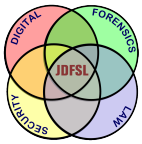

THE JOURNAL OF
DIGITAL FORENSICS, SECURITY AND LAW

Journal of Digital Forensics, Security and Law

Volume 10 | Number 4 Article 7

2015

Exploring The Use Of PLC Debugging Tools For Digital Forensic Investigations On SCADA Systems

Tina Wu
University of Oxford

Jason R.C. Nurse
University of Oxford

Recommended Citation
Wu, Tina and Nurse, Jason R.C. (2015) "Exploring The Use Of PLC Debugging Tools For Digital Forensic Investigations On SCADA Systems," *Journal of Digital Forensics, Security and Law*: Vol. 10 : No. 4 , Article 7.
DOI: https://doi.org/10.15394/jdfsl.2015.1213
Available at: https://commons.erau.edu/jdfsl/vol10/iss4/7

EXPLORING THE USE OF PLC DEBUGGING TOOLS FOR DIGITAL FORENSIC INVESTIGATIONS ON SCADA SYSTEMS

Tina Wu and Jason R.C. Nurse
Cyber Security Centre
Department of Computer Science
University of Oxford
Oxford, United Kingdom
{*tina.wu,jason.nurse*}@cs.ox.ac.uk

ABSTRACT

The Stuxnet malware attack has provided strong evidence for the development of a forensic capability to aid in thorough post-incident investigations. Current live forensic tools are typically used to acquire and examine memory from computers running either Windows or Unix. This makes them incompatible with embedded devices found on SCADA systems that have their own bespoke operating system. Currently, only a limited number of forensics tools have been developed for SCADA systems, with no development of tools to acquire the program code from PLCs. In this paper, we explore this problem with two main hypotheses in mind. Our first hypothesis was that the program code is an important forensic artefact that can be used to determine an attacker's intentions. Our second hypothesis was that PLC debugging tools can be used for forensics to facilitate the acquisition and analysis of the program code from PLCs. With direct access to the memory addresses of the PLC, PLC debugging tools have promising functionalities as a forensic tool, such as the "Snapshot" function that allows users to directly take values from the memory addresses of the PLC, without vendor specific software. As a case example we will focus on PLC Logger as a forensic tool to acquire and analyse the program code on a PLC. Using these two hypotheses we developed two experiments. The results from Experiment 1 provided evidence to indicate that it is possible to acquire the program code using PLC Logger and to identify the attacker's intention, therefore our hypothesis was accepted. In Experiment 2, we used an existing Computer Forensics Tool Testing (CFTT) framework by NIST to test PLC Logger's suitability as a forensic tool to analyse and acquire the program code. Based on the experiment's results, this hypothesis was rejected as PLC Logger had failed half of the tests. This suggests that PLC Logger in its current state has limited suitability as a forensic tool, unless the shortcomings are addressed.

Keywords: PLC debugging, program code, SCADA, digital forensics, NIST, PLCs, attackers

1. INTRODUCTION

Supervisory Control and Data Acquisition (SCADA) systems are used to gather, monitor and analyse real-time data in an automated manner. They provide the underlying architecture for many critical infrastructures including power plants, water, oil and gas distributions systems. Initially SCADA systems were isolated allowing a degree of protection. The adoption of ethernet TCP/IP and wireless technologies (such as IEEE 802.x) in these systems, however has significantly reduced this isolation. The connection to the enterprise network leaves SCADA systems, that were previously air-gapped, vulnerable to an ever-increasing range of external attacks (Zhu, Joseph, & Sastry, 2011).

In recent years there have been several, well-documented attacks including a sophisticated malware designed specifically to attack SCADA systems, known as the Stuxnet virus. This virus exploits multiple vulnerabilities in Windows to target the software used to program Programmable Logic Controllers (PLCs) central to the operation of SCADA systems. One of the key attack mechanisms of Stuxnet was the reprogramming of the program code on the PLCs (Falliere, Murchu, & Chien, 2011). To respond to attacks on SCADA systems, forensics play a critical role. A forensic investigation can be used to identify who carried out the attack, to define their intentions and also to understand their methodology. Not only is it important to use the information discovered during the investigation in order to attribute the attack, it is also essential that it is used to improve security measures on the SCADA system and to share the threat intelligence information with other SCADA sites. In the past, digital forensics focused on normal IT setups with consumer electronics and enterprise systems involving computers with hard drives, the majority using the Windows Operating System (OS). The attack surface of a control-system environment however, is very different to a typical IT environment. Some of the main differences include the fact that there are many embedded devices using bespoke protocols and OSs, and many systems still using outdated and unpatched OSs (Zhu et al., 2011).

Most related research has focused on the development of live forensic frameworks using agents to collect forensic artefacts from SCADA systems such as (Ahmed, Obermeier, Naedele, & Richard, 2012) (Taveras, 2013) (Kilpatrick, Gonzalez, Chandia, Papa, & Shenoi, 2006). Many of these approaches have focused on acquiring evidence from computer systems. However, the focus should arguably shift to acquiring evidence from PLCs as they are a crucial components in SCADA systems (as seen in the Stuxnet case where a critical PLC component was attacked). Moreover the majority of current research in SCADA forensics is theoretical with many authors such as (Taveras, 2013) not having tested the practical use of their proposed approach.

