2023

VIRTUAL PLC PLATFORM FOR SECURITY AND FORENSICS OF INDUSTRIAL CONTROL SYSTEMS

Syed Ali Qasim

VIRTUAL PLC PLATFORM FOR SECURITY AND FORENSICS OF INDUSTRIAL CONTROL SYSTEMS

This dissertation proposal document is submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

SYED ALI QASIM

Bachelor of Science, Lahore University of Management Sciences, Pakistan, 2016

Faculty Adviser: Irfan Ahmed,

Associate Professor, Department of Computer Science

Virginia Commonwealth University

Richmond, Virginia

July, 2023

Acknowledgements

Firstly, I would like to express my deepest gratitude to Allah (SWT) for His countless blessings and opportunities.

My profound appreciation goes to my advisor, Dr. Irfan Ahmed, for his unmatched support and guidance throughout my doctoral journey. I am grateful to Dr. Tamer Nadeem, Dr. Changqing Luo, Dr. Yanxiao Zhao, and Dr. Zhifang Wang for serving on my doctoral committee; their valuable time and insightful feedback have been indispensable.

I extend my thanks to all collaborators at other institutions and my colleagues for their help and support. I am especially thankful to my friends and colleagues at the SAFE lab who made this journey enjoyable and memorable.

Special gratitude goes to my family, who have always been my source of strength and inspiration. I wish to express my heartfelt gratitude to my mother and aunt for their significant sacrifices and tireless support toward my success. A special acknowledgment goes to my brother, whose mentorship has been a beacon throughout my life. Lastly, I am immensely grateful to my wife for her unwavering support and constant encouragement throughout this journey.

Finally, I want to express my deep gratitude to all the individuals who supported and motivated me, and who assisted me throughout this PhD program. I extend my best wishes and prayers for their well-being and success.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Abstract

VIRTUAL PLC PLATFORM FOR SECURITY AND FORENSICS OF INDUSTRIAL CONTROL SYSTEMS

By Syed Ali Qasim

A Virtual PLC Platform for Security and Forensics of Industrial Control Systems submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2023.

Faculty Adviser: Irfan Ahmed,
Associate Professor, Department of Computer Science

Industrial Control Systems (ICS) play a pivotal role in overseeing and regulating critical infrastructures such as nuclear power plants, gas pipelines, electric grid stations, and various commercial industries like manufacturing and packaging. Initially designed to operate in isolation, these systems, with the advent of the Industrial Internet of Things (IIoT), are now incorporated into broader networks for enhanced operational efficiency and economic advantages. This has inadvertently made them appealing targets for cyberattacks. At the heart of ICS are Programmable Logic Controllers (PLCs), key interfaces between the physical and cyber worlds. PLCs process sensor inputs based on user-defined programs, sending output to actuators to control physical processes. Given their significant role, they are often exploited by cyber attackers aiming to sabotage physical processes by manipulating control logic or exploiting PLC vulnerabilities. Therefore, there is a pressing need for tools to conduct forensic analysis of cyberattacks on PLCs, discover vulnerabilities before they can be

exploited by attackers, and gather reliable threat intelligence to bolster ICS security. The proprietary nature of the software and communication protocols used by various PLC vendors means that existing security and forensics tools rely heavily on manual reverse engineering, thus limiting their scalability and efficiency.

To address these limitations, I have developed a Virtual PLC Platform (VPP) capable of performing forensic analyses of various ICS attacks, gathering reliable threat intelligence, and identifying vulnerabilities in PLCs. The VPP, based on the packet replay technique, utilizes network traffic dumps captured during the real PLCs' communication as inputs to learn the PLC template. This template contains the locations and semantics of various fields in the PLC protocol. Upon generating the PLC template, the VPPinitiates a virtual PLC capable of communicating over the network and mimicking a real PLC. Additionally, the VPPincludes a Protocol Reverse Engineering Engine (PREE) module that can be used to reverse-engineer ICS protocols, contributing to the discovery of PLC vulnerabilities and improving the development of forensic and intrusion detection tools. The VPPis fully automated, eliminating the need for manual reverse engineering, and can mimic PLCs from different vendors. This dissertation presents the architecture of the VPPand its applications in forensic analysis of ICS attacks, reverse engineering of ICS protocols, PLC vulnerability discovery, and gathering reliable ICS threat intelligence.

CHAPTER 1

INTRODUCTION

Industrial Control Systems (ICS) are crucial for the monitoring and control of physical and mechanical processes across diverse industries. These systems govern everything from neighborhood traffic signals to high-stakes infrastructures like nuclear power plants, gas pipelines, and electrical power grid stations [1]. Comprising of field sites where actual physical processes are executed, and a control center for process monitoring and management, ICS utilizes Programmable Logic Controllers (PLCs), sensors, and actuators for controlling physical processes. The control center operates ICS services such as the Human-Machine Interface (HMI) and engineering workstations. Originally designed to operate in isolation, these systems are now, due to operational efficiency and economic advantages, connected to wider networks such as enterprise networks and the internet, consequently becoming targets for numerous cyberattacks.

PLCs are central components of an ICS. They receive input from various sensors, process this data according to a user-defined program, known as control logic, and govern the physical processes by sending output signals to the actuators. Due to their pivotal role, PLCs are often the primary target of cyberattacks aimed at sabotaging the physical processes [2, 3, 4, 5, 6, 7]. For instance, the infamous Stuxnet worm infects the control logic of a Siemens S7-300 PLC, modifying the motor speed of centrifuges periodically between 1,410 Hz, 2 Hz, and 1,064 Hz [8, 9]. Stuxnet compromises the Siemens SIMATIC STEP 7 engineering software at the control center and downloads malicious control logic to the PLC at field sites over the network. Beyond compromising the control logic, attackers can also exploit other vulnerabili-

ties in the PLCs. For instance, Ayub et al [**adeen**] identified and exploited various vulnerabilities in the authentication protocol of different PLCs. Similarly, it has been demonstrated that if an attacker possesses knowledge of the PLC communication protocol, they can craft malicious messages to target the PLC's operational state [10].

1.1 Motivation

1.1.1 Lack of Automated Forensic Tools

Network-based attacks on the programs (control-logic) running on PLCs can leave evidence within network traffic. This evidence, if captured, may contain traces of the transfer of malicious control logic. However, current research lacks forensic techniques that can efficiently extract this control logic from network traffic and translate it back to high-level source code for forensic analysis. There are some partial solutions like Laddis, a state-of-the-art forensic tool to recover control logic from ICS network traffic dump [4]. Laddis is essentially a binary control-logic decompiler for Allen-Bradley's RSLogix engineering software and MicroLogix 1400 PLC [11]. It leverages complete knowledge of the PCCC proprietary protocol to extract the control logic from the network traffic, further utilizing a low-level understanding of binary control-logic semantics for decompilation. However, Laddis necessitates tedious and time-consuming manual reverse engineering efforts to understand ICS proprietary network protocols and binary control logic semantics. Given the heterogeneous ICS environment where different PLC vendors employ their own proprietary protocols, software, compilers, etc., it is crucial to develop a fully automated forensic solution. Such a solution would be capable of recovering binary control logic from network dumps and converting it into a human-readable form for forensic analysis.

1.1.2 Vulnerability Detection

The frequency and sophistication of attacks on critical infrastructure are escalating year by year. According to reports [12, 13], cyberattacks on critical infrastructure saw a surge of 41% in the first half of 2021. Over time, these attacks have started to target various components of ICS, such as control-logic and PLC vulnerabilities. To defend our critical infrastructures effectively, the security community must outpace attackers in identifying and patching existing vulnerabilities in the ICS. Unfortunately, current solutions for vulnerability detection [14, 10, 15] heavily rely on manual reverse engineering and lack scalability. Thus, there is an urgent need for automated tools to aid the security community in discovering existing vulnerabilities in ICS.

1.1.3 Threat Intelligence

Reliable threat intelligence plays a vital role in helping security teams protect critical infrastructures. Having a thorough understanding of potential adversaries and their malicious behaviors beforehand contributes to improving security measures. However, existing solutions for ICS threat intelligence [16, 17] are developed through manual reverse engineering and are not scalable. This necessitates the development of automated tools that can help the security community gain insights into attacker behavior and gather reliable threat intelligence.

1.2 Objective

The purpose of this dissertation is to develop a Virtual PLC Platform (VPP) that can strengthen the security community's ability to perform forensic analysis of network-based attacks on PLCs, pinpoint existing vulnerabilities, and generate reliable threat intelligence. This overall objective can be further divided into the

following distinct goals:

1.2.1 Development of the Virtual PLC Platform

The Virtual PLC Platform is envisioned to intake network dumps stemming from communication between any real PLC and its associated software, subsequently learning the location and semantics of various protocol fields autonomously to generate a PLC template. The platform will host a virtual PLC that, leveraging the old network dumps and the PLC template, can replay the network dump, thus replicating the operations of an actual PLC. The salient feature of the virtual PLC platform is its fully automated operation, eliminating the need for laborious manual reverse engineering. Furthermore, the virtual PLC, using prior network dumps, will be able to imitate a real PLC and interact with PLC programming software effectively.

1.2.2 Utilization of the Virtual PLC Platform for Forensic Analysis

In cases where network-based attacks target the control-logic program running on the PLC, capturing the network traffic between the control center and field sites can yield evidence of the transfer of malicious control logic. The goal of this dissertation is to employ the Virtual PLC Platform for forensic analysis of such network-based attacks. In pursuit of this goal, the platform should have the capacity to reconstruct and transmute the binary control logic (present in the traffic dump) into its high-level source code.

1.2.3 Application of the Virtual PLC Platform for Vulnerability Discovery

Proficiency in protocol semantics can considerably assist in uncovering vulnerabilities in the PLC. Attackers can exploit PLC-specific features, such as programming

states or reverse-engineered authentication mechanisms. Therefore, another objective is to empower the virtual PLC platform to comprehend the semantics of proprietary PLC protocols, assisting in the timely identification of potential vulnerabilities.

1.2.4 Employment of the Virtual PLC Platform for Threat Intelligence

Once the virtual PLC platform has a firm grasp on the proprietary protocol, it can seamlessly communicate with any network entity. The virtual PLC could then serve as a honeypot, thereby transforming the Virtual PLC Platform into a valuable reservoir of threat intelligence. This will provide crucial insights into attack trends and attacker behaviors, thereby serving an essential role in the security community's efforts to stay ahead of threats.

1.3 Challenges in Developing Scalable Security and Forensic Tools

There exist several obstacles in realizing our aim of establishing a Virtual PLC Platform, primarily due to the proprietary nature of control-logic formats and ICS protocols. These challenges include:

- **Variability in Binary Control-Logic Formats:** Distinct PLC vendors have implemented their own proprietary compilers to convert the human-readable control-logic into a binary format readable by PLCs. Consequently, the binary control-logic lacks a standardized, open format akin to Linux's ELF and varies across different vendors.

- **Language Support in PLC Programming Software:** PLC programming software typically supports one or more languages as defined by the IEC 61131-3 standard. For example, RsLogix (programming software for Allen-Bradley MicroLogix 1400 and 1100) exclusively supports ladder logic, while SoMachine-

Basic (programming software for SchneiderElectric's Modicon M221 PLC) supports both ladder logic and instruction list. Therefore, for a precise forensic analysis, the binary control-logic needs to be translated back into its respective high-level language.

- **Proprietary Nature of ICS Protocols:** Proprietary ICS protocols serve as the communication medium between a PLC (at the field site) and its programming software. More often than not, these protocol specifications aren't publicly accessible. Even when an open protocol is in use, it usually encapsulates a proprietary layer. For instance, the Modicon-M221 PLC and SoMachine-Basic employ the open Modbus protocol, yet its 'data' field further comprises proprietary fields, such as a control-logic address in PLC memory, function code, and control logic content.

1.4 Contributions

With respect to the objectives mentioned earlier, this dissertation offers the following contributions:

- **Contribution 1: Design and Development of a Virtual PLC Platform (VPP) that Mimics Real PLCs**
  - I designed and developed a Virtual PLC Platform (VPP) that is capable of mimicking real PLCs. The VPP processes network dumps of actual PLC communication and generates a PLC template by learning the location and semantics of various PLC protocol fields. Subsequently, it initiates a virtual PLC capable of network communication similar to a real PLC. The VPP is fully automated, scalable, and has been successfully tested on various PLCs from different vendors and ICS protocols.

- **Contribution 2: Forensic Analysis of Network-based ICS Attacks Using the VPP**
  - I evaluated the forensic analysis capability of the VPP by using it to investigate a "Denial of Engineering Operations" attack. The VPP was successful in recovering the malicious control logic. Similarly, I used the VPP to recover the malicious control logic from the network dumps of a "Control Logic Injection" attack.
- **Contribution 3: Development of a Protocol Reverse Engineering Engine (PREE)**
  - I designed and built a Protocol Reverse Engineering Engine (PREE) adept at dissecting a wide range of Industrial Control Systems (ICS) protocols. This empowers control engineers to formulate heuristics for identifying fields across diverse ICS protocols. Through rigorous testing and evaluation across six different ICS protocols and five PLCs from four vendors, I demonstrated the versatility and effectiveness of PREE.
- **Contribution 4: Development of Control Logic Engine Attack**
  - I developed a new form of cyber-attack called "Control Engine Attack," which targets the control logic engine of Programmable Logic Controllers (PLCs). This innovative attack methodology manipulates inherent PLC features to disable the control logic engine, effectively halting operational processes. I successfully demonstrated the execution and effectiveness of this attack on five industry-standard PLCs, expanding the understanding of potential vulnerabilities in industrial control systems
- **Contribution 5: Demonstration of VPP's Honeypot Capabilities**

– I demonstrated the VPP's ability to act as a PLC honeypot, mimicking different PLCs. I also present a case study using a lab model of an elevator with VPP, performing various ICS attacks on it. This showcases VPP's capacity to engage with an attacker and store attack data.

1.5 Organization of the proposal

Chapter 2 explains the relevant industrial control systems (ICS) background. Chapter 3 explains the architecture of the virtual PLC platform (VPP). The application of VPP for the forensics analysis on network-based attacks on PLC is explained in chapter 4. Chapter 5 presents the Protocol Reverse Engineering Engine. Chapter 6 show the development of a novel, control engine attack on the PLCs. Chapter 7 presents another use case of the virtual PLC platform for threat intelligence and information gathering. Finally chapter 8 concludes the proposal.

CHAPTER 2

BACKGROUND

This chapter will cover the relevant background, section 2.1 provides a detailed picture of an industrial control system followed by PLC programming details in section 2.2.

2.1 Introduction to Industrial Control Systems (ICS)

Figure 1 depicts a typical industrial control system setup tailored for a gas pipeline. This architecture consists of two key components: the control center and the field sites.

Physical Process: In this gas pipeline instance, the physical process involves compressing the gas and channeling it to a remote receiver via a pipeline. The process includes an air compressor, storage and receiver tanks, and solenoid valves. The air compressor pressurizes the air and stores it in a designated storage tank, which is linked to a receiving tank via a pipeline. Solenoid valves seal the tanks and are opened to allow the pressurized air to flow to the receiving tank when necessary.

Field Site: The physical infrastructure of the gas pipeline, located at field sites, is supervised and regulated via a pressure transmitter, solenoid valves, and Programmable Logic Controllers (PLCs). The pressure transmitters, attached to the receiving and storage tanks, relay data to corresponding PLCs. The PLCs are embedded with a control logic to execute two primary functions: first, they open the valves to transport the compressed air to the receiving tank. Second, they continuously monitor the pressure within the tanks, maintaining an optimal level by releasing

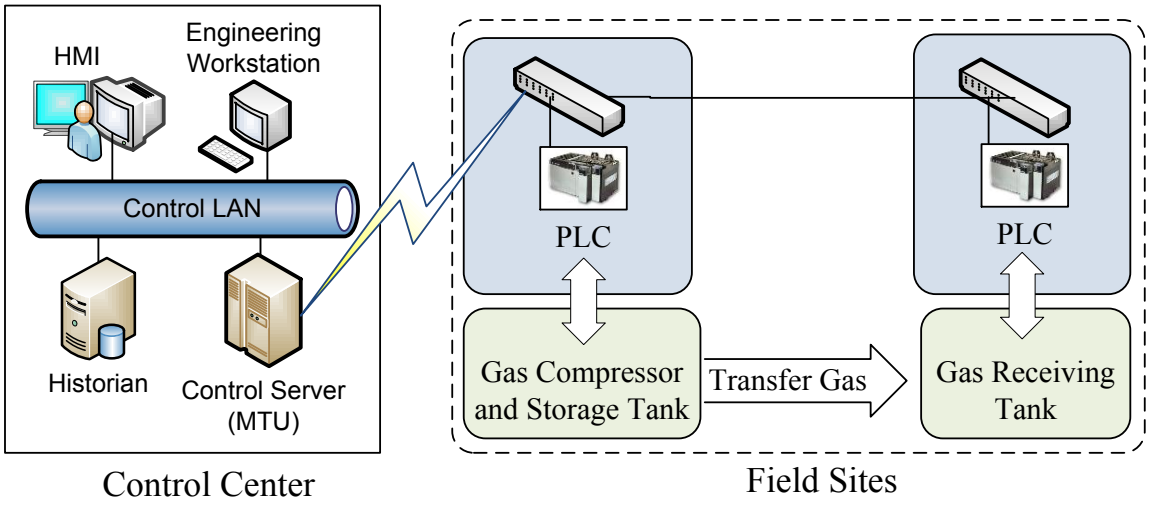


Fig. 1. A simplified representation of an industrial control system implemented for a gas pipeline scenario.

excess air when the pressure becomes too high.

Control Center: The PLCs transmit data to the control center, which includes a Human-Machine Interface (HMI), Historian, and an Engineering Workstation. The HMI provides a graphical display of the current state of the gas pipeline operation. The Historian, a database application, archives the PLC data for potential future processing. The Engineering Workstation employs engineering software for the remote programming, configuration, and maintenance of the PLCs.

2.2 PLC, Engineering Software and Control-Logic

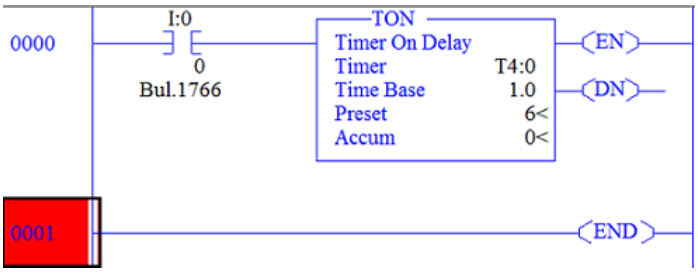
Programmable Logic Controller (PLC): PLCs are robust, embedded devices strategically located at field-sites to directly monitor and control physical processes. Each PLC features input and output modules: the input module connects to sensors, such as temperature and pressure sensors, while the output module interfaces with actuators to effect the desired process state. Internally, the PLC uses control logic to

process input data and determine the appropriate output. Furthermore, PLCs come with network communication capabilities, such as Ethernet or serial ports, enabling them to communicate with ICS services at the control center, like the engineering software.

Engineering Software: The engineering software is a vital tool for programming control-logic in PLCs. Typically proprietary and vendor-specific, these software solutions like SoMachineBasic, RsLogix 500, and CX-Programmer are instrumental in the configuration, programming, and remote maintenance of their respective PLCs.

PLC Communication: PLCs employ a variety of proprietary ICS protocols for communication, including but not limited to Modbus, EtherNet/IP (ENIP), and s7comm. Oftentimes, these PLCs utilize more than one protocol in conjunction, embedding one within another to facilitate communication with the engineering software. For instance, the Schneider Electric Modicon M221 PLC uses the UMAS protocol encapsulated within the Modbus protocol. Likewise, Allen-Bradley MicroLogix 1400 and 1100 PLCs encapsulate the PCCC protocol within the ENIP protocol. This protocol coupling enhances the communication capabilities and flexibility of PLCs within the ICS environment.

PLC Programming: The operation of a Programmable Logic Controller (PLC) is guided by a user-defined program known as control-logic. IEC 61131-3[18], the international standard for programmable controllers, prescribes five distinct languages to create control-logic. These languages are classified into two categories: Textual and Graphical. The Textual category includes 'Structured Text' and 'Instruction List', while the Graphical category encompasses 'Ladder Logic', 'Functional Block Diagram', and 'Sequential Function Chart'. For the purpose of this proposal, we



(a) Ladder Logic

(b) Instruction List

Fig. 2. Different representations of a timer program

have selected 'Ladder Logic' from the graphical languages and 'Instruction List' from the textual languages to illustrate their utility in PLC programming.

- ***Ladder Logic:*** Derived from Relay Logic, Ladder Logic is a graphical language utilized in PLC programming. It is characterized by a diagrammatic structure, with each horizontal line in the program referred to as a *rung*. These rungs are composed of a series of input and output instructions, each defining a specific operation to be performed by the processor [19].

  Figure 2(a) is a ladder logic program consisting of one rung and two instructions: 1) *XIC* (Examine if closed) on left is associated with the input address *I:0/0*, 2) *TON* (timer on delay) on right. The timer instruction has three attributes, i) time base (the unit of time, 1.0 means one second). ii) Preset (maximum time to wait). iii) Accumulator (the time that has passed). It also has two control

bits, *EN* (enable) and *DN*(Done).

When the program executes and the *XIC* is true, it will start the timer and *EN* will become true. The preset is 6 and the time-base is one second. When the timer completes 6 seconds, the *DN* bit turns to true and the accumulator is changed to the preset value.

- ***Instruction List:*** Unlike Ladder Logic, Instruction List resembles assembly language consisting of a sequence of instructions. Figure 2(b) shows an equivalent program in Figure 2(a). The first instruction *BLK* is the start of the timer function block. The second instruction, *LD* (load operator) looks for close edge contact, which is associated with the input *%I0.0*. The contact is closed when bit *%I0.0* is 1. The following instructions are as follows: *IN* represents the input of Timer function block; *Out_BLK* wires the output of timer; *Q* represents the output of timer, and it becomes 1 when the timer expires; *ST* is store operator, which is equivalent to a coil in ladder logic and takes the value of previous logic and is used to store output. Finally, *END_BLK* represents the end of the timer function block [20].

  When the program executes and *LD* is true, it sets *IN* true and starts the timer. The timer has a time-base of 1 second and preset of 6 second. When the timer completes 6 seconds, it sets *Q* (output of timer) true and then both *LD* and *Q* go into *ST*. *LD* and *Q* are in series. When both *LD* and *Q* are true, it will turn the output *ST* true.

Control-logic Transfer: Once a control logic program is composed in one of the high-level languages defined by IEC, a mechanism is required to transfer it to the Programmable Logic Controller (PLC). In this context, the engineering software

provides two essential functionalities - “Upload” and “Download” - which enable the transition of control-logic to and from the PLC.

- **Upload Function:** The ‘Upload’ functionality is a crucial tool enabling remote retrieval of the binary form of control logic from a PLC. This process initiates a decompiling procedure within the engineering software, which translates the binary data back into a high-level source code. As the control engineer triggers the ’Upload’ command, the engineering software facilitates a series of communication exchanges between itself and the PLC, involving session-establishment messages, echo messages, and control-logic messages.

  Initially, the engineering software establishes a connection session with the PLC. Subsequently, it transmits read-request messages targeted at accessing the memory locations of the control logic within the PLC. In reciprocation, the PLC returns the requested data, which constitutes the control logic, encapsulated in the payload of response messages. After the engineering software receives the complete binary control-logic, it directs it towards the decompiler to initiate the decompilation process, which consequently produces the source code in a high-level language format.

- **Download Function:** The ‘Download’ functionality is akin to the ‘Upload’ process, and it facilitates control engineers in transferring a newly crafted or modified control-logic to the PLC from the engineering software. On executing the ‘Download’ command, the engineering software once again establishes a session with the PLC. It then translates the high-level control-logic into a machine-readable binary format. Post this transformation, it dispatches write-request messages to the PLC, guiding it to inscribe the control-logic binary into its memory.

CHAPTER 3

DEVELOPING THE VIRTUAL PLC PLATFORM

This chapter presents the first significant milestone of my dissertation: the design and development of the Virtual PLC Platform (VPP). It demonstrates my deliberate effort to emulate the precise operations of a real Programmable Logic Controller (PLC). I begin by discussing the motivation that led to the inception of the VPP, illustrating its importance in the wider context of PLC systems. Then, I move on to explain the objectives I sought to achieve during the design and development stages. Overcoming the challenges encountered throughout this journey was not an easy task; I delve into these issues and the strategies implemented to navigate them. The evolution of the VPP has been driven by key insights gained from an in-depth understanding of real PLCs, which are also shared in this chapter. Lastly, I provide an in-depth evaluation of the VPP, confirming its capabilities and effectiveness in mimicking the functionalities of an actual PLC.

3.1 PLC Communication Insights

During the analysis of interactions between a real PLC and engineering software, I gleaned several key insights that proved instrumental in the development of the virtual PLC framework, VPP. These observations were critical in shaping the functionalities and performance of the VPPto accurately mimic a real-world PLC

Variety in Message Types During Communication: In a standard communication session between a PLC and the engineering software, post session establishment, a control engineer has the flexibility to perform three operations: monitor the

PLC state, upload the control-logic from the PLC, or download the control-logic to the PLC. Interestingly, the structure of messages for session establishment, PLC state observation, and control-logic upload remains consistent. However, the structure for the control-logic download message deviates from this. For instance, as illustrated in figure 4, the upload request from SoMachineBasic to Modicon M221 PLC solely includes the memory address and the number of bytes to read. Conversely, the download request extends beyond this to incorporate the control logic itself. This understanding proved critical in the design and development of `VPP`.

Deterministic Communication Behavior: Through my observation, I discerned a deterministic behavior exhibited by the communication process. Irrespective of circumstances, the engineering software consistently sends a limited set of unique requests to either establish a session with the PLC or to retrieve (upload) the control logic from the PLC. Notably, when we responded to a request message from the engineering software with a response message taken from a prior network dump, the engineering software's subsequent request message aligned exactly with the next request message in the network traffic dump. This deterministic communication pattern was a key insight in the development of `VPP`.

Control Logic Segmentation: The engineering software implements an interesting strategy while transferring control logic over the network—it partitions the binary control logic into multiple segments, or 'chunks'. In the context of the SchneiderElectric Modicon M221 PLC, the maximum size of a single chunk can be 236 bytes. During both upload and download operations, the engineering software consistently begins reading from and writing to a predetermined memory address. Furthermore, for both operations, the number of bytes read or written for each memory address

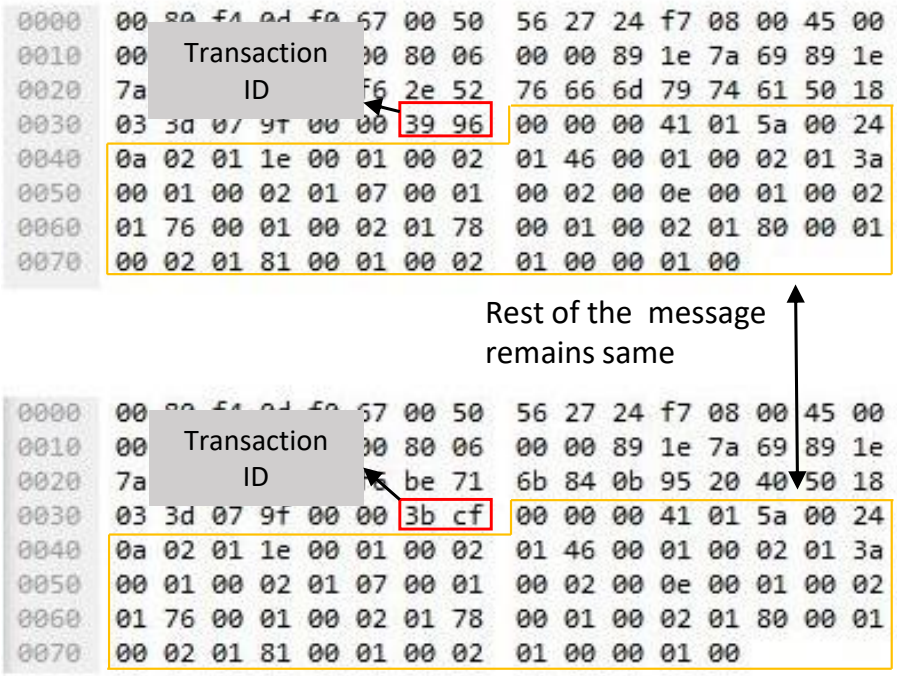


Fig. 3. Same request message from SoMachineBasic to Modicon M221 in two different session.

remains consistent. As seen in Figure 4, during the download request, the engineering software writes 43 bytes of control logic at the address "c404" in the PLC memory. The corresponding upload request replicates this, using the same address and byte count, thus defining the binary chunk. The advantage of this behavior is that `VPP` can utilize the downloaded network capture to reconstruct the control logic without requiring any binary or decompilation information, significantly streamlining the process

3.2 Virtual PLC Platform Design Goals

Building on the insights gained from analysing the communication between a real PLC and the engineering software, I aimed to develop a virtual PLC platform,

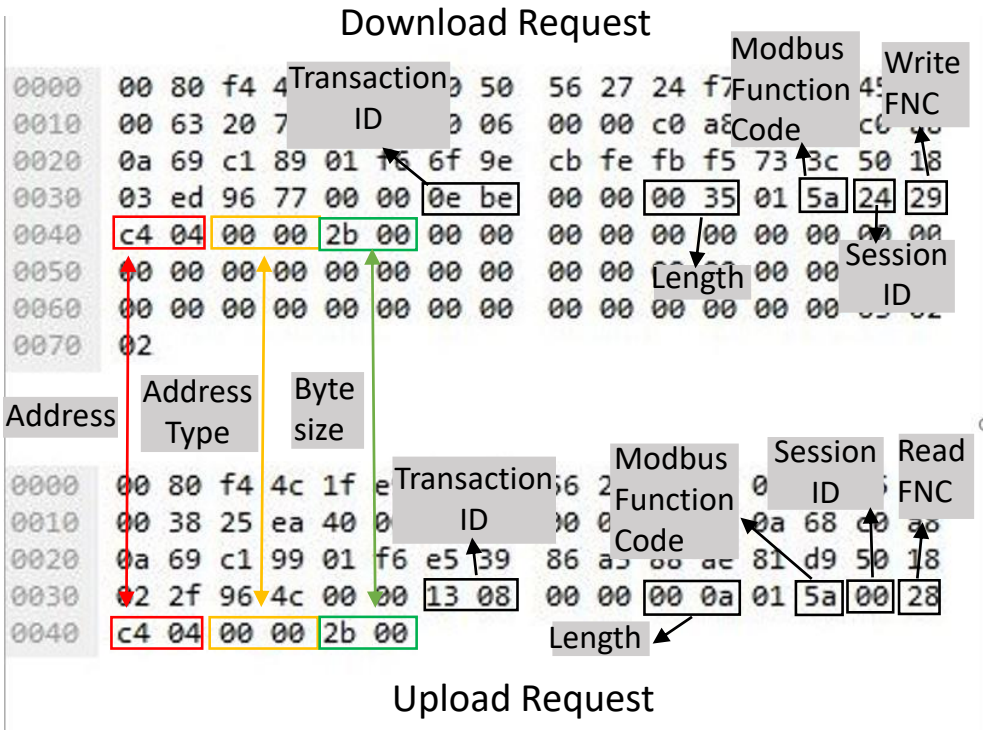


Fig. 4. Comparison of download and upload requests for the same address.

VPP, capable of accurately emulating the behavior of a real PLC. I sought to leverage the deterministic communication patterns, the different types of message structures, and the segmented nature of control logic transfers to create an effective and efficient virtual counterpart of a PLC. These objectives informed the primary design goals of VPP, which are as follows:

1. **Scalable Emulation:** The foremost aim was to create a scalable tool capable of emulating the behavior of a real PLC. This emulation extends to supporting the various operations a control engineer may perform, including observing the PLC state, and uploading or downloading control logic. VPPis designed to mimic these behaviors to create an authentic PLC experience.

2. **Protocol Agnostic:** An essential goal was to ensure VPP remained agnostic to the PLC protocol. This necessitated the tool to have the capacity to mimic PLCs from various vendors, even those utilizing distinct PLC protocols. In this way, VPP would transcend vendor-specific limitations, making it a more universally applicable tool.

3. **PLC Template Generation from Network Dumps:** I intended for VPP to have the ability to leverage network dumps from a real PLC, learn the message structure, location, and semantics of various fields in the ICS protocol, and consequently generate a PLC template. This function would harness the valuable information contained within network dumps to facilitate the generation of an accurate PLC template.

4. **Packet Replay Capability:** To ensure interactive networking, the design goal was to have VPP capable of interacting with any network entity using packet replay technique. This capability would allow for dynamic communication using previously captured network traffic, enhancing the virtual emulation's authenticity.

The design and development of VPP focused on accomplishing these goals, effectively translating the observations from real-world PLC communication into a virtual platform that can mimic a physical PLC with a high degree of accuracy.

3.3 Virtual PLC Platform

3.3.1 Overview

The VPPcan consist of one or more virtual PLCs. Given the network dump of a real PLC's communication, the objective of the virtual PLCs is to replay this net-

work dump, replicating the same network abstraction as a real PLC, and providing the application-level functionalities of a real PLC. This process necessitates three functions: (1) Data Processing: Using the network dump, the VPPshould be capable of organizing the packets to facilitate packet replay and template extraction; (2) Template Generation: The VPPshould learn message structure, various session-dependent fields, and their semantics to update them for new sessions (replay); and (3) Communication Server: In a manner similar to a physical PLC, the virtual PLC should include a communication server and be capable of responding to various request messages.

