STRATIFIED INFERENCE OF INFORMATION IN CYBER-PHYSICAL SYSTEMS BASED ON PHYSICS

A Dissertation
Presented to
The Academic Faculty

By

Qinchen Gu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2020

STRATIFIED INFERENCE OF INFORMATION IN CYBER-PHYSICAL SYSTEMS BASED ON PHYSICS

Approved by:

Dr. Raheem Beyah, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Brendan Saltaformaggio
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Eric Feron
Daniel Guggenheim School of Aerospace Engineering
Georgia Institute of Technology

Dr. Lee Lerner
Center for Development and Application of Internet of Things Technologies
Georgia Tech Research Institute

Dr. Nagi Gebraeel
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: August 18, 2020

To my wife, son, parents and friends

ACKNOWLEDGEMENTS

I would first like to express my gratefulness to Dr. Raheem Beyah for his ongoing support and valuable guidance during my Ph.D studies. Dr. Beyah always provided me with insightful and constructive suggestions when I encountered research problems. Without his help, I would not have been able to finish this work.

Besides my advisor, I would like to thank my dissertation committee members, Dr. Brendan Saltaformaggio, Dr. Eric Feron, Dr. Lee Lerner and Dr. Angelos Keromytis, for their insights and input in the development of this dissertation.

I would also like to thank my previous and current lab members, Dr. Ji, Dr. Liao, Dr. Formby, Weiqing Li, Shukun Yang, Sizhuang Liang, Tohid Shekari, Celine Irvene, Anni Zhou and Binbin Zhao for their help with my research and all of the good times I had with them.

Finally, I would not be where I am today without the love and support of my family and friends. I would like to especially thank my wife and my parents for their continuous support and encouragement during my Ph.D studies.

This work has been funded by the Department of Energy, under the following award: DE-OE0000877.

TABLE OF CONTENTS

Acknowledgments iv

List of Tables x

List of Figures xi

Chapter 1: Introduction 1

1.1 Research Motivation 1

1.2 Research Scope 3

1.3 Background 4

1.3.1 Overview of CPSs 4

1.3.2 Physics-based Defense Techniques in CPSs 5

1.3.3 PLC Programming 7

Chapter 2: Literature Review 9

2.1 Software- and Physics-Based CPS Security Research 9

2.2 Static and Dynamic Analysis of Programs 11

2.2.1 Traditional Computer Programs 11

2.2.2 PLC Programs 15

2.3 Side-Channel Analysis of CPS 17

Chapter 3: Fingerprinting Individual Devices Based on Their Operation Time . 18

3.1 Identifying Threats in Cyber Physical Systems 20

3.2 Formulating the Device Physics-Based Approach 21

3.2.1 Existing CPS Security Research 21

3.2.2 Device Physics-based Fingerprinting Approach 23

3.3 Demonstration Scenario 25

3.3.1 Experiment Setup 29

3.3.2 Extracting Features by Modeling the Physics of Device 30

3.3.3 Classifying Different Devices Based on Their Fingerprints 33

3.3.4 Effect of Network Delay 34

3.3.5 Resistance to False Modeling Attacks 39

3.4 Conclusion 41

Chapter 4: Device Physics Aware Mimicry Attacks 43

4.1 Introduction 43

4.1.1 Observation 46

4.1.2 Challenges 47

4.1.3 Contributions 48

4.1.4 Attacks in CPS 48

4.2 Problem Description 49

4.2.1 Attack Model 51

4.2.2 Formal Definition of the Device Response Mimicry Attack Problem 52

4.3 Methodology 53

4.3.1 Device Physics Modeling 53
4.3.2 Characterization 58
4.3.3 Device Model and Configuration Inference 58
4.3.4 Device Response Packets Synthesis 64
4.4 Experiments 65
4.4.1 Timestamps and Protocols 67
4.4.2 Electric Motor 68
4.4.3 Relay 68
4.4.4 Valve 70
4.4.5 Implementing Timestamped Forged Response Packets 71
4.4.6 Results 72
4.5 Discussion 81
4.5.1 Applicability to Other Field Protocols 81
4.5.2 Applicability to Other Device Types 81
4.5.3 Defending Against CPS Mimicry Attacks 82
4.5.4 Limitations 85
4.6 Conclusion 86

Chapter 5: Identifying the process from its control programs 87

5.1 Introduction 87
5.2 Application Scenario 90
5.3 Building the Structure of PLC Program Binary 92
5.3.1 Understanding the Binary Structure 93

5.3.2 Input, Output and Internal Variables 100
5.3.3 Function Blocks 101
5.4 Building the Automaton 103
5.4.1 Binary Execution Emulation 103
5.4.2 Timers and Counters 104
5.4.3 Fuzzing 105
5.5 Data Collection 107
5.6 Evaluation 111
5.6.1 Classifier 111
5.6.2 Detector 121
5.6.3 Summary 126
5.7 Discussion 126
5.7.1 Generalization 126
5.7.2 Limitations 127
5.8 Conclusion 131

Chapter 6: Identifying the process parameters using side-channel information . 132

6.1 Introduction 133
6.1.1 Audio Side Channel in CPSs 133
6.1.2 Analyzing Audio with Deep Learning 134
6.1.3 Attack Scenario 136
6.2 Audio Side Channel in Electric Motors 138
6.3 Pilot Study: Water Loop Testbed 139

6.4 Case Study: Water Treatment Testbed 142
6.5 Results 145
6.6 Discussion 147
6.7 Conclusion 147

Chapter 7: Conclusion 148

Appendix A: JTAG 150

Appendix B: State Definition of the Standard Function Blocks 154

B.1 Realistic Physical Systems 154

B.1.1 Tank Balancer 154

B.1.2 Stirring System 159

B.1.3 Robot Path 160

B.1.4 Traffic Light 160

Appendix C: Secure Water Treatment (SWaT) 162

LIST OF TABLES

4.1 List of CPS devices and their physical properties 53

4.2 Experiment Settings of the Electric Motor Testbed 68

4.3 Key Parameters of the Relays Taken from Their Specifications 69

4.4 Key Parameters of the Valves Taken from Their Specifications 70

5.1 Stateful standard function blocks 102

5.2 Number of automata in the four categories 111

5.3 The average of *Number of States, Average Degree, Degree Variance* and *Triggers* in all categories of data. 113

5.4 Results of classification. Training Time contains both feature generating and training. Predict Time is the average time spent on reading and predicting one single sample. 121

5.5 Result of detection. Training Time contains both feature generating and training; Predict Time is the average time spent on reading and predicting one single sample. 124

5.6 Precesion and recall of detection methods, where precesion is $TP/(TP + FP)$, and recall is $TP/(TP + FN)$. 126

5.7 Time to generate the automaton. n is the total number of states. p is the number of inputs. σ is the time to perform a single cycle scan. 127

B.1 State definition of the standard function blocks 155

LIST OF FIGURES

1.1 Interconnections among different elements in a CPS. 4

1.2 Structure of a PLC software and hardware stack. 7

3.1 Interconnection between different elements in CPS. 19

3.2 Motor device showing sensor positions and load adjustment. 24

3.3 Plot of angular velocity over time under two different settings. MOI stands for moment of inertia and the numbers are in units of $kg{\cdot}m^2$. 26

3.4 Network packet timing diagram of the experiment setup. 27

3.5 Industrial mixer device that the model emulates. 28

3.6 Precision, recall and accuracy scores of classification on the fingerprints when varying only power input to the motor. 35

3.7 Precision, recall and accuracy scores of classification on the fingerprints when varying only the load connected to the motor. 36

3.8 Precision, recall and accuracy scores of classification on the fingerprints when varying both power and load of the motor. 37

4.1 Attack model used in this chapter, where the attacker injects a false command to the PLC and a forged response to the supervisory host. The attacker needs to first observe the legitimate traffic to infer the actual devices' models and configurations before spoofing the response. 50

4.2 Physical construction and abstract model of permanent magnet DC motor. . 55

4.3 An example showing the model and simulation results of an industrial robot using Wolfram SystemModeler. 61

4.4 Modelica models for a PMDC motor. Code used from Modelica Standard Library is not shown. 63

4.5 Block diagram of Testbed 1 setup. The valve and relay are of specific interest in this study. 65

4.6 Block diagram of Testbed 2 setup. The motor and relay are of specific interest in this study. 66

4.7 Heat map plot of the electric motor's operation curves from different models and under various run-time configurations. Each curve is aggregated over 100 runs. 73

4.8 Performance of the run-time configuration inference of the electric motor. The estimation error tolerance is the allowed distance between the estimated value and the attacker's assumed value. 74

4.9 Comparison of the authentic and spoofed responses from an electric motor. . 75

4.10 Classification performance using relays' operation time. 76

4.11 Histogram of the valves' operation time. Dashed lines indicates the specification values. 78

4.12 Wearing and aging test of relay and valve. 79

4.13 Challenge-response framework to defend against device physics mimicry attacks. 84

5.1 Overall system diagram of LogicFuzzer, which fuzzes the PLC program binary and generates the automaton. 91

5.2 Typical structure of a PLC project. 93

5.3 Program with value assignment 95

5.4 Program with value assignment and logical NOT 95

5.5 Binary diff between value assignment and logical NOT operation. 96

5.6 Program with value assignment and logical AND 97

5.7 Program with value assignment and logical OR 97

5.8 Binary diff showing AND and OR logical operations. 98

5.9 Structure of the disassembly of the PLC program 99

5.10 Flow chart of the fuzzer which generates the automaton 108

5.11 Process and PLC simulator architecture. 110

5.12 States versus transitions. 112

5.13 Visualization of some automata. There are various types of vertices in a graph. E.g., in Stirring System's automata, there are more *pivot-nodes* while in Traffic Light's automata there are more *pass-nodes.* The proportion of different types of vertices can be a effective feature to distinguish graphs. 118

5.14 Similarity score of between each program and the reference programs. . . . 123

5.15 Number of states: normal versus attack. The number of states in the attack version is usually no less than that in normal version. 125

5.16 Mixture level of the tank in the Stirring System controlled with the malicious program, compared with the original requirements. 130

6.1 Waveform and spectrogram representation of the single channel audio signal. The horizontal axis of the spectrogram is in units of "windows". 135

6.2 Spectrogram of the audio collected from an operating water loop system. . . 141

6.3 Top view of the microphone array placement at different angle along the arc with device at the center. 143

6.4 Front view of the microphone array placement matrix showing different angle and horizontal levels relative to the device. 144

6.5 Top view of the microphone array placement at different distance to the device. 144

6.6 CNN's prediction accuracy of the devices' operation status in stage 1 of SWaT under different training size ratio. 145

6.7 CNN's loss versus number of training iterations. 146

A.1 PCB of Schneider M241 PLC with JTAG debugging port 151

A.2 Using J-Link debugger to step through the program and accessing the registers and memory of the PLC 152

A.3 Sample program to demonstrate the input, output and internal variable representations. 153

B.1 Scenarios with different physical systems controlled by PLCs 158

C.1 Human-machine interface (HMI) of the SWaT testbed showing its engineering schematics. 163

C.2 Floor plan of the SWaT testbed showing its physical dimensions. 164

SUMMARY

The field of CPSs is growing rapidly. In recent years, a variety of CPS applications in different domains have flourished. Meanwhile, there have also been more frequent attacks on CPSs. The problem becomes more aggravated as the number of attacks against critical infrastructures increases rapidly. Thus, it is important to develop novel solutions to secure these critical CPSs. This research studies different techniques to infer the critical information of a Cyber-Physical System (CPS) at different levels, leveraging the physics of the CPS. One way of verifying the authenticity and integrity of an operating CPS is to check the fingerprints generated by the static structure and the dynamic operation of the CPS in the "cyber" domain, e.g., network traffic and control programs, or in a side channel, e.g., vibration and sound. A CPS can be physically characterized at three layers, namely, device model and configuration (device), process model (process), and process parameters (parameter) from the lowest to the highest layer. In this research, the correlation between the physics attributes of each layer and its fingerprints in the cyber domains and side channels is studied. Then methodologies to infer critical information of the CPSs from such correlation are studied and evaluated. The outcome from this research can be interpreted as both offensive and defensive techniques. On the one hand, attackers may leverage the device/process/parameter inference techniques to obtain sensitive information about critical infrastructures. Understanding the effectiveness of the inference techniques is a crucial step in discovering the vulnerabilities in these critical infrastructures. On the other hand, for defenders, such inference techniques can also be used to verify the correct operation of the CPS by checking the observed fingerprints against the expected values. These techniques can be used as a basis to develop novel solutions to secure the CPSs.