The novelty of our research is exploring the utility of PLC debugging tools in supporting forensic investigations. Specifically this will be used to acquire the memory addresses from a Siemens S7 PLC and explore the suitability of PLC Logger for forensic investigations on SCADA systems. PLC debugging tools have promising functionalities such as communication with the PLC without using vendor specific software and direct access to the memory of the PLC. However it is unclear whether such tools are useful for forensic investigations.

Our research question is; can PLC debugging tools be used to support digital forensic investigations on SCADA systems? From this research question we developed two hy-

potheses, these are:

- Program code is an important forensic artefact that can be used to determine an attacker's intentions.
- PLC debugging tools can be used as a forensic tool to acquire and analyse the program code from a PLC.

The paper is structured as follows: In Section 2, we discuss related work on SCADA forensics, live forensics data and the installation of agents on a SCADA network and related data acquisition methods. Section 3 discusses the forensic artefacts that can be acquired from PLCs but also artefacts specifically from Siemens S7 PLCs. In Section 4, we present an overview of NIST's Computer Forensics Tool Testing (CFTT) framework and the purpose of its use. In Section 5, we introduce our experimental setup, the PLC Logger tool and the program code. Next, Section 6 presents the experiment results and a critical reflection on the use of the PLC program code to investigate the attacker's intentions, and to test PLC Logger for its suitability as a forensic tool for PLCs using NIST's CFTT framework. Finally Section 7 concludes this paper and presents future work.

2. RELATED RESEARCH

There is a large amount of existing research in the area of forensics on SCADA systems. In particular, several authors have proposed the installation of agents on SCADA networks to conduct live forensic acquisition (Ahmed et al., 2012),(Kilpatrick et al., 2006). Live data acquisition is the extraction of data when the device is still running; in digital forensics this is normally used to recover RAM in the memory of a computer (Jones, 2007). There are two methods of live data acquisitions. The first method requires an agent to be pre-installed on the host computer. Data can then be acquired through the network to extract artefacts in a large network such as an enterprise environment. These agents include open source tools such as Googles GRR Rapid Response[1] and proprietor software such as EnCase Enterprise[2]. The second method extracts data using an agent customised depending on the forensic artefacts required. These include tools such as Mandiant Redline[3] often used in Incident Response (IR) to find signs of malicious activity in the device's memory.

(Kilpatrick et al., 2006) propose implementing agents across the control network, the agents then send network packets back to a data warehouse on an isolated network to prevent external access. This technique relies on trust that the agents have not been compromised, or that the network packets sent back to the data warehouse have not been modified during transit. (Taveras, 2013) proposes the use of a finite state automaton as an agent that would monitor SCADA events in real-time. These events are then compared against a set of rules to determine whether any changes have been made to the state. If changes have been identified the agent then switches to forensic mode to log the information for use in a forensic investigation. However, further research would be required to test this on a testbed simulation to assess its effectiveness on a real SCADA system.

(Radvanovsky & Brodsky, 2013) have identified that obtaining the hexadecimal dump from the memory of the PLC can be useful in a forensic investigation. The file system can be checked for known malware signatures and compared against expected file signatures to determine changes to the

[1] https://github.com/google/grr

[2] https://www.guidancesoftware.com/products/Pages/encase-enterprise/overview.aspx

[3] https://www.mandiant.com/resources/download/redline

file system. The authors have not suggested a practical solution to extract the hexadecimal dump from the PLC, currently there are no methods to achieve this.

(Van der Knijff, 2014) examined an attack scenario and discussed the forensic methods and tools that could possibly be used for a forensic investigation on the SCADA system. The author stated potential artefacts stored in the RAM of the PLC would be lost if rebooted. They have suggested in order to preserve the potential evidence in the RAM it would need to be switched to programming mode. This is an hypothesis they have concluded and have not carried out a practical element to prove their hypothesis is correct. In order to switch the PLC into programming mode specific software would have to be obtained from the vendor. In circumstances where this is unavailable, the author has suggested the use of debugging tools connecting through the Joint Action Test Group (JTAG) port or physical removal of the chips. However it is rare that a SCADA system will have a back-up system that allows a PLC to be taken offline. This could be an issue with the SCADA system owners who would be unlikely to allow this.

A key limitation with the majority of existing work such as (Ahmed et al., 2012)(Patzlaff, 2013) is they have not conducted a practical evaluation of their forensic framework. In order to establish if their implementation is capable of extracting forensic artefacts from PLCs.

3. FORENSIC ARTEFACTS FROM PLCS

3.1 Existing Research

Here we explore the forensics artefacts that can be retrieved from PLCs. In traditional computer forensics, forensic artefacts are normally defined as artefacts that have been exposed by a forensic investigator to reveal the truth about an event that has been left on the system (Clarke, Tryfonas, & Dodge, 2012). On a PLC there are two sources of evidence that can be obtained i.e; the device and network traffic. The focus of our research will be on forensic artefacts retrievable from devices. Network traffic on its own would not be sufficient as it cannot provide hardware level data.

(ENISA, 2013) has identified some types of data that can be extracted from field devices such as date and time, current active processes and logs. They have identified that in order to be able to acquire data from field devices they need to work with control system engineers and vendors to support the investigation.

In terms of a PLC, the forensic process and tools to be used would be highly dependent on the make and model of the PLC. It would therefore be useful to gather information such as vendor of the PLC, firmware version, date and time on the PLC, IP Address, serial number of the PLC, model name, and the type of Computer Processing Unit (CPU).