Figure 5 provides an overview of the virtual PLC platform. In the data management phase, the VPPextracts different sessions from the network dump, identifies request and response messages, and subsequently stores them in a database for further analysis. During the template generation block, the VPPperforms various analytical operations on the communication data stored in the database. This analysis aims to extract the message structure, location, and semantics of different fields in the messages to generate a PLC template. Finally, in the communication interface, the VPPuses the PLC template and the network dumps to instantiate various instances of virtual PLCs, thereby mimicking a real PLC.

3.4 Virtual PLC Platform Design

3.4.1 Data Processing

The initial step for the vPLC entails processing the network dumps, which are collected by monitoring the communication of a real PLC. PLCs utilize various proprietary ICS protocols, such as Modbus, S7comm, and ENIP, for communication. Additionally, they often incorporate other proprietary protocols like PCCC and UMAS,

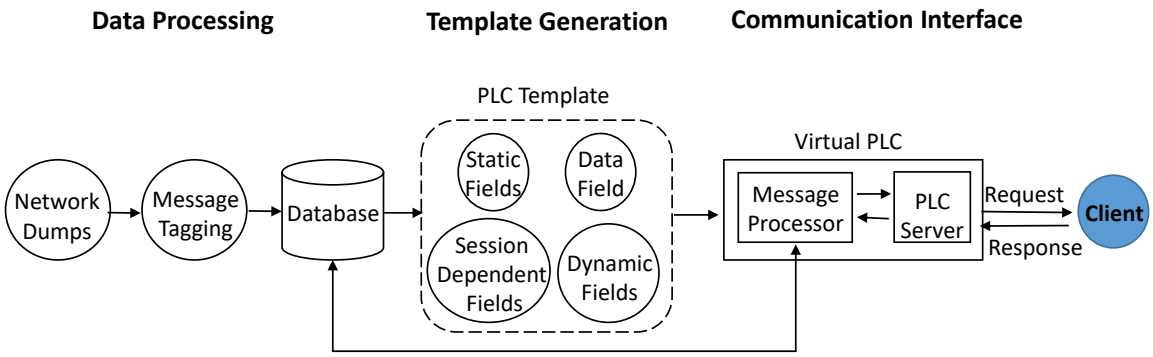


Fig. 5. An overview of virtual PLC platform

which are embedded within these primary protocols (ENIP, Modbus, S7comm, etc.). Given their binary nature, these protocols present an analytical challenge without prior understanding or specialized tools. Therefore, to extract actionable semantic information from network packet captures, a structured methodology is required. This process involves extracting different communication sessions, identifying request and response messages, and organizing them in a database to expedite the analysis process.

Message Tagging: The process of message tagging begins with identifying distinct communication sessions and recognizing the application-level request and response messages. Considering the IP address and port of the PLC, the vPLC can extract these separate communication sessions from the network dumps. The subsequent step involves identifying the request and response messages within these dumps. Since the maximum size of messages in ICS protocols, such as Modbus, is smaller than the maximum TCP/UDP payload size (for instance, Modbus messages are a maximum of 256 bits) add ref, we assume that each TCP/UDP packet contains one application-level request or response message. Given the nature of PLC operations, which act as servers (only responding upon receiving a request), packets whose destination IP

and port match the PLC's are tagged as request messages. Simultaneously, messages whose source IP and port align with the PLC's are tagged as response messages.

Database After messages have been tagged as requests and responses, the ensuing step is to organize them into an efficient and speedy database. Considering the proprietary nature of ICS protocols utilized by PLCs, we treat the application message as a complex binary structure, whose content and structure are not immediately decipherable to us. This approach enables us to remain agnostic of the ICS protocol as we store the messages as hex strings, assuming we have no information about their content.

To learn any semantic information and understand the message structure, it is crucial to pair the request and response messages together. However, during a communication session, network congestion could prevent the client from receiving a response from the PLC. This could lead the client to send another request before getting a response, which means the order of messages in the network dump cannot be relied upon to pair request and response messages.

To navigate this issue, we've developed a queue system for request messages. Each request message enters a queue and, when the vPLC identifies a response message, it retrieves the first message in the queue. It pairs this message with the response and stores this as a key-value pair in the database. This methodology ensures we maintain an organized and effective system for interpreting and learning from the communication data. Algorithm 1 outlines the process of creating databases.

3.4.2 PLC Template Generation

The `VPP`utilizes a packet replay technique to simulate a real PLC's behavior but replaying network traffic poses numerous challenges. Firstly, several session-dependent fields may exist within a message. The `VPP` needs to identify these and

Algorithm 1 Data Processing: Storing the request-response pairs

Require: $filename, src\_ip, plc\_ip, p\_port$

```
pcap ← rdpcap(filename)
table ← empty dictionary
req ← empty string
stack ← empty stack
for each pkt in pcap do
    if pkt contains TCP then
        hex_payload ← hexlify(payload of pkt)
        if source of pkt == src_ip and destination of pkt == plc_ip and length of
    payload ≥ 10 then
            push hex_payload to stack
        else if source of pkt == plc_ip and destination of pkt == src_ip then
            req ← pop first element from stack
            table[req] ← hex_payload
        end if
    end if
end for
```

establish their relationships in request-response messages. Secondly, according to [21], the message structure changes based on the operation performed, e.g., writing to the PLC memory yields a different request-response structure than reading data from it. Therefore, if the network dump provided to the `VPP` was captured during a control logic upload, it could reuse the message after updating the session-dependent fields. However, if captured during a download, the `VPP` needs to ascertain the upload message structure and use the downloaded traffic to populate and transmit the upload message. So `VPP` solves these challenges by and creates a template for each PLC it is mimicking.

3.4.2.1 Identifying Session Dependent Fields

A fundamental step in the development of `VPP`was the identification of session-dependent fields within PLC communications. To achieve this, I implemented a heuristic-based approach, comparing two benign Packet Capture (PCAP) files that contained the same control logic and were going in the same transfer direction, specifically "upload". Given the control logic in both PCAP files was identical, a comparison of the same message from both files would reveal almost identical data, barring session-dependent fields such as the session ID.

For instance, Figure 3 illustrates an example of the same message present in two separate PCAP files, derived from different sessions between a Modicon M221 PLC and its respective engineering software. The similarity between the two messages is striking, with only the Transaction ID varying between the two. This stark contrast highlights the session-dependent nature of the Transaction ID.

To identify these session-dependent fields, `VPP`ingests multiple sets of benign network captures in the form of PCAP files. Importantly, the learning process only

Fig. 6. An overview of identifying session-dependent fields in PLCs protocol

utilizes benign PCAP files, ensuring VPPlearns the correct message format, devoid of any potential anomalies or corruptions present in malicious files. The methodology employed for identifying session-dependent fields is visualized in Figure 6. The pseudo-code is summarized in Algorithm 2.

Pairing: In the initial phase, VPP processes multiple sets of benign PCAP files from different sessions, each of which containing identical control logic and transfer direction (either upload or download). For each request message *K1* in the first PCAP file, a corresponding request message *K2* in the second PCAP file is identified. This identification process relies on two parameters: the size of the message and the similarity of the message strings. VPP systematically compares every request message in the first PCAP file with all the request messages in the second file, and for each message, the framework autonomously detects the most similar message in the second PCAP file. Once these similar messages are identified, they are paired together as *(K1,K2)* for subsequent analysis.

Grouping And Analysis: Having obtained the message pairs, VPP undertakes a differential analysis of each pair, contrasting the two messages character by character.

Algorithm 2 Session Dependent Fields Identification

Let D_1, D_2 be two databases of network dumps
Let $Tuples$ be an empty list
Let $Indices$ be an empty dictionary
for each req_{1i} in D_1 **do**
    Find $match_{req}$ using *findMaxMatch(req_{1i}, D_2)*
    $Tuples$.append([req_{1i},$match_{req}$,res_{1i},$res_{match_{req}}$])
end for
for each T in $Tuples$ **do**
    for $index = 0$ to length(T[0]) **do**
        if $T[0][index] \neq T[1][index]$ **then**
            if $index$ in $Indices$ **then**
                $Indices[index] += 1$
            else
                $Indices[index] = 1$
            end if
        end if
    end for
end for
$threshold$ = tn * length($Tuples$)
$Indices$ = {index: count for index, count in $Indices$.items() if count ¿ $threshold$}
return $Indices$

For each pair, the indices where the messages differ are recorded, indicating the location of session-dependent fields. It's crucial to note that messages between the PLC and the engineering software can vary in length, and thus the location of session-dependent fields may also differ. As a result, `VPP` forms groups based on the length of the messages present in each pair to ensure that all messages within a group share the same format. This procedure is performed for all sets of PCAP files.

Rule Extraction: The final stage of the first learning phase involves examining each group to identify any noise or false positives, following the differential analysis on multiple sets of PCAP files and grouping the potential session-dependent fields. We hypothesize that for each message group, the location or index of session-dependent fields remains consistent. Therefore, at this stage, `VPP` only considers potential session-dependent fields that are consistently present in the majority of messages and discards all other fields that lack consistency. This process is repeated across all PCAP file sets, and the results are aggregated to generate a single set of session-dependent fields for each message group. Despite this method effectively removing noise, two challenges persist: firstly, there are no defined boundaries between fields. This means that if two fields in the protocol are adjacent, they will be considered as one. Secondly, if a session-dependent field is not entirely different across two PCAP files, `VPP` will not be able to extract the complete session-dependent field. For instance, in the message shown in Figure 3, the Transaction ID spans two bytes, from index 0 to 3. However, as the character "3" at index 0 is common in both messages, the differential analysis will identify an incomplete field comprising of three characters, i.e., from index 1 to 3.

Since protocol reverse engineering is not the primary objective of `VPP` , we can afford to overlook the first challenge, as our main goal is to identify and update

session-dependent fields so we can reuse the previously captured network traffic. Even if two session-dependent fields are treated as one, VPP will label this as a combined session-dependent field that needs updating during communication with the engineering software. The second challenge is addressed by the virtual PLC when it begins communication with the engineering software. Upon receiving a request message from the engineering software, the virtual PLC will compare the new request with a similar request in the database. This comparison could potentially generate some false positives. However, the virtual PLC can combine the results of this comparison with the information gathered by VPP to yield the final session-dependent fields. The virtual PLC takes the session-dependent fields learned by VPP as a baseline and only selects those fields from the comparison that are adjacent to, overlapping with, or confined within any of the baseline fields, effectively disregarding the rest.

3.4.2.2 Extracting the Message Structure

Depending on the operation, the structure of the request and response messages changes. When writing data to memory, the client sends a request message containing the write function code, memory address to write to, size of data to write, and the data or control logic itself. In response, the PLC sends a success message if the operation was successful, or an error message otherwise. To read data from the PLC memory, the client sends a request message containing the read function code, the memory address to read from, and the size of the data to read. The PLC then responds with a success message containing the requested data or an error message. Therefore, if the network dump the VPP is replaying contains read operations, the VPP can send the response after updating the session-dependant fields. However, if the replaying dump only contains write operations and the VPP is requested to read, the VPP must extract the structure of the read response message, fill it using

the messages in the dump, and then send it as a response to the read request. To extract the read response structure, the VPP processes two network dumps captured while reading and writing the same control logic. This allows the VPP to identify the different fields present in a read response message. Generally, there are four types of fields: static fields, dynamic fields, session-dependent fields, and control logic or data fields present in the upload response, which are also shown in Figure 7.

1. **Session-Dependent Fields:** These fields vary across different sessions (e.g., Transaction ID). Notably, the value of these fields does not depend on the content of the message but is rather dictated by the unique session they are part of.

2. **Static Fields:** These are the fields that remain consistent across all the upload response messages. Examples include constants such as the Modbus function code or success function code. Their value is independent of the session or the content of the message.

3. **Dynamic Fields:** These fields depend on the content of the message and, therefore, vary across different messages. An example of a dynamic field is the length of the message, which is dependent on the size of the control logic being transferred.

4. **Control Logic:** This part of the message consists of the actual control logic. Its size may vary across different messages, but based on our observations, it consistently follows the aforementioned fields in the message structure.

Identifying Static Fields: Algorithm 3 explains the process of identifying the locations of static fields, the VPPcompares all the request messages in a network dump and labels all the fields that remain constant throughout the session (i.e., fields that

are identical in all request messages) as static in the request. This is repeated for the response messages, and the locations of static fields in both request and response messages are compared to identify their relationships

Algorithm 3 Static Fields Identification

Let D be the database of network dumps
Let $StaticFields_{Req}$, $StaticFields_{Res}$ be empty lists
Initiate $index = 0$
while $index < length(D[0])$ **do**
    Let *value* be the value at *index* in the first request in D
    if all request messages in D have the same value at *index* **then**
        Append *index* to $StaticFields_{Req}$
    end if
    Increment *index*
end while
Repeat the same process for response messages and update $StaticFields_{Res}$
    return $StaticFields_{Req}$, $StaticFields_{Res}$

Identifying Dynamic fields: To locate dynamic fields such as the length of the request message, a heuristic-based approach is employed. A window of two bytes (a common size of length fields in ICS protocols) is rolled on a message and the value inside the window is compared with the length of the message outside the window. If they match, the location of the window is marked as the potential location of the length field. This process is carried out on each read response message in the database, and the location that appears in all request messages is labeled as the length field. Algorithm 4 gives the overview of the dynamic field identification process.

Algorithm 4 Dynamic Fields Identification

```
1: Let D be a database of network dump
2: Let PotentialLocations be an empty list
3: Let LengthFieldLocations be an empty list
4: for each msg in D do
5:     msgLength = length(msg)
6:     for index = 0 to msgLength − 2 do
7:         windowValue = convertToInteger(msg[index : index + 2])
8:         if windowValue = msgLength − index − 2 then
9:             PotentialLocations.append([index,index+2)
10:        end if
11:    end for
12: end for
13: for L in PotentialLocations do
14:    if PotentialLocations.count(L) = length(D) then
15:        LengthFieldLocations.append(L)
16:    end if
17: end for
        return LengthFieldLocations
```

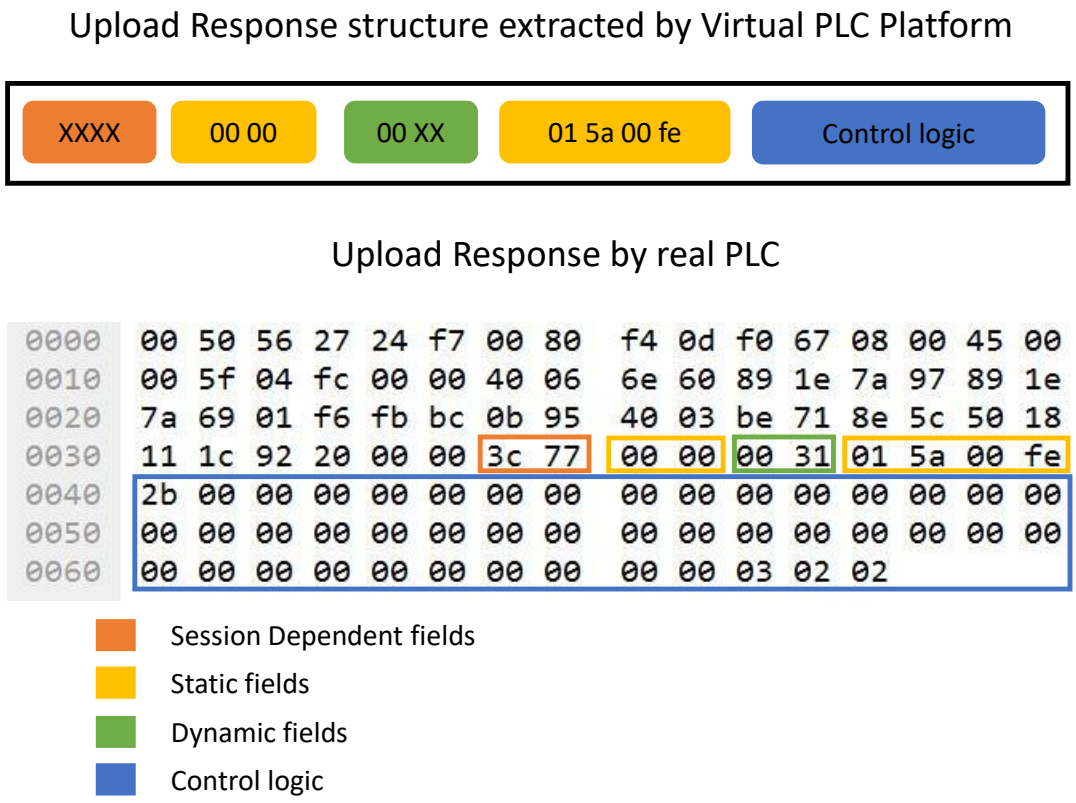


Fig. 7. Comparison of the upload response template generated by VPP and the upload response from a real PLC.

Identifying Control Logic fields: This process consists of two primary steps: pairing request-response messages from upload and download network dumps and identifying control logic.

Step 1:Pairing - In the pairing step, `VPP` pairs the request and response messages present in the upload traffic with the corresponding messages in the download. However, as shown in Figure 4, the length and format of the upload and download request messages differ. In the download request message, the engineering software sends a chunk of control logic to the PLC, accompanied by its size and memory address where the PLC should write the logic. In contrast, the upload request is much shorter, and

the engineering software only requests the PLC to send the control logic chunk present at the address specified in the message.

To pair the messages that read and write on the same physical address, we assume that the control logic will be in the latter part of the download request. Therefore, we use the similarity of the upload request and download request, equal to the length of the upload request, as a heuristic. `VPP` 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.

Step 2:Identifying Control Logic - The control logic is the most vital field for generating the upload template. As mentioned in section 3.1, the engineering software always divides the control logic into the same chunks for both download and upload. This implies that similar messages in the upload and download streams will contain the same control logic piece. The challenge then is to identify the location of control logic in the upload response and download request, enabling its use to create the upload template.

To tackle this, we utilize a heuristic-based approach grounded on the longest common sub-sequence (LCS) present in the download request and upload response. For each message tuple, `VPP` computes the LCS between the download request and upload response and notes the starting and ending index in both messages.

However, in some instances, the control logic part might be smaller than the message header. In such cases, `VPP` might learn an incorrect location for the control logic. To rectify this, after finding the location of the LCS in all download request and upload response message pairs, `VPP` identifies the starting and ending LCS indices that are common in most of the message pairs, using these indices for the final template.

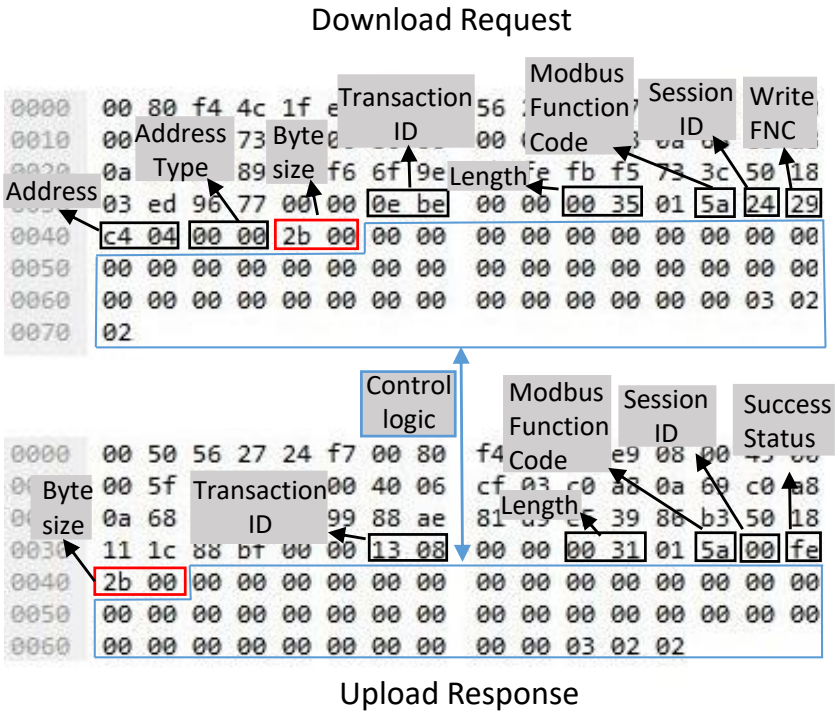


Fig. 8. Comparison of Modicon M221 download request and upload response for the same address.

Figure 8 shows the same control logic in upload response and download request from SoMachineBasic and Modicon M221 PLC.

After identifying the different types of fields, the VPPcompiles them to determine the structure of the read response message. In the end, all this information, including the message structure, the location of different fields in the message, and their relationships in request and response messages, is stored in the PLC template. This template can then be utilized by the VPPto accurately replay the network dump.

3.4.3 Communication Interface - Virtual PLC

Having organized the network dump in the database and generated the PLC template, the next and final component of the `VPP` is the communication interface. The user can instantiate hundreds of virtual PLC instances, providing them with the network dumps to replay (database) and the PLC template. Each virtual PLC mimics the behavior of a real PLC, offering a network abstraction of a real PLC.

PLC server: The virtual PLC consists of two main components: the PLC Server and the Message Processor. The virtual PLC operates a server (running on the same port as a real PLC), enabling communication with clients who can send different request messages to it. Upon receiving a request message, the PLC Server forwards it to the Message Processor, which is tasked with generating an appropriate response message.

Message Processor: Generating Response Messages: For each request message, the Message Processor searches the database for a similar request message, using the message size and string similarity as search parameters. In the first cycle, the Message Processor looks for all request messages in the database with the same length as the current request and finds the message with the highest string similarity. It retrieves the associated response, updates the session-dependent fields for the new session, and sends it to the PLC Server to respond to the client.

If the network dump populates the database and the current operation aligns, the Message Processor will likely find a similar, same-length request in the database. If not, the Message Processor won't find any requests of similar length in the database. In such cases, during the second round, the Message Processor calculates the similarity between the current request message and stored request messages up to the size of the current message.

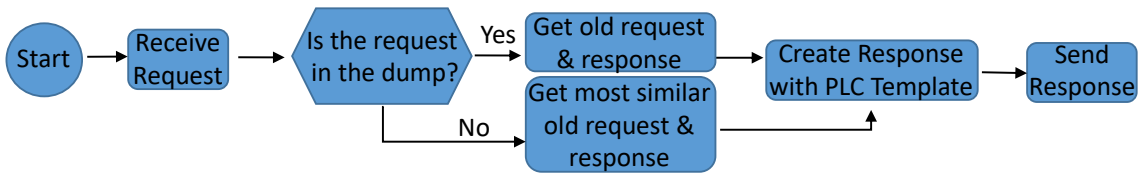


Fig. 9. Flowchart of virtual PLC communication

The Message Processor then selects the message with the highest similarity from the database, retrieves the associated response, and uses it to generate a new response message in line with the PLC template. This response is then sent to the PLC Server, which replies to the client. In this way, by responding to all incoming request messages, the virtual PLC effectively mimics the behavior and operations of a real PLC.

3.5 Evaluation

Upon completion of the Virtual PLC Platform (`VPP`) development, it's crucial to assess its functionalities and the assumptions made during the development process. The `VPP`'s main objective is to ingest network dumps, generate a PLC template, and replay the network dump via a virtual PLC, effectively mimicking a real PLC's behavior. Importantly, `VPP`is designed to replay both upload and download network dumps, so our evaluation considers both aspects. Accordingly, we gauge the performance of the virtual PLC against a variety of metrics.

3.5.1 Evaluation With Upload Network Dump

Lab Setup Our evaluation of the Virtual PLC Platform was conducted on three different PLCs: the Allen-Bradley MicroLogix 1400 Series B, the Allen Bradley MicroLogix 1100 Series B, and the Schneider Electric Modicon M221. For programming

the first two PLCs, we used the RSLogix 500 V9.2.01 software. The Modicon M221 was evaluated using SoMachine Basic v1.6 and v1.4. All the programming software were installed on a Windows 7 virtual machine (VM), while the virtual PLC was running on an Ubuntu v16.04 VM. To ensure a realistic networking environment, the engineering software, PLCs, and virtual PLC were all interconnected via an Ethernet network.

Experiment Methodology In a standard experiment, we first capture the network traffic when engineering software uploads a control logic from a real PLC. Next, the Virtual PLC Platform utilizes the generated pcap files to communicate with the engineering software in an attempt to recover the same control logic. Upon completion, the original and recovered control logic programs are compared manually within the engineering software to evaluate the accuracy and efficacy of the Virtual PLC Platform.

Dataset In order to evaluate the PLCs, we used a varied set of programs to mimic a diverse range of practical scenarios. For the Allen-Bradley MicroLogix 1400 and MicroLogix 1100, 39 and 22 different Ladder logic programs were used respectively. On the other hand, we utilized 52 Instruction List programs for the Modicon M221. These programs were crafted to represent different physical processes, including but not limited to traffic light control, hot water tank management, elevator control, gas pipeline regulation, and vending machines, demonstrating a broad spectrum of complexity and sizes. The details and features of the datasets for MicroLogix 1400 and 1100, and Modicon M221 are illustrated in Tables 1, 2 and 3 respectively.

Table 1. Dataset summary of Ladder logic programs for MicroLogix 1100

File Information		**Rung**				**Instruction**			
File size (KB)	**# of Files**	**Min**	**Max**	**Total**	**Avg.**	**Min**	**Max**	**Total**	**Avg**
0-40	16	2	17	90	5.62	3	48	240	15
41-60	1	4	4	4	4	12	12	12	12
61-80	4	8	63	145	36.25	25	245	543	135.75
81-100	1	13	13	13	13	37	37	37	37
Total	**22**	-	-	**252**	-	-	-	**832**	-

3.5.1.1 Virtual PLC as a Device

The virtual PLC in the VPP platform successfully establishes and maintains a connection with engineering software, mimicking the functionality of a real PLC. We conducted evaluations with two different engineering software, RSLogix and SoMachine Basic, concluding that both of these software recognized the virtual PLC as a genuine device, failing to distinguish it from a real PLC.

Figure 10 illustrates the results of the experiments where the virtual PLC is identified as a genuine MicroLogix 1100, MicroLogix 1400, and Modicon M221 PLC. The experiments were carried out in the following manner:

For connecting Allen-Bradley MicroLogix 1100 and 1400 PLCs to the engineering workstation, the user has to manually configure a driver within the RSLogix Classic. For Ethernet communication, the user can select either EtherNet/IP driver or Ethernet devices driver. When opting for the Ethernet devices driver, the user needs to input the IP address of the PLC device, while the EtherNet/IP driver automatically searches the subnet to discover the PLC devices. In our experiments, we configured

Table 2. Dataset summary of Ladder logic programs for MicroLogix 1400

File Information		**Rung**				**Instruction**			
File size (KB)	**# of Files**	**Min**	**Max**	**Total**	**Avg.**	**Min**	**Max**	**Total**	**Avg**
20-40	21	1	17	99	4.71	1	48	276	13.14
41-60	8	4	48	93	10.33	4	53	344	38.88
61-80	7	8	63	149	22.57	28	245	577	96.166
81-100	2	13	15	28	14	15	37	52	26
101-120	1	10	10	10	10	23	23	23	23
Total	**39**	-	-	**379**	-	-	-	**1272**	-

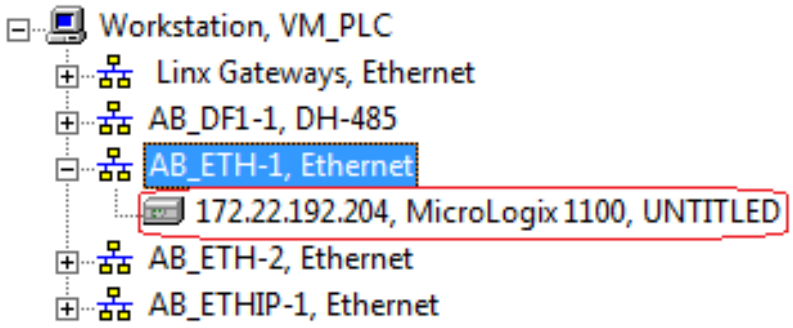
the Ethernet devices driver (AB_ETH-1), providing it with the IP address of the virtual PLC as shown in red circles in Figure 10(a) and 10(b). As a result, RSLinx Classic identified the virtual PLC as genuine MicroLogix 1100 and MicroLogix 1400 PLCs.

Similarly, with SoMachine Basic, the user can either input the IP address of the PLC or browse the subnet using the 'refresh devices' function (marked in the figure). In our experiment, we provided SoMachine Basic with the IP address of the virtual PLC. As seen in Figure 10(c), SoMachine Basic recognized the virtual PLC as a genuine PLC (TM221CE16R).

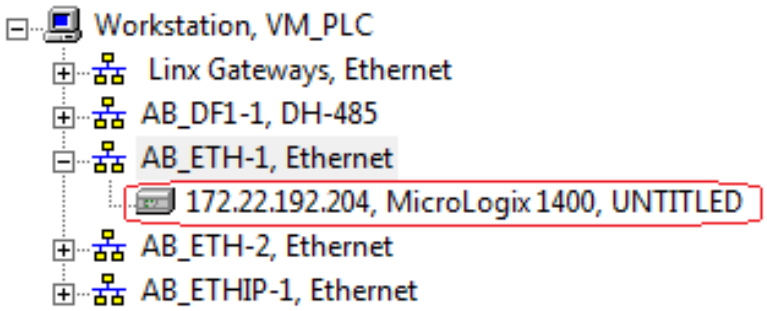
3.5.1.2 Function-Level Accuracy

Successfully mimicking a real PLC requires the virtual PLC to perform three tasks:

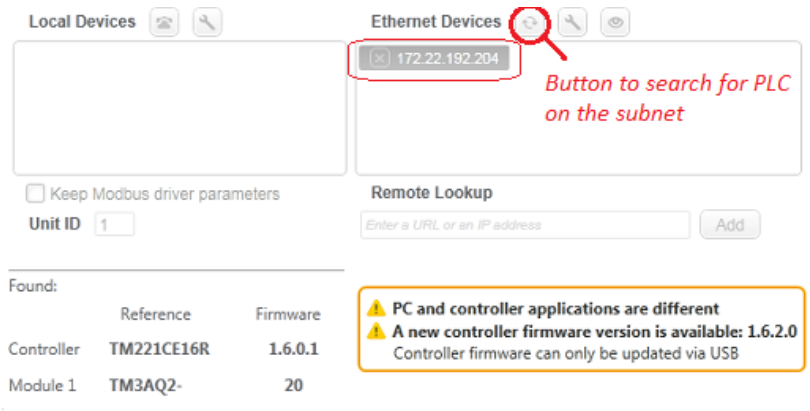
- i) Establish a connection with the engineering software



(a) Virtual PLC identified as real MicroLogix 1100



(b) Virtual PLC identified as real MicroLogix 1400



(c) Virtual PLC identified as real Modicon M221

Fig. 10. Virtual PLC identified as real PLC

Table 3. Dataset summary of Instruction List programs for Modicon M221

File Information		**Rung**				**Instruction**			
File size (KB)	**# of Files**	**Min**	**Max**	**Total**	**Avg.**	**Min**	**Max**	**Total**	**Avg**
60-80	30	1	3	72	2.4	2	23	793	26.4
80-100	14	2	27	107	7.64	7	112	463	33
100-130	4	8	14	43	10.75	20	72	153	38.2
130+	4	12	26	63	16	36	118	269	67.2
Total	**52**	-	-	**286**	-	-	-	**1678**	-

- ii) Handle non-control logic messages, such as echo
- iii) Upon receiving an 'upload' request from the engineering software, correctly upload the control logic (as found in the pcap file)

In this section, we assess the capability of the virtual PLC to establish and maintain a steady connection with the engineering software and upload the correct control logic to the engineering software, given the upload network dump.

Session Establishment and Maintenance Aside from transferring control logic, the engineering software also sends ping (echo) messages and other functional commands to the PLC. To test the robustness of the virtual PLC in establishing and maintaining the connection, we carried out the following experiment: Both RSLogix 500 and SoMachine Basic initiated a connection with the virtual PLC, keeping it open for several minutes without requesting an upload. During these experiments, the virtual PLC successfully maintained the connection in 113 cases.

Transfer Accuracy After successfully establishing and maintaining the session, the next task of the virtual PLC is to correctly upload a given control logic from the provided network dump. As outlined in Section 2.2, the 'upload' function of the engineering software sends a series of read requests to the PLC. Initially, the engineering software retrieves the control logic program's storage information or metadata from the PLC, and then it starts reading the control logic binary from the PLC memory. During the upload process, upon receiving a request message, the virtual PLC searches for a corresponding response message in its database and sends the reply after editing the dynamic or session-dependent fields.

At this stage, any changes other than the dynamic fields can disrupt the connection between the engineering software and PLC or compromise the integrity of the control logic. Our experiments show that the virtual PLC successfully identified and edited the dynamic fields while preserving the integrity of the control logic being uploaded. Furthermore, we analyzed the virtual PLC's ability to reverse-engineer the ICS proprietary protocols.

To evaluate the virtual PLC's accuracy, we manually calculated the number of rungs and instructions in each of the 113 control logic files and transferred them one by one to the engineering software using the virtual PLC. After each upload, we compared the uploaded program with the original files to verify if the number of rungs and instructions were the same.

To further verify the integrity of the control logic transferred by the virtual PLC, we manually compared the order of the instructions in the original control logic and the control logic transferred by the virtual PLC on the rung. Additionally, we compared the values of other variables, such as timer presets and timer bases, with those in the original program.