CHAPTER 1
INTRODUCTION

1.1 Research Motivation

The field of Cyber-Physical Systems, or CPSs, is growing rapidly. In recent years, a variety of CPS applications in different domains have flourished. For example, 80 million smart home devices were delivered worldwide in 2016, a 64 percent increase from 2015 [1]. The global industrial control system (ICS) market, one of the most important areas in CPSs, was valued at 58 billion US dollars in 2014 and expected to be worth 81 billion US dollars by 2021, growing at an annual rate of 4.9% [2]. Meanwhile, attacks targeting CPSs have become more frequent as well. The most well-known attack is Stuxnet, which is a malicious worm targeting the Supervisory Control And Data Acquisition (SCADA) systems, specifically infecting and reprogramming Programmable Logic Controllers (PLCs). It was responsible for causing tremendous damage to Iran's nuclear program, by driving the fast-spinning centrifuges in Iran's nuclear facilities to a break-down. Noticeably, a dossier published by Symantec suggested that the attackers were most likely to have conducted a significant amount of **reconnaissance** [3]. In March 2000, a former contractor of Maroochy Water Services took control of 150 sewage pumping stations using a laptop computer and a radio transmitter. This was not discovered until an engineer examined every signal passing through the system, by which time one million liters of untreated sewage had been released into a stormwater drain [4]. More recently, malware specially crafted to attack the Ukrainian electric utility caused a blackout in a portion of its capital equivalent to a fifth of its total power capacity [5]. The most significant concern regarding attacks targeting CPSs on which we depend is that they pose a threat not only to the equipment in the CPSs themselves, but also to the physical world in which we live. This very threat calls for

innovative and effective techniques to be developed. This very threat calls for innovative and effective techniques to be developed.

With the growing number of threats in this space, it is clear that novel solutions are required. While there have been a plethora of studies and mature techniques in defending attacks against traditional IT systems, the CPS domain is still suffering from lack of adequate attention. The key difference between the attacks on CPSs and those on traditional IT systems lies in the *physical* nature of CPSs. While the goal of attackers in traditional IT systems may be stealing users' private information, those who target CPSs can cause serious damage to the real world. An attack on critical infrastructures may directly threaten people's daily lives, leaving millions of dollars and even human lives at risk. Another unique aspect of the attacks against CPSs is that while some attacks are wide-spread similar to computer malware that aims for better coverage, the most devastating attacks tend to be targeted. As each CPS is configured in a unique manner, attackers would first need the schematics of the ICS [6]. The attackers would then need to know details of the individual device's physical behavior in order to maximize the damage in the following targeted attack. For example, Stuxnet checks a Profibus identification number corresponding to two different models of variable frequency drive (VFD) [1], which are used to control the motors. Two different attack sequences are chosen depending on the type of VFD found.

Many traditional studies in securing the CPSs focus on the network domain knowledge. For example, applying mature techniques adapted from traditional information technology (IT) domains (i.e., firewalls, VPNs, encryption), building an intrusion detection system (IDS) to examine the network traffic and find anomalies in the packets. These techniques are problematic in practice due to their lack of consideration for securing the physical aspects of the CPSs. As shown in numerous real life attacks [4, 7, 5], an insider who has the access to the CPS and some critical information about it cannot be stopped by the network domain defenses. Meanwhile, due to the compatibility with legacy devices

[1] Part number KFC750V3 manufactured by Fararo Paya in Tehran, Iran. Vacon NX VFD manufactured by Vacon in Finland.

and limited computational power and memory space in CPS devices, not all traditional IT security methods can be applied. Therefore, it is critical to study the physics-oriented techniques which can be used to probe and monitor the structural, operational and status information of the CPSs. Such techniques can be used to detect attacks and better secure the CPSs.

1.2 Research Scope

This research aims to study novel techniques which tightly integrate with the physical aspects of the CPSs in order to better secure them. The goal is to leverage these techniques to correlate the knowledge of CPSs with their physical attributes. There are two opposite aspects of applications for such techniques: verifying the correct operation of the CPSs (from the system administrators' perspective) and inferring the knowledge of the CPSs (from the attackers' perspective). The first aspect intends to extract the features of both the static and dynamic information about CPSs from both the cyber domain and the physical domain. These features are then used as a ground truth to check against during the run-time. The second aspect of the goal is to be able to infer the knowledge of the CPSs in terms of individual devices' models and configurations, process structures and process parameters.

Therefore, this goal can be stratified into three layers. Starting from the bottom layer, the first step is to fingerprint individual devices based on their operation time determined by the physical models and configurations. Then, the second step is to identify the process structural information using both static and dynamic analysis of the process control programs. Finally, on the top layer, the third step is to identify the process parameters using side-channel information such as sound and vibrations.

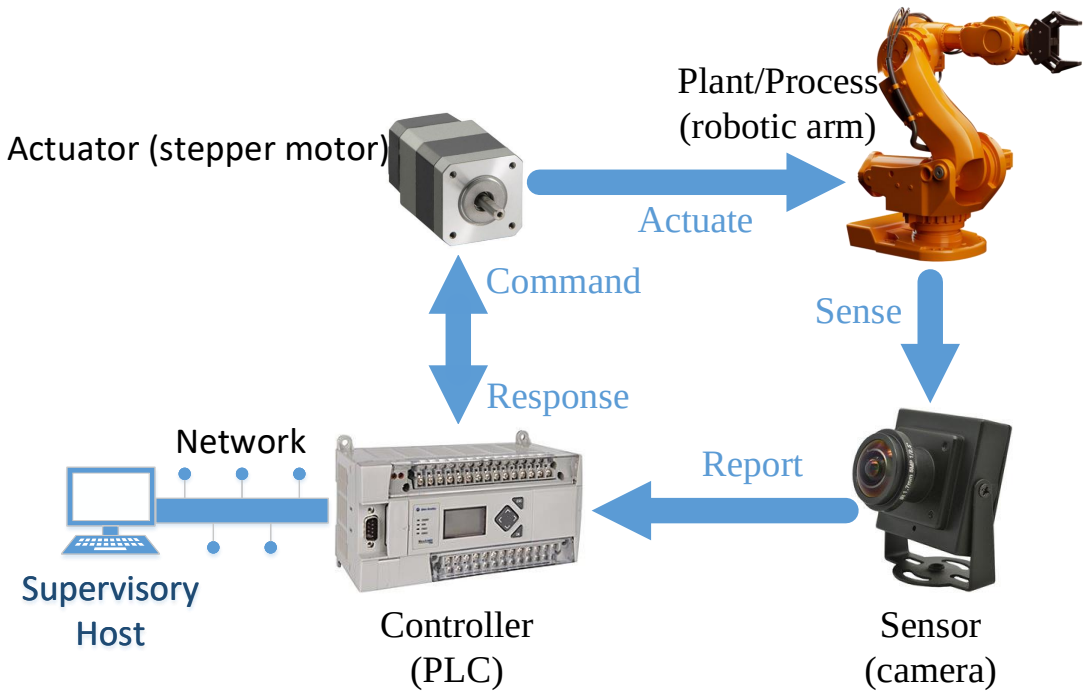


Figure 1.1: Interconnections among different elements in a CPS.

1.3 Background

1.3.1 Overview of CPSs

A CPS is composed of four major types of components, **plants** (also known as processes), **controllers**, **actuators**, and **sensors**. Figure 1.1 shows the interaction among these components in the simplest form of CPS, using the control of a robotic arm as an example. In reality, a CPS can be a cascaded structure, where one or more control loops may be embedded in the plant of a higher level loop.

The **plant** is a physical process that often entails the primary objective of the CPS, such as a room where the temperature must be maintained within a certain range, or a robotic arm that needs to be moved through space in certain sequences. The plant is a combinatorial result of the control commands and the laws of physics.

The **controller** functions as the center of computation in a CPS, and also generates the control commands. Historically, the controller has evolved from mechanical switches

and valves manually operated by human operators, to simple circuits that are hard-wired to follow a routine, and finally digital hardware that are controlled by software, such as a PLC, Remote Terminal Units (RTU), or other microprocessor based embedded systems typically found in a household environment. These controllers are capable of performing network communication and thus may be connected to a computer network for the ease of centralized management. In particular, the PLC as shown in Figure 1.1 can take commands from a host device (e.g., supervisory computers in a SCADA system) and translate the commands embedded in the network packets into electrical signals that drive the actuator. It can also act upon its local information, such as data read by the sensors to maintain the control objective of the plant.

The **actuator** implements the control commands sent from the controller to the plant. For the software controlled controller, the actuator bridges the gap between the controller in the *cyber* domain and the plant in the *physical* domain. There are various forms of actuators including motor, valve, relay, pump, etc. Unlike the speed of information propagation in a computer network, which is predominantly determined by the processing power of the devices and the speed of light, actuators are physical devices bounded by the laws of physics. Thus, there is usually a delay in the control action carried out by the actuators, given the command from the controller. Many actuators are equipped with feedback mechanisms that report back their real-time status to the controller.

The **sensor** is a one-way interface which converts the physical quantities into electrical signals that can be read by the controller.

1.3.2 Physics-based Defense Techniques in CPSs

There are two types of physics-based defense techniques in CPSs: 1) one that uses models of the physics of the process (system); and 2) one that uses models of the device physics.

System-Modeling. This type of CPS defense technique attempt to model the behavior of the physical system in a CPS, usually composed of multiple devices (sensors, actuators,

etc.) and processes. Such models usually leverage the knowledge about the system specifications and system and control theory to seek the detection of potential hazardous states [8, 9, 10]. For example, Cárdenas et al. proposed to use linear system models to detect attacks on networked control systems [11, 12]. They were able to detect stealthy attacks on these systems with linear system models. Urbina et al. studied if physics-based attack detection can limit the impact of stealthy attacks in the ICS and showed that the impact of such attacks can be mitigated by the proper combination and configuration of detection schemes, including a stateful model of the physical system [10]. More recently, McParland et al. proposed a framework to monitor physical constraint violations by leveraging specification-based intrusion detection [9].

Device-Modeling. While system-modeling based techniques focus on the system-level behavior on a CPS, device-modeling techniques can become quite useful in characterizing benign versus malicious operations inside CPSs as well. Such techniques focus on the correct operation of individual devices in a CPS. Similar to a physical system composed of various actuators, controllers, sensors and processes, each device in this system can also be modeled using deterministic equations, per the laws of physics. For example, Formby et al. proposed to tackle the false data injection issued during control command requests to the field devices [13]. The idea was to help ensure the authenticity of the responses by analyzing the observed response against the fingerprints developed by the operation time associated with each device in an ICS, which is a large division in CPS. They claimed to achieve an accuracy as high as 99% using this fingerprinting technique to differentiate authentic mechanical relays from spoofed ones. The authors also showed that it would require a highly knowledgeable and skilled adversary to perform a forgery attack on the fingerprinting technique.

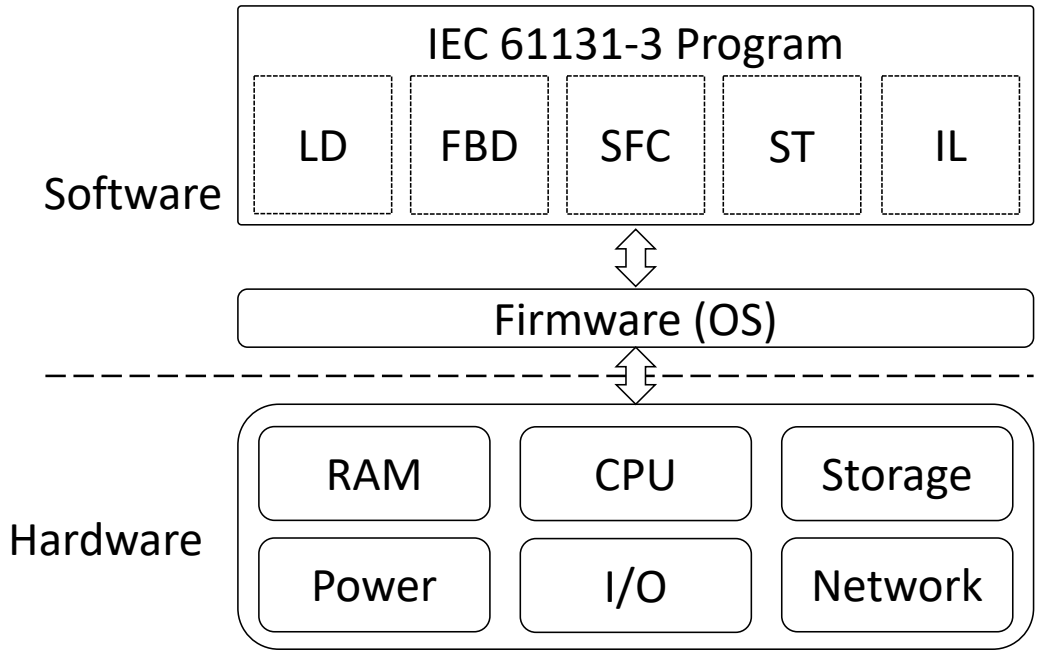


Figure 1.2: Structure of a PLC software and hardware stack.