To the best of our knowledge, no research has currently identified whether there is any useful evidence on PLCs that may be used to support a forensic investigation of a SCADA system. The attack using Stuxnet involved the reprogramming of the PLC program code and the installation of malware. Therefore we hypothesise that there are potential artefacts in the reprogrammed code, that can be used the to establish the attacker's intentions (Falliere et al., 2011). There are other malicious attacks on PLCs such as firmware counterfeiting and modifications (Basnight, 2013) and uploading of malware (McLaughlin, 2011). This paper focuses on a modification attack on the PLC program code, as to our knowledge there has

been limited research in this area (considering this occurred in the Stuxnet attack).

There is a feasible scenario where an attacker could maliciously alter the PLC program code to perform the initial attack. When this is complete the attacker could access the PLC again to revert to the original program code in the PLC or to enter any random program code they choose. In this situation we would not be able to rely on the program code from the PLC without other sources of evidence from the SCADA system. For example captured network traffic and event logs.

Our approach is general, but for this specific test case in the paper, we are using the Siemens S7-1200 PLC. This PLC was chosen for our experiments, as it is a common brand of PLC with wide support and developers regularly creating libraries to support communication with Siemens S7 PLCs. The next section explores the structure of the program code on Siemens S7 PLCs.

3.2 Siemens S7 PLCs

The Siemens S7 protocol is a proprietary protocol used to communicate with PLCs of the Siemens family. The main purpose of the protocol is to exchange data among PLCs and access data from SCADA systems for diagnostics purposes. The Siemens S7 TCP/IP protocol communicate using port 102 and can be programmed using the TIA Portal Step 7 software. According to Siemens, a typical S7 PLC stores program code and configuration data within the Random Access Memory (RAM). The structure of RAM changes frequently as it is the volatile area of the memory, (i.e when power is removed the data stored is lost) (Vidas, 2007).

Siemens structure their program code by storing them in blocks, limited to 1024. The program code uses memory addresses to access information in the memory location. The following are blocks of code used to structure the program code:

- Organisation Block (OB): provides structure to the program and allows the interface between the OS and the user program; it is the entry point of a program;
- Function Code (FC): performs a specific operation on a set of input values. The FC stores the results of this operation in the memory location;
- Data Block (DB): contains program specific data such as numbers and structures;
- Function Block (FB): use an instance DB for its parameters and static data. FB has variable memory located in a DB; and
- System Data Block (SDB): contains information on how the PLC is configured. They are created depending on the number and type of hardware modules that are connected to the PLC (Siemens, 2015).

Forensic artefacts within the Siemens S7 PLCs are the blocks of code (program blocks) such as OB and FBs. Other important artefacts are within the blocks of codes containing memory addresses as shown in Figure 1. The blocks of code and memory addresses can be used to reconstruct the program code back to its raw format, in order to identify modifications made by attacker to determine their intentions. For example in the scenario of a traffic control system, the attacker may change the program code intending to modify the traffic lights sequence to cause accident or traffic chaos.

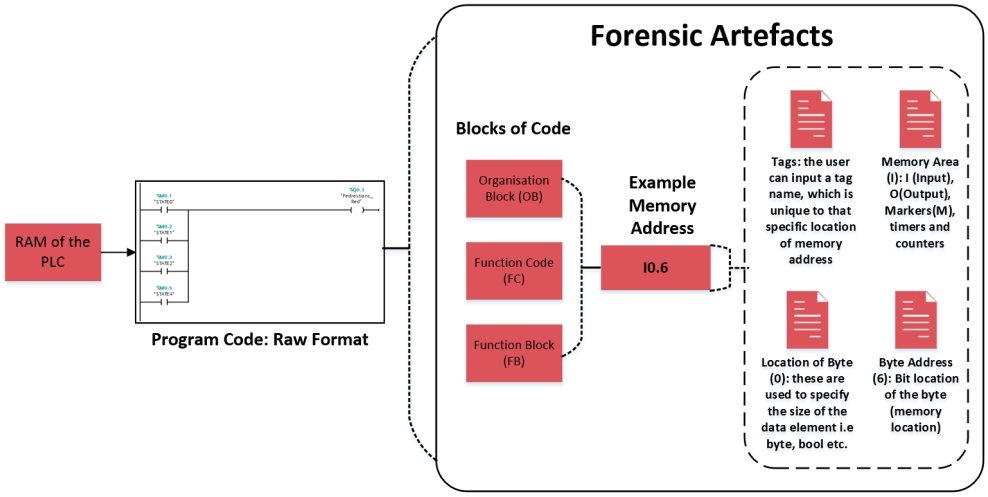

Figure 1. Forensic artefacts of a program code on a Siemens S7 PLCs

4. NIST'S COMPUTER FORENSICS TOOL TESTING (CFTT) FRAMEWORK

The CFTT field is a well researched area with many frameworks developed by the National Institute of Standards and Technology (NIST) (NIST, 2007) to test forensic tools. The CFTT framework has been further refined for specific areas of forensics by (Rick Ayers, 2013),(Flandrin, Buchanan, Macfarlane, Ramsay, & Smales, 2014). Since the CFTT framework involves testing of various computer forensic toolkits the same principles are also applicable to other forensic tools, for example forensic tools for PLCs. The purpose of testing forensic tools is to measure validity of the results, to understand the tool's capabilities and to allow developers to improve the tool, including its functionality and utility.

Currently there are a limited number of fully developed forensic tools for SCADA and crucially no specific framework to test a SCADA forensic tools. The aim of our experiment is to test PLC debuggers, in this case example PLC Logger for its suitability as a forensic acquisition and analysis tool for PLCs. It is not within our scope of this paper to develop a framework to test tools for SCADA systems. Instead we used existing research to test the suitability of PLC Logger. This enabled us to follow an already established standard that is a recognised industry framework.