MicroLogix 1400: For the Allen-Bradley MicroLogix 1400 PLC, we uploaded

Table 4. Transfer accuracy of the virtual PLC

PLC	# of control logic files uploaded	Original Program		Virtual PLC output		Accuracy %
		Rungs	Instructions	Rungs	Instructions	
MircoLogix 1100	22	252	832	252	832	100%
MicroLogix 1400	39	379	1272	379	1272	100%
Modicon M221	52	286	1678	286	1678	100%

39 ladder-logic programs consisting of 379 rungs and 1,272 instructions. The virtual PLC demonstrated 100% accuracy in establishing a connection, basic communication, and control logic upload. Furthermore, in all instances, the original programs and those uploaded by the virtual PLC were identical. The transfer accuracy of the virtual PLC is detailed in Table 4.

MicroLogix 1100: For the accurate evaluation of the virtual PLC with MicroLogix 1100, we utilized 22 ladder-logic programs of varying complexity containing 252 rungs and 832 instructions. The virtual PLC was able to upload all the programs with 100% transfer accuracy.

Modicon M221: For the Modicon M221, we used 52 different programs in Instruction-list format, comprising 286 rungs and 1,678 instructions. These programs varied in terms of complexity, with some as simple as one rung and two instructions per program, to more complex ones with over 20 rungs and 100+ instructions per program. During our experiments, the virtual PLC demonstrated a 100% accuracy rate in uploading the control logic from the pcap files.

3.5.1.3 Packet-Level Accuracy

A primary heuristic in developing the virtual PLC is the deterministic behavior of engineering software. The engineering software employs the same set of messages to initiate a connection or request an upload. Hence, considering the deterministic behavior of the engineering software, if the virtual PLC possesses a complete network traffic log from a previous session, it can utilize this to communicate with the engineering software, with a high probability that all request-messages from the engineering software can be found in the network traffic log. Our experimental results corroborate this theory.

This section evaluates the virtual PLC's ability to identify a given request-message within the database (i.e., the target pcap file). Table 5 presents the packet-level accuracy results of the virtual PLC. During the upload of 52 control logic programs as Modicon M221, the virtual PLC received 8800 request messages from the engineering software. Out of these, 8776 messages, matching in length and with an average similarity of 99.99%, were present in the database. For the remaining 24 messages, the virtual PLC selected the request message with the closest length. Although the average similarity of the messages selected by the virtual PLC, in this case, is 0.58%, it's worth noting that the engineering software accepted the response message from the virtual PLC without causing crashes or errors, and the overall communication behavior remained unchanged. Similarly, for MicroLogix 1400, while uploading 39 control logic files, the virtual PLC received 4219 messages, all of which were present in the database with an average similarity of 100%. For MicroLogix 1100, during the upload process, the virtual PLC received 1639 messages, all of which were present in the database (i.e., target pcap files) with 100% accuracy.

Table 5. Packet-level accuracy of the virtual PLC

PLC	No. of files	Request messages received	Request messages present in DB		Request messages not present in DB	
			No.	Avg. Similarity %	No.	Avg. Similarity %
MicroLogix 1100	22	1639	1639	100%	0	-
MicroLogix 1400	39	4219	4219	100%	0	-
Modicon M221	52	8800	8776	99.99%	24	56.39%

3.5.2 Evaluation With Download Network Dump

3.5.2.1 Experimental Setting

Lab Setup: We evaluated the capability of the virtual PLC to handle a download network dump using a Schneider Electric Modicon M221 PLC and SoMachineBasic V1.6 SP2. The engineering software was installed on a Windows 7 virtual machine, while the virtual PLC was operating on an Ubuntu 18.04.3 LTS machine. All three devices were connected and on the same network subnet.

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 6 shows the summary of our dataset.

Experiment Methodology: A typical experiment includes capturing the network traffic when an engineering software downloads a control logic to a real PLC in the form of PCAP file. This file is then provided to the virtual PLC, and then a connection is established from the engineering software to the virtual PLC and the

Table 6. 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

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 the virtual PLC is manually compared with the original control logic file to find the transfer accuracy of the virtual PLC.

3.5.2.2 Functional-level Accuracy

In this section, we evaluate the functionality of the virtual PLC. The two most critical requirements for the virtual PLC when handling a download dump are:

- The virtual PLC should be able to match the download request messages from one PCAP with the corresponding upload request message in the other PCAP file, and
- The virtual PLC should be capable of generating an upload to produce the upload response message using download network traffic.

Matching Accuracy: When the virtual PLC receives any upload request message, it must find the matching download request from the database, edit the cor-

Table 7. 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	-

responding response message, and send this response to the engineering software. If the response message differs from what the engineering software anticipates, the engineering software will terminate the communication with an error. As shown in Figure 4, the upload and download request messages have different formats, making it a non-trivial task to find the exact match. In our experiments, we found that the similarity and length-based matching approach used by the virtual PLC to match 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 7 summarizes the database look-ups during our experiments. While uploading the 40 control logic files, the virtual PLC received 1,929 read request (upload) messages from the engineering software and was able to successfully find all corresponding write request (download) messages. For evaluation purposes, we compared the address, address type, and control logic size field of the two messages.

Upload Response Structure Accuracy: The second most crucial function of the virtual PLC is to learn the structure of the upload response from the sample PCAP files and use it to generate upload response messages from the target PCAP file. To evaluate the accuracy of the upload template, we manually compared the template with the upload response message from a real PLC to verify if:

- Our template includes all four types of fields (Session-Dependent, Static, Dynamic, and Control Logic).
- The location in the upload template for each field precisely matches the corresponding location in the upload response message from the real PLC.

During our experiments, we found that the virtual PLC was once again able to generate the correct upload template with *100% accuracy* across the entire dataset. Figure 7 and Table 8 demonstrate that the upload template generated by the virtual PLC contains all the required fields and their locations/indices are identical to those found in a response message from the real PLC.

3.5.2.3 Packet-Level Accuracy

One of the main assumptions underpinning the development of the virtual PLC is that during upload and download processes, the engineering software reads and writes control logic on the PLC. If the virtual PLC only has access to download network traffic, it can still locate all the control logic that was written on the PLC and relay it to the engineering software using an upload template. Specifically, for every read request message from the engineering software, there is a corresponding write request message in the target PCAP.

Table 9 provides a summary of the control logic read and write messages received during our experiments. During the transfer of 40 different control logic files,

Table 8. Comparison of the location of different fields in an actual PLC response and a template generated by the virtual PLC

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%

VPP received 1,852 unique read request messages, of which 1,812 had corresponding messages in the database. Upon examining the missing messages, we found that every download PCAP file was missing the same message, which was related to the functional-level.

While our assumption was not 100% accurate, only one out of over 1,800 messages was missing. This issue can be resolved by maintaining a separate database of missing messages and integrating it with the virtual PLC. If a message is not present in the target PCAP (download) file, the virtual PLC can look for the missing message in the second database.

3.5.2.4 Transfer Accuracy

The most crucial metric for evaluating the virtual PLC is the integrity of the control logic transferred by it. If the virtual PLC introduces any changes in the control logic during the upload process, it cannot serve as an effective forensic tool.

Table 9. Summary of control logic read and write messages during the experiments

File Size (kb)	# of Files	Unique Read Messages in Upload	Unique Write Messages 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	-

To determine the transfer accuracy, we uploaded 40 different control logic programs of varying complexities and sizes using the virtual PLC. We then manually compared each program with its original counterpart. Our comparison was comprehensive, encompassing not only the number of rungs and instructions, but also ensuring that each rung and instruction was identical in both versions.

Table 10 provides a summary of the control logic programs uploaded by the virtual PLC. Notably, the virtual PLC was able to successfully upload 40 control logic programs containing 213 rungs and 888 instructions, achieving a transfer accuracy of 100%.

3.6 Conclusion

In this study, we have presented the design and development of a scalable and automated platform for virtual Programmable Logic Controllers (PLCs). The developed system, which effectively mimics the behavior of real PLCs, was extensively tested

on a variety of PLCs from multiple vendors, including Allen-Bradley and Schneider Electric, among others.

The comprehensive evaluations and functional tests performed on our virtual PLC have demonstrated its robustness and capability to accurately mirror real PLCs in critical aspects such as session establishment and maintenance, control logic upload, and packet-level communication accuracy. Moreover, the platform demonstrated an impressive control logic transfer accuracy, maintaining the integrity of the control logic being uploaded.

One of the significant advantages of our system is its scalability. The current design is not limited to the tested PLCs but provides a framework that can be effectively applied to other PLCs, facilitating their representation in a virtual environment. This broadens the scope for testing, evaluation, and optimization efforts, and allows for large-scale and automated PLC interaction scenarios that are typically challenging in physical setups.

This work is a significant step forward in the field of industrial control system security. The developed virtual PLC provides an effective solution for forensic analysis, mimicking the behavior of real PLCs without the requirement of proprietary knowledge about vendor-specific protocols. It holds potential to be a valuable tool for security practitioners and researchers, offering a practical and accessible means to experiment, study, and enhance the security of PLC-based control systems.

As we continue to build upon this work, we aim to further refine our system to increase the range of PLCs it can emulate and enhance its capabilities to handle more complex interaction scenarios, thus contributing to advancements in the cyber-physical system security landscape.

Table 10. Comparison of control logic uploaded by the virtual PLC & real M221 PLC

File Size (kb)	# of Files	M221 PLC		Virtual PLC		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	-

CHAPTER 4

FORENSIC ANALYSIS OF ICS ATTACKS USING VIRTUAL PLC

In this chapter, we present our second contribution, which is the application of our developed virtual PLC platform for forensic analysis of cyber attacks on Industrial Control Systems (ICS). We focus specifically on network-based attacks on PLCs, under the assumption that if the network traffic between the control center and field sites is captured, this traffic would contain evidence of the transfer of the malicious control logic.

As explained earlier, the engineering software can remotely read a control logic binary from a PLC, referred to as the upload function. Additionally, it has a built-in decompiler that can further transform the binary control logic into a human-readable high-level representation. Our core idea is to integrate this decompiler with the previously-captured network traffic of a control logic using the upload function to recover the source code of the binary control logic automatically.

Therefore, for forensic analysis of any network-based attacks, the user can utilize the virtual PLC platform and provide the forensic artifact (network dump) to the virtual PLC. Then, using the engineering software, the user can request the virtual PLC to upload the control logic present in the network dump. This approach leverages the strengths of our virtual PLC system and the inherent capabilities of the engineering software, offering a practical and efficient method for forensic analysis of cyber attacks on ICS.

4.1 Introduction

Industrial control systems (ICS) oversee and manage industrial physical processes, such as nuclear plants, electrical power grids, and gas pipelines [22]. An ICS is composed of a control center and several field sites. The control center operates ICS services like the human-machine interface (HMI) and engineering workstation. Meanwhile, the field sites utilize programmable logic controllers (PLCs), sensors, and actuators to manage the physical processes.

PLCs are the principal targets of cyberattacks designed to sabotage physical processes [2, 23, 4, 5, 6, 7]. They run on a control logic that dictates how a physical process should be managed. Attackers can manipulate this control logic over the network, changing the behavior of the physical process, a method referred to as a *control-logic injection attack* [24, 25, 26, 27]. For example, the infamous Stuxnet worm infects the control logic of a Siemens S7-300 PLC to periodically modify the motor speed of centrifuges, ranging from 1,410 Hz to 2 Hz to 1,064 Hz [9, 8]. Stuxnet can compromise the Siemens SIMATIC STEP 7 engineering software at the control center and inject malicious control logic into the PLC at the field sites over the network. If the network traffic between the control center and the field sites is captured during such an attack, the traffic data will contain evidence of the malicious control logic transfer. However, the current state of research lacks robust forensic techniques that can extract the control logic from the network traffic dump and further transform it back into high-level source code for forensic analysis.

Some partial solutions exist, such as Laddis, a state-of-the-art forensic solution that recovers control logic from an ICS network traffic dump [4]. Laddis is essentially a binary control-logic decompiler for Allen-Bradley's RSLogix engineering software and MicroLogix 1400 PLC [11]. It utilizes a complete understanding of the PCCC

proprietary protocol to extract the control logic from the network traffic and uses a low-level comprehension of binary control-logic semantics for decompilation. Unfortunately, Laddis requires extensive manual reverse engineering efforts, making it a tedious and time-consuming process.

Similo is another forensic solution that addresses some of the shortcomings of Laddis, including the need for manual reverse engineering [28]. However, Similo is designed to investigate control logic theft attacks where the attacker reads the control logic from a PLC over the network. It does 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 these limitations, we propose the use of a virtual PLC platform `VPP`for forensic analysis of network-based cyber attacks on PLCs. `VPP`is capable of replaying network traffic and extracting control logic information, making it an ideal tool for this purpose. Given that the engineering software possesses a built-in decompiler for transforming the control logic binary into a human-readable form, we seek to harness this functionality in conjunction with our virtual PLC. Specifically, we aim to use `VPP`to replay the network dump captured during a cyber attack and subsequently utilize the upload functionality of the engineering software to decompile the control logic from the binary. This integrated approach promises a comprehensive, automated, and efficient solution for forensic analysis of cyber attacks on ICS, particularly focusing on control logic injection attacks.

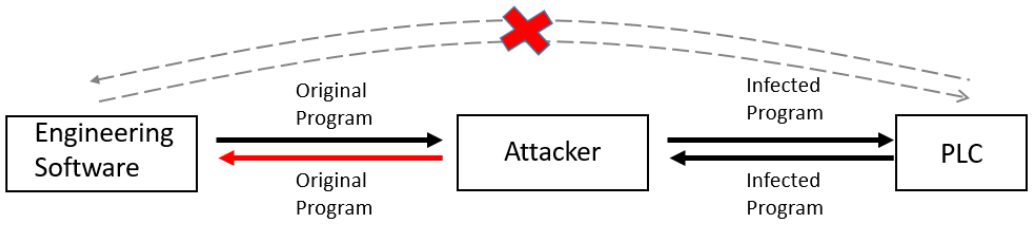


Fig. 11. DEO Attack I: Hiding infected ladder logic from the engineering software

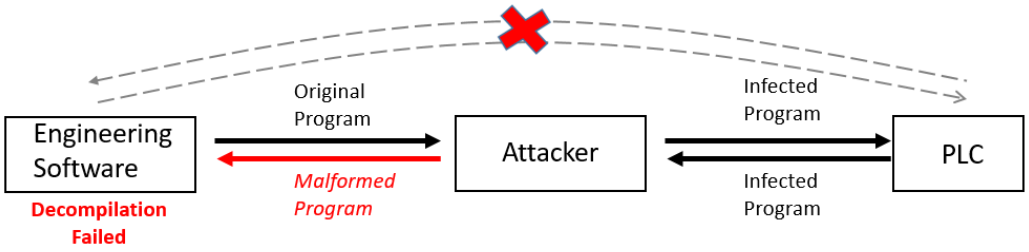


Fig. 12. DEO Attack II: Crashing the decompiler running on Engineering software.

4.2 Network Based Attack on PLCs

4.2.1 Denial of Engineering Operations (DEO) Attack

Recently, Senthivel *et al.* [4] presented denial of engineering operation (DEO) attacks that jeopardize an engineering software's capabilities to perform remote maintenance on a PLC. They demonstrate the attacks on Allen-Bradley MicroLogix 1400-B and RSlogix 500 (engineering software).

DEO Attack I: In DEO I (Figure 11), an attacker performs a man-in-the-middle attack between a target PLC and an engineering workstation (the computer running an engineering software). When the control engineer downloads a control logic program to a compromised PLC, the attacker intercepts the communications and infects this control logic by replacing some part of the code with malicious logic before forwarding it to the PLC. Similarly, when the control engineer tries to upload the control

logic from the PLC, the attacker intercepts the traffic and replaces the infected logic with the original code. In this way, the control engineer remains unaware of the malicious control logic running on the PLC.

Consider the ladder logic program in Figure 2(a), where the timer controls the yellow light in a traffic light signal. The attacker modifies the preset value from 6 seconds to 80 seconds when the program is downloaded to the PLC of the signal. When a control engineer attempts to retrieve the program from the PLC, the attacker intercepts the traffic and changes the preset back to its original value, i.e., 6.

DEO Attack II DEO II is similar to DEO I in that the attacker performs a man-in-the-middle attack between the engineering workstation and PLC, intercepts the communication, and manipulates the traffic as it passes through the attacker's machine. However, in DEO II (Figure 12), the attacker replaces the original code with random (noise) data such as 0xFFFF. When an engineering software receives the malformed logic, it fails to decompile.

4.2.2 Control-Logic Injection Attacks

Control-logic injection attacks represent another critical category of threats to industrial control systems. In these assaults, the attacker downloads malicious control logic to the PLC, potentially disrupting the PLC operation or causing system damage. Stuxnet [9] is a notorious example of control logic injection attacks. In Stuxnet, the attacker first compromises the Siemens SIMATIC STEP 7 software and then targets the S7-300 PLC. The attacker downloads a malicious control logic to the PLC that periodically alters the motor speed of centrifuges from 1410 Hz to 1064 Hz, causing physical damage to the centrifuge. Other examples of control logic injection attacks are as follows.

Kalle *et al.* [29] present CLIK, a control logic infection attack, consisting of four phases. First, it compromises PLC security measures and pilfers the control logic from it. Then, it decompiles the stolen binary of the control logic to inject malicious logic, followed by transferring the infected binary back to the PLC. Finally, it obscures the malicious logic written into the PLC from the engineering software by employing a virtual PLC. This virtual PLC initially captures the network traffic of the original logic, then sends this network traffic to the engineering software when it attempts to read the control logic written inside the PLC.

Yoo *et al.* [24] present two control logic injection attacks, namely 1) data execution and 2) fragmentation and noise padding. In the data execution attack, the attacker exploits the PLC's lack of data execution prevention (DEP) enforcement, transferring the attacker's control logic to the data blocks of the PLC. The attacker then alters the PLC's system control flow to execute the logic located in data blocks. The fragmentation and noise padding attack bypasses deep packet inspection by sending write requests containing the attacker's control logic. Each write request contains one byte of the control logic, while the rest of the packet contains noise. For every subsequent write request, the attacker tries to overwrite the PLC memory region previously filled with noise due to the prior request.

Govil *et al.* [26] introduced a malware written in ladder logic called "ladder logic bomb" that an attacker can insert into the existing control logic of a PLC. These logic bombs are challenging to detect by a control engineer manually verifying the control logic running on the PLC. These bombs can be activated via trigger signals to cause disruption or can persistently damage physical operations over time.

4.3 Problem Statement and Challenges

4.3.1 Problem Statement

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

4.3.2 Challenges in Control-logic Forensics

There are several challenges in achieving our stated goal of control logic forensics due to the proprietary nature of control logic formats and ICS protocols.

- Binary control-logic does not have a standard open format, like Linux ELF, and exists in a vendor-specific proprietary format.

- Engineering software typically supports one or multiple languages defined by the IEC 61131-3 standard. For instance, RsLogix only supports ladder logic, while SoMachine-Basic supports both ladder logic and instruction list. The binary control-logic must be transformed into its respective high-level language.

- Proprietary ICS protocols are employed to transfer a control-logic to a PLC from engineering software. Their specifications are not publicly available. If an open protocol is used, it encapsulates a proprietary layer. For instance, the Modicon-M221 PLC and SoMachine-Basic use the open Modbus protocol. However, its *data* field further contains proprietary fields such as the control-logic address in the PLC memory, function code, and control logic content.

4.4 Forensic Analysis of Attacks Using Virtual PLC Platform

In order to forensically analyze network-based cyber attacks on PLCs, we propose a solution leveraging a virtual PLC platform. The aim is to replay the captured communication between the attacker and the target PLC during the attack, extract the binary control logic, and convert it into a human-readable format. As engineering software typically has a built-in decompiler for converting binary control logic into a readable form, it forms a crucial part of this solution. Our proposed process begins by initializing a virtual PLC on the virtual PLC platform (`VPP`), followed by feeding it with the network traffic dump captured during the attack. Once this setup is ready, we connect this virtual PLC with the engineering software. Using the upload function of the engineering software, we retrieve the control logic that was run on the PLC. This extracted control logic can then be forensically analyzed to understand the specifics of the attack and devise countermeasures.

4.4.1 Forensic Analysis of Denial of Engineering Operations Attacks

4.4.1.1 DEO I

Attack Execution: The man-in-the-middle attack was executed using ARP poisoning through Ettercap. The targeted program for this attack was designed to control a traffic light system, comprising of three timers, each assigned to a different signal light: red, orange, and green. The main aim of the attack was to modify the green light timing. The timer instruction is made up of three parameters, namely, base, preset, and accumulated, with the preset value determining the duration. Therefore, to accomplish this goal, we altered the preset value from 20 to 80 during the program's download to the PLC (MicroLogix 1400) by the Control engineer using a custom-built Ettercap filter. Consequently, the green light now remains ON for 80

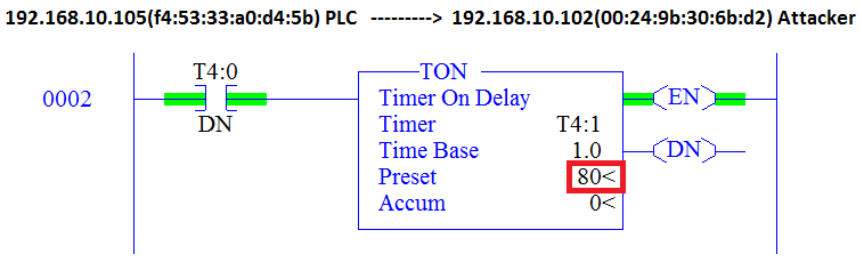


Fig. 13. Control logic from PLC to Attacker

seconds instead of 20. Similarly, when the control engineer uploads the ladder logic program from the PLC, we revert the preset value back to 20, leaving the control engineer to see only the original program on the engineering software. Thus, unbeknownst to the Control engineer, the PLC executes the infected ladder logic.

Forensic Analysis: In order to investigate the DEO attack, we employ a virtual PLC to restore both the original and manipulated instances of the control logic obtained from a network traffic capture, i.e., one instance between the engineering software and the attacker, and the other between the attacker and the PLC. We separate the network traffic based on MAC addresses and subsequently provide these network dumps to the virtual PLC in the form of discrete pcap files. Next, we establish a connection with the virtual PLC using RSLogix software and utilize the upload function to retrieve the control logic. Figures 13 and 14 exhibit the recovered instances of the control logic, where one reflects the original logic and the other reveals the logic manipulated by the attacker, showcasing the alteration of the timer's preset value from 20 to 80.

4.4.1.2 DEO II

Attack Execution: In a manner consistent with the first attack, the assailant employs ARP poisoning (via Ettercap) to instigate a man-in-the-middle attack. The assault takes place when a control engineer attempts to upload the code from a

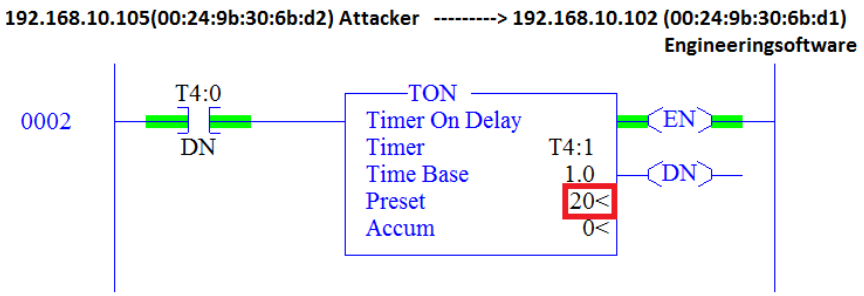
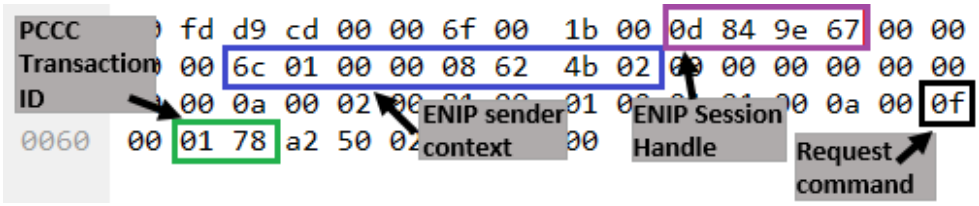


Fig. 14. Control logic from Attacker to Engineering Workstation

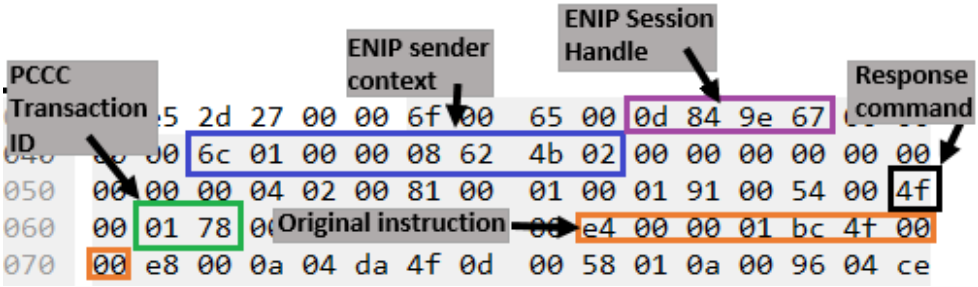
target PLC, the attacker intercepts this communication, and substitutes an authentic instruction with a dysfunctional one. Figure 15(b) and Figure 15(c) delineate the original and distorted messages respectively.

Forensic Analysis: Our examination of the DEO attack incorporates the use of MAC addresses to differentiate the two instances of the control logic, followed by the employment of the virtual PLC to endeavour a recovery of the control logic. To discern the deformed control logic packet within the communication between the engineering software and the attacker, we initiate the *upload* function. However, the altered packet interferes with the engineering software, incapacitating it from further communication with the virtual PLC. The virtual PLC identifies the packet that triggered the disruption, since no subsequent communication is feasible after the transmission of this packet. Figure 15 illustrates the response message from both benign and manipulated control logic. The virtual PLC identifies Figure 15(c).

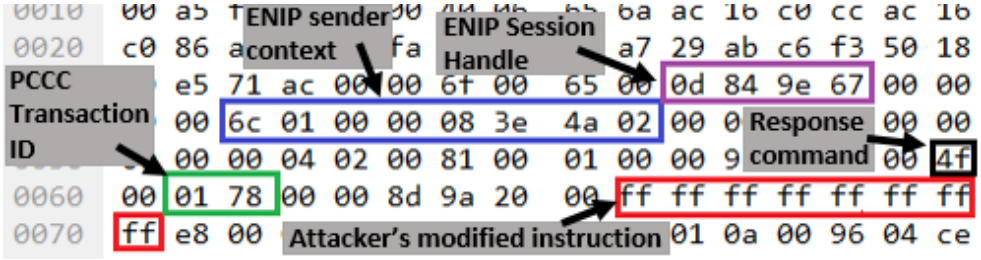
To recover the second (malicious) instance of the control logic between the attacker and the compromised PLC, the virtual PLC invokes the *upload* function once more and successfully transmits the control logic to the engineering software. This results in the recovery of the logic back into high-level source code.



(a) Request message from the engineering software



(b) Response message from PLC to attacker



(c) Distorted response from attacker to Engineering software

Fig. 15. Request and response packets that cause the decompiler to crash

4.4.2 Forensic Analysis of Control-Logic Injection Attacks

Attack Execution: In control-logic injection attacks, the attacker's objective is to inject malicious control logic into the PLC. This can be accomplished in two ways. First, if the attacker has access to the target PLC's IP address and port number, and possesses the capability to generate a malicious control logic binary (as demonstrated in [29, 24, 26]), they can directly connect with the target PLC and overwrite its memory with the malicious control-logic binary. Secondly, the attacker could compromise an engineering workstation (a computer running engineering software), then utilize the present engineering software to author the malicious control-logic, locate the target PLC, and download the harmful control-logic onto it.

Forensic Analysis: Regardless of the method employed by the attacker to execute the control logic injection attack, if the network traffic between the targeted PLC and the attacker has been captured, it will contain evidence of the malicious control logic transfer. Therefore, to retrieve the malicious control-logic from the network traffic dump, a forensic analyst can utilize the virtual PLC platform. Similar to DEO forensics, the analyst can present the forensic artifact (pcap file) to the virtual PLC and then connect with the virtual PLC using the engineering software. The analyst can then leverage the upload functionality of the engineering software to retrieve the control-logic from the network dump. Section 3.5.2.4 confirms that the virtual PLC can upload the control-logic present in the download network stream with 100% transfer accuracy.

4.4.3 Conclusion

In this chapter, we have provided an insight into the persistent threats that Programmable Logic Controllers (PLCs) are subjected to, in the form of DEO and

control-logic injection attacks. As part of this exploration, we discussed the execution of these attacks and underscored the difficulties that researchers and forensic analysts face in performing forensic analyses of these network-based attacks.

To help overcome these challenges, we proposed the use of a Virtual PLC Platform (VPP) as a potential solution. The VPPenables an analyst to analyze network traffic dumps, which often contain vital information about the attacks, and thereby aids in the investigation of PLC attacks. The VPPoffers the ability to replicate the targeted PLC's behavior, hence allowing for the recreation and study of the attacks.

Furthermore, we demonstrated the application of VPPin the forensic analysis of DEO and control-logic injection attacks, by delineating its role in extracting and reconstructing the manipulated control logic from network traffic captures.

The analysis showcased in this chapter confirms that the Virtual PLC Platform can be instrumental in conducting forensic analyses of network-based attacks on PLCs. As such, the VPPemerges as a powerful and promising tool in the realm of cybersecurity, providing an effective way to dissect, understand, and ultimately mitigate threats to PLCs in an industrial control system environment.

CHAPTER 5

PREE: HEURISTIC BUILDER FOR REVERSE ENGINEERING OF NETWORK PROTOCOLS IN INDUSTRIAL CONTROL SYSTEMS

This chapter introduces the third pillar of my research, focusing on the reverse engineering of proprietary protocols used in Industrial Control Systems (ICS). These protocols are instrumental in remote operations of ICS, such as monitoring, controlling, and configuring communication with the control center.

The ability to reverse engineer these protocols paves the way for improved digital forensics techniques in investigating ICS attacks. However, existing methods often fall short due to the complexity and specificities of these protocols.

In response to this challenge, I developed `PREE` (Protocol Reverse Engineering Engine), a heuristic builder. `PREE` leverages knowledge of one ICS protocol to aid in reverse engineering other proprietary ICS protocols. This hypothesis was tested across six ICS protocols using five PLCs from four vendors.

The results demonstrate `PREE`'s efficacy in identifying shared fields across various protocols, surpassing existing reverse engineering tools in terms of accuracy and consistency. Furthermore, `PREE` proves its potential in vulnerability analysis and in investigating various attacks, which will be elaborated further in this chapter.

5.1 Introduction

A Programmable Logic Controller (PLC) is a critical component of Industrial Control Systems (ICS) [30, 22]. These devices are placed at field sites to control physical processes and send their current state to the control center using proprietary

protocols. However, their critical nature makes them a target for attackers over networks to disrupt physical processes [31, 32, 14, 10]. Investigating such attacks is challenging due to the need for an appropriate forensic method to analyze the proprietary protocols used in PLC communication [33, 34, 35].

Protocol knowledge is valuable for security applications such as fuzzing [36, 37, 38], intrusion detection [39, 40, 41], malware injection [31, 42, 29, 43, 44], vulnerability discover [14, 10] and forensics [45, 46, 28, 21]. Since protocols are proprietary, network protocol reverse engineering is typically used to uncover the format and semantics of protocol messages. Existing methods include tedious manual analysis, complex binary analysis [47, 48, 49], or pre-installed capabilities for network traffic analysis [50, 51, 52, 53, 54].

In manual network traffic forensics, the user can compare messages within or across sessions to identify protocol field positions and guess their meanings. Although this is a common approach in security, it faces challenges such as large data volumes, changing control logic message fields, unreadable binary messages, context-specific fields affecting meaning, extracting client-server sessions from network dumps [47], etc. With Industry 4.0, manual semantic forensics is no longer feasible due to the growing connectivity of heterogeneous PLC networks from different vendors, making it difficult for experts to learn all protocols.

These challenges have driven the forensic community to develop automated protocol reverse engineering tools in two directions: Binary (taint) Analysis and Network Trace Analysis. In Binary Analysis, the reverse engineering tool inputs a message to an available executable file of the program or protocol and monitors control flow, called instructions, and memory usage to learn the protocol format and field semantics. In Network Trace Analysis, network traffic between communicating entities is captured and protocol fields and boundaries are identified through machine learning

and data analytics techniques like clustering and differential analysis [54].

ICS protocols support communication between PLCs and enable remote monitoring, control, and configuration by a control center. They inherently overlap and share many standard fields such as function code and PLC memory address. ICS protocols have a consistent usage pattern due to their repetitive control logic operations on PLCs. A pattern recognition tool called Ratcliff/Obershelp was used for fuzzing in a study [38], demonstrating that ICS protocols' consistency makes them suitable for primitive protocol reverse engineering. Therefore, it is hypothesized that knowledge of one ICS protocol can aid in identifying standard fields in others.

This chapter proposes a heuristic builder, the Protocol Reverse Engineering Engine (`PREE`)[55], to allow control engineers to use their ICS protocol knowledge to create heuristics for protocol message fields. `PREE` applies these heuristics to network traffic from an unknown ICS protocol to automatically discover the locations and semantics of similar fields in the protocol messages. It analyzes network dumps at message and session levels and provides data analysis functions to assist heuristic building, such as analyzing message sections and comparing messages within and across sessions.

We evaluated `PREE` on six ICS protocols (Modbus TCP, M221, ENIP, Omron-FINS, CLICK, and PCCC) using five PLCs from four ICS vendors (Modicon M221, Allen Bradley 1400 and 1100, Omron CP1L, and AutomationDirect CLICK Koyo). We used three different techniques (rolling window, vertical window, and frequency table) and created seven heuristics to discover similar fields in multiple protocols. The heuristics effectively identified eight protocol fields, including function code, message type, transmission length, PLC memory address and data size, and session ID.

