Forensic Science International: Digital Investigation 33 (2020) 301013

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS 2020 USA — Proceedings of the Twentieth Annual DFRWS USA

Control Logic Forensics Framework using Built-in Decompiler of Engineering Software in Industrial Control Systems

Syed Ali Qasim [a, *], Jared M. Smith [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:
Control logic
Industrial control systems
Forensics
SCADA

ABSTRACT

In industrial control systems (ICS), attackers inject malicious control-logic into programmable logic controllers (PLCs) to sabotage physical processes, such as nuclear plants, traffic-light signals, elevators, and conveyor belts. For instance, Stuxnet operates by transfering control logic to Siemens S7-300 PLCs over the network to manipulate the motor speed of centrifuges. These devestating attacks are referred to as control-logic injection attacks. Their network traffic, if captured, contains malicious control logic that can be leveraged as a forensic artifact. In this paper, we present Reditus to recover control logic from a suspicious ICS network traffic. Reditus is based on the observation that an engineering software has a built-in decompiler that can transform the control logic into its source-code. Reditus integrates the decompiler with a (previously-captured) set of network traffic from a control-logic to recover the source code of the binary control-logic automatically. We evaluate Reditus on the network traffic of 40 control logic programs transferred from the SoMachine Basic engineering software to a Modicon M221 PLC. Our evaluation successfully demonstrates that Reditus can recover the source-code of a control logic from its network traffic.

1. Introduction

Industrial control systems (ICS) monitor and control industrial physical processes (e.g., nuclear plants, electrical power grids, and gas pipelines (Ahmed et al., 2016)). ICS consists of a control center and various field sites. The control center runs ICS services such as the human–machine interface (HMI) and engineering workstation. The field sites use programmable logic controllers (PLCs), sensors, and actuators to control the physical processes.

PLCs are the main target of a cyber attack to sabotage a physical process (Ahmed et al., 2012; Ahmed et al., 2017; Senthivel et al., 2018; Garcia et al., 2017; Valentine and Farkas, 2011; Kush et al., 2011). They run a control logic that defines how a physical process should be controlled. Attackers target the control logic over the network to manipulate the behavior of a physical process, referred to as a *control-logic injection attack* (Yoo et al., 2019a; Kalle et al., 2019; Govil et al., 2017; Yoo et al., 2019b). For instance, Stuxnet infects the control logic of a Siemens S7-300 PLC to modify the motor speed of centrifuges periodically from 1410 Hz to 2 Hz to 1064 Hz (Falliere et al., Chien; Chen and Abu-Nimeh, 2011). Stuxnet can compromise Siemens SIMATIC STEP 7 engineering software at the control center and download malicious control logic to the PLC at the field sites over the network. If the network traffic between the control center and field sites is captured, the traffic will contain the evidence of the transfer of the malicious control logic. Current research lacks forensic techniques that can extract the control logic from the network traffic dump and further transform it back to a high-level source code for forensic analysis.

There are some existing partial solutions. Laddis is a state-of-the-art forensic solution to recover a control logic from an ICS network traffic dump (Senthivel et al., 2018). Mainly, Laddis is a binary control-logic decompiler for the Allen–Bradley's RSLogix engineering software and MicroLogix 1400 PLC (AllenBradlry, 1400). It uses a complete knowledge of the PCCC proprietary protocol to extract the control logic from the network traffic and further utilize low-level understanding of binary control-logic semantics for decompilation. Unfortunately, Laddis requires tedious and time-consuming manual reverse engineering efforts for exploring the ICS proprietary network protocols and semantics of binary control-logic.

Another state-of-the-art forensic solution is Similo, which addresses some of the short-comings of the Laddis system, including manual reverse engineering (Qasim et al., 2019). Similo is designed to investigate control-logic theft attacks where the attacker reads the control logic from a PLC over the network. However, Similo does

* Corresponding author.
E-mail address: qasimsa@vcu.edu (S.A. Qasim).

https://doi.org/10.1016/j.fsidi.2020.301013

not support the forensic investigation of control logic injection attacks where the attacker transfers a malicious control logic from the engineering software to a target PLC.

To address the shortcomings of these systems, we propose Reditus, a novel control-logic forensics framework for control logic injection attacks. Reditus extracts and decompiles control logic from a network traffic dump automatically *without any manual reverse engineering* and *without knowledge of ICS protocols and underlying binary control-logic format*. Reditus is based on the observation that an engineering software can read a control logic from a PLC remotely referred to as the *upload* function, and has a built-in decompiler that can further transform the control logic into its source-code. Our core idea is to integrate the decompiler with a (previously-captured) network traffic of a control-logic using the *upload* function to recover the source code of the binary control-logic automatically.

The Reditus framework has a virtual-PLC that engages the engineering software using an ICS network traffic. When the engineering software attempts to read a control logic chunk from a PLC, the virtual-PLC receives a request-message, finds its control-logic message from a traffic dump (currently being analyzed), and then creates a response-message to send it back to the engineering software. The Reditus framework is designed to handle several challenges, including redirecting the attack messages of malicious control logic (i.e., engineering software → target PLC) back to engineering software, identifying correct control-logic messages in a traffic dump, updating session-dependent dynamic fields (e.g, transaction ID) in the response messages, and exchanging messages for establishing and maintaining a session.

We evaluate Reditus on a popular Schneider Electric device, the Modicon M221 PLC (Modicon, 1354), and SoMachine-Basic engineering software (Modicon, 1474). Our dataset consists of the network traffic dumps of 40 different control logic programs downloaded to the PLC over the network. Notably, our evaluation demonstrate that Reditus can accurately recover the source-code of a control logic from a network dump.

