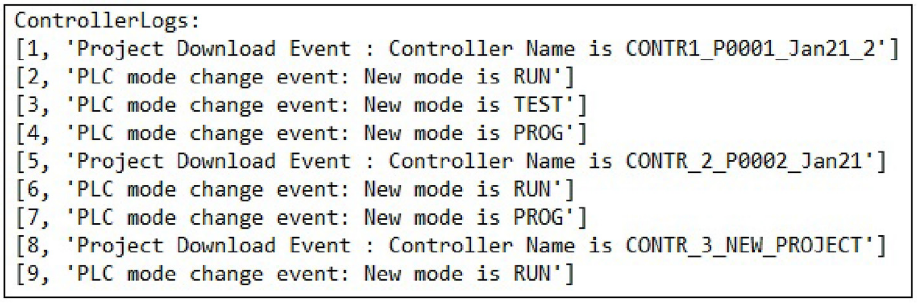
Fig. 10. Analysis tool shows the attacker's machine and username.



Tag Name	Scope (Controller / specific program)	Size (bytes)	Tag value in Hex
Local:5:O ,	controller ,	4 ,	['10', '0', '0', '0']
TAG_1_CONT ,	controller ,	4 ,	['0', '41', '0', '0']
TAG_2_P_MAIN ,	Program:MainProgram ,	4 ,	['0', '8', '0', '0']
TAG_4_MP ,	Program:MainProgram ,	2 ,	['0', '0']
TAG_1_P_MAIN ,	Program:MainProgram ,	4 ,	['0', '0', '0', '0']
TAG_3_P_MAIN ,	Program:MainProgram ,	4 ,	['1', '0', '0', '0']
TAG_1_PROG2 ,	Program:PROG_2 ,	4 ,	['46', '46', '46', '46']
TAG_2_PROG2 ,	Program:PROG_2 ,	4 ,	['0', '0', '0', '0']

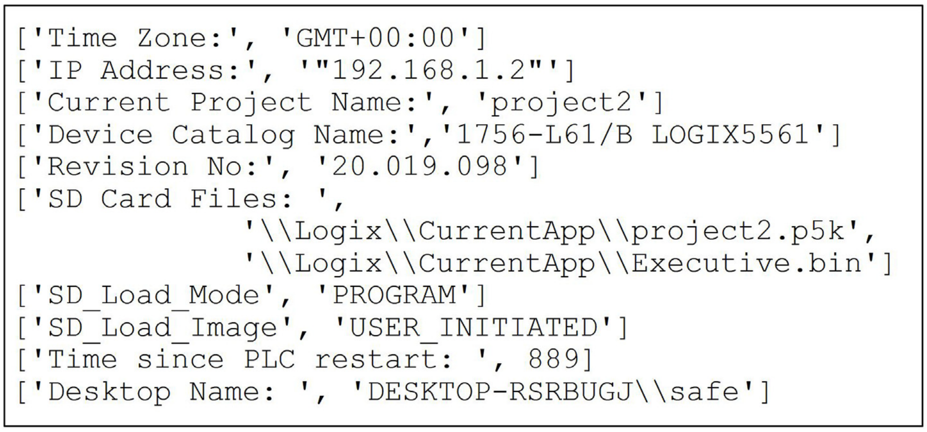
Fig. 11. Physical and Logical IO tags values.

IO Data Values. A PLC communicates with the physical world through IO modules. The state of IO pins in the memory-dump is an important forensic information, as an attacker can modify them and manipulate a physical process. In addition to the physical IO tags, a user can define logical tags in the control-logic for the program's flow-control. Both, physical and logical tags can be assigned to the instructions in the control-logic. Identification of the tags' location helps in the decoding and rules-generation process. Fig. 11 presents the physical and logical tags values extracted from the memory-dump. For instance, TAG_1_CONT with second



```
ControllerLogs:
[1, 'Project Download Event : Controller Name is CONTR1_P0001_Jan21_2']
[2, 'PLC mode change event: New mode is RUN']
[3, 'PLC mode change event: New mode is TEST']
[4, 'PLC mode change event: New mode is PROG']
[5, 'Project Download Event : Controller Name is CONTR_2_P0002_Jan21']
[6, 'PLC mode change event: New mode is RUN']
[7, 'PLC mode change event: New mode is PROG']
[8, 'Project Download Event : Controller Name is CONTR_3_NEW_PROJECT']
[9, 'PLC mode change event: New mode is RUN']
```