1.3.3 PLC Programming

A PLC is a device which has been fortified for the harsh environment in industrial processes, such as water treatment facility or power plant. Computationally, it is no different from any other embedded device based on the Von Neumann architecture. However, since it is specifically designed to provide reliability as well as ease of use for process engineers, it has a few notable characteristics. As can be seen in Figure 1.2, the bottom layer in the software stack is a customized firmware which handles memory and storage access, physical I/O update, and network communication, etc. This firmware is usually a real-time operating system (RTOS), rather than the prevalent Linux or Windows OS. Hence it is unlikely to install or run additional applications other than the control logic written by the user. Unlike the general embedded development tools which uses assembly, C/C++ or other high level programming languages, the control logic program is written in the five languages defined in the IEC 61131-3 standard [14], among which three are graphical (i.e., Ladder diagram (LD), Function block diagram (FBD) and Sequential function chart (SFC)) and

two other are textual (i.e., Structured text (ST) and Instruction list (IL)). These languages are commonly used in the PLC because they have a tight association with the elements in the industrial process. For example, *contact* and *coil* in the LD corresponds to a relay's input and output, respectively. In fact, LD was invented to document the interconnection between relay racks even before the PLC.

Another important difference lies in the execution of the PLC program. The control logic program is written in a way which processes the input readings and performs computations before updating the outputs. Overall, the firmware handles reading the physical inputs (usually connected to sensors) and passing the values to the control program. Then it runs the program and translates the updated output values into the physical state of the outputs (usually connected to actuators). This process is called a **scan cycle**. During the lifetime of a PLC's operation, it performs the scan cycle repeatedly.

CHAPTER 2
LITERATURE REVIEW

2.1 Software- and Physics-Based CPS Security Research

Many existing solutions for protecting CPSs can be divided into two categories: examining the network traffic in order to look for abnormal packets similar to what a traditional intrusion detection system (IDS) does [15]; and modeling the system behavior and comparing the values output from the model's sensors with those from the real-world sensors, which leverages the knowledge about the system specifications, thus seeking to detect potential hazardous states [8, 9, 10].

For the first category, it is possible to provide a level of security of the control system's network, by treating it as an instance of IT networks and applying mature secure access technologies (e.g., virtual private network (VPN), Firewall, etc.). C. Neilson proposed to secure the control system from cyber attacks with traditional IT solutions such as VPN [15]. Although the author listed pros and cons of each solution, none of them took the physical system being controlled into account. This means that standard access control solutions are inadequate because they are not able to stop insiders (attackers) from sending commands that can drive the CPSs into dangerous states, since they have already been granted access to the system. He et al. designed a novel access control and authentication scheme for the home IoT devices, which focuses on device capabilities instead of a per-device granularity [16]. According to the authors, such scheme may provide finer control over the authorization of the IoT devices. Similarly, Schuster et al. applied the situational access control approach used in smartphone frameworks to the IoT domain, and claimed to reduce over-privileging issue which may be used in an attack [17]. These network-based type of solutions only considers the cyber domain of the CPSs, while ignoring the unique

physical attributes. They may overlook important information about the physical state of the system. Such approaches may be the easiest to deploy and is generally agnostic to the specific communication protocols or details of the physical system [15]. However, the amount of protection added by such technologies would be very limited in the industrial control system (ICS) environment, which is one of the biggest subgroups of CPSs. A high cost of deployment, poor support for legacy equipment and rare software patching further aggravate the problem [18]. Moreover, these solutions which only focus on the IT domain may fail to defend against attacks in a CPS, where an attack originated from inside the network (e.g., an insider, malicious device firmware) can hardly be detected.

Fortunately, many researchers realized that there is also a significant component in CPS which falls into the operation technology (OT) domain, and proposed the second category of solutions which take the physical attributes of the system into consideration. Earlier work in the second category focused on modeling the system behavior and comparing the values output from the model's sensors with those from the real-world sensors. Such approach leverages the knowledge about the system specifications, thus seeking to detect potential hazardous states [8, 9, 10]. Solutions in the this category (e.g., modeling of the system physics) attempt to incorporate knowledge specific to the CPSs. Some researchers propose to leverage physics of the system in order to solve this issue. A. Cárdenas et al. identified several challenges for the CPS security research community including new vulnerabilities, threats and consequences of potential attacks on networked control systems, and proposed to use linear system models to detect such attacks [11, 12]. The authors showed that they were able to detect stealthy attacks that change the physical behavior of the targeted control system by incorporating knowledge of the physical system. D. Urbina et al. studied if physics-based attack detection can limit the impact of stealthy attacks in ICS and showed that the impact of such attacks can be mitigated by the proper combination and configuration of detection schemes, including a stateful model of the physical system [10]. More recently, Formby et al. focused on individual devices rather than the entire

system behavior of CPS, and found that CPS devices can be fingerprinted due to their unique physical compositions [13]. Later, Gu et al. studied the feasibility of inferring the device models in CPSs [19]. They claimed that such fingerprint can be overlooked by an average attacker, and can be difficult to mimic due to the limited timing accuracy of embedded devices.

Another recent work proposed a framework to monitor for physical constraint violations by leveraging specification-based intrusion detection. McParland et al. were able to leverage the model of the physical plant and check the model against its corresponding physical limitations in [9]. In their paper, the authors demonstrated their approach with several scenarios, including a boiler with a heater to heat or cool the water depending on the on/off status of the heater. They created scripts to passively monitor the communication and track boiler behavior to alert upon out-of-range conditions. They did so by leveraging control theory to infer the transition states of the system model given the actions defined in the captured packets. This modeling process may require an understanding of the entire physical interactions between all actuators and the plant, as well as between the plant and all the sensors in advance. Compared to the solutions which try to secure the Cyber-Physical Systems (CPSs) by securing the network access to the system, process-modeling based solutions target the underlying physical process, thus increasing the likelihood of detecting an "insider" attack.

2.2 Static and Dynamic Analysis of Programs

2.2.1 Traditional Computer Programs

Program analysis is the process of automatically analyzing the behavior of computer programs regarding a property such as correctness, robustness, safety and liveness. The analysis can be performed without executing the program (static program analysis), during run-time (dynamic program analysis) or in a combination of both. Many tools exist to automatically or semi-automatically process the source code or the compiled (binary) code of

programs to find the vulnerabilities, or analyze the behavior of malicious code.

Static analysis can be performed either on the source or the binary code. The advantages of static analysis include free of danger of executing the destructive payload, and discovery of behaviors hidden during the run-time. On the other hand, it also presents significant challenges in dealing with payload encryption/packing (e.g., program that runs in a custom virtual machine), and the complexity of managing the state of the program (e.g., stack and registers). Moreover, compilers may obfuscate the high level language structures, making the disassembled binary code difficult to understand to a human researcher. Regardless, static analysis can be useful in extracting information that would otherwise be lost during compilation and execution. There are three components of source code analysis: the parser, the internal representation, and the analysis of this representation. The parser converts the source code into an abstract syntax. Most parsers are compiler-based and process the entire language [20, 21]. The internal representations abstract a particular aspect of the program into a form more suitable for automated analysis. Examples of internal representations include the control-flow graph (CFG), the call graph, and the abstract syntax tree (AST) [22]. Other more popular internal representations include static single-assignment (SSA) form, which modifies the control-flow graph such that every variable is assigned exactly once [23, 24]. SSA form simplifies and improves the precision of a variety of data-flow analyses. The value dependence graph (VDG) improves on the results obtained using SSA form. The VDG represents control flow as data flow and thus simplifies analysis [25].

Recent work have also sought to apply source code analysis techniques to the field of CPS. Tian et al. proposed a technique to mine the security and privacy related operations performed by Internet of Things (IoT) app, and attempted to bridge the gap between the app's actual behavior and the user's awareness of such [26]. More specifically, they performed static analysis on the collected IoT app's source code and used natural language processing (NLP) techniques on code annotations and API documents. They parsed the code of each app to create an AST, extracting key components of method names, variable

names and scope. For code annotations such as comments and text strings, they applied Stanford Part-of-speech (POS) Tagging [27] and analyzed the nouns to determine the related contexts. Similarly, Wang et al. instrumented source code of a program on IoT platforms to track data flow and method invocations to capture data provenance such as data creations and derivations [28]. Their approach also started by generating an AST and a call graph from the source code, then performed control flow analysis and data flow analysis over the AST in order to identify all relationships between all data objects. They benchmarked the efficacy of their approach against a corpus of 26 IoT attacks, and achieved promising results (full coverage) of the attack. While both studies have demonstrated effectiveness of using static analysis of the program to discover unintended functionality, and potentially opened up the CPS security field to a large number of mature static analysis tools, such techniques may not be feasible on more generalized CPSs. As a majority of the industrial devices rely on either low level programming languages (e.g., C) which are not object-oriented, or specialized ones such as ladder logic diagram or structured text for PLC, their programs some inherent code structures to be used for extracting contextual information. For example, ladder logic diagrams have no branches and are executed linearly. On the other hand, common elements still exist, such as human-readable comments and text strings. Therefore, NLP techniques may be applied towards extracting the contextual information from the source code of CPS programs.

Dynamic analysis relies on inspection of the software execution. Compared to static analysis techniques, it is agnostic to the complexity of the branching structure in the program, including encryption/packing/obfuscation, and only follows a linear execution path. The users are also exempt from maintaining the states external to the programs. Hence it is more suitable for analyzing large scale programs. Consequently, the downside of dynamic analysis is the localized view of the program's behavior, because the execution path is only limited to the scenarios that are exercised during the analysis [29]. Among various purposes of dynamic analysis of programs, program comprehension has attracted a

large research body. Since comprehension of a program is necessary before making any assumption of the intentions of any program, this domain aligns with one of the purposes of this proposal, i.e., correlate the physical process of a CPS with its corresponding control program.

One of the first papers studying the problem of the program comprehension through dynamic analysis can be dated back to 1972, when Biermann and Feldman synthesized finite-state machines from execution traces [30]. Kelyn and Gingrich proposed structural and behavioral views of object-oriented programs using a tool called TraceGraph, which traced information to animate views of program structures [31]. De Pauw et al. [32, 33, 34] proposed novel visualization views that include matrix visualizations, and used the "execution pattern" notations to visualize traces with scalability in mind. Their work was able to reconstruct interaction diagrams [35] from running programs. Walker et al. [36] presented a tool called AVID, which visualizes dynamic information at the architectural level. It abstracts the number of run-time objects and their interactions in terms of a user-defined, high-level architectural view. Bell [37] introduced an innovative concept of frequency spectrum analysis into the dynamic analysis of program. He demonstrated how the analysis of frequencies of program entities in execution traces can be used to decompose programs and identify related computations. Such techniques are also suitable to characterize programs in the CPS domain, as most of them exhibit statistical pattern in input/output data manipulations. The development of visualization techniques were bifurcated into several other domains. One of the relevant fields is the design and architectural recovery. Heuzeroth et al. [38, 39] combined static and dynamic analyzes to detect design patterns in legacy code. DiscoTect, a tool by Schmerl et al. [40, 41] constructed state machines from event traces in order to generate architectural views. Others studied the behavioral aspects of programs. For example, Heuzeroth et al. analyzed running software by studying interaction patterns. Furthermore, Koskinen et al. [42] used behavioral profiles to illustrate architecturally significant behavioral rules.