5. EXPERIMENTAL SETUP

The purpose of the experiments is to test PLC debugging tools, in this case example we test PLC Logger to explore its suitability as an acquisition and analysis tool to support forensic investigations. The following section provides an explanation of PLC Logger and the program code showing the scenario of a traffic control system required for the experiments. We constructed two experiments based on the two hypotheses, these are:

1. Experiment 1: investigate if the program code once acquired can be used to determine an attacker's intentions.
2. Experiment 2: examined whether PLC Logger is suitable as a forensic acquisi-

tion and analysis tool using an existing NIST CFTT framework.

5.1 Tool: PLC Logger

Based on documentation retrieved about PLC Logger[4], it is used to store and analyse data from various vendors; this includes the Siemens S7, CoDeSys and Modbus TCP protocol-based PLCs. The configuration file stores data about a PLC that is acquired and can be saved as a PLCLogFile (.PLF). The protocol configuration can be stored as a PLCConfigurationFile (.PCF). Finally, any gathered data such as output values from the memory addresses, can then be visualised in a graph. PLC Logger is normally used to record data on PLCs for diagnostic purposes such as tracing faults in machinery and to improve the efficiency of the system. PLC Logger was chosen because it is capable of accessing the memory areas of the PLC while also recording timestamped data. Moreover it has the capability to record data from a number of PLCs simultaneously. As a forensic tool for a SCADA system, this recovering capability would be an important feature, as it is anticipated that there are a number of PLCs on the system.

5.2 The Program Code

For our experiment, program code was developed based on an attack scenario involving a traffic control system. We narrowed down our approach to an example of a single pedestrian crossing with Traffic Lights (TLs). Siemens TIA Portal Step 7 programming software was used to create the program code and upload onto the Siemens S7-1200 PLC. Table 1 shows a block of code from OB1 and contains the memory addresses of the program code. The following provides a further explanation of the program code shown in the table.

[4]http://sourceforge.net/projects/plclogger/

In the first instance, when the program code is uploaded onto the PLC it is in State 0, this means that the TLs are currently green (Q0.2) and the Pedestrian Lights (PLs) are red (Q0.3); see Figure 2. When the PL button (I0.6,I0.7) is pressed, the timer counts for two seconds and then goes to State 1 turning the TLs amber (Q0.1) and the PLs remain red (Q0.3). In State 2 both TLs and PLs are red (Q0.0) and in State 3 the TLs are red (Q0.0) and the PLs green (Q0.4). In State 4 the TLs turn amber (Q0.1) and the PL turn red (Q0.3) before reverting back to State 0.

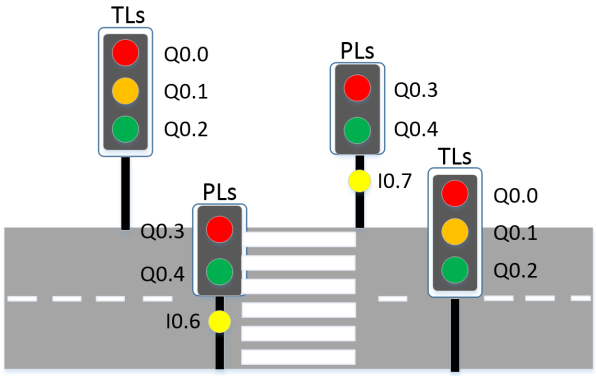

Figure 2. Program Code based on a Scenario of a Traffic Lights and Pedestrian Crossing

Using PLC Logger and the program code the following two experiments were conducted.

5.3 Experiment 1

Our first hypothesis is that by acquiring the program code, we will be able to gain valuable artefacts that could allow us to determine the attacker's intentions. To test this hypothesis we conducted the experiment using the following steps:

1. We uploaded the program code onto the Siemens S7-1200 PLC.
2. We took the memory addresses (shown in Table 1) from the program code and entered these into the PLC Logger configuration interface.

Table 1. Variables stored in the Memory addresses within OB1

Name	Tag Table	Data Type	Address
Start	Variables	Boolean	M0.0
State 0	Variables	Boolean	M0.1
State 1	Variables	Boolean	M0.2
State 2	Variables	Boolean	M0.3
State 3	Variables	Boolean	M0.4
State 4	Variables	Boolean	M0.5
Start P	Variables	Boolean	M0.6
Button	Variables	Boolean	M0.7

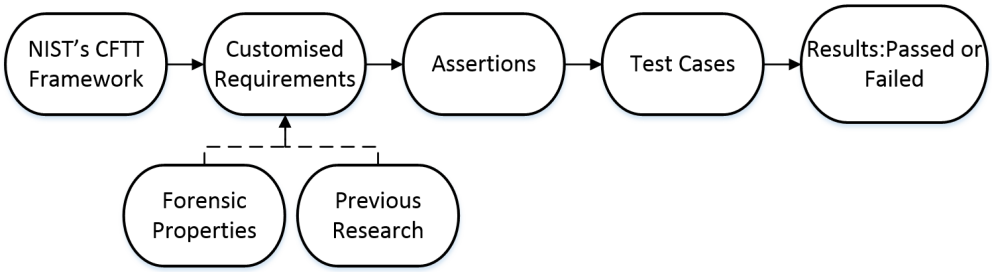

Figure 3. Block diagram of Experiment 2

3. We observed the visualisation graph in PLC Logger to determine the values that the memory addresses output. The only datatypes that were created in the program code are in boolean, meaning it can output only either a value of 1=true or 0=false.
4. The data observed was saved to a local file for later use.
5. Slight modifications were made to the program code to represent an attack scenario on the traffic control system where an attacker modified both PLs and TLs to green. The modified program code was uploaded onto the PLC.
6. We observed the visualisation graph in PLC Logger to determine the memory addresses that output either a value of 1 and 0. The data was saved as a local file to be used later.
7. The data between the two saved files were compared to determine the modifications to the memory addresses, to determine the attackers intentions.