Fig. 12. Controller logs extracted by the analysis tool.



```
['Time Zone:', 'GMT+00:00']
['IP Address:', '"192.168.1.2"']
['Current Project Name:', 'project2']
['Device Catalog Name:','1756-L61/B LOGIX5561']
['Revision No:', '20.019.098']
['SD Card Files: ',
             '\\Logix\\CurrentApp\\project2.p5k',
             '\\Logix\\CurrentApp\\Executive.bin']
['SD_Load_Mode', 'PROGRAM']
['SD_Load_Image', 'USER_INITIATED']
['Time since PLC restart: ', 889]
['Desktop Name: ', 'DESKTOP-RSRBUGJ\\safe']
```

Fig. 13. PLC configuration related logs extracted from the memory.

byte as 41 indicates that pin 8 and pin 14 are HIGH. The observation is confirmed in the engineering snapshot in Fig. 9.

Configuration Parameters. Engineering software also controls configuration parameters such as PLC backup settings, filenames, time-zone information, controller name, etc. For configuration data extraction, unique names of the project and the controller are used to track through the strings search. Fig. 13 presents configuration data extracted from the memory-dump. One of the extracted fields shows information about the last computer's and user's name that connected to the PLC through the engineering software. The information reconciles with the licensing application's snapshot in Fig. 5d. Another interesting data is the PLC up-time information. PLC generates and maintains numerous counters. We found a 6 bytes increasing counter that only resets after PLC restart, and can be used as the up-time estimate to find a concealed reboot event.

Logs Recovery. As per the vendor, there are no logs available for this controller, except for the diagnostic fault-dump generated if the PLC goes into a non-recoverable state. During our analysis, we find forensically important logs for two critical events. First event is the downloading of a new project, and the second one is the change of PLC's mode of operation (RUN/PROGRAM/TEST). The logs are available for all the occurrences of above events after the last reboot. Although we could not find the log's creation time, the information is still helpful for the forensic investigation. Fig. 12 shows the logs generated after manually triggering few change events.

5.1. Memory analysis of the suspect PLC

Once the analysis profile phase is completed, the software can be readily used to extract the running firmware, the control-logic, IO data pins state, configuration details and few critical logs. The artifacts can be compared with the reference data to identify the anomalies. For example, the controller logs presented in Fig. 12 show two instances of PLC mode changes that may not reconcile with user activity logs. The tool is also able to find out the machine and user ID as presented in Fig. 10, which is different from the expected output seen in the RSLogix 5000 view presented in Fig. 5d.

6. Discussion and future work

Although reverse-engineering the firmware and control application binary is a systematic approach to analyze the memory dump, it is a tedious, time-consuming and superfluous task for forensic analysis. PLC market has a variety of hardware architecture, proprietary firmware and control applications, making the reverse-engineering of the firmware and control application binaries difficult. In our differential analysis based approach, we use the application layer interaction to extract the control-logic, IO data states and other artifacts through a much simpler and potentially replicable methodology. One overhead of our approach is the requirement of a PLC for the profile generation exercise. Organizations usually have more than one PLC of the same model, and it is feasible to spare one.

The expansion of our work includes studying more PLCs to generate their memory analysis tool, and ultimately formalizing a generic framework for PLC memory analysis.

7. Conclusion

The study presents the memory forensic analysis of Allen-Bradley PLC 1756-A10 (1756-L61 controller). Dealing with the proprietary hardware, firmware, and control software, the methodology outlines the important steps involved in PLC's memory forensic profile creation. The profile is developed through a closed-loop analysis using a planned memory loading profile set. During each iteration, the resultant memory dump is analyzed through a series of rules based on a combination of carving, list-walking, and heuristics. We identify more than twenty relevant data structures including connecting graphs and linked lists, and successfully recover the firmware, the control-logic, the physical process state, and other forensically important information in the memory dumps. Our solution can be used to instantly acquire these artifacts from any unseen memory dump of the mentioned PLC. As our approach targets the functional layer interaction between the PLC and the engineering software, it can potentially apply on a variety of PLCs with proprietary firmware. This is the first effort to analyze a PLC's complete memory for extracting digital forensic artifacts, and we consider it as a step toward a unified memory analysis framework for PLCs.