2.2.2 PLC Programs

Compared with the traditonal computer programs, analysis of the PLC program has attracted far less attention and much later in the academia. The study of verifying the correctness of a PLC program began as early as the 1990s, though the purpose was far from the idea of computer security. In 1998, Hassapis, Kotini and Doulgeri proposed to use the Hybrid Automata System (HAS) to validate the specifications of the control software written in the Sequential Function Chart (SFC) [43]. Many others [44, 45, 46, 47, 48] used different methods to reinterpret the PLC programs written in five IEC 61131-3 programming languages into Petri Nets[44], Condition/Event (C/E)-systems[46], etc. However, most of these work aimed at verification and validation (V&V) of the PLC programs to ensure the correct behavior of the programs written by the legitimate users, rather than defending against the malicious programs written by the attackers. As a result, none of their work can be applied to a PLC program for which **only the binary is available**.

Ever since the discovery of Stuxnet in 2010 [49], there have been an increasing number of work in studying techniques in both attacking and defending PLC-based systems. The majority of the work focusing on the PLC programs rely on the source code of the program. For example, McLaunghlin and Patrick explored the problem of designing PLC malware that can generate dynamic payload without having a priori knowledge of the target physical process [50, 51]. They proposed to leverage the analysis of the PLC programs taken from inside the control system to infer the structure of the physical plant and use this information to construct a dynamic payload. In their work, the attacker is assumed to use the **source code** of the PLC's original program to search for specific logic structures to carry out an attack. Their work was also limited to using the Instruction List (IL), one of the five programming languages defined by IEC 61131-3. Some researchers have explored using methods which are popular in the traditional software binary analysis domain, such as symbolic and concolic execution, control flow graph (CFG) and data dependency graph to test the correctness of the program [52, 53, 54, 55, 56, 57], which also rely on the source code.

Such methods may only be useful when the objective is V&V, and would fail when facing the problem of analyzing malicious program binaries generated by the attackers. Additionally, these tools may often require certain minimum hardware and software configurations to run, such as a powerful CPU, large RAM and specific platform[55, 56], which are rarely available in most PLCs.

On the other hand, some studies did not rely on the source code of the program. Prähofer, Wirth and Berger presented an approach to extract high-level patterns from traces of PLC programs recorded in real-time when the application interacts with the physical system being controlled[58]. The application is then replayed in the offline mode to reveal its reactive behavior and assist in finding recurring high-level execution patterns in the long run. A similar work done by Prähofer, Schatz and Grimmer augmented the approach with timing information[59]. Although their method does not require the source code, there were a few other drawbacks in their approach such as the cost and lack of flexibility associated with operating the physical system, as well as using a software simulation rather than a real PLC in the implementation. A recent work by Kalle et al. proposed to attack the control logic with first decompiling the binary into the source code, then executing in a virtual PLC[60] built based on the Schneider M221 PLC. Their work inspired us to take a different approach in processing the program binary. Instead of translating the binary back into the source program as done in [60], this work leverages the execution of the binary to discover the behavioral model of the original program. Such behavioral model can be more useful and comprehensive in discovering hidden malicious logic, and more intuitive for humans to understand how a sequence of actions can be carried out.

Several other studies explored the feasibility of replacing the original firmware of the PLC with a malicious variant, without changing the user program running on top of it. Schuett et al. exploited the inherent trust which PLCs place in the firmware verification process, which relies on a cyclic redundancy check (CRC) and a checksum as a validity mechanism[61]. They successfully tested their firmware attack on the Controllogix 1756-L61

PLC and mounted several attacks including a remotely triggered denial-of-service (DoS) attack. Similarly, Garcia et al. proposed a physical model aware attack against the PLC firmware and tested on the Allen Bradley 1769-L18ER-BB1B CompactLogix 5370 L1 Rev. B PLC. Both studies leverage access to the JTAG port on the PLC's printed circuit board (PCB) to download the malicious firmware. Although their work attempts to attack the PLC, the use of JTAG is more applicable for a defender, as it requires physical access to the device.

2.3 Side-Channel Analysis of CPS

Side channels have been used extensively in CPS security research community. Due to the rich physics involved, various side channels can be leveraged to reveal useful information about a CPS, such as power, electro-magnetic (EM) and acoustics. O'Flynn and Chen demonstrated an attack on a secure bootloader of an embedded device, where the firmware files were encrypted with AES-256-CBC [62]. They designed experiment to show that it was feasible to determine the key, the initialization vector (IV), and the secret 4-byte signature added before each encrypted block. Park and Tyagi. used power clues to hack IoT devices [63]. They discovered that the assembly-level program execution within a device can be reconstructed only through power side-channel observations, and were able to identify an instruction with over 80% accuracy through the power side channel. Using AES as an example, Longo et al. leveraged EM side-channel leakage to perform analysis of a system-on-chip (SoC) and software executing on it [64]. This allows a non-invasive means to acquire information from the system. Acoustic side-channel analysis is another attractive direction of the research community, especially for additive manufacturing systems, such as 3D printing. For example, Faruque et al. proposed a novel attack method consisting of audio signal processing, machine learning algorithms, and context-based post-processing to improve the accuracy of object reconstruction [65].

CHAPTER 3

FINGERPRINTING INDIVIDUAL DEVICES BASED ON THEIR OPERATION TIME

Due to the rapid growth in the field of Cyber-Physical Systems, or CPSs, a variety of CPS applications in different domains have flourished recently. For example, 80 million smart home devices were delivered worldwide in 2016, a 64 percent increase from 2015 [1]. The global industrial control system (ICS) market, one of the most important areas in CPSs, was valued at 58 billion US dollars in 2014 and expected to be worth 81 billion US dollars by 2021, growing at an annual rate of 4.9% [2]. Meanwhile, attacks targeting CPSs have become more frequent as well. In March 2000, a former contractor of Maroochy Water Services took control of 150 sewage pumping stations using a laptop computer and a radio transmitter. This was not discovered until an engineer examined every signal passing through the system, by which time one million liters of untreated sewage had been released into a stormwater drain [4]. More recently, malware specially crafted to attack the Ukrainian electric utility caused a blackout in a portion of its capital equivalent to a fifth of its total power capacity [5]. The most significant concern regarding attacks targeting CPSs on which we depend is that they pose a threat not only to the equipment in the CPSs themselves, but also to the physical world in which we live. This very threat calls for innovative and effective techniques to be developed. With the growing number of threats in the space, it is clear that novel solutions are required. One clear approach to improving the security of CPSs is to rely on the process physics as a defensive side-channel. In fact, the authors of [9] make the case for the need to leverage process physics to secure CPSs. While process physics should be used to secure CPSs, physics of the devices also matters. For the rest of this chapter, "device" refers to the "actuator" in the CPSs.

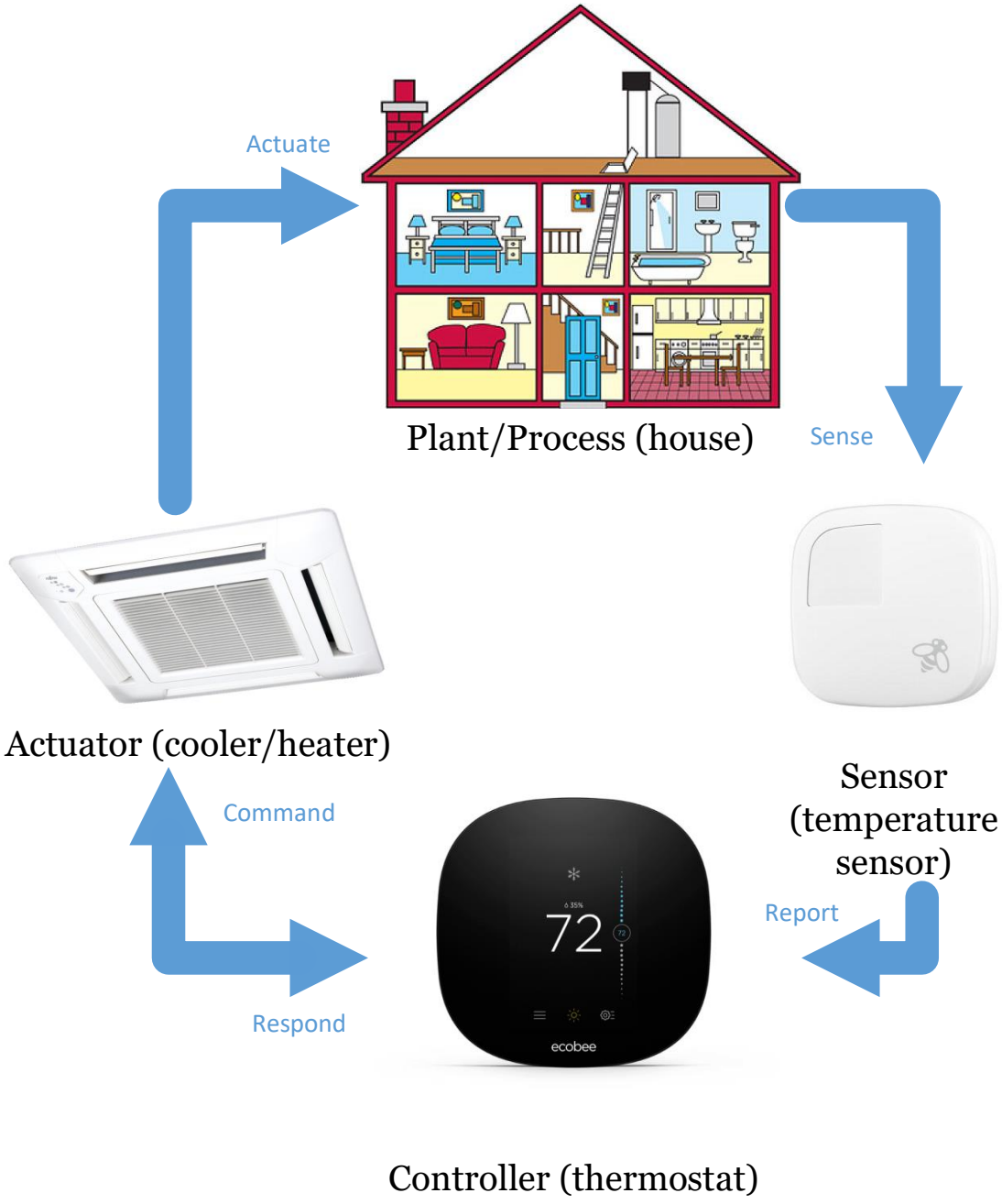


Figure 3.1: Interconnection between different elements in CPS.

3.1 Identifying Threats in Cyber Physical Systems

Common components in CPSs are the process, actuator(s), sensor(s) and controller(s). Figure 3.1 shows the interconnections between these elements using an air conditioning system as an example. Despite various attack vectors to penetrate the system and take control of one or more components, it would be a fair assumption that a potential goal for an attacker is to drive the physical plant to an unsafe state, i.e., attacking either the actuator(s), sensor(s) or controller(s) serves to achieve this goal. One way to do this is spoofing control commands to the actuators or sending incorrect sensor values to the controller. Since defending against direct physical access to the plant falls outside the realm of the cyber world, the scope of the discussion in this chapter is limited to securing the other three components.

The most well-known attack that targeted the controller is Stuxnet [3]. By reprogramming the programmable logic controllers (PLCs), Stuxnet was able to operate the PLCs according to the attacker's intention. More specifically, Stuxnet spreads itself using traditional network infrastructure as well as removable storage media to reach the targeted computer. It then modifies the code on the PLC, causing it to send out commands that drive the centrifuges fast enough to tear themselves apart. It is worth noting that the cost of such an attack is high: four zero-day flaws were exploited and two digital certificates were compromised. Additionally, part of the path Stuxnet took was through air-gapped networks, since the key computer was unlikely to have outbound Internet access. Clearly, many defensive techniques used in the traditional computer network domain should also be used in such environments where security is highly demanded. However, as will be explained later, not all network security tools (e.g., virtual private network (VPN), encryption) can be easily applied in the CPS environment, due to the insufficient computational and memory capabilities of a large portion of legacy devices present in CPSs. Also, note that as illustrated in the Stuxnet example, even though air gapping physically isolates a protected network from insecure networks, it does not ensure 100% security, as attackers

may use flash storage and other media to circumvent this. In fact, attackers may come from the inside, which renders any defensive means against outside attackers useless.