1.1. Contributions

Our contributions can be summarized as follows:

- We present Reditus, a novel forensic framework to investigate control-logic injection attacks.
- We evaluate Reditus on real-world PLCs and engineering software actively used in modern industrial-settings.
- We release our datasets of the network traffic of 40 control logic programs (Reditus-Framework-Git-Repository).

1.2. Roadmap

We have organized the rest of the paper as follows: Section 2 provides the background. Section 3 presents the motivation and challenges for facing state-of-the-art control-logic forensics techniques. Section 4 presents the `Reditus framework`, followed by implementation and evaluation results in sections 5 and 6. Section 7 covers related work, followed by the final takeaways in Section 8.

2. Background

2.1. ICS primer

Fig. 1 provides an overview of an industrial control system environment for a gas pipeline scenario. This setup has a control center and field sites.

2.1.1. Physical Process

In a gas pipeline scenario, the gas is compressed and transported to a remote receiver through a pipeline. The physical process consists of an air compressor, storage and receiver tanks, and solenoid valves. The compressor compresses the air and stores it in a storage tank that is connected with a receiving tank through a pipe. The tanks are closed with solenoid valves. When the compressed air needs to be transported to the receiving tank, the valves are opened to let the air flow through the pipe.

2.1.2. Field site

The gas pipeline infrastructure is located at field sites and are monitored and controlled via a pressure transmitter, solenoid valves, and PLCs. The pressure transmitter is connected to the receiving and storage tanks and also send data to their respective PLCs. The PLCs are programmed with a control logic to achieve the following two controls: first, they open the valves to let the air flow through the pipe to transport the compressed air to the receiving tank. Second, they monitor the pressure of the tanks and maintain a desired level by releasing the air if the pressure is high.

2.1.3. Control center

The PLCs send data to the control center comprised of an HMI, Historian and Engineering Workstation. The HMI displays the current state of the gas pipeline process graphically. The Historian is a database application to store the PLC data for analytics. The Engineering Workstation runs an engineering software for the programming, configuration, and maintenance of the PLCs remotely.

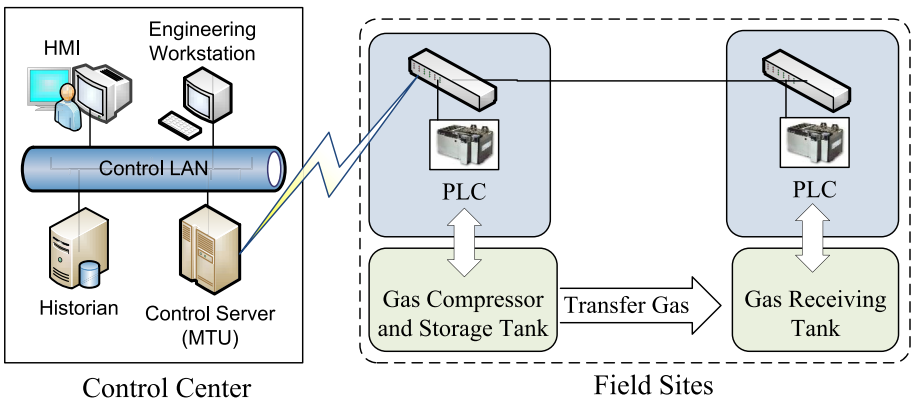

Fig. 1. Industrial control system for a gas pipeline scenario.

2.2. Engineering software and control logic

In practice, engineering software is used to write control logic for PLCs. This proprietary programming software is offered by ICS vendors to configure, program, and perform maintenance on their PLCs. For instance, SoMachine Basic, RsLogix 500, and CX-Programmer are used for the PLCs of Schneider Electric, Allen−Bradley, and Omron respectively.

The IEC 61131-3 (IEC, 1131) standard defines five programming languages for a PLC. Two languages are graphical i.e., ladder logic (LL) and function block diagram (FBD), and three are textual programming languages i.e., sequential function chart (SFC), structured text (ST) and instruction list (IL). Fig. 2 shows a code snippet in both ladder-logic and instruction-list. Ladder-logic represents instructions in graphical symbols. An instruction-list program consists of a sequence of textual instructions, similar to assembly language.

3. Problem statement and challenges

3.1. Problem statement

Given a network traffic dump of a malicious control logic transferred over the network to a target PLC, our goal is to develop a fully-automated forensic solution that can recover the binary control logic from the network dump and then, convert it into a human-readable form for forensic analysis.

3.2. Challenges in control-logic forensics

Several challenges exist to achieve our stated goal of the control logic forensics because of the proprietary control logic format and ICS protocols.

- Binary control-logic does not have a standard open format (such as Linux ELF) and has vendor-specific proprietary format.
- Engineering-software typically supports one or multiple languages defined by IEC 61131-3 standard. For instance, RsLogix only supports ladder logic; SoMachine-Basic supports ladder logic and instruction list. Binary control-logic must be transformed into its respective high-level language.
- Proprietary ICS protocols are used to transfer a control-logic to a PLC from an engineering software. Their specifications are not publicly available. If an open protocol is used, it encapsulates a proprietary layer. For instance, Modicon-M221 PLC and SoMachine-Basic use Modbus open-protocol. However, its *data* field further contains proprietary fields such as the control-logic address in PLC memory, function code, and control logic content.

4. Reditus - a control-logic forensics framework

4.1. Overview

We propose Reditus, a forensics framework to investigate control-logic injection attacks. Reditus consists of a virtual-PLC that can communicate with the engineering software and use its upload functionality to recover the high-level control logic from the network traffic. It is a fully automatic approach and does not require *any* knowledge of ICS protocols or the underlying binary control logic format.