Acknowledgement

This work was supported, in part, by the Virginia Commonwealth Cyber Initiative, an investment in the advancement of cyber R&D, innovation, and workforce development. For more information, visit www.cyberinitiative.org.

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., Obermeier, S., Sudhakaran, S., Roussev, V., 2017. Programmable logic controller forensics. IEEE Secur. Privacy 15 (6), 18–24.
Ayub, A., Yoo, H., Ahmed, I., 2021. Empirical study of plc authentication protocols in industrial control systems. In: 15th IEEE Workshop on Offensive Technologies (WOOT). IEEE.
Basnight, Z.H., 2013. Firmware Counterfeiting and Modification Attacks on Programmable Logic Controllers, Tech. Rep. AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH GRADUATE SCHOOL OF.
Basnight, Z., Butts, J., Lopez, J., Dube, T., 2013. Firmware modification attacks on programmable logic controllers. Int. J. Critical Infrastruct. Protect. 6 (2), 76–84. https://doi.org/10.1016/j.ijcip.2013.04.004. http://www.sciencedirect.com/science/article/pii/S1874548213000231.
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.
Findrik, M., Smith, P., Quill, K., Kieran, M., 2018. Plcblockmon: data logging and extraction on plcs for cyber intrusion detection. In: 5th International Symposium for ICS & SCADA Cyber Security Research 2018, vol. 5, pp. 102–111.
Garcia, L., Brasser, F., Cintuglu, M., Sadeghi, A.-R., Mohammed, O., Zonouz, S., 2017. Hey, My Malware Knows Physics! Attacking Plcs with Physical Model Aware Rootkit. https://doi.org/10.14722/ndss.2017.23313.
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., Van Houten, J., 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: 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. https://doi.org/10.1016/j.fsidi.2020.301013. http://www.sciencedirect.com/science/article/pii/S2666281720302626.
Qasim, S., Ayub, A., Johnson, J., Ahmed, I., 2021. Attacking iec-61131 logic engine in programmable logic controllers. In: Staggs, J., Shenoi, S. (Eds.), Critical Infrastructure Protection XV. Springer International Publishing, Cham.
Rais, M.H., Li, Y., Ahmed, I., 2021a. Spatiotemporal g-code modeling for secure fdm-based 3d printing. In: Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems, ICCPS '21. Association for Computing Machinery, New York, NY, USA, pp. 177–186. https://doi.org/10.1145/3450267.3450545. https://doi.org/10.1145/3450267.3450545.
Rais, M.H., Li, Y., Ahmed, I., 2021b. Dynamic-thermal and localized filament-kinetic attacks on fused filament fabrication based 3d printing process. Add. Manufact. 46, 102200. https://doi.org/10.1016/j.addma.2021.102200. https://www.sciencedirect.com/science/article/pii/S2214860421003614.
Rais, M.H., Awad, R.A., Lopez, J., Ahmed, I., 2021c. Jtag-based plc memory acquisition framework for industrial control systems. Forensic Sci. Int.: Digit. Invest. 37, 301196. https://doi.org/10.1016/j.fsidi.2021.301196. https://www.sciencedirect.com/science/article/pii/S2666281721001049.
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.
Statista. Global plc market share as of 2017, by manufacturer. URL. https://www.statista.com/statistics/897201/global-plc-market-share-by-manufacturer/.
Wu, T., Nurse, J.R., 2015. Exploring the use of plc debugging tools for digital forensic investigations on scada systems. J. Digital Foren. Secur. Law 10 (4), 7.
Yoo, H., Ahmed, I., 2019. 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.
Zaddach, J., Bruno, L., Francillon, A., Balzarotti, D., et al., 2014. Avatar: a framework to support dynamic security analysis of embedded systems' firmwares. NDSS 23, 1–16.
Zhu, R., Zhang, B., Mao, J., Zhang, Q., an Tan, Y., 2017. A methodology for determining the image base of arm-based industrial control system firmware. Int. J. Critical Infrastruct. Protect. 16, 26–35. https://doi.org/10.1016/j.ijcip.2016.12.002. https://www.sciencedirect.com/science/article/pii/S1874548216300014.