5.4 Experiment 2

Our second hypothesis is PLC debugging tools can be used as a forensic tool to acquire and analyse the program code from a PLC, in this case example we use the tool PLC logger. Using NIST's existing CFTT framework we set about testing PLC Logger for its suitability as a forensic analysis and acquisition tool using the methodical approach shown in Figure 3. While NIST's CFTT framework was usable for a general computer forensic toolkit, it would need be customised for these experiments. In this section, we refer to PLC Logger as the "tool" to keep the framework generic. Following NIST's CFTT framework, a set of requirements were defined; these are requirements the tools must adhere to should they be used in a forensic context. Assertions were deduced from the requirements. Assertions are general statements or conditions that can be checked after

a test is executed. The assertions are then transformed into a set of test cases, these are a combination of test parameters required to assess each assertion.

5.4.1 Requirements

The following are requirements that set a base criteria the tool should achieve, these are gathered from a combination of previous research by (Patzlaff, 2013) who have a detailed list of requirements that a forensic tool for a PLC should meet. Additional requirements were developed based on forensic properties that a acquisition and analysis forensic tool should provide (Carrier, 2003). The requirements are defined as follows.

It is necessary that the tool is able to connect to the PLC while it is operational and through a network connection as this is the main approach to acquire the PLC's program code (REQ-01). In a real SCADA system there is no visible way to verify the connection with the PLC, therefore the tool should provide a method of verifying the connection (REQ-02). The tool should be able to acquire the program code from the memory of the PLC (REQ-03). Errors in connection or during acquisition should be reported to the user (REQ-04). The tool should notify the user when the acquisition has been successful (REQ-05). The tool should notify the user when the acquisition has been unsuccessful (REQ-06). The tool should write the acquired data to a file, saved in various file formats when acquisition of the PLC's program code is successful (REQ-07). The tool should not write any data to the memory of the PLC (REQ-08). The tool should acquire the data securely (REQ-09). The tool should automatically be able to acquire the memory addresses (REQ-10). In the next section we outline the assertions developed based on the above requirements.

5.4.2 Assertions (ASRT-#)

The assertions have been categorised into four main forensic properties these are; connectivity, accuracy, usability and verifiability. The assertions have been mapped with the requirements and are shown in Table 2.

5.4.2.1 Connectivity

ASRT-01: The connection between the tool and the PLC should be made possible through the network.

ASRT-02: If the connection between the tool and the PLC is successful, the tool should notify the user.

ASRT-03: If the connection between the tool and the PLC is unsuccessful the tool should notify the user.

ASRT-04: The tool should present acquired program code in a format useful to an digital forensic investigator.

ASRT-05: If the acquisition of the program code is successful or unsuccessful the tool should notify the user.

5.4.2.2 Accuracy

ASRT-06: The tool should not send any data to modify the PLC program code.

ASRT-07: The tool should accurately acquire the program code. For example, if the program code contains 8 DBs and 9 OB blocks, the tool should report this accurately.

5.4.2.3 Usability

ASRT-08: The tool should provide the ability to allow the user to start the acquisition process.

ASRT-09: The acquisition of the program code should be automatic after entering the memory addresses.

ASRT-10: The tool should record and save the data in a format that can be later used in analysis also storing timestamps.

ASRT-11: The tool should have the ability to acquire data while the device is still operational.

Table 2. Requirements for the PLC Debugger tool and associated assertions

	REQ-01	REQ-02	REQ-03	REQ-04	REQ-05	REQ-06	REQ-07	REQ-08	REQ-09	REQ-10
ASRT-01	•									
ASRT-02		•		•						
ASRT-03		•		•						
ASRT-04					•					
ASRT-05				•	•	•				
ASRT-06								•	•	
ASRT-07					•					
ASRT-08			•							
ASRT-09										•
ASRT-10							•			
ASRT-11	•									
ASRT-12							•			
ASRT-13									•	

ASRT-12: If acquisition of the program code is successful, the tool should write the acquired data to a local folder.

5.4.2.4 Verifiability

ASRT-13: The tool should acquire the program code over an encrypted network.

5.4.3 Test Cases (TSC-#)

This section contains a list of test cases. Each of the test cases consists of a list of steps to complete the test and the expected result required to pass the test. The assertions are mapped against the test cases and are shown in Table 3.

5.4.3.1 TSC-01

1. Connect the tool to the PLC via the network
2. Verify the connection status is correct via the tool interface

Expected result: The connection status is reported as correct.

5.4.3.2 TSC-02

1. Connect the tool to the PLC
2. Verify that there has been a successful connection by starting an acquisition

Expected result: The acquisition is reported as completed successfully or unsuccessfully.

5.4.3.3 TSC-03

1. Connect the tool to the PLC
2. Start the acquisition of the PLC memory
3. Examine the captured network traffic using Wireshark[5] (a network protocol analyser)

Expected result: During acquisition, data should not overwrite any program code on the PLC.