4.1.1. Communication with engineering software

Engineering software is equipped with both *upload* and *download* functions. The *download* function transfers a control-logic to a PLC. The *upload* function retrieves a binary control-logic from a PLC and further transforms it to source-code in its respective IEC language.

In both operations, the engineering software communicates with the PLC over a series of request-response messages. Critical to Reditus we observe that starting from the very first message used for establishing the session, the communication is deterministic.

4.1.2. Binary Chunks

To transfer the control logic over the network, the engineering software divides the binary control logic into a number of chunks. The maximum size of one chunk can be 236 bytes. During the download operation, the engineering software always starts writing from the same memory address (i.e "0080"), whereas during upload the software always starts reading from address "d4fe". Moreover, for both download and upload, the number of bytes read or written for each memory address are also same. Fig. 4 shows the comparison of the download and upload request messages from the engineering software to the PLC. In the download request message, the engineering software writes 43 bytes of control logic on address c404 in the PLC memory. In the corresponding upload request, the engineering software uses the same address and number of bytes (i.e the binary chunk). Due to this behaviour of engineering software, Reditus can use the downloaded network capture to recover the control logic without needing any binary or decompilation information.

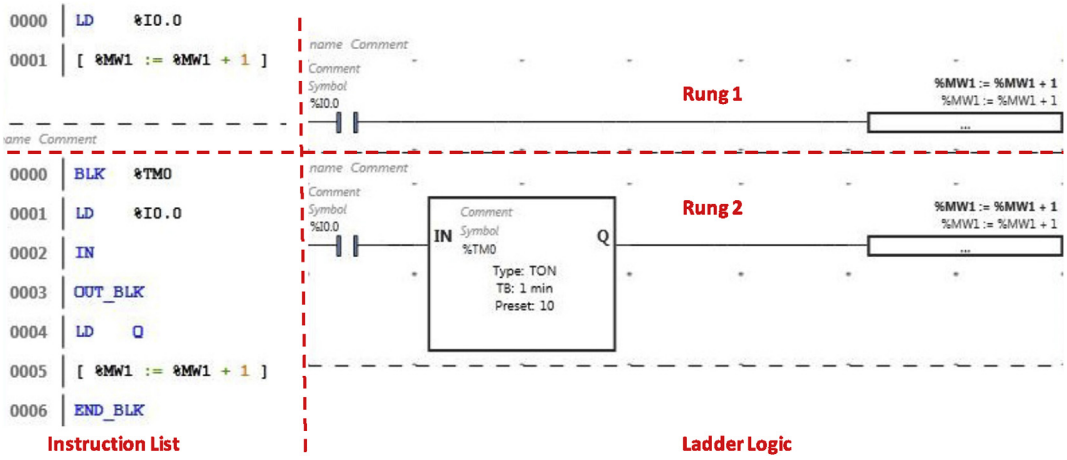

Fig. 2. A code snippet in both instruction list (textual language) and ladder logic (graphical language).

4.1.3. Virtual PLC

Based on the heuristics and observations aforementioned, we developed Reditus as a virtual PLC framework that can take the network traffic captured during the upload or download operation and use it to transfer the control logic within the traffic. To that end, Reditus must learn two relevant pieces of information: first, the locations of all session-dependent, variable fields present in the messages. Second, Reditus must learn the mapping between the download and upload messages to generate a proper response to the upload request using download traffic. With this information, Reditus can communicate with the engineering software using previously captured network traffic after editing the messages according to the new session and request message type (download or upload). In order to achieve this, Reditus goes through a learning phase where it performs differential analysis (Garfinkel et al., 2012) on varying benign network captures. After the learning phase, a forensic analyst can employ Reditus to communicate with the engineering software and perform in-depth forensic analysis on the captured network traffic.

4.2. Learning phase

In the learning phase, Reditus ingests multiple sets of benign network captures in the form of PCAP files to gather the session-dependent fields and upload template using differential analysis. The learning phase consists of two parts. In the first part, Reditus extracts information about the location of session-dependent fields present in the messages. Next, Reditus automatically learns the different types of fields in the upload response message from the PLC and their mapping with the similar fields in download request message. This allows the framework to make a template for the upload messages. The learning is done using only benign PCAP files to make sure that Reditus learns the correct message format. Fig. 3 shows a graphical overview of Reditus.

4.2.1. Session dependant fields

As shown in Fig. 3, the first learning phase in Reditus discovers the session-dependent fields in the messages. We leverage a heuristic-based approach to compare the two benign PCAP files containing the same control logic in the same transfer direction. Since we are using the same control logic in both PCAP files, if we compare the same message from two PCAP files, the majority of the message will be the same and only the session-dependent fields like session ID will differ. Fig. 6 shows the same message present in two PCAP files from different sessions. Clearly, most of the two messages remains the same and only the Transaction ID differs. As shown in Fig. 3, the learning of session-dependent fields consists of Pairing, Grouping and Differential Analysis, and Rule Extraction.

4.2.1.1. Pairing. Reditus first takes multiple sets of two benign PCAP files from different sessions that have the same control logic and transfer direction (both upload or download). Then for each request message *K1* in the first PCAP, it finds the similar request message *K2* in the second PCAP. To identify the similar message, we use two parameters, the size of the message and the similarity of two messages strings. Reditus then compares all the request messages in PCAP1 with all the request messages in PCAP2, and for each message the framework automatically finds the most similar message in the second PCAP. After identifying the most similar message, it pairs them together *(K1,K2)* for further analysis.

4.2.1.2. Grouping and analysis. After acquiring the message pairs, Reditus performs differential analysis on each pair, comparing the two key messages character by character and for each pair it notes the indices where the messages differ. These indices indicate the location of session-dependent fields. It is important to note that messages between the PLC and the engineering software can be of multiple lengths and the location of session-dependent fields may also vary. Therefore, Reditus forms groups based on the length of messages present in each pair to ensures that all messages present in one group share the same message format. This process is done for all different sets of PCAP files.