Our contributions are summarized as follows:

- We present PREE, a heuristic builder for control engineers to use their domain knowledge to reverse engineer ICS protocols.
- We develop eight heuristic algorithms to find eight distinct fields in ICS protocols using three techniques
- We evaluate PREE on six real-world ICS protocols in five PLCs and demonstrate its effectiveness in finding similar fields in different protocols.
- We compare PREE with the existing binary protocol reverse engineering tools like NetPlier, Netzob, and Discoverer.
- We conduct a vulnerability study on CLICK Koyo PLC and develop SNORT rules to investigate and discover several attacks on CLICK PLC to show the application of PREE knowledge.

The remaining chapter is organized as follows: Section 5.2 discusses the background and related work. Section 5.3 presents the PREE architecture and the heuristic algorithms. Section 5.4 and 5.5 presents the PREE implementation and evaluation. Section 5.6 compares PREE with existing reverse engineering tools. Section 5.7 shows offensive and defensive applications of PREE. Section 5.8 concludes the chapter and presents future work.

5.2 Background and Related Work

There are many tools available for reverse engineering protocols to discover protocol message format or the state machine. Most of these tools fall under two techniques; the first is the program analysis technique where protocol binaries are analyzed to reverse engineer the protocol. The second is network trace analysis where different network dumps are analyzed to extract protocol details. Our focus is on tools

developed using network trace analysis and is close to PREE.

Ladi *et al.* [50] presented a four-phase approach to reverse engineer binary protocols. They captured network traffic, constructed and optimized an acyclic graph of the messages exchanged, and assigned pointers at the first byte of each packet to monitor processing. The algorithm starts with a root node and adds nodes for different fields as it moves the pointers of all packets. They developed some heuristics to identify constant bytes, length fields, counters, enumerated types, and highly variable bytes. The approach was evaluated on Modbus and MQTT protocols.

Kim et al. [51] proposed a 4-step method for reverse engineering the Modbus/TCP protocol and creating an intrusion detection system. They used 9 tuples to group similar messages, then applied multiple sequence alignment to categorize bytes into constant, categorical, and variable categories. Next, they identified header/payload boundaries through local sequence alignment and inferred payload fields by categorizing bytes and analyzing their behavior. The result was a successful reverse engineering of the Modbus/TCP protocol and an intrusion detection system.

Wang *et al.* [52] proposed an approach to find feature words in unknown protocols using V-grams and XGBoost. Binary messages were converted to hexadecimal data, grouped by length, and aligned using PMSA. V-grams were generated and feature words were extracted and ranked using XGBoost. They evaluated their approach based on the S7 protocol.

Shim *et al.* [53] proposed a six-stage model for identifying message formats in ICS protocols. They captured communication between PLC and engineering software, then grouped messages based on size and refined groups with K-Means, UPGMA, and mean shift clustering. Then they used a contiguous sequence pattern (CSP) algorithm to extract static/dynamic fields and generated message formats. The approach was evaluated on Modbus/TCP, ENIP, and FTP protocols.

Wu *et al.* [54] presented an HMM-based approach for identifying ICS message formats. They tokenized application layer data into two categories: text (printable bytes) and binary (non-printable) using ASCII encoding. Consecutive printable bytes form one text token and non-printable bytes as a binary token. They then grouped messages with similar token patterns into clusters and inferred different message formats using an HMM-based sequence alignment algorithm. Their approach was evaluated on Modbus/TCP and IEC 61850 protocols.

5.3 Overview of PREE Architecture

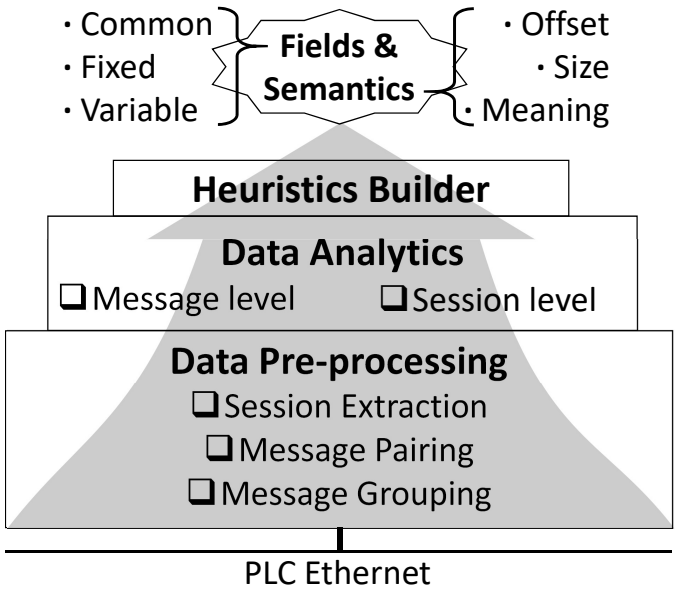


Fig. 16. Protocol Reverse Engineering Engine (PREE) model

`PREE` helps users develop and implement heuristics by using network dumps. Figure 16 illustrates the bottom-up overview of `PREE`. It has a three-layer model: a data pre-processing layer, where network dumps are organized into data structures; a data analytics layer, offering analytics functions for the session and message-level analysis; and a heuristic builder where the user can develop and execute their heuristics for

protocol reverse engineering. PREE works similarly to MySQL. To reverse engineer a protocol, the user provides network dumps with targeted protocol and metadata (e.g. PLC and engineering workstation IPs and ports) to PREE. After processing the dump, the user can use PREE's analytics functions to write heuristics, similar to MySQL queries.

5.3.1 Data Pre-Processing

PREE starts with data pre-processing. This involves extracting client-server sessions, making request-response message pairs, and grouping messages.

Session Extraction: In a network dump, multiple client-server sessions may exist. To analyze them, we must first separate these sessions. PLCs in ICS environments have fixed ports, such as Allen-Bradley MicroLogix 1100 and 1400 using port 44818 and Modicon M221 using port 502. However, the client-side port, used by the engineering software, is often machine-dependent and changes. PREE identifies and separates different sessions by using a four-tuple: source IP, source port, destination IP, and destination port.

Message Pairing: After separating messages from different sessions, the next step is to pair request and response messages and arrange them in order of exchange. This pairing and maintaining the sequence helps identify common fields in request and response messages and fields that show a consistent change along the session.

Message Grouping: Grouping similar messages together is important in ICS protocols that have more than one message format. This helps the user develop heuristics for different groups and discover different message formats in network dumps. Grouping can be based on message payload length or total size.

Table 11. Summary of PREE data analytics functionalities

Function	Description	Type
sim_msg	Find similarity between two messages	Message-Level
find_msg	Search the given sequence of bytes in messages	Message-Level
diff_msg	Find difference between tow messages	Message-Level
h_move	Give all possible substrings and their indices in a message	Message-Level
window_gen	Generates substrings inside a window given message, window size and increment	Message-level
longestSubstringFinder	Find the longest common subsequence of two messages	Session-Level
v_move	Gives array of substring inside the given window for all messages	Session-Level
find_feq	Makes frequency table containing frequency of each byte at each index in the pcap file	Session-Level
freq_match	Find Messages that have bytes with frequency >given threshold	Session-Level
freq_change	Find indices in messages with frequency change lower than given threshold	Session-Level

5.3.2 Data Analytics

PREE's data analytics layer offers useful functions for analyzing network dumps and discovering protocol fields. Table 11 lists the available functions, split into two categories: message-level and session-level analysis.

Message-Level Analysis: During our research, we found that certain protocol fields, such as the "Length field" and "Checksum field", can be identified using the information within the message. The values in the Length and Checksum fields represent the actual length and checksum of a specific section of the message. To aid users in discovering these protocol fields, PREE offers several message-level functions, such as "h_move" and "window_gen" that can be used to identify correlations between different sections of a message.

Session-Level Analysis: The second category focuses on protocol fields that change or show a pattern throughout a session, e.g., the "Transaction ID" present in many ICS protocols increases with every new message in the session. PREE provides the user with several functionalities, such as comparing bytes at the same index in different messages, finding all the values seen at a fixed location in all the messages, etc to perform the session-level analysis.

5.3.3 Heuristic Building

We observe that the common fields in ICS protocols can be divided into three categories based on their behavior during communication. Figure 20 illustrates the classification of different fields in the Modicon M221 message into one of these categories.

Configuration Fields: The fields depend on the ICS environment and can be configured by using engineering software. Their values typically remain constant throughout

the communication session. An example of a configuration field is “PLC ID“.

Fixed Fields: The second category consists of fields with constant values across all messages and sessions. Though it's challenging to gather semantic information from these fields through differential analysis, their patterns can aid in identifying proprietary protocols. Thus, we label them “Protocol Identifiers”.

Variable Fields: The third category includes fields with values in different messages across sessions. For instance, the length, checksum, function code, etc. change per message, while the session ID changes between sessions.

Finding Configuration Fields: No heuristics are required as the values of “Configuration Fields“ are known to the user. They can be located in messages using the “find_msg“ function of `PREE`, which takes the target sequence of bytes (configuration field value) and returns its location or index in all messages of a session if found.

Finding Fixed Fields: Like “Configuration Fields“, no heuristics are required to locate “Fixed Fields“. Users can use the “find_feq“ function to generate a frequency table showing the frequency of values at each index in all messages of the session. Fixed fields can be found where the frequency is 100%, meaning the value stays the same.

5.3.4 Heuristics for Variable Fields

Finding the location and meaning of “Variable fields” is difficult because the variance depends on the field's nature. For example, the “Transaction ID” in a message increases over time, the “Length” and “CRC” fields depend on the payload, and the “Session ID” is initiated by the PLC or engineering software. To handle these variations, we used three techniques and created eight heuristics in total.

Rolling window: In the Rolling Window heuristic, `PREE` employs a sliding window of varying sizes (1,2,...,n bytes) over the message and applies the user-defined function

(which could be designed to find the length, checksum, etc) to all substrings of the messages. If the output of the function matches the value within the window, the location is labeled as a potential field. To minimize false positives, only potential fields that consistently appear across similar messages are selected. Figure 17 shows the implementation of this technique in `PREE`. Using this technique we developed and executed two heuristics to find the length and checksum fields.

- **Length field:** If the user provides the function f(x) that calculates the length of the payload, the value inside any window that matches the output of f(x), the current location of the window can be marked as a length field.
- **Checksum field:** Similarly, if the user has developed a potential checksum function, he can use the rolling window technique to identify the the location of checksum field.

Vertical Window: The value of some fields changes in a fixed pattern throughout a session. This can be detected using the vertical window approach (Figure 18). Using a user-defined function "f(x)", `PREE` moves a window of varying sizes overall messages in a session. For each consecutive message pair, i.e y and y+1, it checks if f(y) = y+1. If this condition is true for all message pairs, the current window location can be labeled a potential protocol field based on f(x). Using this technique we developed two heuristics to find the Transaction ID and PLC Memory Address fields.

- **Transaction ID:** Transaction ID in a protocol increases constantly with each new message. If a user defines f(x) to add a fixed number to x, the sliding window can represent a potential "Transaction ID".
- **PLC Memory Address:** Uploading/downloading control logic involves sending a series of messages with PLC memory address and data size to be read/written

i.e in consecutive messages, the address changes by the size of data written/read. To identify the "PLC Memory Address", the vertical window can use f(x) to add the current memory address and data size.

Frequency table: The frequency table is useful in identifying variable fields in messages that don't have a specific pattern and depend on software or PLC, such as "Function Codes" and "Session ID". The frequency table feature of `PREE` can be used to locate these fields in message headers by creating a table of all messages and storing the frequency and values of each byte at each index in a session. This enables the development of various heuristics to find protocol fields.

- **Session ID:** The Session ID is established in the initial messages between PLC and software and stays constant. To find it, the frequency table can be queried for indices with limited changes and these bytes can be searched in the initial messages. If found, these indices may indicate the "Session ID" in the protocol.

- **Function Code:** The function code is a field with a limited set of codes used by software to send requests to the PLC. If the request is accepted, the PLC replies with a success code. If not, a failure code is sent. In a session with no failures, the function code can be found by querying the frequency table for indices with limited variance in request messages and constant values in response messages. These indices may indicate the location of the "function code" in the ICS protocol. .

- **Message Type ID:** The message type ID is a field that identifies the message as a request or response. It has unique values in request messages and different unique values in response messages. To find this field, separate frequency tables for request and response messages can be created, then compare bytes with 100% frequency in each table.

5.4 Implementation

We developed PREE using Python and Scapy [56]. It is designed as a simple and portable Python library, consisting of four main modules: data processing, data analytics, storage, and query builder. To use PREE, network dumps in the form of pcap files and metadata such as client and server machine IP addresses and port numbers are required.

Data Processing: PREE receives pcap files and metadata (IP & port no.), extracts sessions from the network dump, and identifies request and response messages. It stores them as an ordered dictionary, where requests are keys and responses are values. Multiple pcap files can be processed, creating dictionaries for each. Messages with varying lengths are grouped by length for analysis.

Data Analytics: The data analytics module provides the user with different functionalities to analyze the network dumps and find various relations, patterns, and trends across different messages. The details of functions provided by the data analytics module are in Table 11.

Heuristics Builder: The Heuristics Builder allows for interaction between the user and the PREE. The user can download the PREE and use it either within their programs or through a Python shell. With the help of various functions, the user can process pcap files and make queries to gather information necessary for creating and implementing heuristics.

Storage: The storage module offers various functions to store intermediate results and final protocol message formats, as one protocol field may lead to others.

5.5 Evaluation

5.5.1 Data Collection

For evaluating PREE and our heuristic algorithms, we analyzed six widely-used ICS protocols such as Modbus, EtherNet/IP, etc [57, 58]. We generated network dumps by connecting to different PLCs using their engineering software and capturing network communication using Wireshark. To ensure diversity and comprehensive coverage of message formats, we performed various actions such as transferring control logic between PLC and engineering software, changing PLC mode, etc.

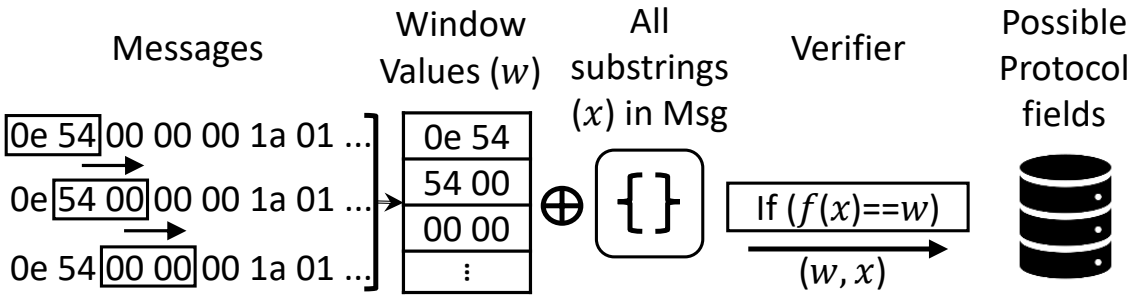


Fig. 17. Rolling window approach to find message-level fields

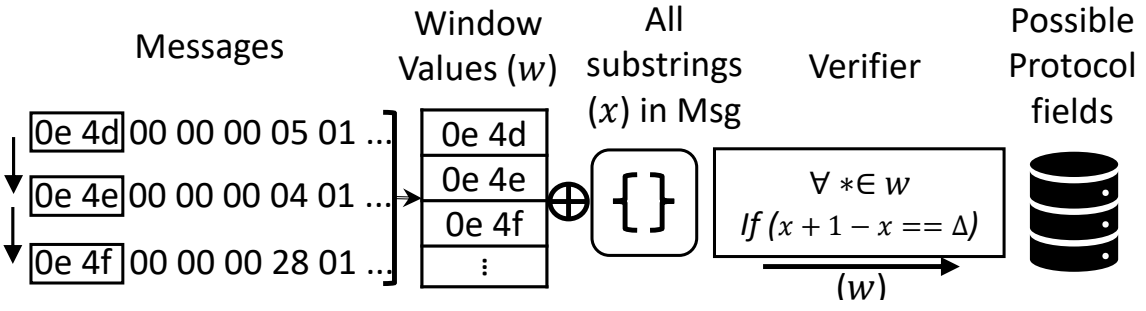


Fig. 18. Vertical window approach to find session-level fields

5.5.2 Evaluation Metrics

Several studies have manually reverse-engineered some of the protocols evaluated in this paper, identifying the location and meaning of certain protocol fields for use in offensive or defensive applications. For the purposes of this paper, we consider the established location and meaning information for each protocol's fields as the **ground truth**. With the ground truth defined, we evaluated `PREE` using three metrics as shown in Figure 19.

Coverage evaluates the percentage of messages covered by `PREE` as protocol fields and is calculated as the ratio of bytes labeled by `PREE` to the total bytes in the message.

Conciseness measures the stability of how `PREE` identified the protocol fields compared to the ground truth and is calculated as the ratio of fields extracted by `PREE` to the total number of fields in the ground truth.

Perfection evaluates the quality of how we perfectly extracted out of the existing ground truth fields. It is the same as having true-positive as a numerator but divided by the total number of ground truth fields.

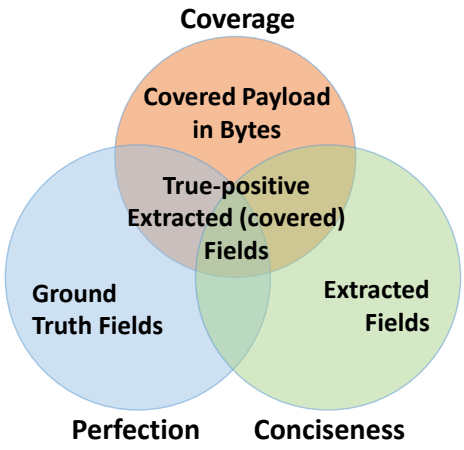


Fig. 19. Three metrics used for evaluating PREE

Table 12. Common fields in different ICS protocols

Semantic	Modbus	Modbus M221	ENIP	PCCC	CLICK	Omron FINS	Field Type
PLC ID	✓						Configuration
Transaction	✓		✓	✓	✓	✓	Variable
Session ID			✓			✓	Variable
Message Type ID			✓	✓	✓	✓	Variable
Message Length	✓	✓	✓	✓	✓	✓	Variable
Function Code		✓	✓	✓	✓	✓	Variable
PLC Memory Data Size		✓	✓	✓	✓	✓	Variable
PLC Memory Address		✓	✓	✓	✓	✓	Variable
Protocol Identifiers	✓	✓	✓	✓	✓	✓	Fixed

$$Conciseness = \frac{\#\ of\ extracted\ ground\ truth\ fields}{\#\ of\ extracted\ fields}$$

$$Coverage = \frac{\#\ of\ labeled\ bytes}{\#\ of\ extracted\ bytes}$$

$$Perfection = \frac{\#\ of\ extracted\ ground\ truth\ fields}{\#\ of\ total\ ground\ truth\ fields}$$

5.5.3 Evaluation Methodology

Table 12 shows that some fields are common across ICS protocols. After collecting network dumps from multiple protocols, we applied various heuristic algorithms created with `PREE` to each protocol, to determine the location and meaning of these fields. To assess conciseness, perfection, and coverage, We compared our results with ground truth from previous ICS protocol reverse engineering studies.

5.5.4 Modbus

In our experiments, we applied different heuristic algorithms developed with PREE to identify fields listed in Table 12 in the Modbus protocol. Using the rolling window heuristic, we found the "Length field" is 2 bytes located at bytes 5-6th, representing the length of the region from byte 6 to the end of the message (payload). The "Transaction ID" was identified using the vertical window heuristic and was found to be two bytes, represented by the first two bytes of the message, and incrementing by one with each new message. Finally, using a frequency table with 100% frequency (bytes that had the same value in all the messages), we identified the Protocol Identifiers. As shown in Figure 20, PREE was able to achieve 100% coverage and identified 4 fields in the Modbus message. Table 13 compares the location and semantics of fields identified by PREE and the ground truth. PREE was able to achieve 100% conciseness and perfection. Our results align with the results of previous manual reverse engineering studies on Modbus protocol (ground truth) [21, 29].

Table 13. Comparison of PREE and ground truth in Modbus

Field	PREE Location	Ground Truth Location	PREE Semantic	Ground Truth Semantic	#PREE types	# Ground Truth types
1	1-2	1-2	Transaction ID	Transaction ID	1	1
2	5-6	5-6	Length	Length	1	1
3	3-4	3-4	Protocol ID	Protocol ID	1	1
4	7	7	Protocol ID	Protocol ID	1	1

5.5.5 UMAS

We used the frequency table heuristic to identify the first byte as a "Protocol Identifier" in the Modbus payload. We found a 2-byte "Length field" at bytes 7-8 that represents the remaining message length. We identified the M221 function code as the 3rd byte, which changed in request messages and was constant in response messages. The "PLC Memory Address" was found at bytes 4-5. Figure 21 shows that `PREE` was able to achieve 100 % coverage in request and 98% coverage in response messages. We found 5 fields in the UMAS message. Furthermore, as shown in Table 14, the fields and semantics identified by `PREE` also matched with the manual forensic studies [42, 28, 29] and achieved 100% conciseness and 80% perfection.

5.5.6 ENIP

ENIP is widely used by Allen-Bradley PLCs, such as the MicroLogix 1400 and MicroLogix 1100, for communication with RSLogix engineering software. For evaluation, we captured the communication between a MicroLogix 1400 PLC and RSLogix during different engineering operations. Using the rolling window technique we found three 2-byte "length fields" at offsets 3-4, 35-36, and 54-55, indicating multiple data

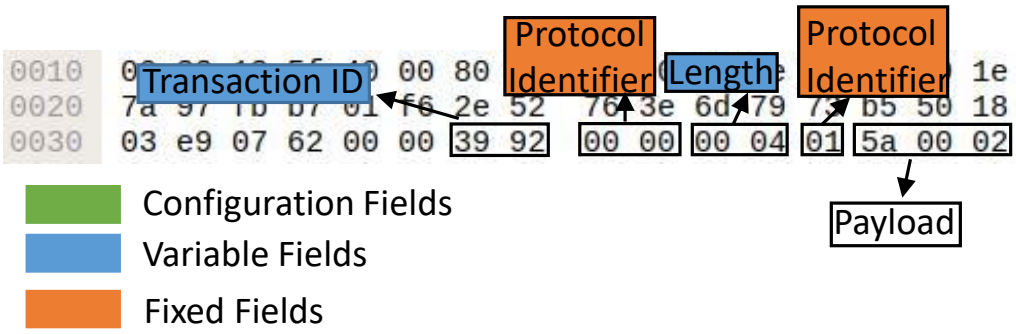


Fig. 20. Fields identified in the Modbus message

Table 14. Comparison of PREE and ground truth in UMAS

Field	PREE Location	Ground Truth Location	PREE Semantic	Ground Truth Semantic	#PREE types	# Ground Truth types
1	1	1	Protocol ID	Protocol ID	1	1
2	3	3	Function Code	Function Code	1	1
3	4-5	4-5	PLC Memory Address	PLC Memory Address	1	1
4	8-9	8-9	Length	PLC Memory Data Size	1	1

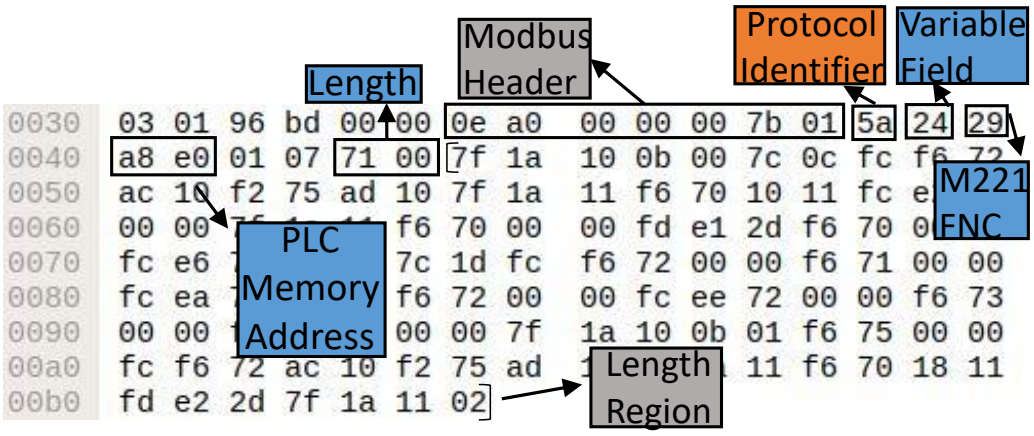


Fig. 21. Fields identified in UMAS request messages

layers. A 100% frequency threshold in the frequency table identified multiple protocol IDs, and a 90% frequency revealed a 4-byte session ID at bytes 5-8. The vertical window approach showed a constant 2-byte "Transaction ID" at bytes 13-14, incrementing by 2 in each message. A 6-byte session field was also found but varied in different sessions. Furthermore, frequency table heuristics identified an additional field: "Message Type ID" at bytes 29-30 with values "0500" in requests and "0004" in responses. Finally, we found the location of IP address of the PLC at 37-50th byte by directly searching it in the message as Field 12 in Table 15. As shown in Figure 22 we could identify 14 fields in the ENIP message and achieve a 98% coverage. Table 15 shows that not only the location and semantics discovered by PREE matched with the existing manual reverse engineering efforts [42, 28, 29], it was also able to identify an extra field *PLC IP Address* (configuration field) that was not discovered (hence marked *NA*, not applicable) in the previous works. PREE achieved 100% conciseness and perfection.

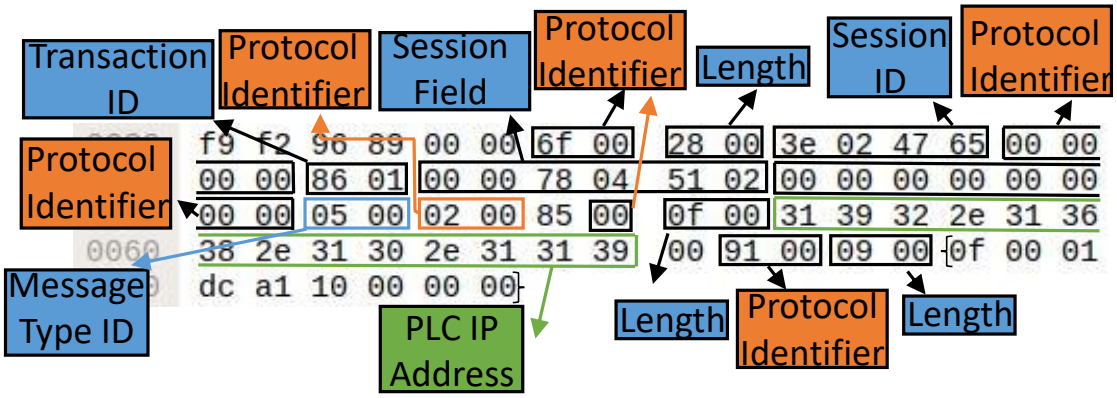


Fig. 22. Fields identified in ENIP request Message

Table 15. Comparison of PREE and ground truth in ENIP

Field	PREE Location	Ground Truth Location	PREE Semantic	Ground Truth Semantic	#PREE types	# Ground Truth types
1	1-2	1	Protocol ID	Protocol ID	1	1
2	3-4	3	Length	*NA*	1	1
3	5-8	4-5	Session ID	*NA*	1	1
4	9-12	8-9	Protocol ID	PLC Memory Data Size	1	1
5	13-14	13-14	Transaction ID	Transaction ID	1	1
6	15-20	15-20	Session Field	Session Field	1	1
7	21-28	21-28	Protocol ID	Protocol ID	1	1
8	29-30	29-30	Message Type	Message Type	2	2
9	31-32	31-32	Protocol ID	Protocol ID	1	1
10	34	34	Protocol ID	Protocol ID	1	1
11	35-36	35-36	Length	Length	1	1
12	37-50	*NA*	PLC IP	*NA*	1	1
13	52-53	52-53	Protocol ID	Protocol ID	1	1
14	54-55	54-55	Length	Length	1	1

5.5.7 PCCC

PCCC is a proprietary protocol used by many Allen-Bradley PLCs. For MicroLogix 1400 and 1100, PCCC messages are embedded in ENIP payloads. After analyzing the ENIP protocol, we applied heuristic algorithms to find PCCC protocol fields. Using the frequency table technique, we found the "Message Type ID" at first and a protocol identifier at the 2nd byte. The "Message Type ID" remained "0f" for all request messages and "4f" for all response messages. We identified the "Transaction ID" at the 3-4th byte in the PCCC message, which increments by 4 with each new message. Using rolling window heuristics, we identified the "Function Code" at the 5th byte and the "PLC Memory Data Size" at the 6th byte. As shown in figure 23, we were able to identify 5 fields in the PCCC protocol and achieve 60% coverage. Table 16 shows that the location and semantics of different fields match with previous works works [46, 43] done on PCCC. Furthermore, we identified a one-byte *Protocol Id* at 2nd byte that was not labeled in the previous work (*NA*). `PREE` achieved 100% conciseness and 62.5% perfection for PCCC protocol.

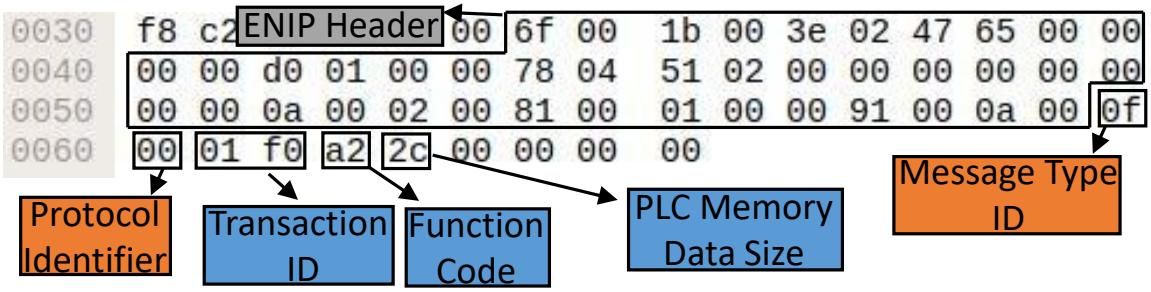


Fig. 23. Fields Identified in PCCC request Message

5.5.8 CLICK

AutomationDirect has developed its own application layer proprietary protocol for communication between CLICK PLC and its engineering software. Using the

frequency table technique with 100% frequency, we were able to identify "Protocol Identifiers". We also found a variable field at the 12th byte that was the same in all the request-response messages except one. We also detected a one-byte "length" field at the 9th byte and the "PLC Memory Data Size" field using the rolling window technique. The "Transaction ID", a two-byte field, was found at the 5-6th byte using the vertical window heuristic. Additionally, the Memory Address heuristic helped us identify the "PLC Memory Address" field's location (16-19th byte) and the read and write function codes at the 13th and 14th bytes. As shown in Figure 24, we identified 8 fields in the CLICK protocol, achieving 85% coverage in request messages. `PREE`'s results matched the existing study on the CLICK protocol [14], achieving 100% conciseness and perfection.

Table 16. Comparison of PREE and ground truth in PCCC

Field	PREE Location	Ground Truth Location	PREE Semantic	Ground Truth Semantic	#PREE types	# Ground Truth types
1	1	1	Message ID	Message ID	2	2
2	2	*NA*	Protocol ID	*NA*	1	*NA*
3	3-4	3-4	Transaction ID	Transaction ID	1	1
4	5	5	Function code	Function code	1	1
5	6	6	Length	PLC Memory Data Size	1	1

Table 17. Comparison of PREE and ground truth in CLICK

Field	PREE Location	Ground Truth Location	PREE Semantic	Ground Truth Semantic	#PREE types	# Ground Truth types
1	1-4	1-4	Protocol ID	Protocol ID	1	1
2	5-6	5-6	Transaction ID	Transaction ID	1	1
3	9	9	Length	Length	1	1
4	10-11	10-11	Protocol ID	Protocol ID	1	1
5	15	15	PLC Memory Data Size	PLC Memory Data Size	1	1
6	16-19	16-19	PLC Memory Address	PLC Memory Address	1	1
7	20	20	Length	PLC Memory Data Size	1	1

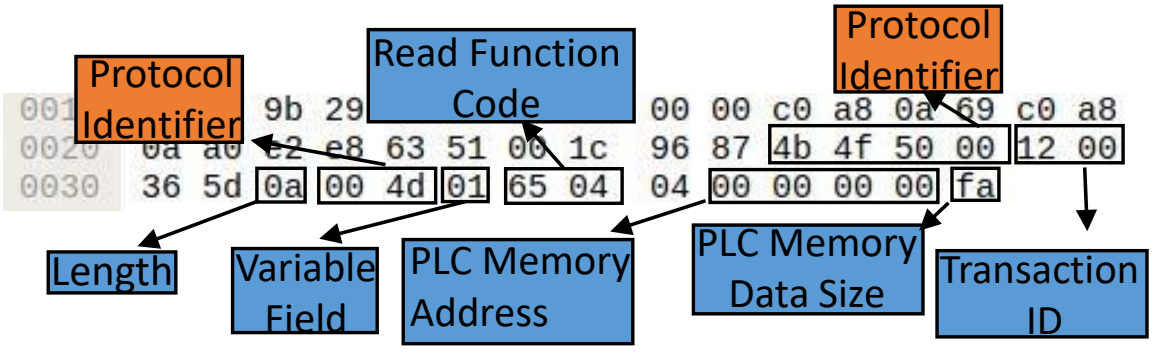


Fig. 24. Fields identified in CLICK PLC request message

5.5.9 OMRON FINS Protocol

The Omron FINS protocol is a proprietary protocol used by OMRON PLCs for communication with engineering software. We used the OMRON CP1L PLC and CX-Programmer in our experiments. Using the frequency table technique with a 100% threshold, we identified the "Protocol Identifier" and "Message Type ID" fields in OMRON FINS network dumps. The two-byte "length field" was found at 7-8th bytes using the rolling window technique. It indicates the number of bytes after it in the message. The one-byte "Transaction ID" was discovered at the 26th byte with the vertical window heuristic. It increases by one in new command messages and stays constant in command-response messages. As shown in Figure 25 we identified 12 fields and achieved 23% coverage. Table 18 show the details of the location and semantics of different fields identified using `PREE`. As we did not find any ground truth for the OMRON FINS protocol, the ground truth location and semantic columns show *NA*. Similarly, we were not able to compute the conciseness and perfection as shown in Table 19.