5.4.3.4 TSC-04

1. Connect an unsupported PLC model to the tool
2. Verify that the PLC is identified as unsupported

Expected result: The connection status is reporting as an unsupported PLC in the tool.

5.4.3.5 TSC-05

1. Start the acquisition of the PLC program code
2. Turn off the PLC

Expected result: The acquisition is aborted and the tool notifies the user of the disconnection.

5.4.3.6 TSC-06

- Start the acquisition of the PLC program code
- Wait until the acquisition is completed successfully

[5] https://www.wireshark.org/

- Export and save the acquired data to a file

Expected result: The tool saves the acquired data to a file.

5.4.3.7 TSC-07

- Connect a supported PLC to the tool
- Enter the memory addresses to acquire
- Use the tool to start the acquisition

Expected result: The acquisition is completed successfully.

5.5 Limitations

There are several limitations in our experiments. Firstly, PLC Logger reads values over a network connection which would increase traffic overhead on the network. Further research would be required to establish effects on industrial equipment.

Our experiments were limited to the use of a specific brand of Siemens PLC and one PLC debugging tool. However our experimental setup can be applied to other brands of PLCs and other PLC debugging tools.

We based our experiment on one scenario with the program code specifically developed for these experiments and was not collected from a real SCADA system. Therefore, additional testing would be required and program codes would need to be collected from real SCADA systems. We have developed our traffic light scenario for the experiments as close as possible to model a real-life traffic light crossing. A real-life traffic light scenario will be more complex when compared to our demonstration since ours is only a small component of the traffic light system. PLC programming concepts are common to all manufacturers and differ in program languages, I/O addressing, memory organisation and instruction sets mean that they are never perfectly interchangeable between different makers, even products from the same manufacturer may not be compatible.

6. RESULTS AND DISCUSSION

In this section, we present and discuss the results from our experiments. The results from testing PLC Logger are presented. Then we used NIST's CFTT framework to test PLC Logger to determine its suitability as a forensic acquisition and analysis tool for PLCs.

6.1 Experiment 1

In PLC Logger, the following are the memory addresses that have output a value of 1 (True): Pedestrian Red (Q0.3), Green (Q0.2), Start Light (M0.0), State0 (M0.1) and StartP (M0.6). This indicates the PLC is currently in State 0, with the TLs green and the PLs red. Slight modifications were made to the memory addresses in the program code, to cause both the TLs and PLs to turn green. PLC Logger was again used to acquire the memory addresses and found that the following had been modified; Pedestrian Red (Q0.3) outputs a value of 0 (False) and Pedestrian Green (Q0.4) of 1 (True). Thus, both TLs and PLs were showing green.

One of our research aims was to investigate whether the PLC program code can be of use in forensic investigations, for instance to determine an attacker's intentions. The results from the experiment indicate that PLC Logger can be used to communicate and acquire data from the memory addresses of the PLC without any problems. In this scenario, by acquiring the memory addresses from the PLC we were able to identify that the attacker had made slight modifications to the memory addresses, allowing us to determine their intention. In this situation it would have been their intention to cause both TLs and PLs to turn green, potentially to cause an accident, disrupt the flow of traffic flow or cause distress. The results from Experiment 1 show the hypothesis was accepted, as we were able to show that the

program code was useful for a forensic investigation by identifying the attacker's intention. In the next section a methodological approach is used to further test PLC Logger.

6.2 Experiment 2

We used an existing CFTT framework to explore the suitability as a forensic acquisition and analysis tool for PLCs. First, developing a set of requirements, assertions and then test cases. Each test case consists of the steps required to complete the test and the expected results needed to pass the test. The reason for not using a numbering grading scale as the one seen in (Anobah, Saleem, & Popov, 2014) is because it's main use there was for the comparison of forensic tools. Whereas our intention was to explore whether PLC Logger was suitable forensic tool for PLCs. An overview of the results are shown in Table 3.

The complete results can be found in Appendix A; in this section only the main results are discussed.

The test that failed was TSC-01 with the assertion ASRT-02 referring to the notification of successful connection between PLC Logger and the PLC if successful. The tool failed this assertion as it does not notify the user when the PLC is online or when an unsupported PLC is connected. We found that when the wrong IP address of the PLC was entered, the tool's interface would display a red exclamation indicator. But this was the also the case when a user had entered an incorrect memory address. This meant there was no clear indication which was incorrect as it could either be the memory address, IP address or both.

TSC-04 with the assertion ASRT-03 refers to PLC Logger notifying the user when the PLC is disconnected from the network. The tool failed the test as it does not notify the user when the PLC is disconnected. It was identified the graphical interface in PLC Logger continues to output the value 0, which could be an indication that the PLC is disconnected.

TSC-02 with the assertion ASRT-05, referring to the PLC Logger notifying the user if the acquisition of the PLC is successful or unsuccessful. This assertion failed the test, as there are no notifications in the interface of PLC Logger until the user has input the correct IP address of the PLC and memory addresses. If these are not input correctly the interface in PLC Logger remains at the default setting.

Assertions ASRT-01 to 05 showed it had failed on connectivity issues. However these can easily be overcome by implementing simple features such as notifications to the user when a PLC has been disconnected from PLC Logger.

In TSC-03 with the assertion ASRT-06, refers to PLC Logger making modifications to the program code during acquisition. This assertion was to test for accuracy and failed, after examination of the captured network traffic, it was not possible to determine whether any modifications were made. With any live forensic tool the investigator would want to prevent modifications to the original data as much as possible to preserve the integrity of the evidence. Therefore further research would be required on PLC Logger to determine the type of data sent and received during the acquisition process on the PLC (Jones, 2007).