4.2.1.3. Rule extraction. After performing differential analysis on multiple sets of PCAP files and grouping the possible session-dependent fields, the final step of the first learning phase examines each group to identify noise or any false positives. We assume that for each message group, the location or index of session-dependent field is consistent, so at this state Reditus takes only those possible session-dependent fields that are consistent in the majority of the messages and discards all other fields that are inconsistent. This process is also repeated for all the PCAP sets and the results are again combined in an identical manner to get one set of session-dependent fields for each message group. Though this step removes the noise, two problems remain: first, there is no definitive boundry between the fields, i.e if two fields in the protocol are adjacent they will be considered as one. Second, if any session-dependent field is not completely different in two PCAP files Reditus will not be able to extract the complete session-dependent field. For example in the message shown in Fig. 6, the Transaction ID is of two bytes i.e from index 0 to 3. Since "3″ at index 0 is common in both messages, the differential analysis will identify incomplete field consisting of three characters i.e. from index 1 to 3.

Since protocol reverse engineering is not the objective of Reditus, we can ignore the first problem as we are only interested in identifying and updating the session-dependent fields so we can reuse the previously captured network traffic. Even if two session-dependent fields appear as one, Reditus will update this combined session-dependent field in the response message while communicating with the engineering software. Reditus addresses the second problem in the testing phase by doing a comparison of a newly received request and similar request in the database. It is possible that the test phase comparison may produce some false positives, so Reditus combines the result of this comparison with information gathered in the learning phase to produce the final session-dependent fields. Reditus then takes the session-dependent fields learned previously as a baseline and only selects the fields from the testing phase that are adjacent to, overlapping, or confined in any of the baseline fields, ignoring the rest.

4.2.2. Upload template

If the network stream under investigation was captured during uploading a program from the PLC to the engineering software, then Reditus can easily upload the control logic present in the PCAP file to the engineering software by only modifying the session-dependent fields present in the message. However, if the network traffic contains download traffic, then in order to respond to the upload request message, Reditus must generate a response from scratch, since the format of upload response is different from download as shown in Fig. 5. To generate the upload template, Reditus again uses a heuristic-based approach. By examining the upload responses messages from a real PLC to the engineering software, we observed four types of fields present in the upload response, which are also shown in Fig. 7.

1. Session-Dependent Fields: Vary over different sessions (e.g. Transaction ID). The value of these fields does not depend on the content of the message.

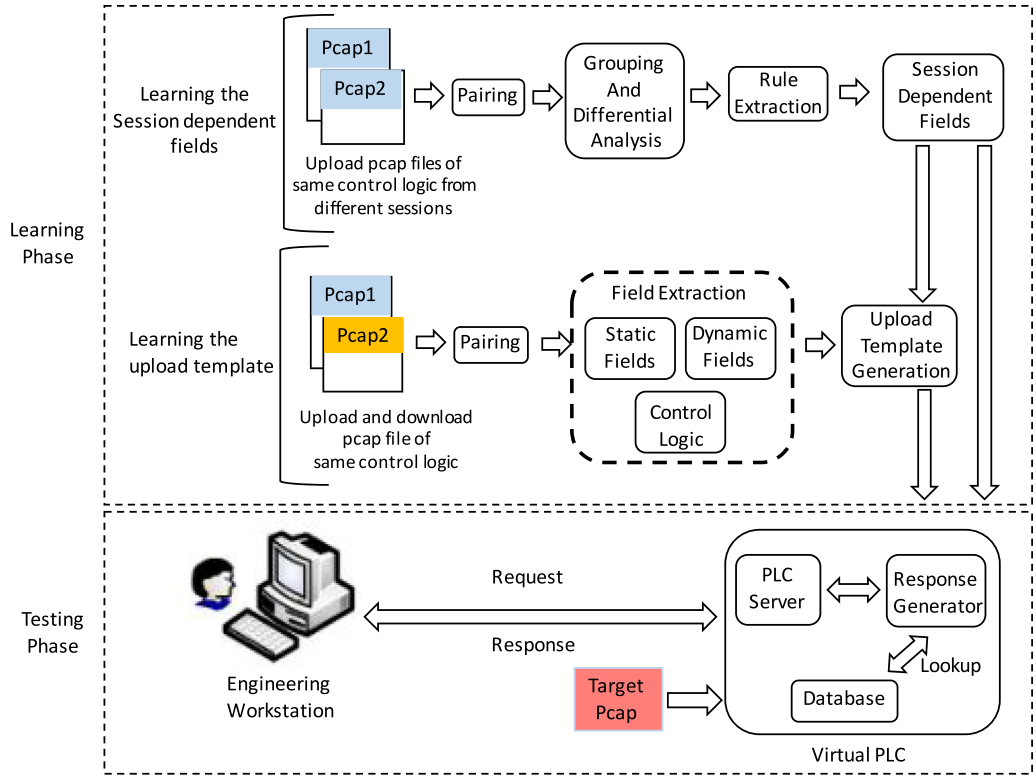

Fig. 3. Overview of Reditus.