Compared to the controllers, actuators and sensors are more tightly coupled with physical plants, as they serve as the direct inputs and outputs, respectively, of the physical process. Attacking sensor devices can be carried out in two ways: integrity attacks and Denial-of-Service (DoS) attacks [8]. In the former case, the attacker can inject any sensor value so the controller receives false non-zero values, performing a false data injection attack. In the latter case, the attacker simply cuts off the communication link between the sensor and the controller. Attacking the actuators may be carried out by compromising the actuator and impersonating it so the attacker can send out false reports back to the controller (e.g., the breaker or valve has opened). Since the actions of an actuator apply directly to the underlying plant and thus may potentially cause the fastest and most effective damage, it is clearly a critical point to defend in a control system. Thus, this work aims to secure the Cyber-Physical Systems (CPS) at the individual actuator level, by leveraging the fingerprints generated by the physical devices. The fingerprints can be used to authenticate the actuators, ensuring that these responses are not spoofed by attackers.

To summarize the problem, assume the global set of all devices in CPSs consisting of devices of various models and configurations. Given a finite-time observation of any device in a network, the goal of the device physics-based fingerprint method is to identify which model and configuration the observation corresponds to.

3.2 Formulating the Device Physics-Based Approach

3.2.1 Existing CPS Security Research

Many existing solutions for protecting CPSs can be divided into two categories: examining the network traffic in order to look for abnormal packets similar to what a traditional intrusion detection system (IDS) does [15]; and modeling the system behavior and comparing the values output from the model's sensors with those from the real-world sensors, which

leverages the knowledge about the system specifications, thus seeking to detect potential hazardous states [8, 9, 10].

For the first category, it is possible to provide a level of security of the control system's network, by treating it as an instance of IT networks and applying mature secure access technologies (e.g., virtual private network (VPN), Firewall, etc.). C. Neilson proposed to secure the control system from cyber attacks with traditional IT solutions such as VPN [15]. Although the author listed pros and cons of each solution, none of them took the physical system being controlled into account. This means that standard access control solutions are inadequate because they are not able to stop insiders (attackers) from sending commands that can drive the CPSs into dangerous states, since they have already been granted access to the system. This network-based type of solution only considers the cyber domain but ignores the unique physical attributes, which may overlook important information about the physical state of the system. Such an approach may be the easiest to deploy and is generally agnostic to the specific communication protocols or details of the physical system [15]. However, the amount of protection added by such technologies would be very limited in the industrial control system (ICS) environment, which is one of the biggest subgroups of CPSs. A high cost of deployment, poor support for legacy equipment and rare software patching further aggravate the problem [18].

Solutions in the second category (e.g., modeling of the system physics) attempt to incorporate knowledge specific to the CPSs. Some researchers propose to leverage physics of the system in order to solve this issue. A. Cárdenas et al. identified several challenges for the CPS security research community including new vulnerabilities, threats and consequences of potential attacks on networked control systems, and proposed to use linear system models to detect such attacks [11, 12]. The authors showed that they were able to detect stealthy attacks that change the physical behavior of the targeted control system by incorporating knowledge of the physical system. D. Urbina et al. studied if physics-based attack detection can limit the impact of stealthy attacks in ICS and showed that the impact

of such attacks can be mitigated by the proper combination and configuration of detection schemes, including a stateful model of the physical system [10].

A more recent work proposed a framework to monitor for physical constraint violations by leveraging specification-based intrusion detection. McParland et al. were able to leverage the model of the physical plant and check the model against its corresponding physical limitations in [9]. In their paper, the authors demonstrated their approach with several scenarios, including a boiler with a heater to heat or cool the water depending on the on/off status of the heater. They created scripts to passively monitor the communication and track boiler behavior to alert upon out-of-range conditions. They did so by leveraging control theory to infer the transition states of the system model given the actions defined in the captured packets. This modeling process may require an understanding of the entire physical interactions between all actuators and the plant, as well as between the plant and all the sensors in advance. Compared to the solutions which try to secure the Cyber-Physical Systems (CPSs) by securing the network access to the system, process-modeling based solutions target the underlying physical process, thus increasing the likelihood of detecting an "insider" attack.

3.2.2 Device Physics-based Fingerprinting Approach

In [13], Formby et al. proposed to fingerprint CPS devices by measuring the network level actuator response time, which in turn is determined by the physical properties inherent to each individual device. More specifically, for an actuator as shown in Figure 3.2, they monitor the time it takes between the event that a command from the controller is being sent to the actuator (e.g., a TCP packet containing the command to open a valve) and the event that the corresponding response is received by the controller (e.g., a TCP packet containing the response from the actuator that confirms the valve is open). In this chapter, it is shown that this time difference, called the "operation time", is tightly coupled with the physical characteristics of the device and experiments are performed to validate this hypothesis. The

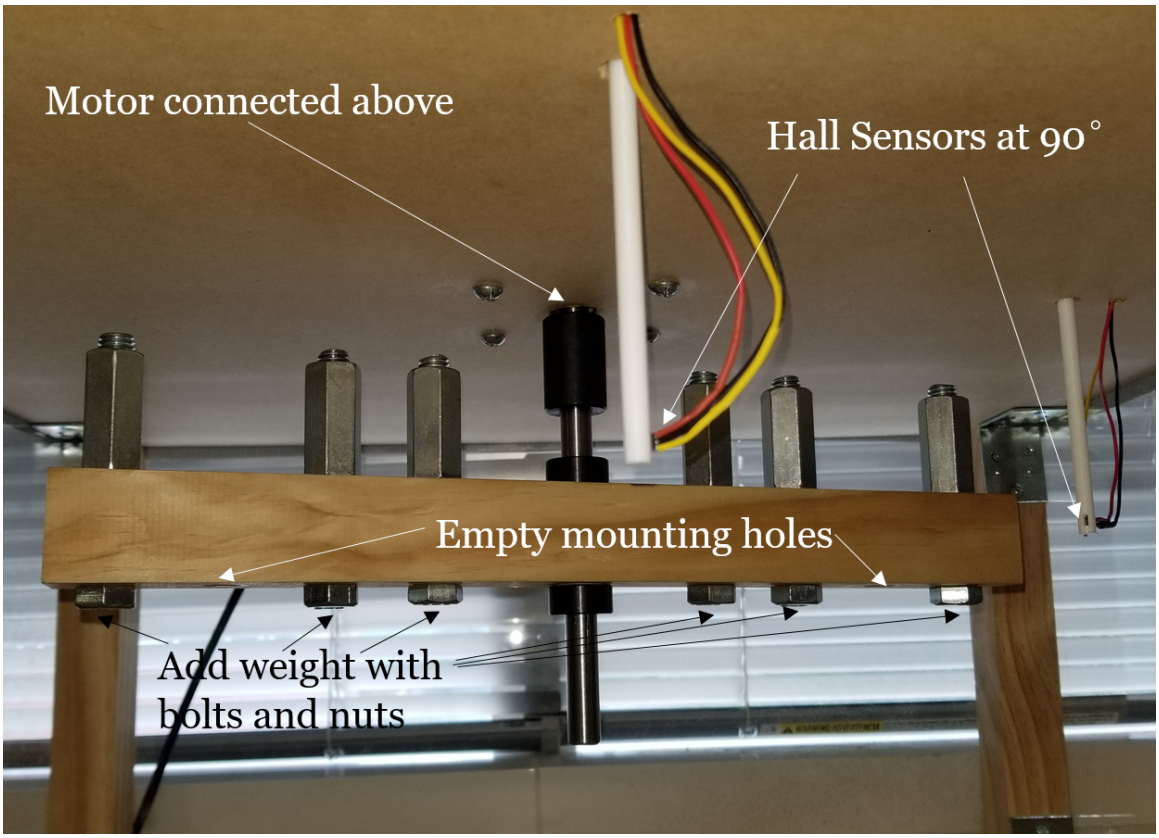


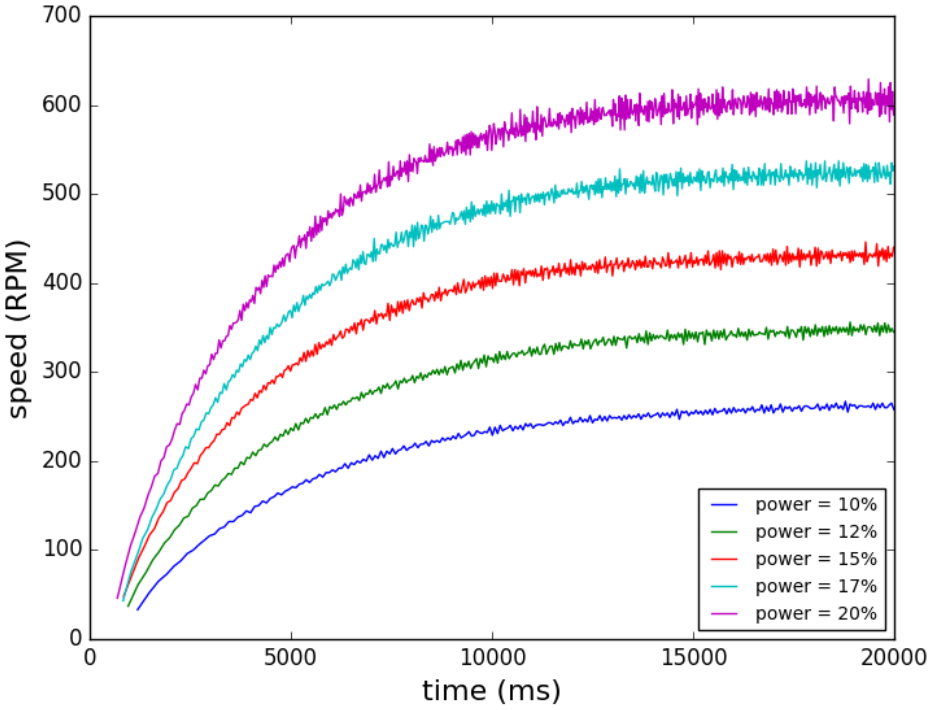
Figure 3.2: Motor device showing sensor positions and load adjustment.

operation time for a certain actuator is one method to fingerprint the device. In contrast to the concept in the general network domain, where the fingerprint of a device is mostly related to hardware and software components and configurations, that of a CPS device can be additionally affected by the unique physical composition of the device (i.e., device physics). Thus, timing features of the messages returned by an actuator related to the operation it performs can be used as the main attributes for fingerprinting the device.

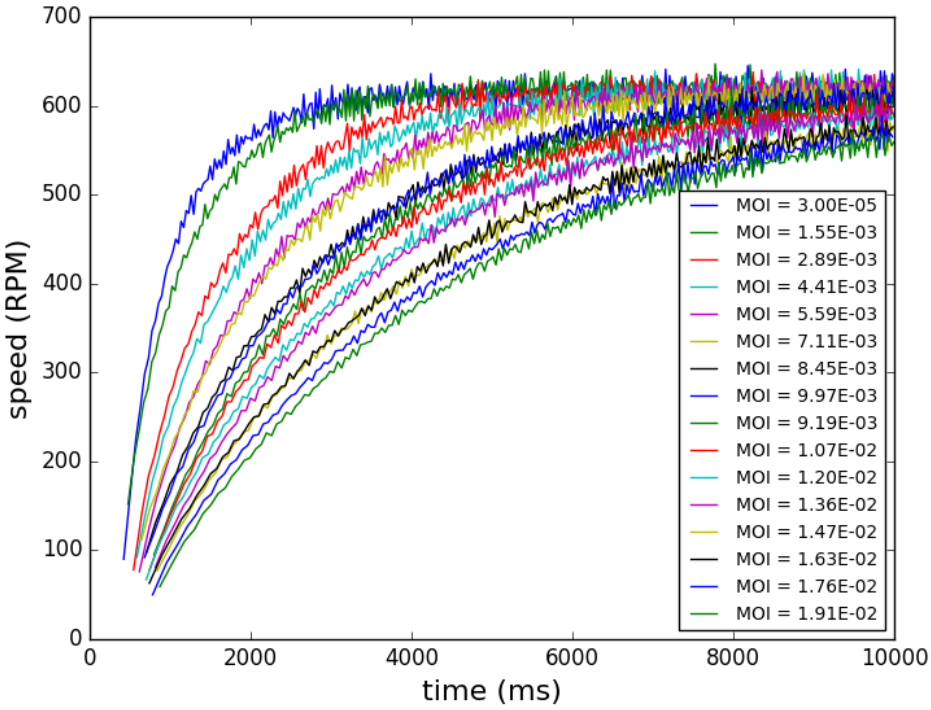
Device-modeling techniques can be used in conjunction with the technique proposed in [9] (process-modeling techniques) to better secure CPSs. While the work in [9] examines and monitors the physical system as a whole (i.e., the physical process), this work combines the information from both "cyber" and "physical" domains and monitors the signatures at the individual device level. Because the behavior of an individual device is largely dependent on its own state and attributes, this approach requires less knowledge from the overall system dynamics, and thus is still useful when the system gets complicated and difficult to model. Furthermore, by combining information at the individual device level with that at the system level, it becomes much more difficult for an attacker to exploit vulnerabilities when device physics is also used to detect attacks. Because, in addition to having to monitor and model the dynamics of the process, the attacker would need to forge actuator response times with strict timing and value constraints for each device in order to circumvent the detection mechanism.