The next tests was TSC-02 with the assertion ASRT-07, referring to PLC Logger reporting the complete program code. This was to test for accuracy, however PLC Logger failed as it did not provide specific memory addresses that can be entered into PLC Logger. A solution would be to use a combination of PLC Logger and a Snap 7 Client. A Snap 7 Client [6] is used to communicate with

[6] http://snap7.sourceforge.net/

Table 3. Assertions mapped against the tests cases and the results from testing PLC Logger using an existing CFTT framework, more details are shown in Appendix A

	TSC-01	TSC-02	TSC-03	TSC-04	TSC-05	TSC-06	TSC-07
ASRT-01	Passed						
ASRT-02	Failed	Passed					
ASRT-03				Failed			
ASRT-04		Passed					
ASRT-05		Failed			Failed		
ASRT-06			Failed				
ASRT-07		Failed					
ASRT-08		Passed					
ASRT-09		Failed					Passed
ASRT-10						Passed	
ASRT-11	Passed						
ASRT-12						Passed	
ASRT-13			Failed				

Siemens S7 PLC. The client can be used as the supporting tool using the directory function to read only the memory addresses of the PLC. Also to gather information such as the number of program blocks (such as OB, FB) and the memory addresses within each program block.

TSC-02 associated with the assertion ASRT-9, refers to the automatic acquisition of the program code. This assertion failed, as it was found the user would have to manually enter the memory addresses.

Assertions 08 to 12 passed 5 out of 6 on usability as it allowed the user to easily identify how to start the acquisition process and save the data in a suitable format for later analysis.

TSC-03 next, with the assertion ASRT-13, refers to the acquisition over an encrypted network. Encryption during the acquisition process will prevent an attacker from intercepting the data acquisition and to ensure the data stream is secure. However, PLC Logger does not provide an option for this feature.

Another finding during testing was that, PLC Logger does not provide the option to delete memory data areas once they have been entered. A solution to this was deleting the configuration file in PLC Logger's installation folder.

The results from Experiment 2 show that PLC Logger failed half of the tests. This demonstrates that PLC Logger in its current state would not be suitable for use as an acquisition tool on SCADA systems. It still has potential, features such as the "Snapshot" function allowing users to directly take values from the memory addresses of the PLC. However, PLC Logger's shortcomings need to be addressed before it can be used in a forensic context.

Taking into consideration that PLC Logger is not a tool specifically developed for forensic purposes, functions such as encrypted secure network would not be expected to be present.

7. CONCLUSION AND FURTHER RESEARCH

In this paper, our first hypothesis was, that the program code once acquired could be used to identify the attacker's intention. The

findings from Experiment 1 indicate our hypothesis was accepted and by using PLC Logger to acquire the program code, we were able to extract useful forensic artefacts that could be useful in determining an attacker's intention.

Our second, hypothesis was that PLC debugging tools can be used as a forensic tool to acquire and analyse the program code from a PLC. In Experiment 2 we used NIST's CFTT framework to establish a method for testing forensic tools for PLCs and also to understand the tool's capabilities. The results showed this hypothesis was rejected as PLC Logger had failed half of the tests. Considering PLC Logger was designed to be used for diagnostic purposes on PLCs, and not as a forensic tool, this could be viewed as a expected result.

The use of NIST's CFTT framework would be ideal for initial testing of other acquisition and analysis tools for PLCs. Helping to identify any possible problems with tools and whether it would be a suitable tool for forensic investigations on SCADA systems. The results from these experiments can be used to improve PLC Logger for use as a forensic tool for SCADA systems. They can also be used as a specification for the initial design of an acquisition tool.

Currently we have conducted all of our experiments only on the Siemens S7 PLC. PLC Logger also supports Modbus TCP/IP, therefore further research should be carried out to determine whether it would be useful for other PLC brands. A practical problem, because PLC Logger was developed by open source developers, may at some point choose not to develop it further or fix bugs in the software. Further research would be required to add the additional forensic features that were found to be missing during the tool testing experiments. For example, the encrypted secure network was one of the missing features. A next step in the research would be to evaluate our research contributions by comparing it against a tool with similar features.

Current research suggests that remote attestation be implemented to solve the problem of attacks manipulating the PLC program code. However this is theoretical and has not been tested in practical experiments. It has been suggested that remote attestation is weak, as a verifier cannot prove that the device has not been compromised only that it is working as expected (Valente, Barreto, & Cardenas, 2014).

ACKNOWLEDGEMENTS

We would like to thank EPSRC for the funding of Tina Wu's DPhil programme. Tina would also like to thank Adrian Campos for teaching her how to program PLCs.

REFERENCES

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

Anobah, M., Saleem, S., & Popov, O. (2014). Testing framework for mobile device forensics tools. *Journal of Digital Forensics, Security and Law*, *9*(2), 221–234.

Basnight, Z. H. (2013). Firmware Counterfeiting and Modification Attacks on Programmable Logic Controllers. Retrieved from `http://oai.dtic.mil/oai/oai?verb=getRecord\&metadataPrefix=html\&identifier=ADA583401`

Carrier, B. (2003). Defining digital forensic examination and analysis tools using abstraction layers. *International Journal of digital evidence*, *1*(4), 1–12.