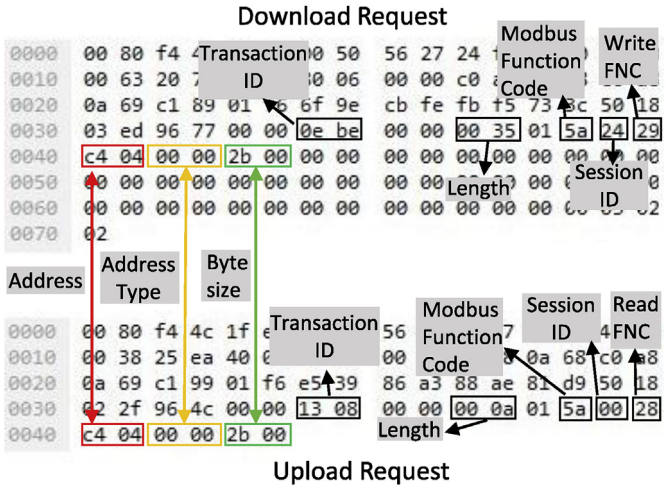

Fig. 4. Comparison of download and upload requests for the same address.

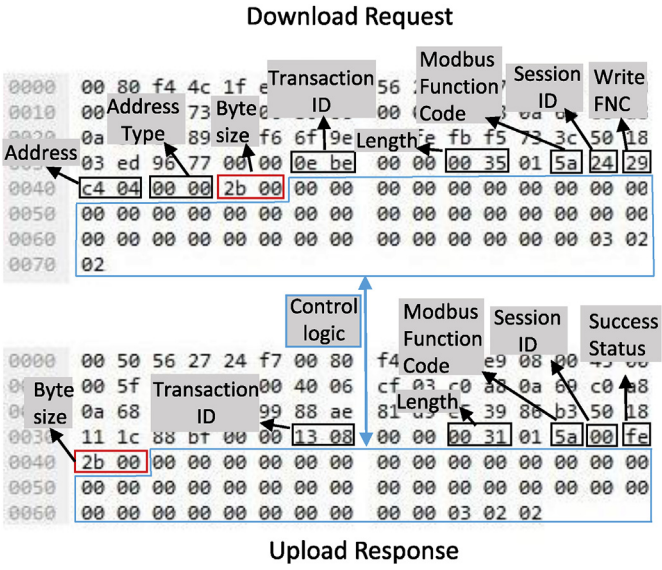

Fig. 5. Comparison of download request and upload response for the same address.

2. Static Fields: Remain constant over all the upload response messages such as the Modbus function code or success function code.
3. Dynamic Fields: Depend on the content of the message and vary in different messages (e.g. length of the message which depends on the size of the control logic being transfered)
4. Control Logic: Control logic part of the message. Its size may vary in different messages, but according to our observation, it always comes after the aforementioned fields.

Since Reditus has already identified the session-dependent fields in the first part of learning, in the second part, the framework tries to find the remaining three fields to make the upload template. To accomplish this, Reditus takes sets of two benign PCAP files that contain the same control logic but have a different transfer direction, i.e one upload and the other download. These files only contain the request-response messages containing the control logic. Similar to previous learning, the first step is pairing, followed by identifying the dynamic, control logic, and static fields. Finally, Reditus combines this knowledge to form the upload template.

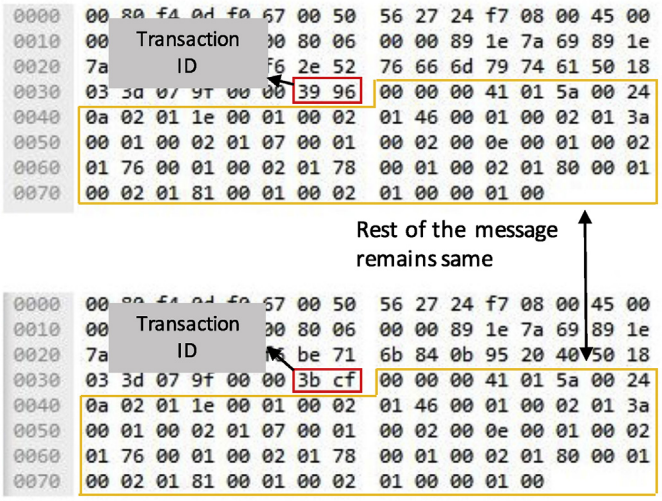

Fig. 6. Same request message in two different session.

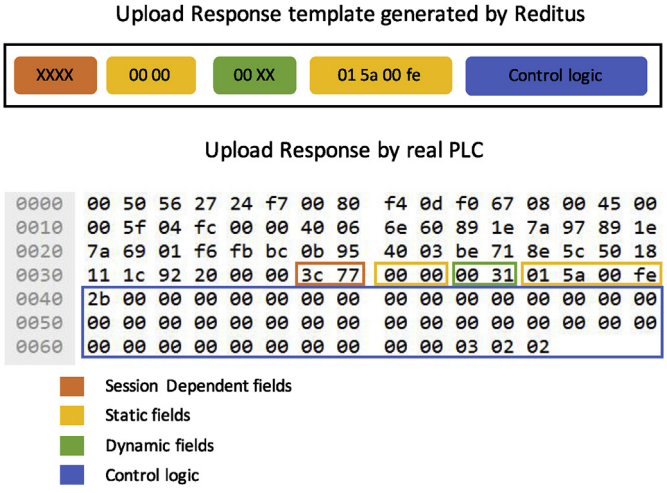

Fig. 7. Comparison of the upload response template generated by Reditus and the update response from a real PLC.

4.2.2.1. Pairing. In the first step, Reditus pairs the request and response message present in the upload traffic with the corresponding message in the download. But unlike the first phase, the length and format of the upload and download request message is different as shown in Fig. 4. In the download request message, the engineering software sends a chunk of control logic to the PLC along with its size and memory address where the PLC should write the logic. Notably, the upload request is much shorter and the engineering software only asks the PLC to send the control logic chunk present on the address provided in the message. To pair the messages reading and writing on the same physical address, we assume that the control logic will be in the later part of the download request. To pair similar messages, we use the similarity of the upload request and download request equal to the length of upload request. Reditus then calculates the similarity of an upload request message with all of the request messages in the download PCAP file, forming a four element tuple of upload and download requests and responses where the similarity is highest among the messages.