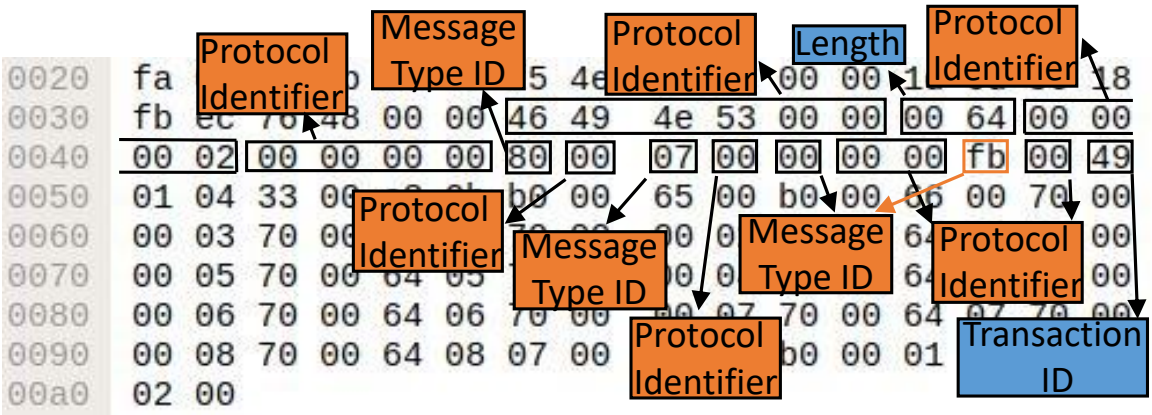


Fig. 25. Fields identified in OMRON FINS command message

Table 18. Comparison of PREE and ground truth in OMRON FINS

Field	PREE Location	Ground Truth Location	PREE Semantic	Ground Truth Semantic	#PREE types	# Ground Truth types
1	1-6	*NA*	Protocol ID	*NA*	1	1
2	7-8	*NA*	Length	*NA*	1	1
3	9-16	*NA*	Length	*NA*	1	1
4	17	*NA*	Message Type ID	*NA*	1	1
5	18	*NA*	Protocol ID	*NA*	1	1
6	19	*NA*	Message Type ID	*NA*	1	1
7	20	*NA*	Protocol ID	*NA*	1	1
8	21	*NA*	Message Type ID	*NA*	1	1
9	22-23	*NA*	Protocol ID	*NA*	1	1
10	24	*NA*	Message Type ID	*NA*	1	1
11	25	*NA*	Protocol ID	*NA*	1	1
12	26	*NA*	Transaction ID	*NA*	1	1

Table 19. Summary of fields identified by PREE

Results	Modbus TCP	Modbus M221	CLICK	ENIP	PCCC	Omron FINS
Ground Truth Fields	4	5	6	13	8	15
PREE Identified	4	4	6	14	5	13
Conciseness	100%	100%	100%	100%	100%	-
Perfection	100%	80%	100%	100%	62.5%	-

5.6 Comparison with Existing Tools

5.6.1 Comparison Metrics

In this section, we will compare the proposed PREE with other protocol reverse engineering tools. There are five evaluation metrics: **V-measure, homogeneity, completeness, conciseness, and perfection** [59]. The score of the **V-measure** $\boldsymbol{v}$ can be computed from **homogeneity** $\boldsymbol{h}$ and **completeness** $\boldsymbol{c}$ as:

$$v = 2 \cdot \frac{h \cdot c}{h + c} \tag{5.1}$$

$$h = \frac{\sum_i^k \#\ of\ elements\ in\ cluster_i\ from\ single\ class}{\sum_i^k \#\ of\ elements\ in\ cluster}$$

$$c = \frac{\sum_i^c \#\ of\ elements\ in\ class_i\ assigned\ to\ single\ class}{\sum_i^c \#\ of\ elements\ in\ class}$$

where k represents the number of clusters, and c represents the number of classes (keywords).

Evaluation metrics such as $\boldsymbol{h}$ and $\boldsymbol{c}$ for probabilistic approaches indicate how well the clustering results are classified into classes. Since PREE is not based on probabilistic methods, it is considered that clustering has been successfully performed if the semantics of the key fields are correctly identified. If the ground truth keyword,

in this case, function code (service code) is inferred, it has h and c score of 100%. For instance, assuming a network using only two function codes, h decreases if the network is classified as one without being able to distinguish between the two, and c decreases if more than two are excessively classified.

5.6.2 Existing/comparison tools

NetPlier proposed by [60] is the most recent state-of-the-art automatic protocol reverse engineering study and provides protocol reverse engineering with probabilistic approaches based on Netzob [61] and MAFFT [62]. The main probabilistic approaches based on NetPlier can be evaluated with homogeneity and completeness scores, which indicate how well clustering contains one class (keyword) [59]. They also performed a comparative analysis with the other protocol reverse engineering tools Netzob and Discoverer [63].

5.6.3 Experiment methodology

For fairness, we compared the results of `PREE` and other tools with a Modbus dataset known to have been used by NetPlier [64] as shown in Table 20. According to [65], NetPlier collected this dataset through a network security monitoring tool called Bro [66]and used this dataset for anomaly detection. Even though NetPlier extracted and evaluated 1000 messages from the dataset, the exact purification method was not disclosed. We evaluate all of the Modbus packets in the dataset.

5.6.4 Comparison Results

As shown in Fig. 26, the proprietary protocols UMAS (embedded in Modbus) and PCCC (embedded in ENIP) were compared using `PREE` and NetPlier. To compare the two, additional development was done for NetPlier. A total of 64,525 Modbus

Table 20. Comparison of PREE with existing tools in Modbus

	PREE	NetPlier	Netzob	Discoverer
Homogeneity	100%	100%	73%	100%
Completeness	100%	100%	70%	55%
Conciseness	100%	70%	59%	4%
Perfection	100%	5%	8%	5%

(UMAS) packets were used for the evaluation and 16,601 for ENIP (PCCC). PREE extracted function codes flawlessly in all four protocols. However, due to the embedded proprietary protocols in Modbus and ENIP, packet lengths became more variable, leading to a drop in NetPlier's performance of over 30%. Nonetheless, NetPlier showed better performance in ENIP (PCCC) than in Modbus (UMAS), and the completeness and homogeneity varied depending on the protocol's propensity.

NetPlier infers keyword candidates to cluster classes accurately. Function codes are typically used as the key field to indicate the class, as is the case in most other situations. However, in instances where a proprietary protocol is embedded in a known protocol (e.g., UMAS in Modbus), there are two kinds of function code fields, one in the known protocol and one in the proprietary protocol. As a result, even if NetPlier correctly infers the function code in the known protocol (Modbus) as a keyword, maintaining a homogeneity of 1.0, the completeness remains low because the proprietary function code is the true classification. NetPlier, Netzob, and Discoverer were unable to infer proprietary function fields such as Modbus (UMAS) and ENIP (PCCC). This trend occurred due to two main reasons. Firstly, the probabilistic approach is not appropriate for analyzing proprietary protocols with dual structures.

Secondly, entropy-based alignment performed by MAFFT does not handle proprietary fields located within the payload effectively.

The performance of proprietary protocols, such as Modbus (UMAS) and ENIP (PCCC), was inversely proportional to the number of internal function types. Although performance tended to increase on a large scale as the number of packets increased, it never exceeded a certain value. This is due to the proprietary function field not being recognized as a possible keyword candidate based on entropy. It's worth noting that poor metrics were not solely caused by proprietary protocols. The example packets' usage patterns differed somewhat from the actual behavior of PLCs.

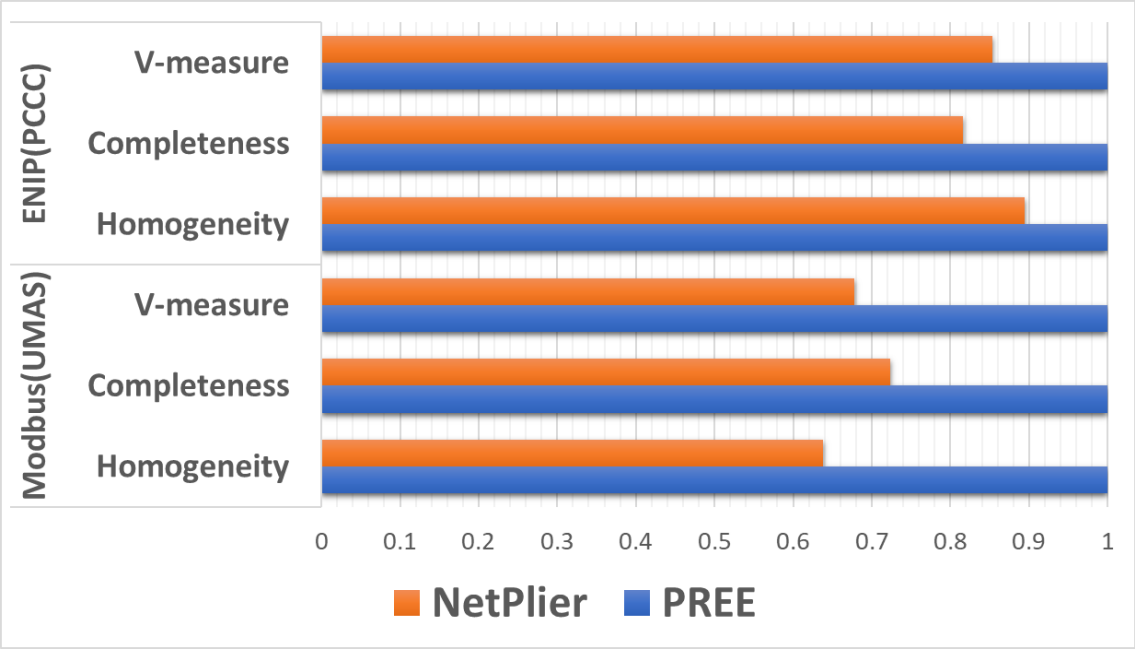


Fig. 26. Comparison of PREE with NetPlier on proprietary protocols (UMAS embedded in Modbus and PCCC embedded in ENIP)

5.7 PREE Applications for Vulnerability Study and Forensic Analysis of Different Attacks

5.7.1 PREE Application 1: Vulnerability Study on CLICK PLC

Network-based attacks on PLCs often require knowledge of proprietary protocols. For example, reverse engineering was used to exploit PLC authentication in Adeen et al.[14]. Kalle et al.[29] leveraged knowledge of the Modbus protocol for a control logic injection attack on Schneider Electric's Modicon M221. Syed et al.[10] demonstrated a new type of attack that disrupts the physical process by targeting the control engine (responsible for executing control logic). They identified messages for starting/stopping the control engine of a PLC by analyzing network communication between the PLC and software, then modified fields and replayed the messages to stop the engine. We extended their work by conducting a control engine attack on the CLICK PLC using protocol knowledge obtained through `PREE`.

Adversary Model: We assume the adversary is inside the ICS network and can communicate with the target PLC, sniff its communication with engineering software, initiate connections, and send malicious messages.

Experimental Setup: We used AutomationDirect's CLICK Koyo PLC model C0-10DD2E-D with firmware version 2.60. The engineering software was running on Windows 10 in a virtual machine, and the attacker scripts ran on an Ubuntu 18.04 virtual machine, both in the same network.

Attack Implementation: We started by changing the mode of a PLC using the engineering software, captured the network traffic, and analyzed the differences to identify the messages responsible for switching the PLC from start to stop. Once identified (Figure 27), we created a python script that modifies the message using the protocol knowledge from `PREE` and sends it to the target PLC to change its operation

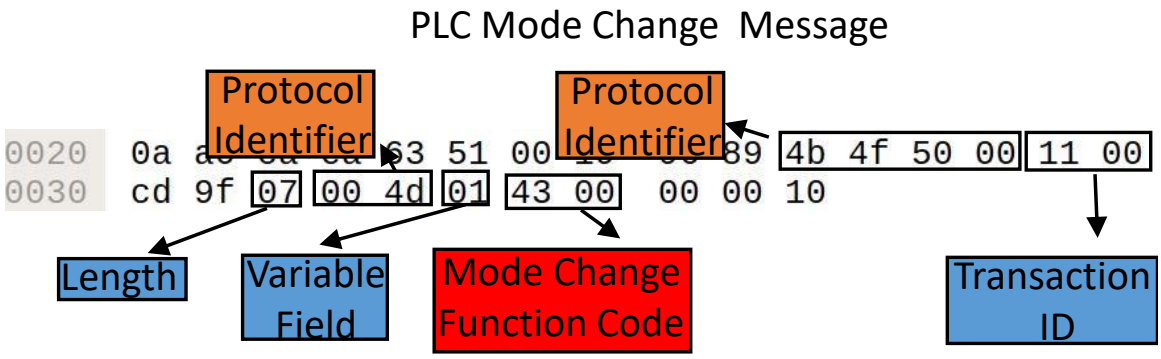


Fig. 27. Message of a control engine attack on CLICK PLC, showing the function code used to change the PLC mode

mode.

5.7.2 PREE Application II: Forensic Analysis of Different Attacks on CLICK PLC Using SNORT

The communication between a PLC and the attacker, if captured during an attack, can serve as a forensic artifact. A forensic investigator with knowledge of the PLC protocol can use SNORT to analyze network dumps (pcap files) for evidence of an attack. SNORT is a widely used IPS/IDS tool, and it is used in projects such as [67]. It analyzes messages using user-defined rules and generates alerts for matching packets.

Snort Rule for Detecting Control Engine Attack: To investigate if control logic engine attacks happened on a CLICK PLC, we used `PREE` to analyze attack messages and find fields in the CLICK protocol. Figure 28 shows the SNORT rules we developed. While analyzing a network dump a SNORT rule raises an alert if it finds a message containing the signature of a PLC mode change i.e“4b 4f 50 00” at offset 0 and “07 00 4d 01 43 00” starting from offset 8. Our evaluation shows that

this rule effectively detects control engine attacks on a CLICK PLC.

Snort Rule for Control Logic Injection Attack: To detect control logic injection attacks on the CLICK PLC, where an attacker tries to download malicious control logic to the PLC, we need to identify the request message used for writing data and its signature. We captured network communication while downloading a benign control logic on the PLC and identified write request messages (those with the largest size and larger than the corresponding response message). Using `PREE`, we extracted protocol identifiers, length, and function code and developed a SNORT rule as shown in Figure 28. The SNORT rule raises an alert when it detects a message with the signature "4b 4f 50 00" at offset 0 and "0a 00 4d 01 65 05" starting from offset 8. In this way, the forensic investigator can analyze the network dump to find evidence of a control logic injection attack on the targeted PLC.

Snort Rule for Control Logic Theft Attack To detect control logic theft attacks, where an attacker tries to read the logic on a PLC, we need to identify the unique signature of read request messages. We captured the communication between the PLC and engineering software during a control logic upload and found the read request messages by looking for smaller requests compared to other requests and smaller than their corresponding response. Then we used `PREE` to discover different fields in these messages and developed the snort rule that raises an alert whenever it detects a message in the network dump with the following signature; bytes "4b 4f 50 00" at offset 0, and "0a 00 4d 01 65 04" starting from offset 8. In this way, a forensic investigator can discover if a control logic theft attack happened on the targeted PLC.

Snort Rule Template for Detecting Control Engine Attack

```
alert udp any any -> PLCIP 25425 (content:"|4b 4f 50 00|";offset:0;
 depth:4; content:"|07 00 4d 01 43 00|"; offset:8;
 depth:6; msg:“PLC Mode change attempted“)
```

Snort Rule Template for Detecting Control Logic Injection Attack

```
alert udp any any -> PLCIP 25425 (content:"|4b 4f 50 00|";offset:0;
 depth:4; content:"|0a 00 4d 01 65 05|"; offset:8;
 depth:6; msg:“Control Logic write attempt“)
```

Snort Rule Template for Detecting Control Logic Theft Attack

```
alert udp any any -> PLCIP 25425 (content:"|4b 4f 50 00|";offset:0;
 depth:4; content:"|0a 00 4d 01 65 04|"; offset:8;
 depth:6; msg:“Control Logic Read attempt“)
```

Fig. 28. Snort rules to detect different attacks on CLICK PLC

5.8 Conclusion

In industrial control systems, proprietary protocols are commonly used to establish communication between PLCs and their engineering software. The knowledge of the location and meaning of various fields in these protocols can enhance the effectiveness of existing security solutions, support the development of new tools, and aid the forensic community in investigating network-based attacks on PLCs. Consequently, we introduce a novel tool for reversing proprietary ICS protocols: the protocol reverse engineering engine `PREE`.

Our observations revealed that many ICS protocols have similar characteristics and share common fields due to operational requirements. This led us to propose the hypothesis that knowledge of one ICS protocol can aid in reverse engineering other proprietary protocols. In this chapter, we present `PREE`, a tool that helps users develop heuristics for identifying fields in proprietary ICS protocols. To test our

hypothesis, we employed three techniques to develop seven heuristics, which were applied to six different protocols (Modbus, UMAS, ENIP, PCCC, CLICK, OMRON FINS). Our evaluation results indicate that `PREE` is able to identify several common fields in these protocols, such as "Length", "Transaction ID", "Message Type ID", etc. Furthermore, we showed the practical application of protocol knowledge to investigate 3 different network-based attacks on CLICK PLC.

CHAPTER 6

ATTACKING IEC-61131 LOGIC ENGINE IN PROGRAMMABLE LOGIC CONTROLLERS IN INDUSTRIAL CONTROL SYSTEMS

This chapter explores the fourth contribution of my research, introducing a new dimension to cyber-attacks on Programmable Logic Controllers (PLCs) in industrial control systems (ICS) - the Control Engine Attack. Traditionally, most cyberattacks have centered on injecting malicious control logic into a PLC, with the aim of sabotaging a physical process. However, we now shift focus to the control logic engine of a PLC, the very core that runs the control logic, revealing its vulnerability to cyberattacks.

This research demonstrates how a cyberattack can effectively disable a PLC's control logic engine by exploiting inherent features, such as program mode and engine start/stop operations. Two novel case studies on Control Engine Attacks are presented, utilizing real-world industrial PLCs. These include a sophisticated PLC with security features and traditional PLCs devoid of security measures.

The insights gleaned from these case studies aim to enhance understanding within the ICS research community and industry on attack vectors targeting the control logic engine. By evaluating these attacks in the context of a power substation, a multi-floor elevator, and a conveyor belt, this chapter underscores the real-world implications of these cyberattacks, which can bring physical processes to a halt.

6.1 Introduction

Industrial control systems (ICS) monitor and control industrial physical and infrastructure processes such as power grids, nuclear plants, water treatment facilities, and gas pipelines [68, 69, 2]. An ICS environment consists of a control center and a field site. The control center includes ICS services such as the human-machine interface (HMI) and engineering workstation, while the field sites contain the actual physical processes monitored and controlled via sensors, actuators, and programmable logic controllers (PLCs). PLCs are embedded devices that directly automate industrial processes [14]. They run control logic programs written in IEC 61131 languages such as instruction list, ladder diagram, and structured text, which define how a physical process is controlled.

A PLC's control logic is a common target of cyberattacks aimed at sabotaging a physical process [70]. However, in existing literature, the focus of control logic attacks is predominantly on injecting malicious control logic into a target PLC over the network to disrupt or cause damage to the underlying physical process [71, 10, 21, 72, 29, 4, 8, 73, 22]. For instance, Stuxnet, a piece of ICS malware, targets Siemens Step 7 engineering software and S7-300 PLCs in a nuclear plant facility to inject malicious control logic [74].

This chapter introduces a new dimension of control logic attacks by targeting the control engine (which runs the control logic) in a PLC. We demonstrate that a cyberattack can successfully disable an IEC-61131 control-logic engine, thereby halting a physical process controlled by a target PLC by exploiting the PLC design features such as program mode and starting/stopping engine. We develop two novel case studies on control logic engine attacks by utilizing the attacks from the MITRE ATT&CK knowledge base such as denial of control (T0813), loss of availability (T0826), ma-

nipulation of control (T0831), unauthorized command message (T0855), and man in the middle (T0830) [75]. We apply the MITRE ATT&CK to the logic engine in four real-world PLCs used in industrial settings. The first case study covers the Schweitzer Engineering Laboratory (SEL) Real-Time Automation Controller (SEL RTAC 3505), which is equipped with security features such as device access control and encrypted traffic. The second case study involves three traditional PLCs with no security features: Schneider Electric's Modicon M221, and Allen-Bradley's Micrologix 1400 and 1100 PLCs. The case studies discuss the internals of the control logic engine attacks, including proprietary PLC communication protocols, and aid the ICS research community and industry in understanding the attack vectors on the control logic engine.

We evaluate the effectiveness of the control engine attacks on a power substation, a conveyor belt, and a 4-floor elevator to demonstrate their real-world impact. For instance, a PLC controls the sorting of different types of objects on the conveyor belt using sensors and an air solenoid. In the substation, a PLC opens a circuit breaker when the voltmeter reports a voltage level higher than a given threshold. The control engine attacks halt the conveyor belt and prevent the substation from controlling high voltage. In the elevator, a user can select a floor both from inside and outside the elevator as an input to a PLC. In response, the PLC moves the elevator to the selected floor.

Our contributions are threefold:

- We introduce a new attack vector that targets an IEC-61131 control-logic engine in a PLC to halt a physical process.
- We successfully utilize the MITRE ATT&CK knowledge base to develop and demonstrate control engine attacks on four real-world PLCs.

	Senthival et al.	Yoo et al.	Govil et al.	Stuxnet	Kalle et al.	McLaughlin et al.	Garcia et al.	Schuett et al.	This work
Firmware Modification							x	x	
Malicious Control Logic	x	x	x	x	x	x			
Control Engine State									x

Table 21. The comparison of different attacks on PLCs.

- We evaluate the effectiveness of the control engine attacks on connected physical processes, namely a power substation, a 4-floor elevator, and a conveyor belt to demonstrate their real-world impact of halting a physical process.

6.2 Related Work

Existing control logic attacks in the literature either target a control logic code running on a PLC (also referred to as control logic injection attacks [76, 77, 78]), or compromise the PLC firmware to manipulate control logic execution [5].

Senthival *et al.* [4] present three types of denial of engineering operations (DEO) attacks. In the first DEO attack, the attacker intercepts the network traffic between the engineering workstation and a target PLC, replacing the original ladder diagram

program with an infected one and vice versa when a control-logic program is downloaded and uploaded, respectively. In the second DEO attack, a man-in-the-middle attacker replaces part of the original ladder diagram program with noise when a control logic program is uploaded, causing the engineering software to crash. The third DEO attack also crashes the implementing software, but in this case, the attacker remotely downloads an infected control logic to a PLC instead of performing a man-in-the-middle attack.

Kalle *et al.* [29] present CLIK, a control logic infection attack, comprising four phases. First, it compromises PLC security measures and steals the control logic from it. Then, it decompiles the stolen binary of the control logic to inject the malicious logic, followed by transferring the infected binary back to the PLC. Finally, it hides the malicious logic written into the PLC from the engineering software by employing a virtual PLC that captures the original logic's network traffic and sends this network traffic to the engineering software when it attempts to read the control logic written inside the PLC.

Similar to CLIK, McLaughlin *et al.* present SABOT [79], a tool that first uploads the targeted PLCs' control logic bytecode and decompiles it into a logical model to find a mapping between the devices connected to the PLC and variables within the control logic. The attacker can then change this mapping arbitrarily and download the control logic back to the PLC to cause damage to the plant. SABOT assumes that the attacker has knowledge of ICS operations.

Yoo *et al.* [72] present two control logic injection attacks, namely 1) data execution and 2) fragmentation and noise padding. In the data execution attack, the attacker exploits the fact that the PLCs do not enforce data execution prevention (DEP) and transfers the attacker's control logic to the data blocks of the PLC. The attacker then changes the PLC's system control flow to execute the logic located

in data blocks. The fragmentation and noise padding attack subverts deep packet inspection by sending write requests with the attacker's control logic. Each write request contains one byte of the control logic while the rest of the packet includes noise. For every subsequent write request, the attacker attempts to overwrite the PLC memory region previously written with noise due to the previous request.

Govil *et al.* [26] presented malware written in ladder logic called ladder logic bomb that an attacker can insert into the existing control logic of a PLC. These logic bombs are hard to detect by a control engineer manually validating the control logic running on the PLC. These bombs can either be activated via trigger signals to cause disruption or can persistently damage the physical operations over time.

Garcia *et al.*[5] presented Harvey, a model-aware rootkit that sits in a PLC firmware using JTAG (Joint Test Action Group)[33]. From the legitimate input data, Harvey generates fake, real-looking input. The PLC processes this input according to the control logic and generates output commands to actuators. Harvey blocks this output at the firmware level and sends the malicious output generated by the attacker's code to the sensors. This abstraction helps Harvey deceive the control engineer monitoring the HMI.

Schuett *et al.* [80] evaluated the possibility of modifying the PLC firmware to execute remotely-triggered attacks. They first reverse-engineered the PLC framework and added modifications. This modified firmware is repackaged and installed on the PLC. Using this compromised firmware, the attacker can perform time-based or remotely-triggered denial of service attacks on the PLC.

6.3 Attacking the Control Logic Engine

6.3.1 Adversary Model

We assume the adversary has access to the ICS network and can communicate with the target controller to launch a control logic engine attack. The attacker could use a real-world IT attack method (such as an infected USB stick or a vulnerable web server) to infiltrate the ICS network and disable the controller from running the control logic. However, discussing IT attacks falls beyond the scope of this paper. While within the ICS network, we also assume that the attacker possesses the following capabilities:

- The ability to read the communication between a PLC and an engineering workstation.
- The capability to drop or modify any message in the communication by positioning themselves as a man-in-the-middle.
- The ability to initiate a connection with a PLC to send malicious messages remotely.

6.3.2 Overview of the Case Studies

We introduce two innovative case studies that explore IEC-61131 control logic engine attacks on real-world PLCs used in industrial settings. A control logic engine attack is defined as "an attack that disrupts or impairs the normal functioning of a control logic engine." The case studies examine cyberattacks that can halt a control-logic engine from executing control logic. Our approach to conduct these studies involves utilizing the MITRE ATT&CK [75] knowledge base on real-world PLCs to target the control-logic engine. In particular, the studies deploy a subset of the

Table 22. Subsets of MITRE ATT&CK utilized on four PLCs in the case studies

PLC	Manipulation of Control	Loss of Availability	Denial of View	Denial of Control	Man in the Middle	Network Sniffing	Unauthorized Command Message
SEL RTAC 3505	✓				✓	✓	
Modicon M221	✓	✓		✓		✓	✓
MicroLogix 1100	✓		✓	✓	✓	✓	✓
MicroLogix 1400	✓		✓	✓	✓	✓	✓

following attacks (along with their IDs) from the knowledge base to demonstrate the control-logic engine attacks. Table 22 summarizes the attack subsets used from the MITRE ATT&CK knowledge base to illustrate control logic engine attacks on the PLCs.

- *Manipulation of Control (T0831).* The attacker manipulates physical process control within the industrial environment.
- *Loss of Availability (T0826).* The attacker disrupts a component to prevent the operator from delivering products or services.
- *Denial of View (T0815).* The attacker disrupts or prevents the operator from viewing the status of an ICS environment.
- *Denial of Control (T0813).* The attacker temporarily prevents the operator from interacting with process controls.

- *Man in the Middle (T0830).* An attacker within the ICS network can intercept, modify, or drop the packets exchanged between the engineering workstation and the PLC.

- *Network Sniffing (T0842).* An attacker within the ICS network can attempt to sniff the network traffic to gain information about its target.

- *Unauthorized Command Message (T0855).* Attackers may send unauthorized command messages to industrial control systems devices to cause them to function improperly.

The first case study focuses on an SEL RTAC device, equipped with security features such as encrypted traffic and device-level access control. The RTAC has a component known as the 'Logic engine,' which is responsible for running the controller's control logic. With this component as the attacker's primary target, she positions herself as a man-in-the-middle between the engineering software and the PLC to prevent the controller from executing the control logic in two ways: 1) by modifying the packet responsible for starting the logic engine, and 2) by dropping this packet entirely. Note that initial communication with the controller is encrypted using transport layer security (TLS). Unencrypted communication begins once a legitimate user has logged in, thereby enabling the man-in-the-middle attack. For more details, see Section 6.4.

The second case study focuses on traditional PLCs that lack built-in security features. It involves three PLCs: Modicon M221, MicroLogix 1100, and MicroLogix 1400. These PLCs differ from the RTAC in two ways: First, these PLCs do not have a separate logic engine, and the processor assumes this role. Depending on the controller type, the PLC either needs to be in 'run' mode or must receive a 'start controller' command to run the control logic. Second, unlike the case with

SEL-RTAC, most communications with these PLCs are unencrypted. To attack the control logic engine, the attacker can craft a specific message and then send it to the PLC to remotely change its state. For more details, see Section 6.5

6.4 Case Study I: SEL-3505 RTAC

6.4.1 Controller Details

The SEL-3505 Real-Time Automation Controller, developed by Schweitzer Engineering Laboratories, comes equipped with an IEC 61131 control logic engine. It provides a web interface that allows monitoring and configuration of the network interface, system logs, user accounts, and security settings. Control engineers can utilize the AcSELerator RTAC SEL-5033 software [81] to write control logic, configure protocol communication, read/write projects, and start or stop the logic engine.

Concerning device-level security, the RTAC employs ex-GUARD [82], a whitelist-based system that controls the execution of different tasks. Any tasks not approved by the whitelist are blocked from operation [81].

The RTAC 3505 communicates with the AcSELerator software via port 5432. The majority of this communication—including session establishment, user authentication, and reading and writing of projects on RTAC—is encrypted using TLS encryption. However, after a user successfully logs in, the controller opens another port (1217) and begins a second communication channel to share the state of the RTAC in real-time. Intriguingly, the communication on port 1217 is not encrypted.

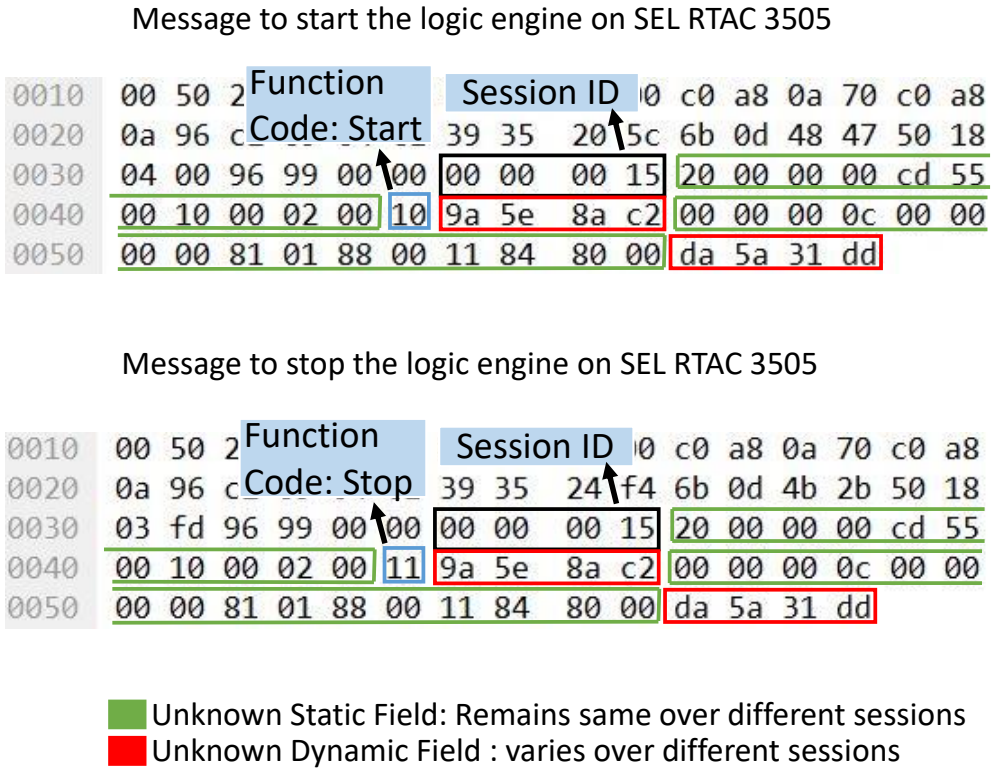


Fig. 29. Messages sent by AcSELerator software to SEL RTAC 3505 to start and stop logic engine

6.4.2 Vulnerability

In examining the internal communication of the RTAC, we discovered that the RTAC sends unencrypted commands on port 1217 to start or stop the logic engine. Furthermore, we were able to identify the packets that carry these commands. We proceeded to reverse-engineer the commands to better understand the function codes and other fields such as the session ID. Figure 29 illustrates the two request packets that the AcSELerator sends to the RTAC to start and stop the logic engine. We found that the session ID increments by three in every new session, and the function codes for starting and stopping the logic engine are `0x10` and `0x11`, respectively. Additionally, we observed that the remaining parts of the messages stay the same

across different sessions (referred to as unknown static fields), indicating that a deep understanding of their semantics isn't necessary to perform the attacks.