3.3 Demonstration Scenario

In this chapter, the technique in [13] is generalized to understand more about why the technique worked, and it will be concluded that device physics can also be widely applied to CPS security. Specifically, an experiment is designed with several controllable variables to test the robustness of the device physics-based fingerprinting technique. A high-level network diagram can be seen in Figure 3.4, as a PC acts as a host device sending out commands to a PLC as well as receiving responses from the programmable logic controller



(a) 5 power input settings with fixed load.



(b) 16 load settings with fixed power.

Figure 3.3: Plot of angular velocity over time under two different settings. MOI stands for moment of inertia and the numbers are in units of $kg{\cdot}m^2$.

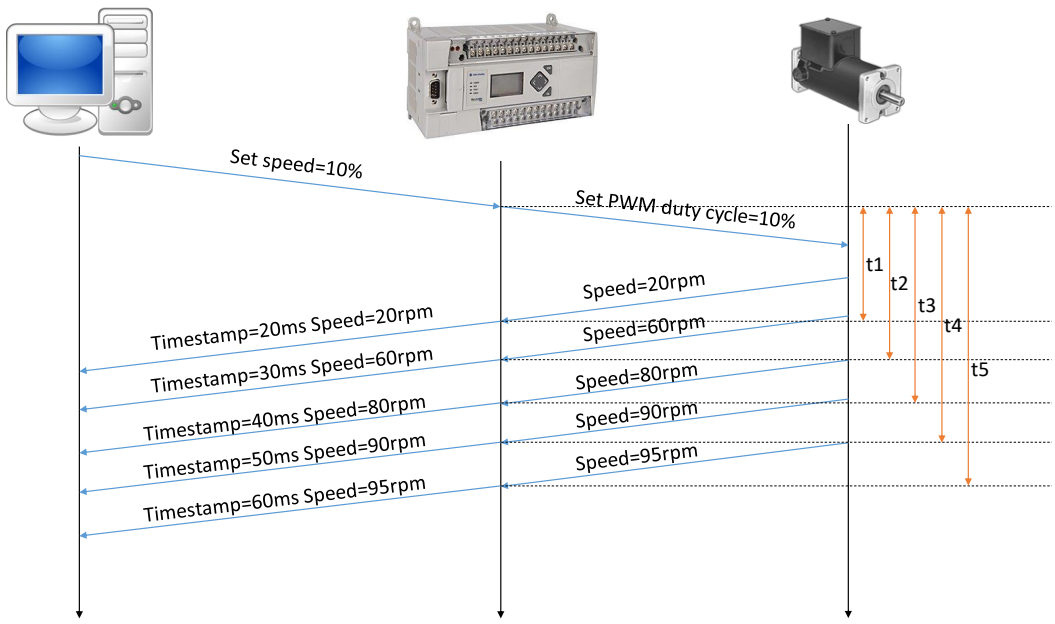


Figure 3.4: Network packet timing diagram of the experiment setup.

(PLC) regarding the status of the physical devices. The PLC then connects to actuators and sensors and executes the command on the corresponding devices. For the actuator, a motor is picked as it is an extensively used device in the industrial control system (ICS) environment. It is also a high-valued target for attackers since a motor usually outputs a much higher power than many other actuators such as valves and relays, thus may deal more damage to the surrounding equipment and personnel when being sabotaged. An example of such sabotage is faking the response of a request to reduce the revolutions per minute (RPM) of a motor, leading to a potentially unsafe scenario for individuals in the plant or the plant in general. The experiment emulates the use case of a motor as in an industrial mixer shown in Figure 3.5, as such equipment is widely used and usually comes in many variants. For example, a series of six heavy duty electric mixers can be found from a company named INDCO [66], with horsepower ranging from 1/3 HP to 3HP. The various horsepower ratings represent different possible options when choosing a motor at a specific point within a CPS, and may generate different fingerprints. At the end of the mixing shaft of the mixer is a blade which can propel the fluid being stirred. The fluid exerts a drag

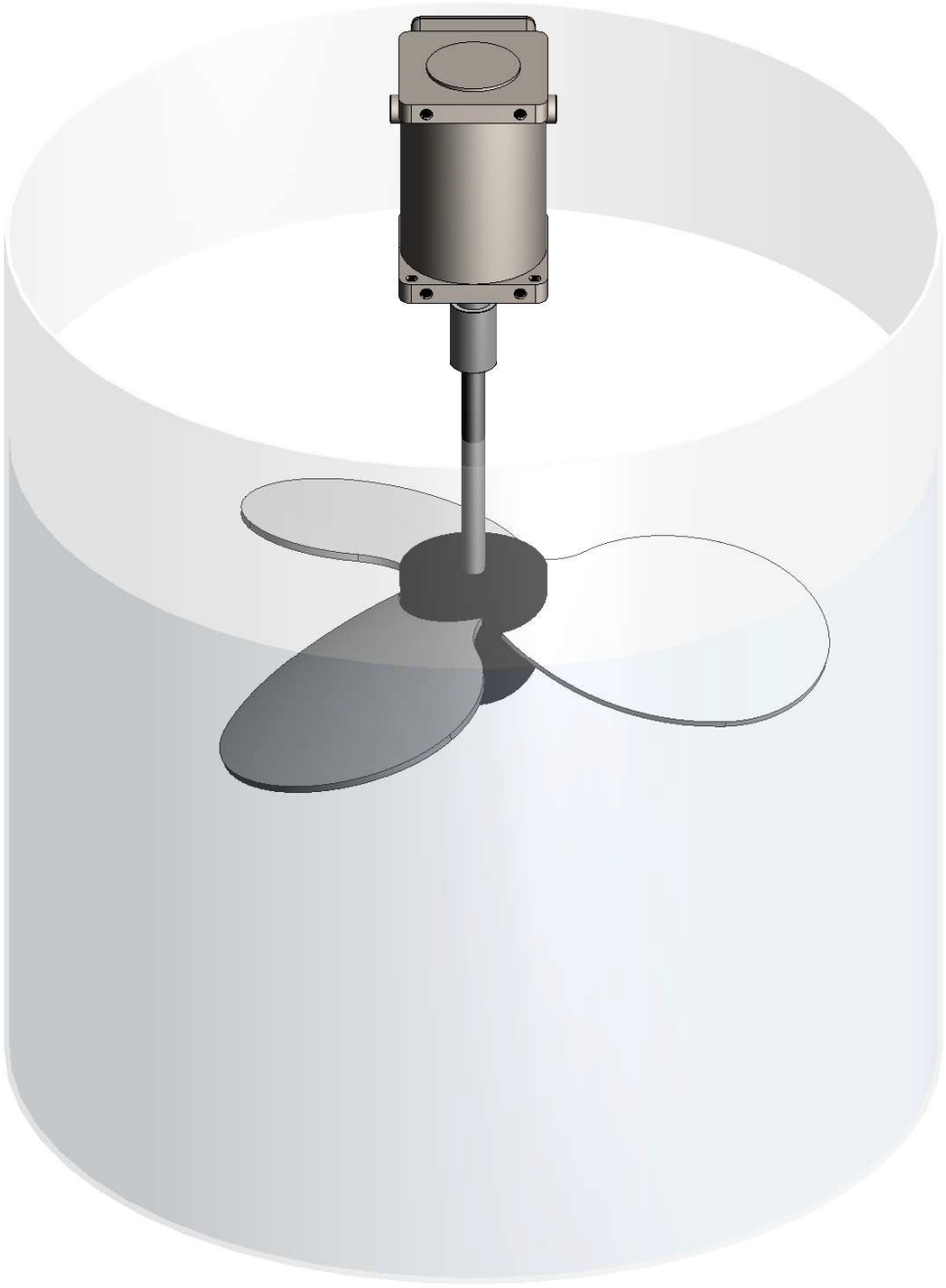


Figure 3.5: Industrial mixer device that the model emulates.

force on the blade which slows it down. When the mixer is in use, the fluid being stirred usually consists of the same material, while the viscosity of the fluid can vary depending on its concentration and density (e.g., different concentrations of maple syrup solution). Such differences also make a difference in the fingerprints generated by the mixer in response to control commands sent from the host. To emulate the mixer and quantify the behavior of the device from both the cyber and physical domains, a device model as shown in Figure 3.2 is leveraged. Five different levels of power input are set to a single electric motor used in the experiment (to emulate variants with different horsepower), and 16 different load levels on the motor output (to emulate various fluid viscosity), giving a total of 80 different operating configurations for the motor. Each configuration generates a fingerprint that is later used to classify the corresponding configuration. In an industrial environment, the total number of configurations may vary. Thus, a discussion will follow later about the effect of a different number of configurations on the performance of this fingerprinting method. The goal of the experiments is to show that a remotely observable fingerprint can be generated according to the physical attributes of an actuator to effectively identify the different configurations of the device (which naturally extends to identifying different devices).

3.3.1 Experiment Setup

For the communication protocol between the host PC and PLC, Modbus is chosen as it is an open standard protocol widely used in ICSs, and is easier to implement compared to DNP3. The Modbus protocol does not inherently contain timestamp information in its packet, therefore this issue is addressed with a modification in the PLC ladder logic program and the PLC is utilized to timestamp the command and response packets to achieve real-time accuracy. The PLC acts as a Modbus slave and waits for read and write requests from the host PC running a Python script which acts as a Modbus master. The host sends commands to the PLC to set the operating speed of the motor and receives a series of response packets from the PLC containing the measurement of the angular velocity of the

rotating load along with timestamps at which the measurement is taken, as depicted in Figure 3.4. Note that the timestamps are measured in reference to the time when the PLC received the corresponding command. The operating speed can be adjusted in a range of values by changing the pulse-width modulation (PWM) output of the motor driver connected in between the PLC and the motor. The motor spins a load that can be adjusted by adding or removing weights on it as shown in Figure 3.2. The base of the load is a uniform light-weight wood bar with eight mounting holes positioned vertically at proportional distance to the center of spinning axis. The mounting holes are symmetrical to the spinning axis to keep the center of mass aligned with the center of rotation, thus minimizing the rotating imbalance and the wobbling movement of rotating structures. A bolt and a coupling nut of known masses are mounted at each hole to adjust the moment of inertia of the overall rotating load. Two Hall effect sensors are placed near the circular track of the tips of the rotating bar that produce signals when either of the two magnets attached to the tips of the load passes by. These signals are then picked up by the programmable logic controller (PLC) connecting to the sensors, and used to send responses back to the host.