4.2.2.2. Dynamic fields. The length of the message is a very common field in almost every protocol, so in order to generate the upload template, Reditus needs to find the location of length field in the message. In developing Reditus, we assumed that the size of the length field is two bytes because it is the average size of the length field in ICS protocols. To find the exact location of length field for each upload request message, Reditus slides a window of two bytes, i.e 4 characters, calculates the length of message after the window, and compares it with the value inside the sliding window. If the length of remaining message to the left of window is equal to the value inside the window, then the indices of the window become a potential length field. Unfortunately, this process can produce false positives as its possible that by chance the value with the window matches the length of the remaining message. For example, if the value inside the window is *00 01* and there is only one byte after the window, it will be considered the length field, as shown at the end of the messages in the Fig. 6. To remove these false positives, Reditus calculates the possible length fields in all the message tuples and only then selects the fields that are present in all the messages tuples. In this way, all false positives are eliminated and Reditus can acquire the exact location of length field in the message.

4.2.2.3. Static fields. Another important component of upload messages are the static fields, the part of message that remains same in all the upload response messages (e.g modbus function code). To identify the static fields, Reditus compares all the upload response messages with each other and identifies the indices where the all the message have the same value.

4.2.2.4. Identifying control logic. The control logic is the most important field for generating the upload template. As mentioned in section 4.1, the engineering software always divides the control logic into the same chunks for both download and upload. We assume then that similar messages in upload and download stream will have the same control logic piece present in them. The challenge then is identifying the location of control logic in the upload response and download request so that it can be used to make the upload template. For this purpose, we used a heuristic-based approach based on the longest common sub-sequence (LCS) present in the download request and upload response. For each message tuple, Reditus computes the LCS between the download request and upload response and notes the starting and ending index in both the messages. It is possible that in some messages the control logic part is smaller than the message header, so in that case, Reditus will learn an incorrect location for the control logic. To resolve this issue, after finding the location of the LCS in all download request and upload response message pairs, Reditus finds the starting and ending LCS indices that are common in most of the message pairs and uses it for the final template.

4.2.2.5. Generating template. With the control logic in hand, Reditus combines all fields together to generate the final template. It first forms a string where it takes an upload response message from the upload PCAP files and normalizes it by replacing all the values except the static fields with *X* as shown in Fig. 7. This serves as a basic upload template. During the testing phase, Reditus uses this template and then updates it according to the session-dependent, dynamic, and control logic fields to generate the final response message.

4.3. Testing phase

In the testing phase, Reditus runs a virtual PLC and takes the network capture under investigation in the form of PCAP file. First, it generates a database of request and response messages present in the PCAP. Then it starts a PLC server on the same port as the real

PLC, i.e 502 in the case of the Modicon M221, and waits for request messages from the engineering workstation. Once Reditus is running, the control engineer can connect with it using the engineering software and the *IP* address of the computer running Reditus. Once the PLC server receives any request messages from the engineering software, it forwards them to the Response Generator, which make the response message using the messages present in database, upload template, and session-dependent fields and gives the message to the PLC server that sends the response to the engineering software.

4.3.1. Response generator

The most important component of our virtual PLC finds the most similar request message in the database for all the new request messages from the engineering software and updates the response according to the new session. In order to do this, the virtual PLC uses the same similarity-based approach used for pairing the messages in section 4.2.2 and finds the request message from the database that has the maximum similarity with the new request. After correctly identifying the request message similar to the new request, the next step is to modify the corresponding response message according the new session. If the network traffic under investigation contains the control logic upload, Reditus only needs to update the session-dependent fields. For download traffic, it extracts the control logic portion using the information learned in section 4.2.2 and puts it in the upload template generated earlier. Finally, Reditus updates the session-dependent fields and dynamic fields such as length in this newly crafted response message and forwards the message to the PLC server which sends the response message to the engineering software. We assume that the upload or download direction information is provided to Reditus by the user.

5. Implementation

Reditus is developed in Python. It takes the network captures in the form of PCAP files sas input. Based on the IP address, Reditus filters the request and response message and stores the transport layer payload of the request message and response message as a key and value for a dictionary. The payload is stored as a hexidecimal string. We used Scapy (Scapy. [link].L https) for creating, filtering, and modifying network packets. The PLC server is implemented using a server socket on port 502 (the same port used by the real Modicon M221 PLC). To measure the similarity of two messages, we used the ratio() function of the SequenceMatcher class from Python's difflib library (diflib. [link].L https). For any two given messages, difflib provides a similarity ratio between *[0,1]*. 1 if the two messages are identical and 0 if there is no similarity. To find the longest common sub-sequence (LCS), we wrote a program using dynamic programming. The program takes the two strings and returns a string representing the LCS.

6. Evaluation

6.1. Experimental setting

6.1.1. Lab setup

We evaluated Reditus on a Schneider Electric Modicon M221 PLC and SoMachine Basic V1.6 SP2. The engineering software was installed on a Windows 7 virtual machine and Reditus was running on an Ubuntu 18.04.3 LTS machine. All three devices were connected and on the same network subnet.

6.1.2. Dataset

The dataset used for the evaluation consisted of 40 control logic programs of varying complexity and sizes, both in terms of the number of rungs and the number of instructions. Table 1 shows the summary of our dataset.