Clarke, N., Tryfonas, T., & Dodge, R. (2012). *Proceedings of the 7th international workshop on digital forensics and incident analysis WDFIA 2012*. University of Plymouth.

ENISA. (2013). *Can we learn from scada security incidents?* (Tech. Rep.). Enisa.

Falliere, N., Murchu, L. O., & Chien, E. (2011). W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*.

Flandrin, F., Buchanan, W. J., Macfarlane, R., Ramsay, B., & Smales, A. (2014). Evaluating digital forensic tools (DFTs).

Jones, R. (2007). Safer live forensic acquisition. *University of Kent*. Retrieved from `https://www.cs.kent.ac.uk/pubs/ug/2007/co620-projects/forensic/report.pdf`

Kilpatrick, T., Gonzalez, J., Chandia, R., Papa, M., & Shenoi, S. (2006). An architecture for SCADA network forensics. *Advances in Digital Forensics II*, *222*, 273–285\r364. Retrieved from `<GotoISI>://000240980400022`

McLaughlin, S. (2011). On dynamic malware payloads aimed at programmable logic controllers. *Proceedings of the 6th USENIX conference on Hot topics in security. HotSec*, *11*, 10.

NIST. (2007, December). *General test methodology for computer forensic tools.*

Patzlaff, H. (2013). *D7.1 preliminary report on forensic analysis for industrial systems* (Tech. Rep.). The CRISALIS Consortium.

Radvanovsky, R., & Brodsky, J. (2013). *Handbook of SCADA/control systems security*. Taylor & Francis. Retrieved from `https://books.google.co.uk/books?id=ukqGhkpOkdMC`

Rick Ayers, W. J., Sam Brothers. (2013, September). *Guidelines on mobile device forensics.*

Siemens. (2015, January). S7-1200 easy book [Computer software manual].

Taveras, P. (2013). SCADA live forensics: real time data acquisition process to detect, prevent or evaluate critical situations. *European Scientific Journal*, *9*(21).

Valente, J., Barreto, C., & Cardenas, A. a. (2014). Cyber-Physical Systems Attestation. *IEEE International Conference on Distributed Computing in Sensor Systems*, 354–357.

Van der Knijff, R. M. (2014). Control systems/scada forensics, what's the difference? *Digital Investigation*. doi:

10.1016/j.diin.2014.06.007

Vidas, T. (2007). The acquisition and analysis of random access memory. *Journal of Digital Forensic Practice*, *1*(4), 315–323.

Zhu, B., Joseph, A., & Sastry, S. (2011). A taxonomy of cyber attacks on scada systems. In *Proceedings of the 2011 international conference on internet of things and 4th international conference on cyber, physical and social computing* (pp. 380–388).

APPENDIX

A. TOOL TESTING RESULTS

A.1 ASRT-01

A.1.1 TSC-01 [PASSED]

The tool was capable of using the network through the PLC's IP address to acquire the memory addresses stored on the PLC.

A.2 ASRT-02

A.2.1 TSC-01 [FAILED]

The tool does not provide the user with a notification when the connection to the PLC is successful

A.2.2 TSC-02 [PASSED]

The tool does not notify the user if the acquisition is successful however the tool provides graphical visualisation that an connection has been established and acquisition can be visualised in the tool.

A.3 ASRT-03

A.3.1 TSC-04 [FAILED]

The tool does not notify the user when the connection between the PLC and the tool is unsuccessful.

A.4 ASRT-04

A.4.1 TSC-02 [PASSED]

The tool outputs values within the memory addresses with useful data that can be linked back to the program code. For example the tool was able to determine whether the values in the memory addresses were either True or False

A.5 ASRT-05

A.5.1 TSC-02 [FAILED]

The tool does not notify user if the acquisition is successful or unsuccessful. However it was identified if an memory area added that is not within the memory range of the PLC then a red exclamation is shown in PLC Logger

A.5.2 TSC-05 [FAILED]

The tool does not notify the user when the PLC is disconnected from the network. This can viewed in the graphical visualisation to whether the memory addresses are outputting any values

A.6 ASRT-06

A.6.1 TSC-03 [FAILED]

After examining the captured network traffic between the PLC and PLC Logger it was undetermined whether there were any modifications were made during live acquisition.

A.7 ASRT-07

A.7.1 TSC-02 [FAILED]

The tool only allows the user to view the memory addresses but is unable to allow the user to view the number of program blocks within the program code

A.8 ASRT-08

A.8.1 TSC-02 [PASSED]

The tool has a button labelled with a "Play" button and allows the user to start data capturing based on the customised configurations

A.9 ASRT-09

A.9.1 TSC-02 [FAILED]

The tool requires some input from the user, in order to be able to acquire data from the memory addresses the user would need to manually enter each memory address as the tool does not automatically detect the memory addresses in the PLC.

A.9.2 TSC-07 [PASSED]

Once the memory addresses are entered into the tool the acquisition process can be completed

A.10 ASRT-10

A.10.1 TSC-06 [PASSED]

The tool provided the option to export the data in two formats MySQL and CSV files. The tool successfully allows the user to save the files to be used later for further analysis including the timestamps.

A.11 ASRT-11

A.11.1 TSC-01 [PASSED]

The tool is capable of acquiring data over the network while the PLC is online

A.12 ASRT-12

A.12.1 TSC-06 [PASSED]

The tool was capable of saving the acquired data to a local folder

A.13 ASRT-13

A.13.1 TSC-03 [FAILED]

The tool does not provide an option to allow the user to have an encrypted secure acquisition