6.4.3 MITRE ATT&CK

In this case study, we utilize the following strategies from the MITRE ATT&CK knowledge base.

Network Sniffing (T0842): Network Sniffing serves as the initial step for discovering vulnerabilities and launching the final attack. Given that the attacker's machine is on the same network as the legitimate engineering workstation and the controller, she can conveniently sniff the network traffic between the legitimate parties to find potential vulnerabilities. Through this process, we can identify the port where the communication is unencrypted and the necessary fields required for designing our Ettercap filters [83].

Man in the Middle (T0830): Upon sniffing the network traffic and identifying the packets responsible for starting or stopping the logic engine, the attacker can develop Ettercap filters, poison the ARP cache of the target machines, and place herself as a man-in-the-middle between the AcSELerator software and the SEL RTAC device. This position allows her to modify the content of the packets being sent from the engineering software to the device or drop these packets altogether.

Manipulation of Control (T0831): This strategy serves as an extension of the aforementioned man-in-the-middle attack. Through the steps taken to establish a man-in-the-middle attack, the attacker can prevent the control logic engine from running the control logic on the PLC, consequently halting the execution of the underlying physical process.

Pseudocode for RTAC filters

```
Input: TCP packet
1:  if (packet_src == AcSELerator & packet_dst == RTAC & packet_port ==1217)
  2:  if (packet_payload_contains (static_fields) )
    3:  Modify/Drop
```

Fig. 30. Illustration of the Ettercap filter implementation.

6.4.4 Attack Implementation

Leveraging the information obtained, we developed two Ettercap filters, namely the DropFilter and Start-StopFilter. These filters can successfully identify the messages that contain the start' and stop' commands for the logic engine, modify them (i.e., convert start' to stop' and vice versa), or altogether drop them. This action results in the logic engine halting its operations or obstructs the control engineer from accessing the logic engine.

Initially, the filters identify the messages from the AcSELerator to the RTAC using their respective IP addresses and port 1217. As shown in Figure 29, a large portion of the messages containing the start' and stop' commands remains the same across different sessions; we refer to this as the unknown static field. The filters search for these static bytes in the TCP payload of the message to identify the correct one. Once the message is correctly identified, the DropFilter can utilize the "drop()" command to drop the message. Similarly, the Start-StopFilter can change the function code located at the 15th index in the TCP payload from start (`0x10`) to stop (`0x11`) and vice versa.

6.4.4.1 Evaluation

6.4.4.2 Experimental Settings

We evaluate the control logic engine attack within a simulated power substation environment. This environment consists of an SEL-3505 RTAC connected to an engineering workstation, a circuit breaker, and a simulated voltage measurement device designed to behave as a voltmeter. The RTAC is programmed to open the circuit breaker whenever the voltmeter reports a voltage level higher than a pre-configured threshold. This threshold is typically set by operators in the control center. In case the circuit breaker does not open promptly after the voltage rises beyond the safe limit, expensive power equipment could be potentially damaged or destroyed. The system is monitored via a Human-Machine Interface (HMI), as depicted in Figure 31. Note that in our evaluation setup, the "vRTAC" displays the system's ground truth state, even if the SEL RTAC fails. This feature allows us to identify real-time discrepancies between the actual system state and the state reported by the RTAC during the attack.

6.4.4.3 Attack Execution and Evaluation

At system startup, the SEL RTAC 3505 automatically activates its logic engine. Therefore, our first step involves stopping the logic engine. This process emulates typical maintenance or reprogramming operations. Subsequently, we initiate Ettercap's ARP spoofing attack against the RTAC and the engineering workstation, armed with our custom packet filters. Whenever an operator attempts to start the logic engine, the attacker intercepts the command, modifies the function code or drops the packet before it reaches the RTAC. As a result, the logic engine never restarts.

Following the failure of the logic engine to restart due to our attack (even after

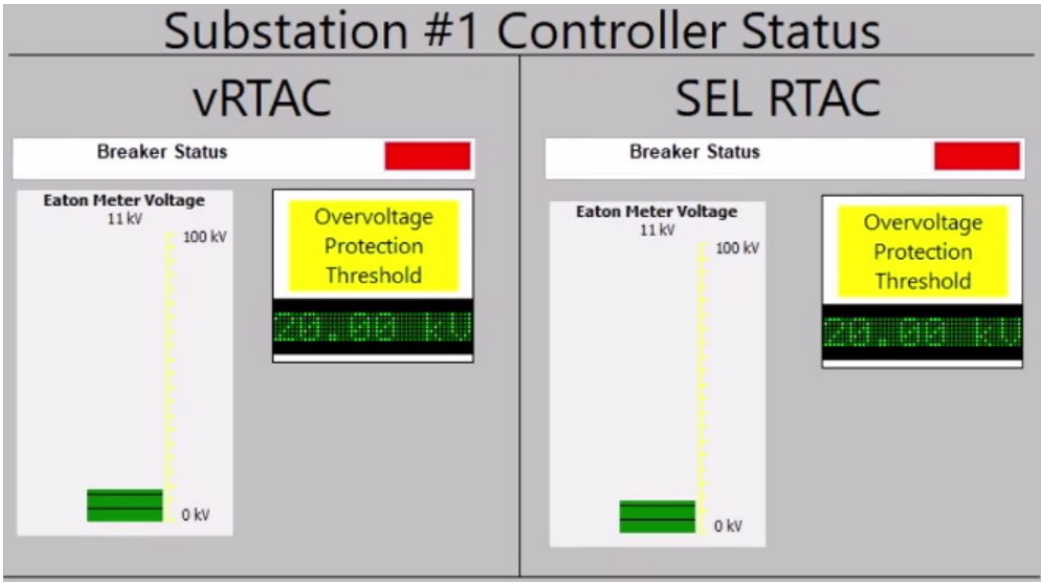


Fig. 31. HMI showing ground truth (left) and SEL RTAC state (right) for a circuit breaker (red means closed) and voltmeter with a given over-voltage protection threshold

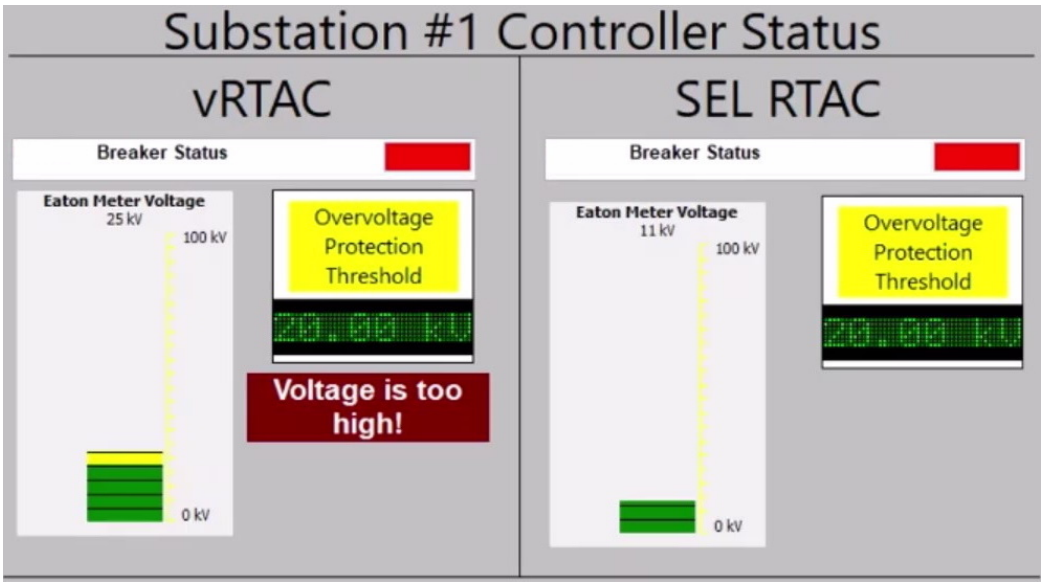


Fig. 32. HMI showing SEL RTAC is no longer reporting new values while voltage has surpassed the threshold

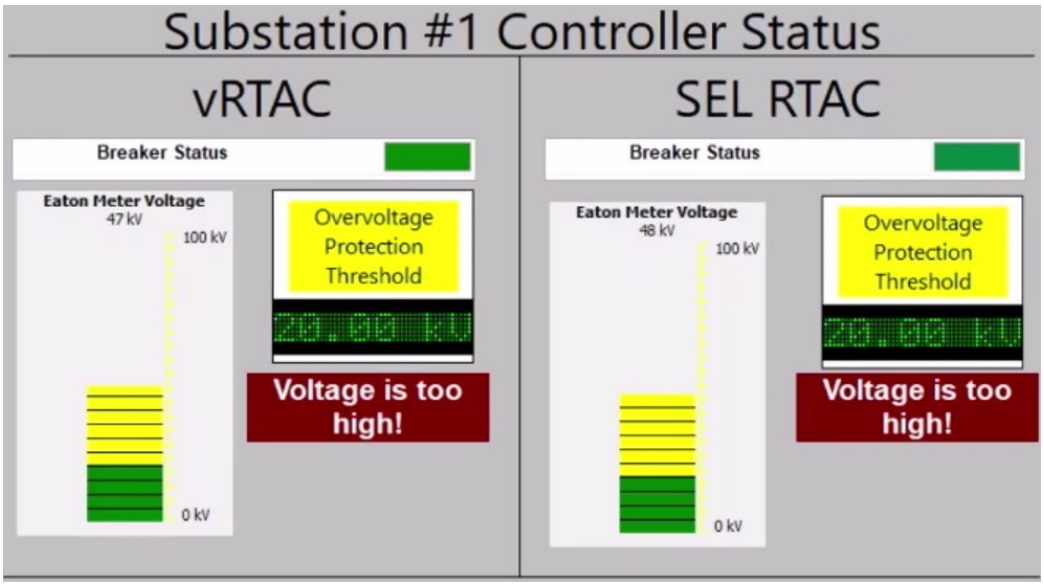


Fig. 33. HMI showing updated SEL RTAC after the logic engine is enabled and the circuit breaker is now open (green means open)

repeated start command attempts), the RTAC can no longer monitor or control the power system devices. If a short circuit occurs in the system, leading to a high voltage detection by the voltmeter, the controller needed to open the circuit breaker no longer functions. Consequently, power continues to flow unchecked, potentially reaching critical equipment such as transformers. This can result in severe equipment damage or destruction, necessitating costly repairs or replacements and potentially leading to a power outage. Our evaluation state is depicted in Figure 32. As observed, the SEL RTAC no longer reports current voltage values, and the breaker fails to open when the voltage exceeds the threshold.

Once the Ettercap attack is stopped, an operator can finally restart the RTAC's logic engine. The RTAC would then be able to detect the high voltage from the voltmeter and open the circuit breaker. However, in an operational power system with real-time demands, such delayed action is too late to prevent system damage. Figure 33 illustrates the state after re-enabling the RTAC logic engine. Although the

breaker eventually opens, it fails to prevent the significant damage already incurred in an operational power system.

6.5 Case Study II: Traditional PLCs

The case study involves the PLCs of two vendors: Schneider Electric's Modicon M221, and Allen-Bradley's MicroLogix 1400 & 1100. Unlike SEL RTAC (refer to Section 6.4), the PLCs do not have the security features to protect the communication and PLC device such as encryption and access-control.

6.5.1 Case Study II (a): Schneider Electric's Modicon M221

6.5.1.1 Controller Details

The Schneider Electric Modicon M221 is a nano Programmable Logic Controller (PLC) designed to control manufacturing processes. Control engineers can utilize the vendor-provided engineering software, SoMachine Basic, to write control logic, monitor the physical process, and manage the state of the M221. SoMachine Basic supports two IEC-61131 languages, namely, the Ladder Diagram (LD) and Instruction List (IL), for writing control logic. Engineers can download a control logic program to the PLC, effectively writing to the PLC's memory. Furthermore, engineers can start or stop the execution of the control logic, i.e., the logic engine on the M221, via SoMachine Basic. The communication between the M221 and SoMachine Basic is unencrypted, and it utilizes a proprietary protocol on port 502, encapsulated in Modbus/TCP. Notably, the M221 allows only one connection at a time.

6.5.1.2 Vulnerability

In addition to the fact that the network communication between the M221 PLC and its engineering software is unencrypted, we exploit two additional features of the

PLC. Firstly, the PLC's state, which enables or disables it from running the control logic program, can be changed remotely via the engineering software. Secondly, the PLC only permits one engineering software to connect to it at any given time.

6.5.1.3 MITRE ATT&CK

The case study utilizes the following attacks from the MITRE ATT&CK knowledge base:

Network Sniffing (T0842): As the initial step in launching the attack, network sniffing enables the attacker to monitor the network traffic between the engineering software, SoMachine Basic, and the PLC. This provides the attacker with critical information about protocol details that could assist in reaching her target.

Unauthorized Command Message (T0855): Given that communication with the PLC is unencrypted, the attacker can remotely send crafted messages to the PLC. By using the protocol information obtained from network sniffing, the attacker can construct a message that instructs the PLC to stop running the control logic program.

Loss of Availability (T0826): As M221 only allows a connection from one machine at a time, the attacker can render it unavailable for control engineers by maintaining the established session as part of her previous attack, thus not allowing any other connections.

Denial of Control (T0813): This attack is similar to the loss of availability. In this scenario, the attacker maintains the open session with the PLC, which prevents the control engineer from interacting with the process control.

Manipulation of Control (T0831): This attack represents the ultimate goal of the attacker. She employs network sniffing and unauthorized command messaging to stop the controller from running the control logic program, thus manipulating the PLC's control process.

6.5.1.4 Attack Implementation

Through differential analysis and manual efforts, we managed to reverse engineer the proprietary protocol of the M221 and identified the packets that the SoMachine-Basic software sends to start and stop the PLC's logic engine. The packets can be seen in Figures 35 and 36. Function codes `0x40` and `0x41` are used to start and stop the controller, respectively. If the request message from the engineering software is successful and accepted by the PLC, the PLC sends back a success message to acknowledge the change, as shown in Figure 37. Using this information, we wrote a Python script that first establishes a session with the Modicon M221 PLC, and then sends crafted messages to start or stop the execution of the control logic running on the PLC.

6.5.1.5 Experimental Settings

We evaluated the attack on a lab-functional model of a real conveyor belt used in an industrial environment. The details of the model can be seen in Figure 34. The conveyor belt sorts different types of objects with the help of sensors and manipulates them using an air solenoid. The system is controlled by the Modicon M221 PLC. The SoMachine Basic software runs on a Windows 7 Virtual Machine, while the attack scripts run on an Ubuntu 16.04 Virtual Machine. In our scenario, we assume that the attacker has infiltrated the ICS network, meaning the PLC, the engineering workstation, and the attacker's machine are all on the same network.

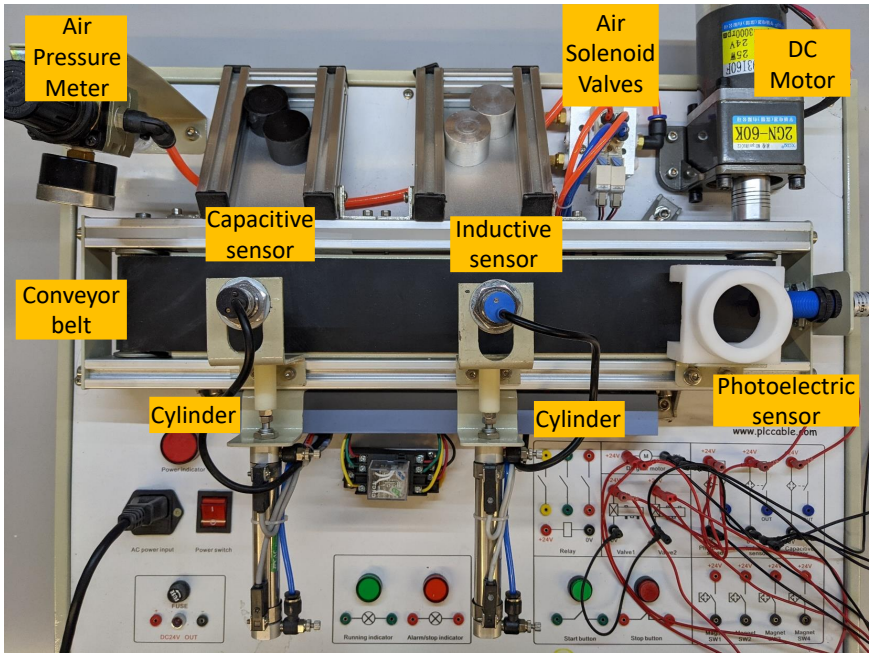


Fig. 34. Top-view of the fully-functional conveyor belt model

6.5.1.6 Attack Execution and Evaluation

Initiating with a network scan, the attacker identifies the IP address of the PLC, then proceeds to launch the attack program. The attacker first establishes a Modbus protocol session with the Modicon M221 PLC, then sends the 'Stop controller' request to the PLC. On receipt of this request, the PLC ceases to execute the control logic, leading to a halt in the physical process. As the PLC only allows one connection at a time, the control engineer is barred from communicating with the PLC and executing the 'Start controller' command, as long as the attacker maintains the Modbus session active.

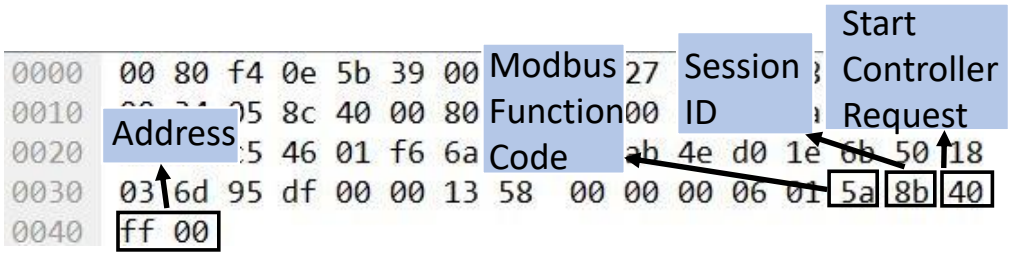


Fig. 35. Request message to "START" the Modicon M221 PLC

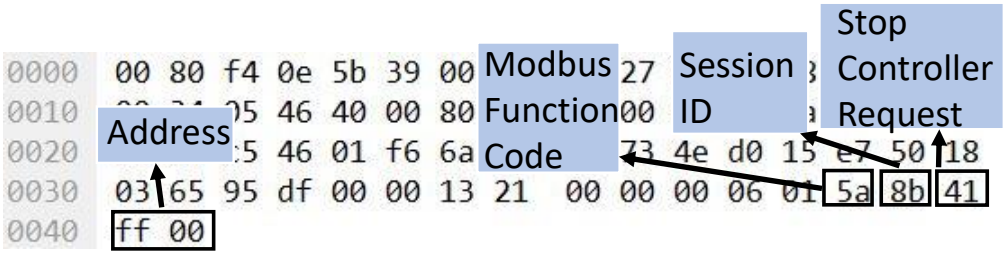


Fig. 36. Request message to "STOP" the Modicon M221 PLC

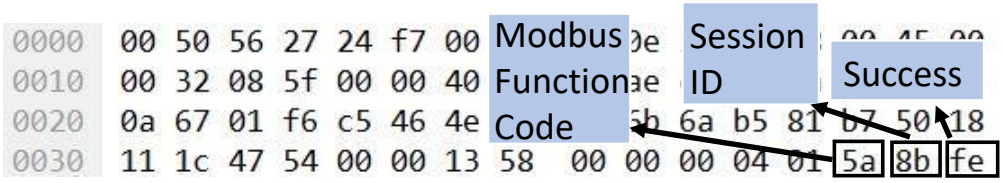


Fig. 37. Response from the Modicon M221 PLC with success function code

6.5.2 Case Study II (b): Allen-Bradley's MicroLogix 1400 & 1100

6.5.2.1 Controller Details.

Allen-Bradley MicroLogix 1400 and 1100 belong to the MicroLogix family and share several similarities. Both PLCs can be monitored and controlled using the RSLogix 500 engineering software, and they communicate with RSLogix using an unencrypted PCCC protocol encapsulated in EtherNet/IP [46]. RSLogix supports ladder diagram (LD) for writing the control logic. After establishing a connection with the PLC, the control engineer can upload the control logic (i.e., read the control logic running on the PLC) or download newly created logic to the PLC (i.e., write to the PLC's memory).

Both PLCs have three modes of operation: Run, Program, and Remote. In the 'Run' mode, the PLC executes the control logic and oversees the physical process. For maintenance or control logic changes, the PLC must be put into 'Program' mode, which allows for modifications. In 'Program' mode, the PLC's logic engine is paused and it does not execute any control logic as long as it remains in this mode.

The PLC's mode can be physically changed from the command line interface provided on both these PLCs. However, typically for operational convenience, it is set to 'Remote' mode, which allows the control engineer to remotely change the mode from 'Run' to 'Program', and vice versa, from the engineering software. Both PLCs utilize an unencrypted PCCC protocol encapsulated in EtherNet/IP for communication with the engineering software.

6.5.2.2 Vulnerability

We exploit an inherent functionality in MicroLogix PLCs to change operational modes to 'Run', 'Program', and 'Remote', where the PLCs in 'Program' do not

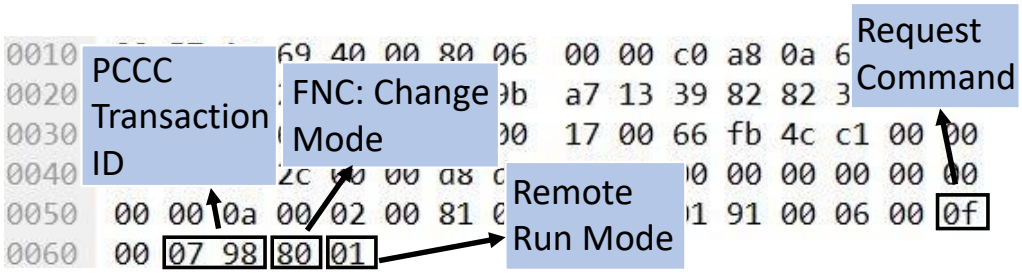


Fig. 38. Request message to set the MicroLogix 1400 PLC to Remote-Run Mode

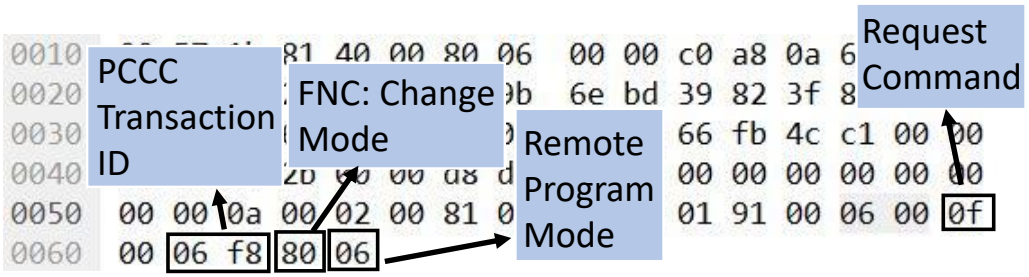


Fig. 39. Request message to set the MicroLogix 1400 PLC to Remote-Program Mode

execute control logic and wait for an operator to update their control logic and configurations.

6.5.2.3 MITRE ATT&CK

The following attacks from the MITRE ATT&CK knowledge base are utilized for the case study on MicroLogix PLCs.

Network Sniffing (T0842): The attacker employs network sniffing to identify the communication protocol between the PLC and its engineering software. Through this, she discerns the packets that are instrumental in changing the PLC state, which can potentially disrupt the running physical process.

Unauthorised Command Message (T0855): Using the information obtained,

she composes a well-crafted message capable of remotely altering the PLC state from 'run' to 'program', effectively halting the control logic program's execution.

Manipulation of Control (T0831): Resulting from the above attack, the attacker is successful in disrupting the control logic program's execution on the PLC, thereby halting the physical process it was controlling.

Denial of Control (T0813): As the attacker modifies the state of the PLC from 'run' to 'program', she temporarily obstructs the control engineers from interacting with the process controls.

Man in the Middle (T0830): In addition to changing the state of the PLC and halting the physical process, the attacker also aims to conceal this state alteration from the control engineer. Consequently, she employs ARP cache poisoning on the engineering software and the PLC and positions herself as a man-in-the-middle between her targets. This enables her to alter the PLC state from 'program' to 'run' when the engineering software requests to read the state.

Denial of View (T0815) Through the above man-in-the-middle attack, the attacker deceives the control engineer, who remains under the impression that the PLC is still in 'Run' mode and controlling the physical process. In reality, however, the process has been halted.

6.5.2.4 Attack Implementation

We successfully reverse-engineered the request messages that RSLogix sends to alter the PLC mode to Remote-Run or Remote-Program through manual processes. Figures 38 and 39 display the messages transmitted to set the PLC into Remote-Run and Remote-Program modes, respectively. As depicted in the figures, the function code `0x80` is employed to modify the mode. This function code is followed by either `0x01` for Remote-Run mode or `0x06` for Remote-Program mode.

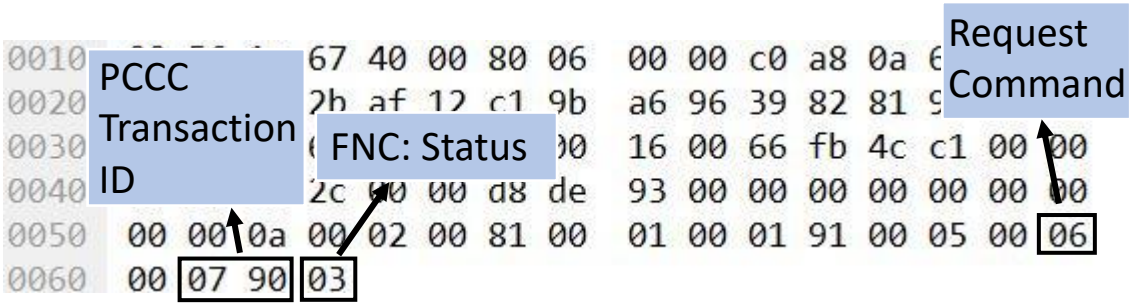


Fig. 40. Request message to sent the MicroLogix 1400 PLC to inquire current status

We also discovered that RSLogix software periodically queries the PLC's status. As demonstrated in Figure 40, the engineering software sends a request message featuring the function code `0x03` to inquire about the PLC's status. The differential analysis of response messages during Remote-Run and Remote-Program modes reveals that a function code of `0x21` is used for Remote-Run mode (Figure 41), while `0x26` is used for Remote-Program mode (Figure 42).

Leveraging this information, we developed a program that begins an ENIP session with the target PLC and dispatches the mode-change messages to place the PLC in Remote-Program mode, thereby halting the control logic's execution on the PLC. Given that the RSLogix software periodically checks the status, the control engineer can detect the mode change. Hence, to deceive the RSLogix software, we developed an Ettercap filter capable of identifying the status response message and altering the Remote-Program function code to Remote-Run.

6.5.2.5 Experimental Settings

Considering that both Micrologix 1400 and 1100 employ the same communication protocol and function codes, we conducted our logic engine attack test on a Micrologix 1400 connected to a lab functional model of an elevator in our setting. The

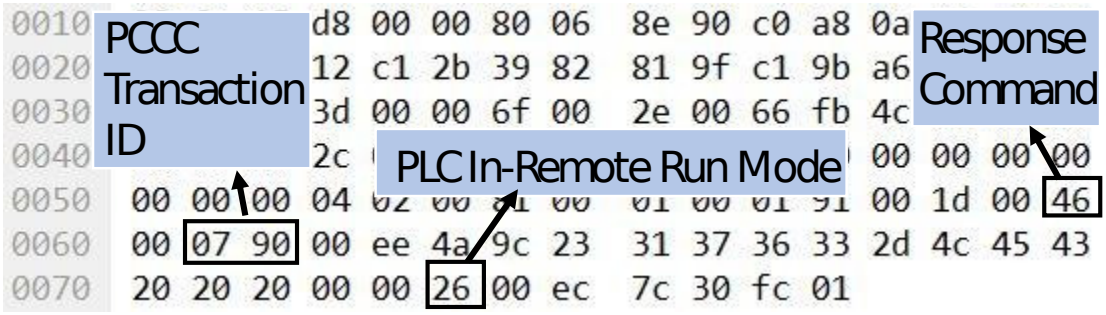


Fig. 41. Response message from the MicroLogix 1400 PLC when in Remote-Run mode

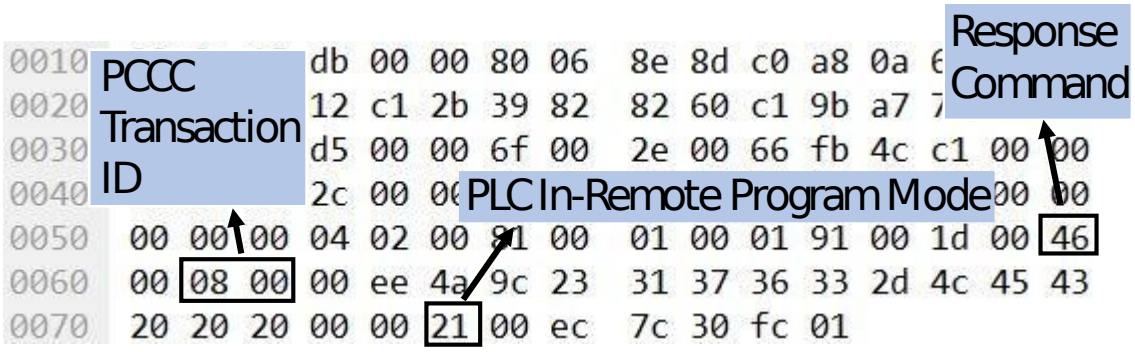


Fig. 42. Response message from the MicroLogix 1400 PLC when in Remote-Program mode

elevator model has four floors and operates identically to a real elevator, as depicted in Figure 43. A user can select a floor from both inside and outside the elevator to provide input to the PLC. In response, the PLC moves the elevator to the chosen floor.

RSLogix 500 software, which can communicate with the PLC, operates on a Windows 7 Virtual Machine (referred to as the engineering workstation). The attacker employs a machine running Ubuntu 18.04.3 LTS. As with previous experiments, the PLC, engineering workstation, and attacker machine are all on the same network.

6.5.2.6 Attack Execution and Evaluation

In carrying out the attack, the perpetrator first conducts a man-in-the-middle attack using ARP poisoning, followed by establishing an ENIP session with the Micrologix 1400 PLC that controls the elevator. The attacker then sends a request message to the PLC, instructing it to switch to Remote-Program mode. As a consequence, the PLC ceases to execute the control logic, leading to a halt in the operation of the elevator.

To prevent the control engineer from being informed about the current status of the PLC, the attacker deploys the Ettercap filters as described in the previous section. These filters modify the function code for Remote-Program mode to that of Remote-Run mode. Consequently, the control engineer remains oblivious to the ongoing attack, and the elevator operation remains disrupted.

6.6 Mitigation

The major vulnerability with most of the PLCs lies in their unencrypted communication. Even though RTAC uses TLS for most of its communications, the exchanges occurring on port 1217 are unencrypted. This unencrypted communication facilitates

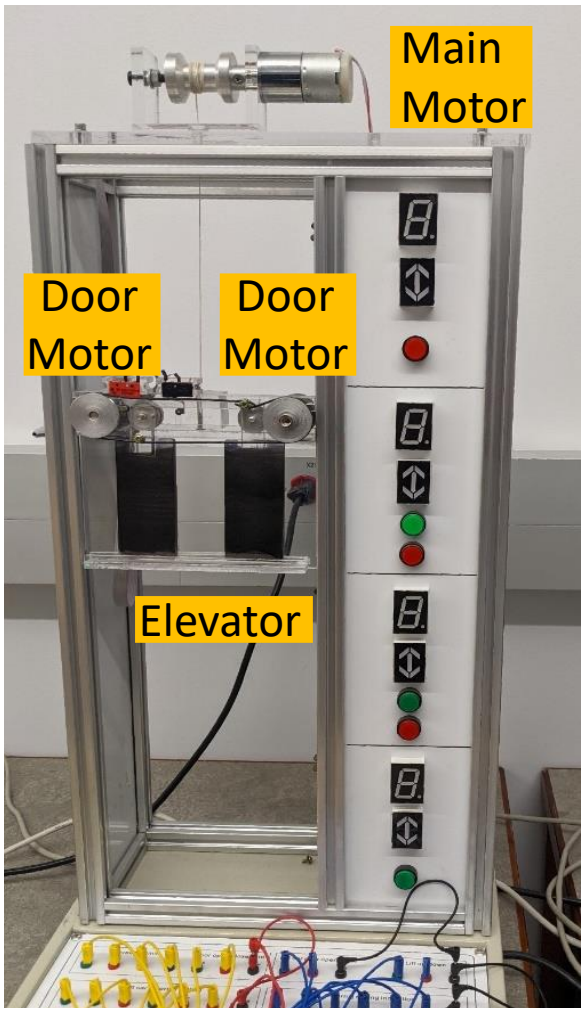


Fig. 43. Front-view of the fully-functional elevator model

protocol reverse engineering and eases the launching of attacks.

PLCs such as Modicon M221 and Micrologix 1400 incorporate some security layers, such as password protection, which safeguard the control logic from unauthorized reading and, in certain cases, writing. However, these security measures are inadequate since attackers can change the PLCs' operational state without requiring authentication, effectively preventing the execution of control logic. As such, it is recommended that password protections should be extended to cover PLC state changes as well.

Further, the M221 PLC exhibits a default feature that allows unauthorized users to establish connections without any authentication requirements. Also, it only permits one user to connect at a given time. This leaves room for attacks requiring an established connection with the PLC, such as those demonstrated in our study. An attacker can connect to the PLC, stop the controller from running the control logic program, and maintain the session active, effectively blocking legitimate field engineers from regaining control over the PLC. To prevent such attacks, a password protection scheme should be implemented for PLC connections.