6.1.3. Experiment methodology

A typical experiment replicates a control-logic injection attack by downloading a control logic file to the Modicon M221 PLC and capturing the Modbus network traffic in the form of PCAP file. This file is then provided to Reditus, and then a connection is established from the engineering software to Reditus and the upload function of the engineering software is used to acquire the high-level control logic present in the PCAP file. Finally, the control logic uploaded by Reditus is manually compared with the original control logic file to find the transfer accuracy of Reditus.

6.2. Functional-level accuracy

In this section, we will evaluate the functionality of Reditus. The two most important requirements of Reditus are: 1) Reditus should be able to match the download request messages from one PCAP with the corresponding upload request message in the other PCAP file, and 2) Reditus should be able to generate a upload to generate the upload response message using download network traffic.

6.2.1. Matching accuracy

When Reditus receives any upload request message, it needs to find the matching download request from the database, edit the corresponding response message, and send the response to the engineering software. If the response message is different from what the engineering software was expecting, the engineering software will terminate the communication with an error. As shown in Fig. 4, the upload and download request messages have different formats, so finding the exact match is non-trivial. In our experiments, we found that the similarity and length based matching approach used by Reditus for matching the upload request message with the download request message present in the database works with *100% accuracy* across our entire dataset for a real PLC.

Table 2 summarizes the database look-ups during our experiments. While uploading the 40 control logic files, Reditus received 1929 read request (upload) messages from the engineering software and was able successfully find all corresponding write request (download) messages. For evaluation purpose, we compared the address, address type, and control logic size field of the two messages.

6.2.2. Upload template accuracy

The second most important function of Reditus is to learn the upload response template from the sample PCAP files and use it to generate upload response messages from the target PCAP file. To evaluate the accuracy of upload template, we manually compared the template with the upload response message from a real PLC to check if 1) our template contains all four types of fields (Session-Dependent, Static, Dynamic and Control Logic) and 2) if the location

Table 1
Summary of our Dataset for M221 PLC.

File Size (kb)	# of Files	Rungs				Instructions			
		Min	Max	Avg	Total	Min	Max	Avg	Total
60–80	24	1	5	2.75	66	2	23	10.75	258
81–90	5	2	5	3.8	19	8	16	10.2	51
91–100	4	5	16	9	36	19	112	50	200
101–120	4	8	14	10	40	20	72	36.5	146
120+	3	12	26	17.3	52	36	118	77.66	233
Total	40	–	–	–	213	–	–	–	888

Table 2
Summary of database look ups.

File Size (kb)	# of Files	# Read Messages Received	# Successful lookups	Match Accuracy %
60-80	24	1105	1105	100%
81-90	5	265	265	100%
91-100	4	192	192	100%
101-120	4	198	198	100%
120+	3	169	169	100%
Total	40	1929	1929	–

in the upload template for each field is exactly the same as in the upload response message from the real PLC. During our experiments we found that Reditus was again able to generate the correct upload template with *100% accuracy* across the entire dataset. Fig. 7 and Table 3 show that the upload template generated by Reditus has all the required fields and locations/indices as a response message from the real PLC.

6.3. Packet-level accuracy

The main assumption in the development of Reditus is that during upload and download, the engineering software reads and writes control logic on the PLC, and if Reditus only has download network traffic it can find all the control logic that was written on the PLC and can send it to the engineering software using an upload template. Specifically, for every read message from the engineering software, there is a write message in the target PCAP. Table 4 shows the summary of control logic read and write messages during our experiments. While transferring 40 different control logic files, Reditus received 1852 unique read messages out of which 1812 were present in the database. Upon examining the missing messages, we found that every download PCAP file was missing the same message, related to the functional-level. Although our assumption was not 100% accurate, only one of more than 1800 messages was missing. This problem can be resolved by keeping a separate database of missing messages and connecting it with Reditus. If the target PCAP (download) file does not have any message, Reditus can look for the missing message in the second database.

6.4. Transfer accuracy

The most important metric to evaluate Reditus is the integrity of the control logic transferred by Reditus. If Reditus introduces any change in the control logic during the upload process it cannot be used as a forensic tool. In order to find the transfer accuracy, we uploaded 40 different control logic programs of various complexities and sizes using Reditus and then manually compared each of them with the original control logic program. We not only compared the number of rungs and instructions but also made sure that each rung and instruction is the same in both versions. Table 5 highlights the summary of control logic programs uploaded by Reditus. Notably, Reditus successfully uploaded 40 control logic programs containing 213 rungs and 888 instructions with 100% transfer accuracy.

7. Related work

There are few tools available that can perform forensic analysis and acquire the high-level representation of control logic from ICS network traffic. Senthival et al. (Senthivel et al., 2017) introduced a tool named “cutter” that can parse the network traffic of PCCC protocol and extract forensic artifacts such as SMTP client configuration, control logic binary, and other system configuration files. Cutter is limited to extracting different types of PCCC files and is not able to convert the control logic binary to high-level representation. To address this limitation, the authors took another manual reverse engineering approach to develop Laddis (Senthivel et al., 2018). Laddis can decompile the low-level binary of ladder logic to a higher-level representation. Although Laddis can decompile the ladder logic program from the network traffic on both directions i.e upload and download, it only works for the PCCC protocol and Allen–Bradley RSLogix 500 engineering software.

Kalle et al. (Kalle et al., 2019) developed Eupheus, a decompiler that can transform the low-level control logic in the form of machine code of an RX630 to instruction list program. It was evaluated on SoMachine Basic and Modicon M221 PLC. The authors also presented a virtual PLC that can communicate with the engineering software. Using Eupheus and a virtual PLC, the authors performed a remote control logic injection attack on the Modicon M221 PLC. Similar to Laddis, Eupheus and the virtual PLC are based on manual

Table 3
Comparison of the location of different fields in an actual PLC response and a template generated by Reditus.