3.3.2 Extracting Features by Modeling the Physics of Device

To classify the fingerprints generated by the mixer under different configurations, features must be extracted from the raw sensor readings. Because each configuration differs only in its physical attributes, a straight-forward source of features to be used for classification is the mathematical model which corresponds to the dynamics of the device. Thus, derive a simple mathematical model of the device is derived, starting with Newton's second law for rotation,

$$\tau - \hat{\tau} = I\alpha \tag{3.1}$$

where τ is the torque exerted by the motor to the load, $\hat{\tau}$ is the frictional torque assumed to be constant relative to the angular velocity of the load, I is the overall moment of inertia of the load, and α is the angular acceleration of the load. For a DC motor like the one used in

the experiment, the relationship between the torque it generates and the input voltage can be derived from equations,

$$i = \frac{E_s - E_o}{R} \tag{3.2}$$

$$P = E_o i \tag{3.3}$$

$$E_o = ZnF/60 \tag{3.4}$$

where i is the current through the armature in the motor, E_s is the source DC voltage and E_o is the induced voltage in the armature conductors as they cut the magnetic field produced by the permanent magnet inside the motor, P is the mechanical power of the motor, n is the rotation speed in revolutions per minute (RPM), and R, Z and F are the armature resistance, winding coefficient, and flux per pole respectively, and are constant regarding the specific motor build. Turning attention to the torque, it is known that the mechanical power P is given by the expression

$$P = \tau n \times \frac{2\pi}{60} \tag{3.5}$$

Thus, the equation becomes

$$\tau = \frac{ZF(E_s - ZnF/60)}{2\pi R} \tag{3.6}$$

Now consider the case where the load is initially at rest and accelerated by applying a time-invariant voltage E_s onto the motor. The angular velocity ω satisfies the equation

$$\omega(t) = \int \alpha dt = \int \frac{ZF(E_s - ZnF/60)}{2\pi RI} dt - \int \frac{\hat{\tau}}{I} dt \tag{3.7}$$

under the boundary condition that $\omega(0) = 0$. By solving the differential equation and substituting $n = \frac{30}{\pi}\omega$, an exponential decay equation is obtained

$$\omega = -Ae^{-t/B} + A \tag{3.8}$$

where $A = (\frac{ZF}{2\pi R}E_s - \hat{\tau})\frac{4\pi^2 R}{Z^2F^2}$ and $B = \frac{4\pi^2 R}{Z^2F^2}I$. In the experiment, all timestamped velocities are measured at the time when a magnet passed by a sensor and the instantaneous velocity is calculated based on the inverse of the time it takes since the last time a magnet passed by. Therefore, during an acceleration process such calculated velocity is slightly lower than the actual value at the timestamp. Thus, a delay variable t_d is introduced into the equation

$$\omega = -Ae^{\frac{t-t_d}{B}} + A \tag{3.9}$$

Recall that the only two variables in the equation are the moment of inertia I and the supply voltage E_s which determine how the angular velocity ω changes over time. It will later be shown that the configuration of the mixer device consisting of these two variables can be inferred from the features extracted from the raw sensor readings in the response packets (i.e., corresponding timestamps and angular velocity measurements). Because of the asymptotic nature of the curve described by Equation (3.9) and the noise introduced in actual measurements as can be seen in Figure 3.3, it becomes infeasible to define an exact operation time as it has been done in [13]. Instead of using a single event to mark the completion of an operation, the timestamps are characterized in reference to the angular velocity measurements and a trend of "operation times" is obtained by the series of timestamps generated in response to a command. Thus, by fitting Equation (3.9) to the measured timestamps and angular velocity values from the experiments, various features can be generated on which to classify, consisting of the coefficients in the equation.

3.3.3 Classifying Different Devices Based on Their Fingerprints

There are two stages in the aforementioned approach, namely a training stage and a classification stage for fingerprinting each actuator. During the first stage, a number of the operation time is generated and stored in a database. In the classification stage, a series of measurements of the operation times are taken from the devices in the control system and compared against the database generated during the training stage. From the test results, conclusions can be drawn about whether the actuator as seen from network traffic corresponds to the actual configuration of the device or one of its variants.

To collect data needed for training and testing, both the input power to the motor as well as the load driven by the motor are altered and timestamped angular velocity measurement response are taken under a total of 80 different configurations formed by the combination of the two variables. Recall that the various input power emulates variants of the same model of mixer with different horsepower, and various loads corresponding to different viscosity (i.e., concentrations) of fluid being stirred by the motor-drive mixer. For each configuration, 50 runs of measurement are performed by accelerating the load from rest to a stable angular velocity. In this case, the capture time for each run is heuristically set to 30 seconds. Each measurement run generates a fingerprint associated with the physical characteristics dependent on the two variables. To get some intuition on the separability of these fingerprints generated by different configurations, one out of the 50 runs is randomly picked for each configuration, and the angular velocity measurements are plotted against corresponding timestamps in Figure 3.3. For clarity, the measurements under two set of configurations are shown, namely varying power with fixed load, denoted as S_{power}; and varying load with fixed power, denoted as S_{load}. This shows that the same configuration can generate stable fingerprints distinguishable from different configurations.

As a next step, a classifier is built to quantitatively measure how well these fingerprints can be used to identify the configuration of a device (or, as a natural extension, a number of different devices). The features used by the classifier are extracted by fitting Equation 3.9

to the selected angular velocity/time measurements in each run (i.e., the first 20 seconds of data after the command was sent) and taking the coefficients generated after the fitting (i.e., A, B and t_d). The entire measurement dataset is split into training and testing sets using stratified K-fold method, with K set to 10. For the aforementioned experiments, three basic supervised machine learning classification methods are chosen, namely decision tree, naive Bayes and k-nearest neighbors (KNN) implemented in the Python scikit-learn machine learning library. These classification algorithms generated highly accurate classification results. For example, the naive Bayes classifier achieves both 1.0 precision and recall scores when varying only the power input to the motor, and 0.98 precision and 0.97 recall scores when varying only the load connected to the motor. All classifiers have a slight performance drop when classifying fingerprints generated by all 80 configurations when combining the two variables, namely the input power and load. For example, the decision tree classifier achieves a 0.89 precision score and 0.89 recall score. This difference can be taken as a deficiency in the aforementioned approach. Hence it becomes a challenge when the device has so many possible configurations that they have similar signatures. However, this does not become an unsolvable problem in an industrial environment, as the number of different design configurations of a device (thus the fingerprints) is small. Furthermore, a device is usually expected to operate within a reasonable range of states, which allows slight deviations from its theoretical operating state. Such deviations can sometimes cause the fingerprints to be difficult to differentiate from those generated by a slightly different device configuration.

3.3.4 Effect of Network Delay

As timestamps play a significant role in the fingerprinting technique, their integrity is critical to the success of the proposed technique. Generally, there are three options for obtaining the actuator timing values in Cyber-Physical Systems (CPSs): 1. If the protocol natively supports timestamps (e.g., distributed network protocol (DNP3)), the operation time can

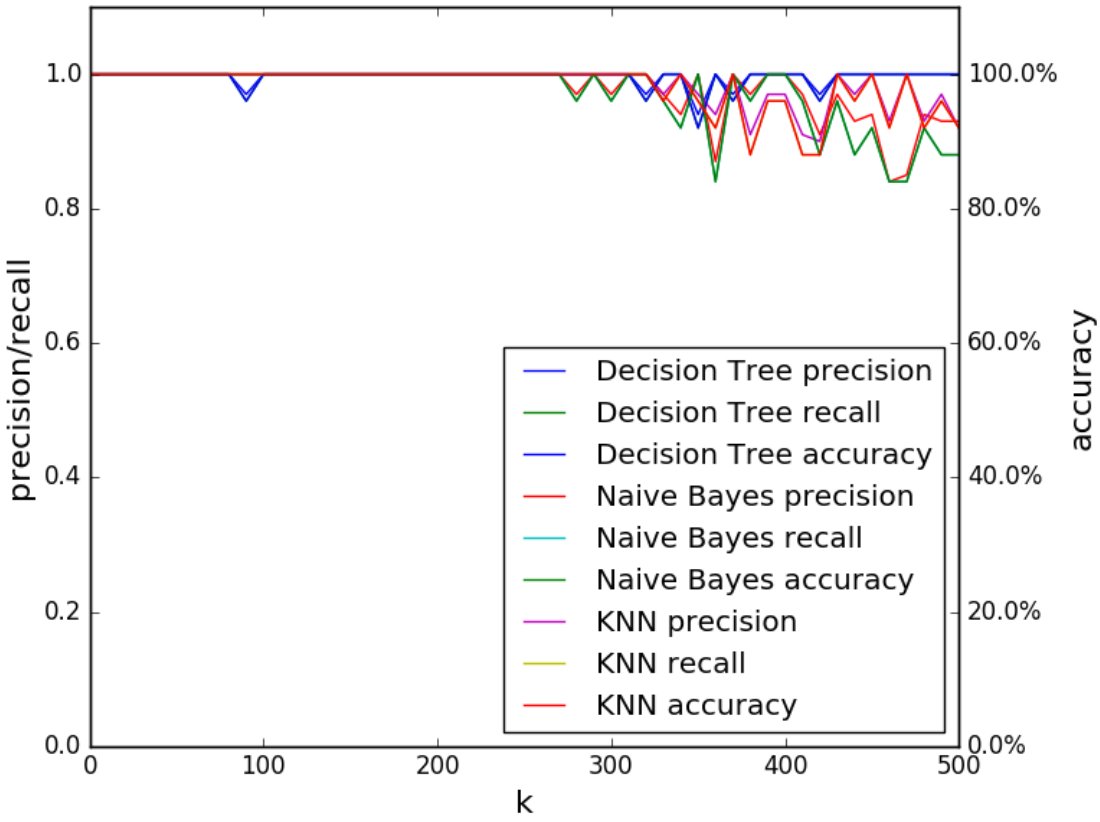


Figure 3.6: Precision, recall and accuracy scores of classification on the fingerprints when varying only power input to the motor.

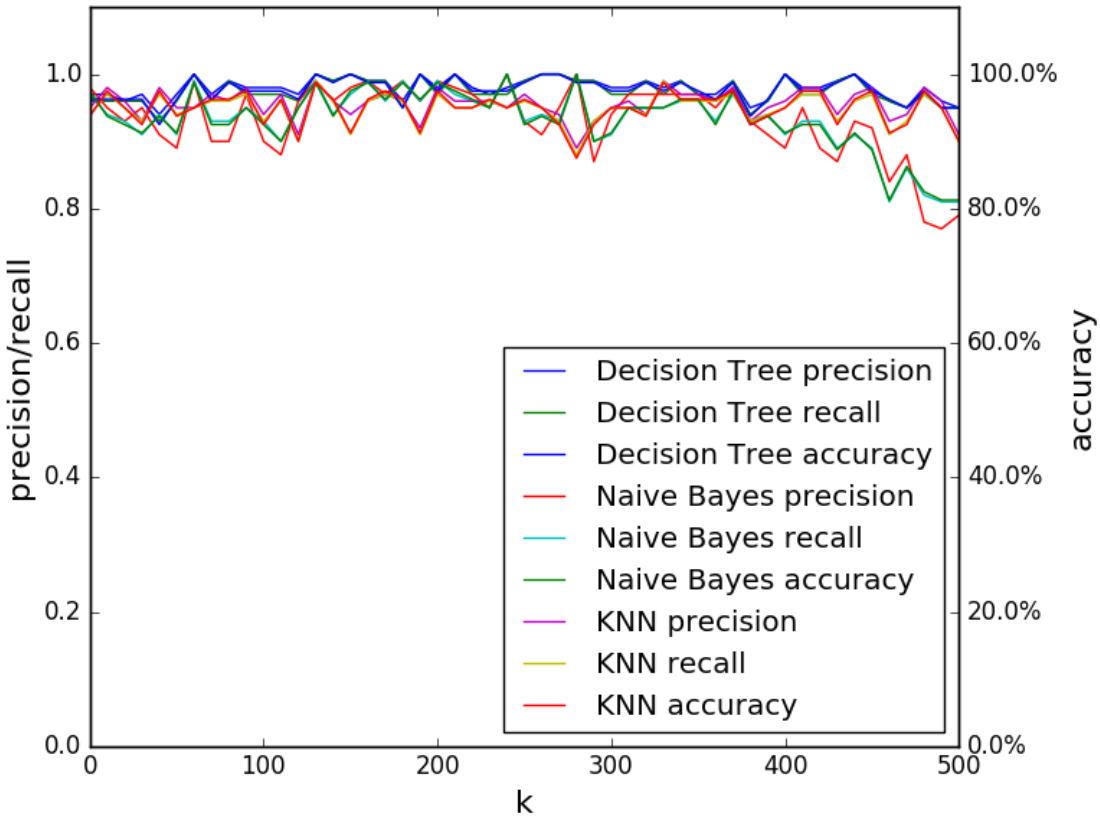


Figure 3.7: Precision, recall and accuracy scores of classification on the fingerprints when varying only the load connected to the motor.