Man-in-the-middle attacks, another prevalent threat, can be mitigated by implementing techniques such as DHCP snooping and ARP inspection [84]. These measures can significantly enhance the security of industrial control systems.

6.7 Conclusion

In this chapter, we introduced a novel category of control-logic attacks targeting Programmable Logic Controllers (PLCs). Our approach diverges from traditional tactics that inject malicious control logic into a PLC. Instead, we target the IEC-61131 control-logic engine, which is responsible for executing the PLC control logic. We utilized attack methods from the MITRE ATT&CK knowledge base to illustrate our control logic engine attacks through case studies involving four real PLCs, namely the SEL-3505 RTAC, Modicon M221, and MicroLogix 1400 and 1100.

These case studies effectively halted the operations of three distinct physical processes in real-world scenarios: a power substation, a conveyor belt, and an elevator. By demonstrating these attacks, we aim to enlighten the industrial control systems (ICS) research community and the broader industry about the potential attack vectors related to the control logic engine. This knowledge can foster the development of stronger security measures to protect these vital components of industrial infras-

tructure.

CHAPTER 7

USING VIRTUAL PLC PLATFORM AS A HONEYPOT FOR ICS THREAT INTELLIGENCE

This chapter presents my fifth contribution, focusing on the utilization of a virtual PLC platform as a honeypot for gathering threat intelligence. Programmable Logic Controllers (PLCs) play a vital role in industrial control systems (ICS), managing critical processes in manufacturing and power generation. To counter increasingly sophisticated cyberattacks, the security community relies on PLC honeypots to gather information about attackers' tools and techniques.

Existing PLC honeypots have limitations in effectively engaging advanced attackers or providing comprehensive functionalities. This chapter introduces the virtual PLC platform (VPP) as a protocol-agnostic and scalable PLC honeypot that accurately emulates real PLCs. In addition to the standard features of the virtual PLC platform, I added additional functionalities such as identifying the location of function codes used by actual PLCs and mapping them with application-level operations. These modifications enable advanced capabilities like control logic transfer, PLC authentication, and different PLC modes, enhancing the `VPP`'s ability to engage attackers effectively.

The chapter demonstrates the VPP's capability to emulate various PLCs and provides experimental evidence of its ability to replicate a wide range of functional and operational features. Furthermore, a case study involving an elevator lab model showcases the VPP's effectiveness in engaging attackers and storing attack data for subsequent analysis. This innovative honeypot framework significantly advances the

field of threat intelligence for ICS by facilitating more efficient engagement with attackers and generating actionable insights to enhance system security and resilience.

7.1 Introduction

Programmable Logic Controllers (PLCs) are crucial to modern industrial control systems (ICS), directly monitoring and controlling various processes such as manufacturing, power generation, and chemical processing. They execute user-defined control logic based on sensor input signals and send output to actuators. PLCs also communicate with other systems like human-machine interfaces (HMIs) and supervisory control and data acquisition (SCADA) systems. Due to their critical role, PLCs have become prime targets for increasingly sophisticated cyberattacks[**clinjection**, **doe**, 85, 86, 8, 87, 32], requiring the security community to develop effective threat intelligence capabilities to stay ahead of attackers.

PLC honeypots are one such solution that can help security professionals in gathering valuable threat intelligence by attracting and monitoring attackers targeting ICS. While physical honeypots offer effective intelligence gathering, they come with high hardware and deployment costs. While physical honeypots are an effective means of gathering threat intelligence, they can have a high hardware and deployment cost[88]. Fortunately, many researchers have focused on developing virtual honeypots [89, 90, 91, 92, 93, 94, 95] that can be used to simulate the behavior of a real PLC. There are two types of virtual honeypots: low-interaction and high-interaction. Low-interaction honeypots simulate only parts of a real system's functionality, making them easy to set up and resource-efficient but limited in functionality and easily detectable by attackers. High-interaction honeypots aim to emulate the entire PLC system, including hardware, software, and network environments, offering more comprehensive data on attacker behavior but requiring greater resources and expertise to

set up and maintain.

However, existing ICS honeypots have limitations that hinder effective engagement with attackers and useful threat intelligence collection. Low-interaction honeypots are unable to engage and attract sophisticated attackers, while high-interaction ones lack operational-level functionalities such as control logic transfer, PLC modes, and support for different PLC functions. Due to the lack of operational-level support, most high-interaction honeypots can be easily identified and may not be able to engage the attacker for longer sessions. To address this, more advanced honeypots with operational-level capabilities are needed for effective engagement.

In this chapter, I demonstrate the use of a virtual PLC platform as a protocol-agnostic, scalable PLC honeypot that can mimic a real PLC by replaying network dumps. To achieve this goal, I have added new features to the PLC template, such as function code identification and mapping. Furthermore, the virtual PLC includes two additional components: "PLC state" and "Data storage," which track the PLC's state and gather data for threat intelligence. With these enhancements, the virtual PLC platform can effectively function as a honeypot and provide several application-level features comparable to those of a real PLC. There are three main contributions of this chapter:

- We introduce the application of the virtual PLC platform, as a scalable and protocol-agnostic honeypot framework capable of providing application-level features.

- We demonstrate `VPP`'s ability to mimic different PLCs and provide experimental evidence of its capabilities to replicate various functional and operational features.

- We present a case study using a lab model of an elevator with `VPP`, performing

various ICS attacks on it, showcasing its capacity to engage the attacker and store attack data.

7.2 BACKGROUND AND RELATED WORK

7.2.1 Operational and Functional Features of PLC

PLCs monitor and control the physical process. The user can use the engineering software to configure and program a plc. Along with transferring the control logic, the PLC also has some other operational and functional features:

- **Authentication:** PLCs offer password protection for login and state changes.
- **Report I/O data:** Connected to engineering software or HMI, PLCs reply to I/O-related requests to show the ICS state.
- **PLC modes:** PLCs have modes like "Run" and "Program." "Program" mode allows control logic writing, while "Run" mode executes control logic and blocks write memory operations.
- **Session Establishment:** PLCs accept remote connections and have session establishment protocols in their firmware.
- **Session Maintenance:** PLCs respond to incoming requests for operations like read, write memory, authentication, mode change, and reading I/O data.

7.2.2 Related Work on ICS Honeypots

Honeypots are decoy systems designed to replace real systems, attracting and engaging attackers. They can be classified into low-interaction or high-interaction honeypots based on the functionality and level of engagement they provide. Numerous PLC honeypots have been developed for ICS [89, 92, 90, 91, 93, 95, 94]. Among

Table 23. Summary of existing PLC honeypots in literature and their features ●= Complete Implementation ◐= Partial Implementation ○= No Implementation

Honeypot	ICS Protocols			Required	Standard Operational and Functional Features							Device Discovery		Additional IT Services		
	Modbus	ENIP	S7comm	ICS Protocol Libraries	Control Logic Upload	Control Logic Download	Authen-tication	Exchange I/O Data	PLC mode	Session Establisment	Session Maintaince	Engineering Software	Network Scanning	HTTP	FTP	SNMP
Conpot	◐	◐	◐	YES	○	○	○	○	○	○	○	○	◐	●	○	●
HoneyNet	◐	○	○	YES	○	○	○	○	○	○	○	○	◐	●	●	●
ICSpot	◐	○	●	YES	●	●	○	●	○	●	●	●	●	●	○	●
NeuPot	◐	○	○	YES	○	○	○	●	○	○	○	○	○	○	○	○
HoneyPLC	○	○	●	YES	●	●	○	○	○	●	●	●	●	●	○	●
CryPHL	○	○	●	YES	○	○	◐	○	○	◐	◐	◐	○	●	○	●
s7CommTrace	○	○	●	NO	○	○	○	○	○	◐	○	○	●	●	○	●

the existing honeypots, HoneyPLC [91] and ICSpot [94] are closest to VPPin terms of the functionalities they provide. HoneyPLC supports control logic upload and download functionality for Siemens PLCs and can establish and maintain a session with STEP7 engineering software. ICSpot, based on HoneyPLC, provides all the features of HoneyPLC and includes dedicated modules for simulating I/O data from the physical process. Table 23 summarizes our assessment of various features and capabilities of existing honeypots, such as protocol coverage (the number of different protocols a honeypot supports), control logic transfer (whether the honeypot can upload and download control logic), device discovery (if the honeypot is identified as a real PLC by engineering software and other network scanning tools), and lastly, the operational and functional features the honeypot provides (application-level interaction).

7.2.3 Limitations of State-of-the-art Honeypots

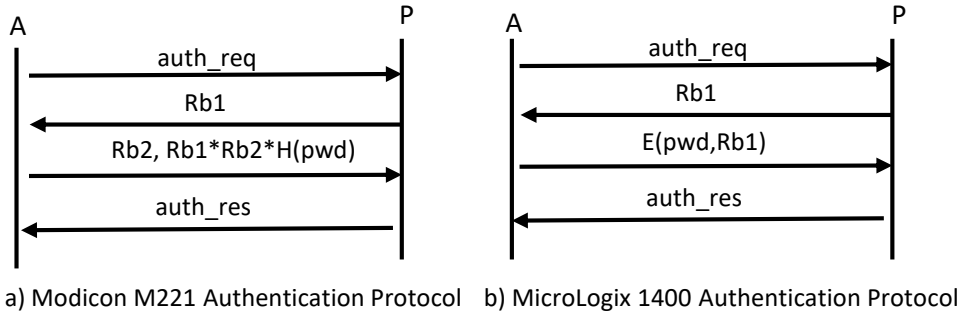
State-of-the-art honeypots have limitations that reduce their ability to engage attackers and collect valuable threat intelligence data. Some of these limitations are: **Limited coverage (L1)**: Current honeypots, both low and high interaction, cover a limited number of PLCs. They rely on libraries and frameworks for specific PLCs, developed through reverse engineering ICS protocols. HoneyPLC, CryPLH, and ICSpot

use the Snam7[96] framework for S7comm server simulation, while many Modbus-based honeypots depend on open-source Modbus libraries such as libmodbus[97]. These dependencies restrict the supported PLC range.

PLC Memory Manipulation (L2): Major ICS attacks, like Stuxnet[9], aim to manipulate PLC memory. Attackers seek to write malicious control logic or malware to disrupt the physical process. However, only HoneyPLC and ICSpot offer control logic download capability. The absence of control logic capture in most honeypots limits their attack surface and threat intelligence generation, making them unable to detect various attack types.

Lack of Operational and Functional features (L3): PLCs offer features like authentication, modes, session establishment, and maintenance, which enhance the attack surface and enable better attacker engagement. However, inaccurate emulation of these features may reveal honeypot identities. ICSpot and HoneyPLC support application-level sessions, CryPLH and S7CommTrace partially support Siemens PLCs, but none support PLC mode features. Only CryPLH offers authentication, while ICSpot (S7comm-based) and NeuPot (Modbus-based) provide I/O data exchange.

In light of these limitations, current honeypots exhibit reduced efficiency in engaging with attackers and gathering threat intelligence. Consequently, there is a pressing need for a dynamic, scalable honeypot capable of replicating the diverse operational and functional features characteristic of real PLCs.



a) Modicon M221 Authentication Protocol b) MicroLogix 1400 Authentication Protocol

Fig. 44. Client Authentication Protocol used by different PLCs

7.3 Virtual PLC Platform - Enabling Application-level PLC Functionalities at Scale

7.3.1 Challenges in Developing a Scalable Honeypot

Developing a scalable and dynamic honeypot for different PLC operational features presents various challenges.

Diverse Application logic/Implementation (C1): Heterogeneity in PLC application logic complicates honeypot functionality at the application level. Without standardized formats or APIs, application logic differs among manufacturers, as seen in the authentication protocols of Schneider Electric's Modicon M221 and Allen-Bradley MicroLogix 1400 (Figure 44). Variations in hashing algorithms and mechanisms [14] pose challenges in replicating application logic.

Proprietary PLC protocols (C2): Honeypots face challenges in replicating application-level functionalities due to proprietary ICS protocols used by PLCs for network communication. For instance, M221 employs the UMAS protocol within Modbus, whereas MicroLogix 1400 uses the PCCC protocol encapsulated by ENIP protocol. Honeypots need knowledge of PLC protocol fields and semantics, posing a significant research challenge.

Lack of PLC State Machine knowledge (C3):PLCs possess various programming modes or states, like "Run" and "Program" modes, each having distinct state-switching mechanisms. For instance, MicroLogix 1400 directly enters "Run" mode after control logic download completion and begins execution, while Modicon M221 requires a user command ('Start') in an intermediate state to initiate control logic execution. Honeypots need to accurately implement PLC state machines to deceive attackers, but the lack of public documentation requires extensive manual experimentation.

7.3.2 Virtual PLC Platform Framework

To address the limitations and challenges, we introduce the application of a virtual PLC platform as a honeypot. As demonstrated in 3.4.2.1, the virtual PLC platform is capable of identifying session-dependent fields in various ICS protocols and effectively replaying network traffic, addressing challenge C2. To overcome challenges C1 and C3 and handle application-level PLC functionalities, a new module called "Function Code Mapping" is incorporated into the PLC template. Additionally, the virtual PLC is enhanced with a PLC state module, which showcases changes in the PLC's state to the attacker. The modified framework of the Virtual PLC Platform is illustrated in Figure 45.

7.3.2.1 Handling PLC Functionalities (C1 & C3)

The Virtual PLC Platform addresses heterogeneous application logic implementations across PLCs by providing an abstraction layer. This approach is based on observations of numerous functional and operational features. For example, Figure 44 shows the authentication protocol of Modicon M221 and Micrologix 1400 PLCs. In both instances, the PLC merely sends the initial random number (seed) and ver-

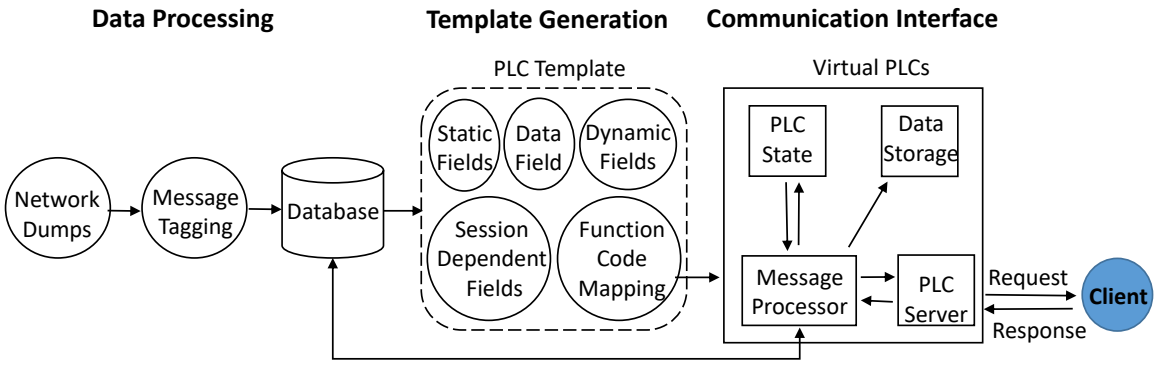


Fig. 45. An overview of virtual PLC platform

ifies the received password hash. These operations can be managed by replaying old network messages, negating the need for application implementation in the VPP. Consequently, the VPPremains agnostic of application logic implementation and can effortlessly mimic various operational and functional features by replaying suitable response messages.

To achieve this, the VPPinitially identifies request and response messages corresponding to diverse operational features in the network dump and subsequently replays them according to the live session. This is done by first identifying the function or command code present in most ICS protocols and then correlating these codes with various operational features. The VPPleverages this information when interacting with the client and responding to different requests based on their function code. Figure 46 shows the process of identifying function code and mapping it to different operational features.

Function Code: Function codes in ICS protocols are used to issue commands for various tasks on PLCs. Manufacturers assign unique function codes to operations like reading or writing data in memory, starting or stopping the PLC, and running diagnostics.

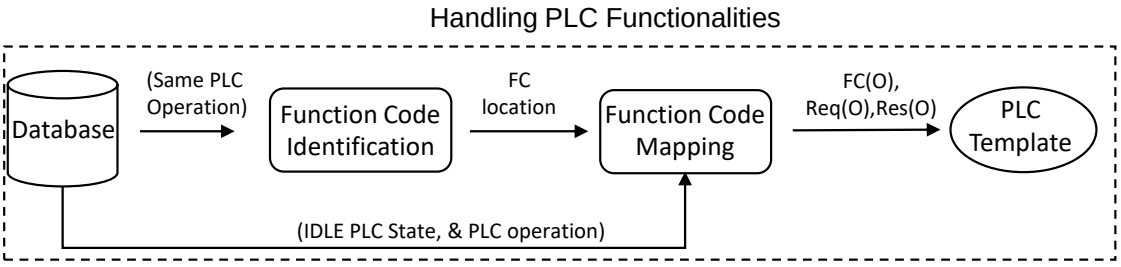


Fig. 46. Process of identifying and mapping function codes with different operations

When the PLC receives a request message from engineering software, it processes the message based on the function code value. A valid function code triggers the associated code execution and a successful response. Otherwise, an error message is sent. For example, Figure 47 shows the message exchange between an M221 PLC and SoMachineBasic engineering software. To upload control logic or read data from the PLC memory, the software sends a request message with a "28" function code, memory address, address type, and byte quantity to read. With the correct structure, function code, and required information, the PLC performs a "Read" operation and sends the requested data to the engineering software using the function code "fe," indicating a successful operation.

Function Code Identification In a session between engineering software and a PLC, the function code field can have multiple values. Unlike other dynamic fields in ICS messages, such as "Transaction ID" that exhibit a fixed pattern with a unique increment in each new message, the function code field lacks a consistent pattern. This absence of a pattern makes it challenging to locate the function code field using pattern recognition techniques or simple differential analysis. To overcome this challenge, we have developed our own heuristics based on our knowledge of the ICS domain.

Observation: Through our observations, we have noticed that each PLC sup-

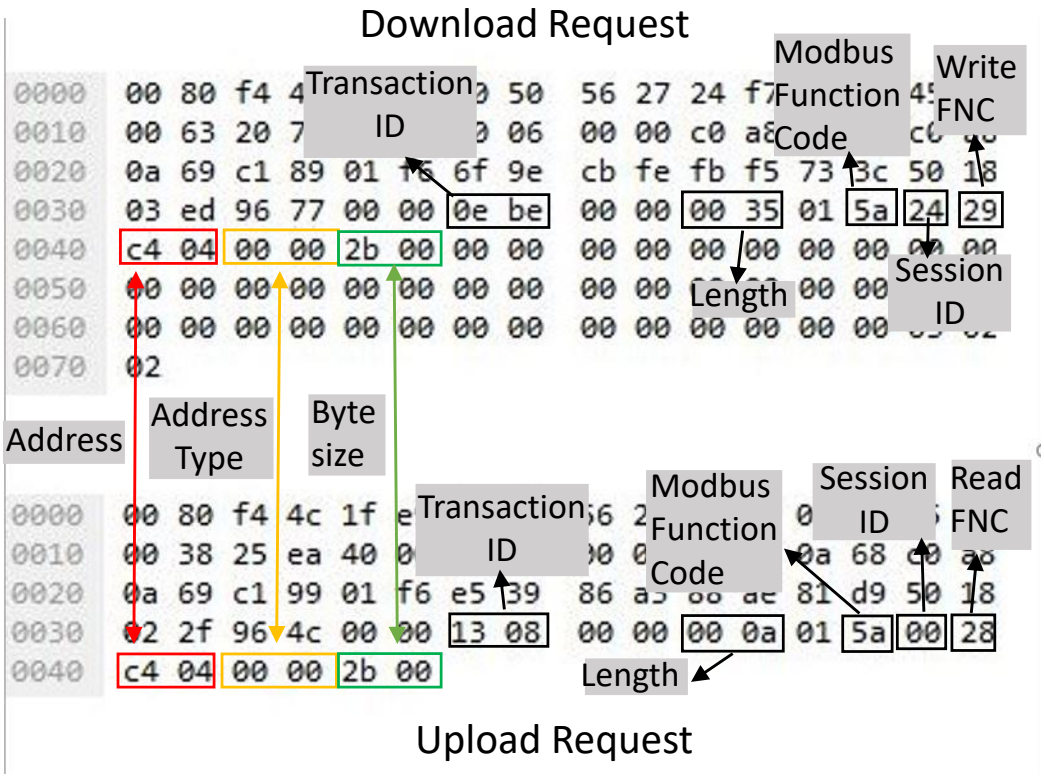


Fig. 47. Request-Response message to read and write a control logic on M221 memory

ports a limited number of function code values, some of which are used repeatedly during a session. Additionally, when the engineering software sends a request to the PLC, the PLC responds with a "success" or "failure/error" message to indicate whether the request has been processed. For example, when downloading control logic to the PLC memory, the engineering software sends multiple "READ" requests (FC:29) to the PLC, and the PLC responds to each request with either a success (FC:fe) or a failure (FC:FF) message.

Heuristic: Based on our observations, we propose a heuristic for identifying the location of the function code within an ICS message. We divide the request and response messages into separate groups. Since the function code is typically represented by one byte, we compare the messages within each group byte by byte

and determine the number of unique values at each location. The location of the function code is identified by an index that shows a variation within a given threshold in the request messages and has at most two unique values in the response messages.

Function Code in UMAS Protocol: To locate the function code field in the UMAS protocol used by the M221 PLC, we established a session with the PLC using the engineering software and performed an "Upload" operation. By capturing the network dump and dividing the request and response messages into two groups based on IP address and Port numbers, we compared the messages byte by byte. Indices in the request message group showing minimal changes below a specified threshold were labeled. Similarly, in the response message group, we labeled indices with at most two values (e.g., success and failure). Comparing the function code candidates from both groups, we selected the common index as the function code location.

Function Code Mapping Once the location of the function code in a message is identified, the next step is to map the function codes to their corresponding application-level functionalities. To accomplish this, we conducted manual operations on the PLC, such as changing the PLC's state from start to stop and vice versa, enabling or disabling password protection, and performing successful and unsuccessful authentication attempts. During each operation, we captured the network communication between the PLC and the engineering software.

From each network capture, we extracted all the function codes used during that session. By comparing the function codes extracted from the application-level operations with those from a benign session (a simple connection to the PLC), we eliminated the function codes common to both. The remaining function codes were then associated with specific application-level PLC functionalities. This mapping information was stored in the PLC template.

By following this process, we successfully identified the function codes linked to various application-level PLC functionalities and incorporated this information into the PLC template.

7.3.2.2 Handling the state of the PLC

Maintaining the state of the virtual PLC is crucial for emulating a real PLC and providing realistic behavior. To achieve this, two components were added to the virtual PLC platform: PLC State and Data Storage. Additionally, the Message Processor was modified to update the PLC state upon receiving requests from users.

PLC State: The PLC State is a critical module in the `VPP` framework. It stores the initial state of the virtual PLC, including parameters such as the PLC mode and password. This component dynamically adjusts the state based on client requests, allowing users to remotely modify PLC functionalities. This capability enhances the deception aspect of `VPP` by providing a convincing environment for potential attackers. By adapting the PLC state according to client interactions, `VPP` ensures flexibility and realistic responses, thereby improving its effectiveness in threat detection and prevention.

Data Storage: Data Storage is another essential component within the `VPP` architecture. It is responsible for preserving all communication between the virtual PLC and clients. This repository of interaction data serves as a valuable resource for generating threat intelligence and performing forensic analysis. The captured data can be analyzed to identify patterns indicative of malicious activity, investigate security incidents, or gain insights into the techniques and strategies employed by attackers. By maintaining a comprehensive log of communications, the Data Storage component supports both real-time threat response and post-incident investigations.

7.4 Evaluation

Existing PLC honeypots fail to provide application-level interaction and operational features like a real PLC. These features are essential for expanding the honeypot's attack surface and gathering meaningful threat intelligence. In this section, we evaluate VPP's capability to support these features.

7.4.1 Experimental Setup and Methodology

Our experimental setup included Allen-Bradley Micrologix 1400, Micrologix 1100, and Schneider Electric Modicon M221 PLCs. We used SoMachineBasic and RsLogix 500 engineering software, running on a Windows 10 VM (engineering workstation), to configure and program the Schneider Electric and Allen-Bradley PLCs. The VPP ran on an Ubuntu 18 VM, and all devices were on the same network. To evaluate VPP's various functions, we first executed a targeted function on the real PLC using the engineering software and captured the network traffic. Then, we provided the network dump to VPP and performed the same function on it. Finally, we assessed if VPP could deliver the same features and functionalities as the real PLC.

7.4.2 Device Discovery

To establish communication with a PLC, the first step is discovering it on the network. It is crucial for a honeypot to be scannable and identifiable as a PLC. The PLC can be discovered on the network using engineering software such as RSLogix and SoMachineBasic. We tested VPP's discoverability using these softwares.

Methodology: In this experiment, first we discover the real PLC using the RSlogix software and capture the network communication between the PLC and the engineering software. To connect these PLCs to the engineering software, we had to manually

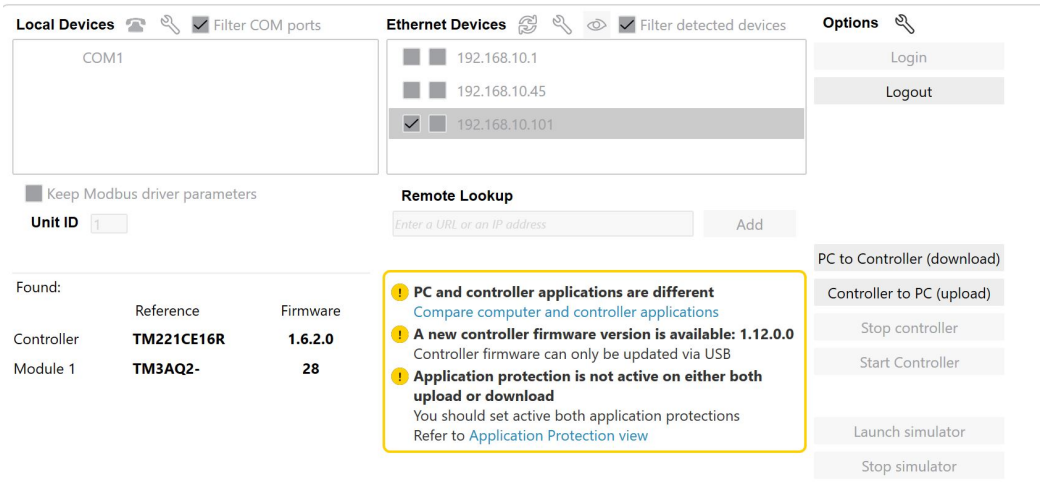


Fig. 48. Virtual PLC Platform identified as a real Modicon M221 PLC in SoMachineBasic

configure a driver in the RSlinx Classic using the IP address of the PLC. Then we provide the captured network dump to the `VPP` along with the PLC template and started the `VPP` server. In the end, we again used the discovery function available in the engineering software to identify the IP address of the `VPP`.

Evaluation Criteria: For this experiment, we set the criteria that `VPP` should be discovered by RsLogix as a real MicroLogix 1400 and 1100 PLC, and by SoMachineBasic as a Modicon M221 PLC.

Results: The `VPP` was identified by the RsLogix as a real Mircologix 1400 and MicroLogix 1100 PLC. Whereas SoMachineBasic identified it as a Modicon M221 PLC. As shown in figure 48.

7.4.3 Operational and Functional Features

7.4.3.1 Session Establishment and Maintenance

PLCs, acting like servers, must establish and maintain communication sessions. After verifying that `VPP` is a genuine PLC, we tested its ability to establish and

maintain communication sessions. The VPP should also send appropriate responses to various messages from engineering software during a session.

Methodology: We used network dumps captured from a real plc to initialize and run VPP for each PLC, then used engineering software to discover and establish communication sessions. We ran multiple experiments with varying session durations, recording request-response messages.

Evaluation Criteria: We set these criteria: 1) VPP should handle various message types; 2) VPP should send appropriate responses for each request; 3) VPP should maintain sessions of different lengths (5, 15, 30 mins); 4) No timeouts or disconnects should occur.

Results: VPP successfully established and maintained communication with engineering software in all experiments. As shown in Table 24, VPP managed communication sessions for set durations. For all three PLCs, VPP successfully responded to hundreds of requests in 5, 15, and 30-minute sessions, handling 9 unique function codes for Modicon M221 and 2 unique function codes for both the Micrologix PLCs. The number of unique function codes remained the same for all sessions because engineering software sends the same request messages repeatedly in idle conditions, inquiring about the PLC state and reading IO values.

7.4.3.2 Authentication:

Many PLCs provide an authentication feature where a user can set a password. When the user tries to establish a communication session or perform any of the critical operations such as writing the control logic, the PLC asks for the password that was set earlier. If the password is correct the PLC allows the operation otherwise it denies it and sends an error message. Since the implementation of the authentication mechanism can be different in different PLCs, it is difficult to enable this in the

Table 24. Summary of messages exchanged between a VPP and the engineering software for various sessions.

PLC	Session Length (min)	# of req msg	# of res msg	# of unique FC	Session Timeout/ Disconnects
MicroLogix 1100	5	8153	8153	2	0
	15	24460	24460	2	0
	30	48926	48926	2	0
MicroLogix 1400	5	10569	10569	2	0
	15	31720	31720	2	0
	30	63460	63460	2	0
Modicon M221	5	2678	2678	9	0
	15	8098	8098	9	0
	30	16126	16126	9	0

VPP without manual reverse engineering which undermines its scalability. Therefore, instead of developing the authentication mechanism, similar to CryPLH[92], VPP provides an abstraction to the attacker i.e for each authentication request the VPP can accept or deny it without verifying the password. To implement this, the user can enable or disables the authentication and set number of the authentication attempts in the PLC template. Whenever the VPP receives any authentication request it responds with a success or failure response based on the template.

Methodology: Initially, we set password authentication on the real Modicon M221 PLC using engineering software and attempted authentications with correct and incorrect passwords, capturing network communication. Next, we identified messages and function codes for authentication using the method in 7.3.2, adding this information to the PLC template. Finally, we ran VPP and made some authentication attempts.

Evaluation Criteria: For this experiment, we set the criteria that VPP should respond to authentication requests and approve or disapprove them according to the PLC template.

Results: In our experiments, VPP successfully managed authentication requests according to the PLC template. Additionally, it captured the attacker's password hash, which can be used for forensic analysis.

7.4.3.3 PLC Modes

PLC mode, an operational feature altered using engineering software, can be exploited to disrupt physical processes, as noted by Syed et al. [10]. It is crucial for VPP to replicate PLC modes to effectively engage attackers. We evaluated this on Micrologix 1400, 1100, and Modicon M221 PLCs.

Methodology: We established sessions with real PLCs and changed their modes using the engineering software i.e "Run" to "program" (for Micrologix 1400 and 1100) or "start" to "stop" and vice versa for Modicon M221. We captured the network communication, identified functions and messages for PLC modes, and updated the PLC template. Finally, we tested mode changes on `VPP` multiple times.

Evaluation Criteria: 1) `VPP` should respond to mode change requests and support all modes. 2) `VPP` should update and return its mode based on user requests.

Result: `VPP` successfully changed its mode in each cycle and maintained connection without disruption, indicating its ability to engage attackers targeting PLC modes. Additionally, `VPP` logged communication for potential threat

7.4.3.4 Control Logic Download

Control logic download is a prime target in PLC cyber attacks, where attackers aim to disrupt physical processes by injecting malicious control logic. Thus, it is crucial for `VPP` to allow users to download malicious control logic and store it for threat intelligence gathering. We tested `VPP`'s capability to handle control logic downloads.

Methodology: We initiated `VPP` with the Modicon M221 PLC template and established a communication session using SoMachineBasic software. We then downloaded control logic programs of varying sizes and complexities onto `VPP` using the engineering software, capturing the network communication for evaluating the upload functionality.

Evaluation Criteria: The evaluation criteria required `VPP` to handle all control logic download messages without timeouts or session disconnects, ensuring the user receives confirmation of control logic download from the engineering software.

Results: Our experiments demonstrated that `VPP` successfully handled all download requests. As seen in Table 25, we downloaded 40 control logic programs to the `VPP`.

Table 25. Summary of control logic download operations on M221 PLC

File Size (Kb)	# of Programms	Avg no. of Rungs	Avg no.of Instructions	Total Req Msgs	No. of Write msg	No. of Connection Timeouts
60-80	24	2.75	10.75	9050	1126	0
81-90	5	3.8	10.2	1888	255	0
91-100	5	9	25.8	1996	245	0
101-120	3	10.66	26.66	1178	151	0
120+	3	17.3	77.66	1176	173	0
Total	40	-	-	15288	1950	0

During these operations, it received 15,288 request messages, including 1,950 messages to write control logic on PLC memory. `VPP` effectively responded to all messages and stored the communication, which can be used to extract the control logic binary

7.4.3.5 Control Logic Upload

Control logic upload is an essential feature for `VPP` to support, as it is often used in cyber attacks [79, 29]. Attackers perform reconnaissance by uploading or reading control logic from the PLC memory, gaining insights into the physical process's inputs, outputs, and current state. This allows them to create efficient and impactful malicious control logic. Additionally, after downloading malicious control logic to the `VPP`, attackers may perform upload operations to verify that the desired malicious control logic is running on the PLC.