Field type	Indices in upload response from PLC	Indices identified by Reditus in template	Template Accuracy
Session dependant	0,1,2,3	0,1,2,3	100%
Static	4,5,6,7,12,13,14,15 ,16,17,18,19	4,5,6,7,12,13,14,15 ,16,17,18,19	100%
Dynamic	8,9,10,11	8,9,10,11	100%
Control Logic	20-end of message	20-end of message	100%

Table 4
Summary of control logic read and write messages during the experiments.

File Size (kb)	# of Files	Unique Read Message in Upload	Unique Write Message in Download	Messages missing in Download	Message missing per file
60–80	24	1060	1036	24	1
81–90	5	255	250	5	1
91–100	4	184	180	4	1
101–120	4	191	187	4	1
120+	3	162	159	3	1
Total	40	1852	1812	40	–

Table 5
Comparison of control logic uploaded by Reditus and the original M221 PLC.

File Size (kb)	# of Files	M221 PLC		Reditus		Upload Accuracy
		Rungs	Instructions	Rungs	Instructions	
60–80	24	66	258	66	258	100%
81–90	5	19	51	19	51	100%
91–100	4	36	200	36	200	100%
101–120	4	40	146	40	146	100%
120+	3	52	233	52	233	100%
Total	40	213	888	213	888	–

reverse engineering and cannot be used for any other ICS protocol or PLC. Though SoMachine Basic and Modicon M221 provide both the Ladder Logic and Instruction List options to represent the control logic, Eupheus can only convert the low-level binary to an instruction list program and will require additional effort to show the Ladder Logic representation.

Qasim et al. (Qasim et al., 2019) developed Similo to recover control logic from the ICS network traffic. Similar to Reditus, Similo can learn the protocol fields automatically from the network traffic. However, it only works if the control logic is being uploaded in the network capture which limit its use for the forensic investigation of control logic injection attacks such as Stuxnet, where the attackers only download the control logic to the PLC.

8. Conclusion

In this paper we have presented Reditus, a new forensic framework for forensic analysis of control logic injection attacks. Reditus can recover the control logic from then network dump by integrating the decompiler in the engineering software with the virtual PLC. Reditus is fully automatic, and does not require any explicit knowledge of the ICS protocol or underlying binary format. We evaluated `Reditus on` a Modicon M221 PLC and SoMachine Basic engineering software and successfully recovered 40 control logic programs from network traffic captures.

Acknowledgements

This work was supported, in part, by the Virginia Commonwealth Cyber Initiative (CCI) and ORAU Ralph E. Powe Junior Faculty Enhancement Award.

References

diflib [link]. URL. https://docs.python.org/3/library/difflib.html.

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

Ahmed, I., Roussev, V., Johnson, W., Senthivel, S., Sudhakaran, S., 2016. A SCADA system testbed for cybersecurity and forensic research and pedagogy. In: Proceedings of the 2nd Annual Industrial Control System Security Workshop. ICSS).

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

AllenBradlry. Product specifications. URL. https://ab.rockwellautomation.com/Programmable-Controllers/MicroLogix-1400#overview.

Chen, T.M., Abu-Nimeh, S., 2011. Lessons from stuxnet. Computer 44 (4), 91–93.

Falliere, N., Murchu, L.O., Chien, E.. W32.stuxnet dossier. URL. https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

Garcia, L., Brasser, F., Cintuglu, M., Sadeghi, A.R., Mohammed, O., Zonouz, S.A., 2017. Hey, my malware knows physics! attacking plcs with physical model aware rootkit. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, NDSS,2017. Internet Society, Reston, VA, USA. https://doi.org/10.14722/ndss.2017.23313. URL.

Garfinkel, S., Nelson, A.J., Young, J., 2012. A general strategy for differential forensic analysis. the Proceedings of the Twelfth Annual DFRWS Conference Digit. Invest. 9, S50–S59. https://doi.org/10.1016/j.diin.2012.05.003. URL. http://www.sciencedirect.com/science/article/pii/S174228761200028X.

Govil, N., Agrawal, A., Tippenhauer, N.O.. On ladder logic bombs in industrial control systemsdoi:2017arXiv170205241G. URL. http://adsabs.harvard.edu/abs/2017arXiv170205241G.

IEC. IEC 61131-3. https://www.sis.se/api/document/preview/562735/.

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).

Kush, N.S., Foo, E., Ahmed, E., Ahmed, I., Clark, A., 2011. Gap analysis of intrusion detection in smart grids. In: Valli, C. (Ed.), Proceedings of 2nd International Cyber Resilience Conference. secau-Security Research Centre, Australia, pp. 38–46.

Modicon. SoMachine basic operating guide. https://download.schneider-electric.com/files?p_File_Name=EIO0000001354.10.pdf.

Modicon. SoMachine basic - generic functions library guide. https://www.schneider-electric.com/en/download/document/EIO0000001474/.

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.

Reditus-Framework-Git-Repository [link]. URL. https://gitlab.com/sqasim1/reditus-framework.

Scapy. [link]. URL https://scapy.net/

Senthivel, S., Ahmed, I., Roussev, V., 2017. SCADA network forensics of the PCCC protocol. Digit. Invest. 22 (S), S57–S65.

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

Valentine, S., Farkas, C., 2011. Software security: application-level vulnerabilities in scada systems. In: 2011 IEEE International Conference on Information Reuse Integration, pp. 498–499. https://doi.org/10.1109/IRI.2011.6009603.

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.

Yoo, H., Kalle, S., Smith, J., Ahmed, I., 2019. Overshadow plc to detect remote control-logic injection attacks. In: Perdisci, R., Maurice, C., Giacinto, G., Almgren, M. (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment. Springer International Publishing, Cham, pp. 109–132.