Methodology: To evaluate `VPP`'s control logic upload functionality, we began by running `VPP` with the M221 template and providing it with network dumps captured during the control logic download experiments. We then used the engineering soft-

Table 26. Summary of control logic upload operations and the transfer accuracy on M221 PLC

File Size (Kb)	# of prog.	Total Rungs		Total Instructions		Total Req Msgs	No. of Read Msgs	# of Connection Timeouts
		Upload	Download	Upload	Download			
60-80	24	66	66	258	258	5124	1228	0
81-90	5	19	19	51	51	917	265	0
91-100	5	45	45	129	129	894	249	0
101-120	3	32	32	80	80	516	141	0
120+	3	52	52	233	233	509	168	0
Total	40	214	214	751	751	7960	2051	0

ware's upload functionality to read the control logic that was previously downloaded to the PLC. Finally, we compared the downloaded and uploaded control logic to assess the transfer accuracy. For each control logic, we manually compared individual rungs and instructions.

Evaluation Criteria: The evaluation criteria for this experiment are: 1)`VPP` must handle all read control logic requests, 2) engineering software must successfully upload the entire control logic from `VPP` without errors or session disconnects, and 3) the uploaded control logic should match the one previously downloaded, with the same rungs and instructions in order.

Results: As shown in table 26, our evaluation demonstrates that `VPP` successfully uploaded all 40 control logic files using the network dump. Additionally, we compared the uploaded control logic program with the original program, and they had identical rungs and instructions in the same order.

7.5 Case Study: Virtual PLC Platform for Elevator

After evaluating various application-level functionalities of VPP, we conducted a case study involving multiple cyberattacks found in the literature. This study assessed the VPP' ability to engage with attacks and collect attack artifacts. We also analyzed the data gathered by the VPP to detect traces of executed cyberattacks.

Experimental Setting & Methodology: In this case study, we used a lab model elevator controlled by a Modicon M221 PLC. A Windows 10 VM (engineering workstation) with SoMachineBasic was employed for programming and monitoring the PLC, and an Ubuntu 20 VM (attacker VM) executed various attacks. All devices shared the same network. We collected network dumps by performing operations on the M221 PLC to create a PLC template. Then, the VPP was started using these dumps and the M221 template. We conducted multiple ICS attacks on the VPPindividually, capturing network communication. Finally, we analyzed the network dumps for attack evidence and the attacker's footprint.

7.5.1 Cyber Attacks On Virtual PLC Platform

We executed multiple ICS attacks on the VPP:

Control Logic Injection Attack The control logic injection attack [72] is a type of cyber attack where an attacker attempts to download harmful control logic onto the memory of a PLC in order to disrupt the physical process being controlled by the PLC. This can be achieved through the use of engineering software specific to the targeted PLC model or through the use of custom scripts that repeatedly send "write" requests to the PLC with the malicious payload.

Control Logic Theft Attack (Reconnaissance) The Control Logic Theft Attack (Reconnaissance) [79, 25]is a type of cyber-attack that involves gathering information about a physical process through the analysis of the control logic written on the Programmable Logic Controller (PLC) responsible for controlling the process. The attacker's aim is to gain a better understanding of the physical process, which can then be used to develop customized attacks targeting the specific process. This attack typically involves the reconnaissance phase, which is a standard initial step in most cyber-attacks, where the attacker seeks to learn more about the target system. The information obtained through the Control Logic Theft Attack can be used to develop sophisticated attacks, which may result in significant damage to the physical process.

Control Engine Attack A Control Engine Attack [10]is a type of cyber-attack that differs from other attacks targeting the control logic running on a Programmable Logic Controller (PLC). Instead of targeting the control logic itself, the attacker targets the control engine responsible for executing the control logic on the PLC. To download new control logic onto the PLC, the PLC must first halt the execution of the current program and enter a "program" or "stop" state. The attacker can exploit this feature of the PLC and send a command to change the state of the PLC, disrupting the physical process.

Brute Force Authentication Lastly, the data collection capability was tested by performing a brute force authentication attempt. This can be achieved by using the engineering software of the PLC or by sending authentication request messages to the PLC through a Python script. The objective of this attack was to determine the effectiveness of `VPP` in detecting and logging authentication attempts made by

unauthorized users. By performing a brute force authentication attempt, the VPP was tested for its ability to detect and respond to malicious access attempts.

subsectionAnalysing the Forensic ArtifactsAfter executing attacks on VPP and capturing the communication, we analyzed the network dumps to identify the attacker's footprints. This analysis helped determine the attacker's methods and actions during the attack. The information can be used to reconstruct the attack and assess its impact on the system.

Identifying Control Logic Injection and Reconnaissance Attacks: We analyzed network dumps from the control logic injection and reconnaissance attacks using Eupheus, a control logic decompiler by Sushma et al. [29]. Eupheus converts control logic binaries to Instruction List format. We identified write request messages with function code '29' to extract the control logic binary from the network dump. To recognize control logic theft attacks, we looked for read function code '28' messages. Our analysis found 61 unique read messages, suggesting the attacker attempted to access the PLC's control logic.

Detecting Control Engine Attack: To detect control engine attacks, we developed a Python script that searches the VPP network dump for request messages where the attacker attempts to change the M221 PLC mode from start to stop. In the M221 PLC, function code '40' is used to start the PLC and '41' to stop the PLC. The script filters request messages accordingly.

$$\forall r \in Reqs, \quad \begin{cases} \text{PLC START}, & \text{if } r.\text{tcppayload}[9] = 40 \\ \text{PLC STOP}, & \text{if } r.\text{tcppayload}[9] = 41 \\ \text{Continue}, & \text{otherwise} \end{cases}$$

Detecting Authentication Attempts: We used a custom Python script using

algorithm 3 to detect and analyze password authentication attempts on the VPP . During the function code extraction and mapping phase, we identified messages and function codes used for M221 authentication. Authentication in M221 occurs in two steps. First, the attacker sends an authentication initialization message with a function code '03', requesting the PLC for the seed to compute the password hash. The PLC responds with a hash seed. The attacker then sends the computed password hash using the seed provided by the M221 PLC. The password hash message has a unique function code '6d'. We programmed the VPP to engage the attacker, responding to the initial authentication request by repeating an old authentication message and rejecting the authentication with an error function code 'fd' when the attacker sends the password hash.

Algorithm 5 Algorithm to detect PLC Authentication Attempt

```
r ← Reqs
if r.tcppayload[9] == "03" then
    Authentication Attempt
end if
if r.tcppayload[9] == "6D" then
    Password Hash
end if
```

7.6 Conclusion

As attacks on ICS systems continue to grow in both frequency and sophistication, it is imperative for the security community to understand attacker behavior and capabilities. In this paper, we introduce VPP, a scalable, protocol-agnostic honeypot. Our experimental results show that VPP outperforms existing state-of-the-art honeypots in the literature by providing application-level functionalities. Moreover,

the PLC template generation feature of VPP highlights its scalability, allowing the security community to configure it to operate as a variety of (out-of-the-box) PLCs. To illustrate VPP's effective engagement with attackers, we conducted a case study using a lab model of an elevator. Throughout this case study, we launched several attacks on the PLC, demonstrating that VPP not only successfully engages with the attacker but also generates valuable data for forensic analysis and the production of trustworthy threat intelligence.

CHAPTER 8

CONCLUSION

The increasing connectivity of industrial control systems (ICS) with enterprise networks and the internet has made them vulnerable to adversaries. Programmable Logic Controllers (PLCs) in ICS are frequently targeted by attackers aiming to disrupt physical processes. However, due to the proprietary nature of PLC communication protocols and engineering software, there is a limited availability of security and forensic tools for these PLCs. Therefore, the objective of this dissertation was to develop a virtual PLC platform capable of using captured ICS network traffic to communicate with engineering software. The platform's applications include performing forensic analysis of network-based ICS attacks, vulnerability analysis, and collecting ICS threat intelligence.

The first contribution of this research was the development of a virtual PLC platform that effectively emulates a real PLC. The platform utilizes captured ICS network dumps to learn session-dependent fields and protocol templates. The tool is fully automated and scalable for multiple PLCs. Extensive testing was conducted on several PLCs, such as Schneider Electric Modicon M221, Allen-Bradley MicroLogix 1400, and MicroLogix 1100, demonstrating successful communication between the virtual PLC platform and engineering software. Future work will involve testing the virtual PLC platform with additional PLCs.

The second contribution focused on the forensic analysis of various network-based attacks on ICS. Using the virtual PLC platform, we demonstrated forensic analysis of "Denial of Engineering Operation" and "Control Logic Injection" attacks, successfully

recovering control logic from network traffic with 100

The third contribution involved the design and implementation of a Protocol Reverse Engineering Engine (PREE). PREE is capable of dissecting a wide range of ICS protocols, empowering control engineers to formulate heuristics for identifying fields across diverse ICS protocols. Through rigorous testing and evaluation across six different ICS protocols and five PLCs from various vendors, PREE demonstrated its versatility and effectiveness.

The fourth contribution introduced a new attack on PLCs called the Control Logic Engine attack. This attack targets the control engine of a PLC responsible for executing the control logic, effectively halting the physical process. The attack was successfully conducted on five different PLCs, including CLICK, MicroLogix 1400, MicroLogix 1100, Modicon M221, and SEL RTAC-3505.

Finally, the virtual PLC platform demonstrated its capability to act as a PLC honeypot, mimicking various PLCs. A case study was presented using a lab model of an elevator with the virtual PLC platform, showcasing its ability to engage with an attacker and store attack data.

In conclusion, this dissertation has contributed to the development of a virtual PLC platform, conducted forensic analysis of network-based attacks on ICS, designed a Protocol Reverse Engineering Engine, demonstrated a new attack on PLCs, and showcased the virtual PLC platform's honeypot functionality. These contributions enhance the understanding and security of industrial control systems, providing valuable insights for threat detection, prevention, and forensic analysis in the field of ICS security.

Appendix A

ABBREVIATIONS

VCU	Virginia Commonwealth University
RVA	Richmond Virginia
ICS	Industrial Control System
DEO	Denial of Engineering Operations
PLC	Programmable Logic Controllers
VPP	Virtual PLC Platform
PREE	Protocol Reverse Engineering Engine

Appendix B

LIST OF PUBLICATIONS BY THE CANDIDATE, SYED ALI QASIM

- **[WiNTECH 2023]. Syed Ali Qasim**, Muhammad Taqi Raza, Irfan Ahmed, "vPLC: A scalable PLC testbed for IIoT Research", In the proceedings of the 17th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization, 2023. ***Submitted***

- **[HOST 2024]. Syed Ali Qasim**, Muhammad Taqi Raza, Irfan Ahmed, "PLCpot: Application Dialogue Replay based Scalable PLC Honeypot for Industrial Control Systems", In the proceedings of IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2024. ***Submitted***

- **[DFRWS USA]. Syed Ali Qasim**, Wooyeon Jo, Irfan Ahmed, "PREE: Heuristic Builder for Reverse Engineering of Network Protocols in Industrial Control Systems", In the 23rd Annual Digital Forensics Research Conference (DFRWS USA'23), July 2023, Baltimore, Maryland, USA

- **[IEEE Security and Privacy Journal].** Adeen Ayub, Wooyeon Jo, **Syed Ali Qasim**, Irfan Ahmed, "How are industrial control systems insecure by design? A deeper insight into real-world PLCs", In IEEE Security and Privacy journal, 2023

- **[IEEE Access Journal].** Masrik Dahir, **Syed Ali Qasim**, Irfan Ahmed, "Cronus: An Automated Feedback Tool for Concept Maps", In IEEE Access Journal, August 2021

- **[ICCIP]. Syed Ali Qasim**, Adeen Ayub, Jordan A Johnson, Irfan Ahmed, "Attacking the IEC-61131 Logic Engine in Programmable Logic Controllers in Industrial Control Systems", In the 15th IFIP International Conference on Critical Infrastructure Protection (ICCIP), March 2021, Arlington, Virginia.

- **[DFRWS USA]. Syed Ali Qasim**, Jared Smith, Irfan Ahmed, " Control Logic Forensics Framework using Built-in Decompiler of Engineering Software in Industrial Control Systems", In the 20th Annual Digital Forensics Research Conference (DFRWS'20), July 2020, Memphis, TN. (held virtually)
  (BEST STUDENT PAPER AWARD)

- **[ISC]. Syed Ali Qasim**, Juan Lopez Jr, Irfan Ahmed, "Automated Reconstruction of Control Logic for Programmable Logic Controller Forensics", In the 22nd Information Security Conference (ISC'19), September 2019, New York.

REFERENCES

[1] Irfan Ahmed et al. "A SCADA System Testbed for Cybersecurity and Forensic Research and Pedagogy". In: *Proceedings of the 2nd Annual Industrial Control System Security Workshop (ICSS)*. Los Angeles, CA, USA, 2016. ISBN: 978-1-4503-4788-4.

[2] I. Ahmed et al. "SCADA Systems: Challenges for Forensic Investigators". In: *Computer* 45.12 (Dec. 2012), pp. 44–51. ISSN: 0018-9162.

[3] I. Ahmed et al. "Programmable Logic Controller Forensics". In: *IEEE Security Privacy* 15.6 (Nov. 2017), pp. 18–24. ISSN: 1540-7993.

[4] Saranyan Senthivel et al. "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. Tempe, AZ, USA: ACM, 2018, pp. 319–329. ISBN: 978-1-4503-5632-9.

[5] Luis Garcia et al. "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. San Diego, CA, USA: Internet Society, 2017. ISBN: 1-891562-46-0. URL: `http://dx.doi.org/10.14722/ndss.2017.23313`.

[6] S. Valentine and C. Farkas. "Software security: Application-level vulnerabilities in SCADA systems". In: *2011 IEEE International Conference on Information Reuse Integration*. Aug. 2011, pp. 498–499. DOI: `10.1109/IRI.2011.6009603`.

[7] Nishchal Singh Kush et al. “Gap analysis of intrusion detection in smart grids”. In: *Proceedings of 2nd International Cyber Resilience Conference*. Ed. by C Valli. Australia: secau-Security Research Centre, 2011, pp. 38–46.

[8] T. M. Chen and S. Abu-Nimeh. “Lessons from Stuxnet”. In: *Computer* 44.4 (Apr. 2011), pp. 91–93. ISSN: 0018-9162. DOI: 10.1109/MC.2011.115.

[9] Nicolas Falliere, Liam O Murchu, and Eric Chien. “W32.Stuxnet Dossier”. In: (2011). URL: https://www.symantec.com/content/en/us/enterprise/media/%20security_response/whitepapers/w32_stuxnet_dossier.pdf.

[10] Syed Ali Qasim et al. “Attacking the IEC 61131 Logic Engine in Programmable Logic Controllers”. In: *Critical Infrastructure Protection XV*. Ed. by Jason Staggs and Sujeet Shenoi. Cham: Springer International Publishing, 2022, pp. 73–95. ISBN: 978-3-030-93511-5.

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

[12] James Spiro. *Cyberattacks on critical infrastructure jump by 41% in first half of 2021*. Aug. 2021. URL: https:www.calcalistech.com/ctech/articles/0,7340,L-3915536,00.html.

[13] URL: https://security.claroty.com/1H-vulnerability-report-2021.

[14] Adeen Ayub, Hyunguk Yoo, and Irfan Ahmed. “Empirical Study of PLC Authentication Protocols in Industrial Control Systems”. In: *15th IEEE Workshop on Offensive Technologies (WOOT)*. IEEE, 2021.

[15] Adeen Ayub et al. “How Are Industrial Control Systems Insecure by Design? A Deeper Insight Into Real-World Programmable Logic Controllers”. In: *IEEE Security Privacy* 21.4 (2023), pp. 10–19. DOI: 10.1109/MSEC.2023.3271273.

[16] Daniele Antonioli, Anand Agrawal, and Nils Ole Tippenhauer. "Towards High-Interaction Virtual ICS Honeypots-in-a-Box". In: *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*. CPS-SPC '16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 13–22. ISBN: 9781450345682. DOI: 10.1145/2994487.2994493. URL: https://doi.org/10.1145/2994487.2994493.

[17] Emmanouil Vasilomanolakis et al. "Multi-stage attack detection and signature generation with ICS honeypots". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 2016, pp. 1227–1232. DOI: 10.1109/NOMS.2016.7502992.

[18] Ramakrishnan Ramanathan. "The IEC 61131-3 programming languages features for industrial control systems". In: *2014 World Automation Congress (WAC)*. 2014, pp. 598–603. DOI: 10.1109/WAC.2014.6936062.

[19] AllenBradlry. *User Manual*. URL: https://literature.rockwellautomation.com/idc/groups/literature/documents/um/1763-um001_-en-p.pdf.

[20] Modicon. *SoMachine Basic - Generic Functions Library Guide*. https://www.schneider-electric.com/en/download/document/EIO0000001474/.

[21] Syed Ali Qasim, Jared M. Smith, and Irfan Ahmed. "Control Logic Forensics Framework using Built-in Decompiler of Engineering Software in Industrial Control Systems". In: *Forensic Science International: Digital Investigation* 33 (2020), p. 301013. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2020.301013. URL: https://www.sciencedirect.com/science/article/pii/S2666281720302626.

[22] Irfan Ahmed et al. "A SCADA System Testbed for Cybersecurity and Forensic Research and Pedagogy". In: *Proceedings of the 2nd Annual Industrial Control*

System Security Workshop (ICSS). Los Angeles, CA, USA, 2016. ISBN: 978-1-4503-4788-4.

[23] I. Ahmed et al. "Programmable Logic Controller Forensics". In: *IEEE Security Privacy* 15.6 (Nov. 2017), pp. 18–24. ISSN: 1540-7993.

[24] Hyunguk Yoo and Irfan Ahmed. "Control Logic Injection Attacks on Industrial Control Systems". In: *ICT Systems Security and Privacy Protection*. Ed. by Gurpreet Dhillon et al. Cham: Springer International Publishing, 2019, pp. 33–48. ISBN: 978-3-030-22312-0.

[25] Sushma Kalle et al. "CLIK on PLCs! Attacking Control Logic with Decompilation and Virtual PLC". In: *Proceeding of the 2019 NDSS Workshop on Binary Analysis Research (BAR)*. 2019.

[26] Naman Govil, Anand Agrawal, and Nils Ole Tippenhauer. "On Ladder Logic Bombs in Industrial Control Systems". In: *Computer Security*. Ed. by Sokratis K. Katsikas et al. Cham: Springer International Publishing, 2018, pp. 110–126.

[27] Hyunguk Yoo et al. "Overshadow PLC to Detect Remote Control-Logic Injection Attacks". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Roberto Perdisci et al. Cham: Springer International Publishing, 2019, pp. 109–132. ISBN: 978-3-030-22038-9.

[28] Syed Ali Qasim, Juan Lopez, and Irfan Ahmed. "Automated Reconstruction of Control Logic for Programmable Logic Controller Forensics". In: *Information Security*. Ed. by Zhiqiang Lin, Charalampos Papamanthou, and Michalis Polychronakis. Cham: Springer International Publishing, 2019, pp. 402–422. ISBN: 978-3-030-30215-3.

[29] Sushma Kalle et al. “CLIK on PLCs! Attacking control logic with decompilation and virtual PLC”. In: *Binary Analysis Research (BAR) Workshop, Network and Distributed System Security Symposium (NDSS)*. 2019.

[30] Irfan Ahmed et al. “SCADA Systems: Challenges for Forensic Investigators”. In: *Computer* 45 (2012), pp. 44–51.

[31] Adeen Ayub et al. “Gadgets of Gadgets in Industrial Control Systems: Return Oriented Programming Attacks on PLCs”. In: *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE. 2023.

[32] Nauman Zubair et al. “Control Logic Obfuscation Attack in Industrial Control Systems”. In: *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE. 2022, pp. 227–232.

[33] Muhammad Haris Rais et al. “JTAG-based PLC memory acquisition framework for industrial control systems”. In: *Forensic Science International: Digital Investigation* 37 (2021), p. 301196.

[34] Muhammad Haris Rais et al. “Memory forensic analysis of a programmable logic controller in industrial control systems”. In: *Forensic Science International: Digital Investigation* 40 (2022), p. 301339.

[35] Rima Asmar Awad et al. “Towards generic memory forensic framework for programmable logic controllers”. In: *Forensic Science International: Digital Investigation* 44 (2023). Selected papers of the Tenth Annual DFRWS EU Conference, p. 301513. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2023.301513. URL: https://www.sciencedirect.com/science/article/pii/S2666281723000148.

[36] Zhengxiong Luo et al. “Polar: Function Code Aware Fuzz Testing of ICS Protocol”. In: *ACM Trans. Embed. Comput. Syst.* 18.5s (Oct. 2019). ISSN: 1539-9087.

[37] Zhengxiong Luo et al. “ICS Protocol Fuzzing: Coverage Guided Packet Crack and Generation”. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020, pp. 1–6.

[38] Matthias Niedermaier, Florian Fischer, and Alexander von Bodisco. “PropFuzz — An IT-security fuzzing framework for proprietary ICS protocols”. In: *2017 International Conference on Applied Electronics (AE)*. 2017, pp. 1–4.

[39] Huan Yang, Liang Cheng, and Mooi Choo Chuah. “Deep-Learning-Based Network Intrusion Detection for SCADA Systems”. In: *2019 IEEE Conference on Communications and Network Security (CNS)*. 2019, pp. 1–7.

[40] Huiping Li, Bin Wang, and Xin Xie. “An Improved Content-Based Outlier Detection Method for ICS Intrusion Detection”. In: *EURASIP J. Wirel. Commun. Netw.* 2020.1 (May 2020). ISSN: 1687-1472.

[41] Hyunguk Yoo et al. “Overshadow PLC to Detect Remote Control-Logic Injection Attacks”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Roberto Perdisci et al. Cham: Springer International Publishing, 2019, pp. 109–132. ISBN: 978-3-030-22038-9.

[42] Hyunguk Yoo and Irfan Ahmed. “Control Logic Injection Attacks on Industrial Control Systems”. In: *ICT Systems Security and Privacy Protection*. Ed. by Gurpreet Dhillon et al. Cham: Springer International Publishing, 2019, pp. 33–48.

[43] Saranyan Senthivel et al. “Denial of engineering operations attacks in industrial control systems”. In: *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. 2018, pp. 319–329.

[44] Nauman Zubair et al. “PEM: Remote forensic acquisition of PLC memory in industrial control systems”. In: *Forensic Science International: Digital Investigation* 40 (2022), p. 301336.

[45] Irfan Ahmed et al. “Programmable Logic Controller Forensics”. In: *IEEE Security Privacy* 15.6 (2017), pp. 18–24.

[46] Saranyan Senthivel, Irfan Ahmed, and Vassil Roussev. “SCADA network forensics of the PCCC protocol”. In: *Digital Investigation* 22 (2017), S57–S65. ISSN: 1742-2876.

[47] John Narayan, Sandeep K. Shukla, and T. Charles Clancy. “A Survey of Automatic Protocol Reverse Engineering Tools”. In: *ACM Comput. Surv.* 48.3 (Dec. 2015). ISSN: 0360-0300. DOI: 10.1145/2840724. URL: https://doi.org/10.1145/2840724.

[48] Zhiqiang Lin et al. “Automatic protocol format reverse engineering through context-aware monitored execution.” In: *NDSS*. Vol. 8. 2008, pp. 1–15.

[49] Yeop Chang et al. “One Step More: Automatic ICS Protocol Field Analysis”. In: *Critical Information Infrastructures Security*. Ed. by Gregorio D’Agostino and Antonio Scala. Cham: Springer International Publishing, 2018, pp. 241–252. ISBN: 978-3-319-99843-5.

[50] Gergő Ládi, Levente Buttyán, and Tamás Holczer. “Message Format and Field Semantics Inference for Binary Protocols Using Recorded Network Traffic”.

In: *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2018, pp. 1–6.

[51] Hyunjin Kim et al. "Unknown Payload Anomaly Detection Based on Format and Field Semantics Inference in Cyber-Physical Infrastructure Systems". In: *IEEE Access* 9 (2021), pp. 75542–75552.

[52] Rui Wang, Yijie Shi, and Jinkou Ding. "Reverse Engineering of Industrial Control Protocol By XGBoost with V-gram". In: *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*. IEEE. 2020, pp. 172–176.

[53] Kyu-Seok Shim et al. "Clustering method in protocol reverse engineering for industrial protocols". In: *International Journal of Network Management* 30 (June 2020).

[54] Zewei Wu et al. "How to Reverse Engineer ICS Protocols Using Pair-HMM". In: *Information and Communication Technology for Intelligent Systems*. Ed. by Suresh Chandra Satapathy and Amit Joshi. Singapore: Springer Singapore, 2019, pp. 115–125. ISBN: 978-981-13-1747-7.

[55] Syed Ali Qasim, Wooyeon Jo, and Irfan Ahmed. "PREE: Heuristic builder for reverse engineering of network protocols in industrial control systems". In: *Forensic Science International: Digital Investigation* 45 (2023), p. 301565. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2023.301565. URL: https://www.sciencedirect.com/science/article/pii/S2666281723000744.

[56] Philippe Biondi and the Scapy community. https://scapy.net/. 2022.

[57] Zhengxiong Luo et al. “Polar: Function Code Aware Fuzz Testing of ICS Protocol”. In: *ACM Transactions on Embedded Computing Systems* 18 (Oct. 2019), pp. 1–22. DOI: 10.1145/3358227.

[58] Wang Yusheng et al. “Intrusion Detection of Industrial Control System Based on Modbus TCP Protocol”. In: *2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS)*. 2017, pp. 156–162. DOI: 10.1109/ISADS.2017.29.

[59] Andrew Rosenberg and Julia Hirschberg. “V-measure: A conditional entropy-based external cluster evaluation measure”. In: *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*. 2007, pp. 410–420.

[60] Yapeng Ye et al. “NetPlier: Probabilistic Network Protocol Reverse Engineering from Message Traces.” In: *NDSS*. 2021, pp. 1–18.

[61] Georges Bossert, Frédéric Guihéry, and Guillaume Hiet. “Towards automated protocol reverse engineering using semantic information”. In: *Proceedings of the 9th ACM symposium on Information, computer and communications security*. 2014, pp. 51–62.

[62] Kazutaka Katoh et al. “MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform”. In: *Nucleic acids research* 30.14 (2002), pp. 3059–3066.

[63] Weidong Cui, Jayanthkumar Kannan, and Helen J Wang. “Discoverer: Automatic Protocol Reverse Engineering from Network Traces.” In: *USENIX Security Symposium*. 2007, pp. 1–14.

[64] Yapeng Ye et al. *Netplier Tool Data*. https://github.com/netplier-tool/NetPlier/tree/master/data. 2021.

[65] Wenyu Ren, Timothy Yardley, and Klara Nahrstedt. "Edmand: edge-based multi-level anomaly detection for scada networks". In: *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE. 2018, pp. 1–7.

[66] Vern Paxson. "Bro: a system for detecting network intruders in real-time". In: *Computer networks* 31.23-24 (1999), pp. 2435–2463.

[67] *Quickdraw-Snort*. https://github.com/digitalbond/Quickdraw-Snort. 2022.

[68] N. Kush et al. "Gap analysis of intrusion detection in smart grids". In: *2nd International Cyber Resilience Conference (ICRC 2011)*. 2011, pp. 38–46.

[69] I. Ahmed et al. "Programmable Logic Controller Forensics". In: *IEEE Security Privacy* 15.6 (Nov. 2017), pp. 18–24. ISSN: 1540-7993.

[70] S. Bhatia, S. Behal, and Irfan Ahmed. "Distributed Denial of Service Attacks and Defense Mechanisms: Current Landscape and Future Directions". In: *Versatile Cybersecurity*. Vol. 72. Cham: Springer International Publishing, 2018.

[71] Hyunguk Yoo et al. "Overshadow PLC to detect remote control-logic injection attacks". In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2019, pp. 109–132.

[72] Hyunguk Yoo and Irfan Ahmed. "Control logic injection attacks on industrial control systems". In: *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer. 2019, pp. 33–48.

[73] Muhammad Haris Rais, Ye Li, and Irfan Ahmed. "Spatiotemporal G-code Modeling for Secure FDM-based 3D Printing". In: *Proceedings of the ACM/IEEE twelfth International Conference on Cyber-Physical Systems*. ICCPS '21. Nashville, TN: Association for Computing Machinery, 2021.

[74] Nicolas Falliere, Liam O Murchu, and Eric Chien. "W32.stuxnet dossier". In: *White paper, Symantec Corp., Security Response* 5.6 (2011), p. 29.

[75] "MITRE". *MITRE ATT&CK*. 2020. URL: https://collaborate.mitre.org/attackics/index.php/Main_Page.

[76] Ruimin Sun et al. *SoK: Attacks on Industrial Control Logic and Formal Verification-Based Defenses*. 2020. arXiv: 2006.04806 [cs.CR].

[77] R. E. Johnson. "Survey of SCADA security challenges and potential attack vectors". In: *2010 International Conference for Internet Technology and Secured Transactions*. 2010, pp. 1–5.

[78] Stephen Dunlap Carl Schuett Jonathan Butts. "An evaluation of modification attacks on programmable logic controllers". In: *International Journal of Critical Infrastructure Protection* 7 (1 2014), pp. 61–68. ISSN: 1874-5482. URL: https://doi.org/10.1016/j.ijcip.2014.01.004..

[79] Stephen McLaughlin and Patrick McDaniel. "SABOT: Specification-Based Payload Generation for Programmable Logic Controllers". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, pp. 439–449. ISBN: 9781450316514. DOI: 10.1145/2382196.2382244. URL: https://doi.org/10.1145/2382196.2382244.

[80] Carl Schuett, Jonathan Butts, and Stephen Dunlap. “An evaluation of modification attacks on programmable logic controllers”. In: *International Journal of Critical Infrastructure Protection* 7.1 (2014), pp. 61–68.

[81] ”Schweitzer Engineering Laboratories”. *SEL-3505/SEL-3505-3 Real-Time Automation Controller Data sheet*. URL: https://cms-cdn.selinc.com/assets/Literature/Product%20Literature/Data%20Sheets/3505_DS_20200224.pdf?v=20200305-193459.

[82] *Exe-GUARD*. URL: https://www.energy.gov/sites/prod/files/2017/04/f34/SEL_Exe-guard_FactSheet.pdf.

[83] ”ETTERCAP”. *THE ETTERCAP PROJECT*. 2021. URL: https://www.ettercap-project.org/.

[84] H. A. S. Adjei et al. “SSL Stripping Technique (DHCP Snooping and ARP Spoofing Inspection)”. In: *2021 23rd International Conference on Advanced Communication Technology (ICACT)*. 2021, pp. 187–193. DOI: 10.23919/ICACT51234.2021.9370460.

[85] Dragos and Dragos. *CRASHOVERRIDE: Analyzing the malware that attacks power grids*. Apr. 2022. URL: https://www.dragos.com/resource/crashoverride-analyzing-the-malware-that-attacks-power-grids/.

[86] Yassine Mekdad et al. “A Threat Model Method for ICS Malware: The TRISIS Case”. In: *Proceedings of the 18th ACM International Conference on Computing Frontiers*. CF ’21. Virtual Event, Italy: Association for Computing Machinery, 2021, pp. 221–228. ISBN: 9781450384049. DOI: 10.1145/3457388.3458868. URL: https://doi.org/10.1145/3457388.3458868.

[87] Kevin E. Hemsley and Dr. Ronald E. Fisher. *History of industrial control system cyber incidents*. Dec. 2018. URL: https://www.osti.gov/servlets/purl/1505628.

[88] Jianzhou You et al. "HoneyVP: A Cost-Effective Hybrid Honeypot Architecture for Industrial Control Systems". In: *ICC 2021 - IEEE International Conference on Communications*. 2021, pp. 1–6. DOI: 10.1109/ICC42927.2021.9500567.

[89] Arthur Jicha, Mark Patton, and Hsinchun Chen. "SCADA honeypots: An in-depth analysis of Conpot". In: *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*. 2016, pp. 196–198. DOI: 10.1109/ISI.2016.7745468.

[90] Susan Wade. "SCADA Honeynets: The attractiveness of honeypots as critical infrastructure security tools for the detection and analysis of advanced threats". PhD thesis. 2011.

[91] Efrén López-Morales et al. "HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 279–291. ISBN: 9781450370899. DOI: 10.1145/3372297.3423356. URL: https://doi.org/10.1145/3372297.3423356.

[92] Daniela Buza et al. "CryPLH: Protecting Smart Energy Systems from Targeted Attacks with a PLC Honeypot". In: *International Workshop on Smart Grid Security*. 2014.

[93] Feng Xiao, Enhong Chen, and Qiang Xu. "S7commTrace: A High Interactive Honeypot for Industrial Control System Based on S7 Protocol". In: *Inter-*

national Conference on Information, Communications and Signal Processing. 2017.

[94] Mauro Conti, Francesco Trolese, and Federico Turrin. "ICSpot: A High-Interaction Honeypot for Industrial Control Systems". In: *2022 International Symposium on Networks, Computers and Communications (ISNCC)*. 2022, pp. 1–4. DOI: 10.1109/ISNCC55209.2022.9851732.

[95] Yao Shan et al. "NeuPot: A Neural Network-Based Honeypot for Detecting Cyber Threats in Industrial Control Systems". In: *IEEE Transactions on Industrial Informatics* (2023), pp. 1–10. DOI: 10.1109/TII.2023.3240739.

[96] Stéphane Raimbault. *Step7 Open Source Ethernet Communication Suite*. URL: https://snap7.sourceforge.net/.

[97] Stéphane Raimbault. *Libmodbus*. URL: https://libmodbus.org/.

VITA

Syed Ali Qasim received his BS in Computer Science from Lahore University of Management Sciences, Pakistan in 2016. He joined the Doctor of Philosophy program at Virginia Commonwealth University, Richmond, Virginia in 2017. He is currently working as a research assistant under the supervision of Dr. Irfan Ahmed in Security and Forensics Engineering lab at VCU. His research interests are in developing automated tools for security and forensic analysis of industrial control systems and internet of things.