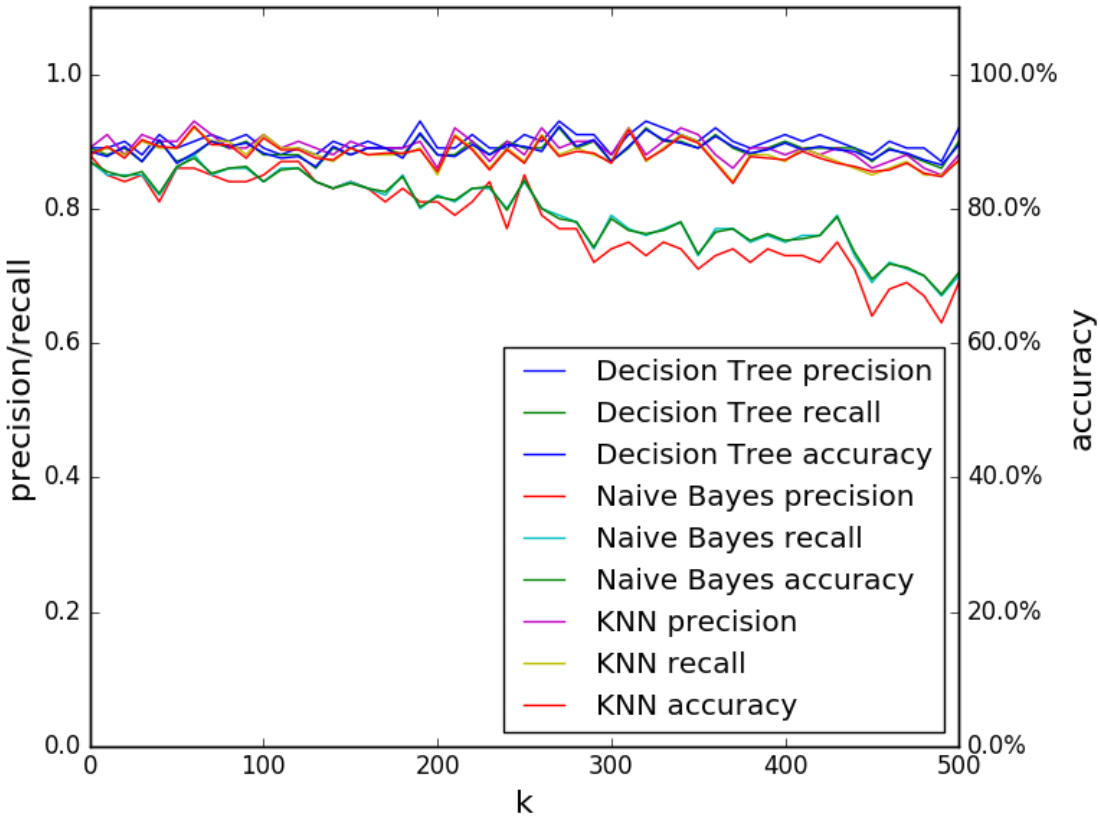


Figure 3.8: Precision, recall and accuracy scores of classification on the fingerprints when varying both power and load of the motor.

simply be sent as part of the packet. 2. Protocols that do not have native support for timestamps (e.g., Modbus as used in the experiment) can have programmable logic controllers (PLCs) store such information as values in their registers, and later send these values back to the host. 3. Neither of the above two methods are available. For the first two options, a timestamp is taken by the PLC in real time and is relatively accurate (the results above illustrate this scenario). In the last option, the operation time is taken using a tapping point which monitors the packets flowing in the network. In such case, network delay can add to the measurement of operation time and thus impact the accuracy of the timestamps.

Recall that the results thus far represent scenarios (1) and (2) above where the scheme is impervious to network perturbations. This is because the actuation time is placed as a value inside the packet. Here, the time can be inferred from the timing of the request/response packets received/sent from/to the actuator. Thus, network delay *would* affect the technique. This method would have to be used if the hardware and protocol used in the CPS do not support the ability to transmit timestamps in the packet, the timestamp in the packet is not trustworthy, or the timestamp is not available (e.g., encrypted). In this scenario, a network traffic monitoring device would be used to timestamp the packets locally. To better understand the effect of network delay on the performance of the fingerprinting technique, a statistical model fitted to real data [67] is used to generate network delay time values without having to perform an enormous amount of experiments. Thus, physical devices may receive the command from the controller earlier or later than in the previous scenario due to the network perturbations.

The network delay values generated through the statistical model are taken and added to the collected timestamps to simulate the data collected by the traffic monitoring device (tapping point) under the influence of network congestion and delay. The shape parameter k is tweaked in the gamma distribution function used in the model while keeping the scale parameter θ at default value 1.0. The mean is thus $k\theta = k$ and variance is $k\theta^2 = k$, with the unit of time in ms. Note that jitter can be modeled by the variance as its definition is

the variation in the delay of received packets. The classification tasks are repeated under different network delay distributions determined by k as well as under three different set of configurations, S_{power}, S_{load} and S_{both}, which denotes varying only power input with fixed load, varying only load with fixed power, and varying both, respectively. The classification result is plotted in Figure 3.6, Figure 3.7 and Figure 3.8. Note, that a larger k value corresponds to more severe network delay and 0 simply means no delay. The performance of almost all classifiers degrades as k increases, however both Decision Tree classifier and KNN classifier remain at > 0.8 precision and recall, and $> 80\%$ accuracy even under a large network delay and jitter. The result suggests that the fingerprinting technique can be robust under the influence of network delay and jitter up to a reasonable level (approximately $500ms$).

3.3.5 Resistance to False Modeling Attacks

As illustrated above, physics-based device fingerprinting can be used to authenticate actuators. Accordingly, it is important to understand the efficacy of such an approach when under attack. Thus, the concept of a false modeling attack is introduced here. In a false modeling attack, it is assumed that an attacker launches the attack on a CPS either inside or outside of the CPS network, and his objective is to sabotage the physical system (i.e., the process) by spoofing commands to the actuators and sending emulated response to the controllers in the Cyber-Physical System (CPS). It can be further assumed that the attacker has some, but not complete knowledge of the physical components in the system. Such an assumption is usually reasonable in most cases, as even a system administrator may not have all details about the underlying physical system, such as the exact model of every device. In order to achieve his objective, the attacker would need to deceive the detection mechanism with network response packets as long as possible, so that he could perform a long enough attack to cause significant damage to the system. As the attacker does not hold every detail of the system, he needs to make assumptions about the missing details in

order to generate the response packets. Without complete knowledge of the exact model of a device missing, the attacker has to randomly guess the model and generate the response packets based on the potentially incorrect model, e.g., assuming a motor of unknown model to be a 24V, 2 horsepower model driving a load equivalent of $0.2kg \cdot m^2$. Hence, this attack is deemed a false modeling attack.

The proposed device physics-based fingerprinting technique can be used to defend against false modeling attacks. Specifically, a two-phased approach is used. During the first phase, a classifier is trained with the fingerprint of each device that is of interest. Such fingerprints could be generated experimentally (black box model), or could be generated by an accurate modeling of the device (white box model) [13]. A more general method could be a mix of both (gray box model). Because the attacker may have limited details of the detection method and actual models of the devices used in a CPS, it is very likely that he will assume a device model different from the one actually being used. Under ideal conditions, the possibility P_{diff} of the attacker guessing the wrong device model is $P_{diff} = 1 - 1/N_{model}$, where N_{model} is the number of models a device can have. Note that the aforementioned technique may fail to detect the spoofed response packets if the attacker picks the correct device model or a similar one with the fingerprints in the packets close enough to the expected response generated by the actual device.

A simulated test is performed to experimentally measure the performance of the aforementioned technique against false modeling attacks. Using the same classifiers trained in Section 3.3.3, the classifiers are tested with the fingerprints generated by randomly chosen configurations of the motor. These test data represent the fingerprints generated by an attacker who does not know which model of the motor is used in the actual CPS. The experiment is again performed under three different device configuration combinations, S_{power}, S_{load} and S_{both}. Under each combination, the attacker correspondingly chooses a device model among the configurations. The results show that the detection rate is very close to the ideal value. For example, when varying only power input (5 different values), the

naive Bayes classifier correctly identifies 79.1% of the attacks on average. When varying only the load (16 different values), 93.7% of the attacks can be correctly identified. The number gets even higher to 98.9% when varying both parameters and all 80 configurations are available. Apparently, with an increasing number of models and configurations for a device, the device physics-based fingerprinting technique has an increasing success rate at detecting false modeling attacks. Clearly, as the recall in some cases is not equal to 1.0, false alarm will occur. However, this problem can be mitigated by: 1) adjusting the tolerance of the difference between the inferred configuration and the true configuration, thus taking a trade-off between precision and recall depending on the specific application, and 2) incorporating the device physics method with other ones, e.g., process physics methods as mentioned at the beginning of this chapter. In reality, if there is a high cost associated with the actions taken in the case of false alarms, a less aggressive defense can be taken at first to minimize the loss due to the shutdown, such as checking network logs to determine if there has been an intrusion.

3.4 Conclusion

Device-modeling is an effective way to improve the security of CPSs. Further, it naturally synergizes with the process modeling technique described in [9] to enhance the security of CPSs. The proposed technique is suitable to be used in a CPS environment as it takes into account both network-side information and the unique physical attributes of the actuators in a CPS, unlike traditional methods which only consider the network aspect of the system. The method is evaluated using experiments involving an emulated device commonly used in an industrial control system (ICS) environment. Its robustness is also evaluated against network delay and jitter, as well as under false modeling attacks.

In the future, the physics-based model extraction process can be automated. Another important consideration for this work is the effect of wear and tear on the devices' operating times which can potentially compromise the effectiveness of the technique over time.

Since the fingerprints are tightly associated with the physical operation of the device, they are subject to change gradually as the device inevitably degrades. Taking this factor into account might not only help to reduce false alarms as device fingerprints deviate from the time they were first collected and stored for reference, but also increases the difficulty for the attacker to spoof the device.

CHAPTER 4

DEVICE PHYSICS AWARE MIMICRY ATTACKS

Chapter 3 proposed to improve the security of CPSs by authenticating the CPS devices through the device operation times in the response packets from the devices, due to the strong correlation between the timing fingerprints and the physics of the devices. Although such a technique may be effective in defending against naive attackers, an advanced attacker may monitor the operation of the CPS before launching a device physics aware mimicry attack. In this chapter, it is shown how the spoofed response packets can be crafted by an attacker to deceive the CPS device authentication method based on the device operation times. Specifically, the timing and physical measurements embedded in the packets are used to reconstruct the devices in the physical system, which can be used to spoof response packets corresponding to the actual model and configuration of the devices in the CPS. The performance of this technique is demonstrated on realistic testbeds with real devices. Finally, an upgraded defense mechanism is proposed which may be used against such mimicry attacks.

4.1 Introduction

With the increased proliferation of Cyber-Physical Systems (CPSs), there have also been more frequent attacks on CPSs. While some attacks are wide-spread similar to computer malware that aims for better coverage, the most devastating attacks tend to be targeted. The most well-known such attack on a CPS is Stuxnet, which is a malicious worm targeting the Supervisory Control And Data Acquisition (SCADA) systems, specifically infecting and reprogramming Programmable Logic Controllers (PLCs). It was responsible for causing tremendous damage to Iran's nuclear program, by driving the fast-spinning centrifuges in Iran's nuclear facilities to a failed state. Although there have been no official conclusion

as to who is responsible for this attack, the size and sophistication of the worm have led researchers to believe that at least one nation-state was involved [68]. Noticeably, a dossier published by Symantec suggested that the attackers were most likely to have conducted a significant amount of **reconnaissance** [3]. As each PLC is configured in a unique manner, the attackers would first need the schematics of the industrial control system (ICS) [6]. An attacker would then need to know details of the individual device's physical behavior in order to maximize the damage in the following targeted attack. For example, Stuxnet checks a Profibus identification number corresponding to two different models of variable frequency drive (VFD) [1], which are used to control the motors. Two different attack sequences are chosen depending on the type of VFD found. A similar incident where attackers retrieved information about the target system before mounting the actual attack occurred in December 2015. A piece of malware specially crafted to attack a Ukrainian electric utility caused a blackout in a portion of its capital equivalent to a fifth of its total power capacity [5]. It even sabotaged power distribution equipment, complicating the restoration of power [7]. A critical step in this attack was to seize control of the SCADA system and to remotely shut down substations. The attack was found to be a premeditated multi-level invasion. The attackers were thought to have hidden in the IT network of a utility company for six months, **collecting data to figure out the inner-workings of the system** before performing the actual attacks.

The key difference between the attacks on CPSs and those on traditional IT systems is the *physical* nature of CPSs. While the goal of attackers in traditional IT systems may be stealing users' private information, those who target CPSs can cause serious damage to the real world. An attack on critical infrastructures may directly threaten people's daily lives, leaving millions of dollars and even human lives at risk. However, the physical nature of CPSs can be a double-edged sword. For example, many studies have proposed to leverage the physical domain as a channel to secure CPSs [8, 9, 10, 12, 69, 13]. Specifically in [13],

[1] Part number KFC750V3 manufactured by Fararo Paya in Tehran, Iran. Vacon NX VFD manufactured by Vacon in Finland.

Formby et al. modeled the physics of ICS devices and demonstrated that the operation times of these devices can be used to generate fingerprints, which are capable of verifying the integrity of the devices' response packets subject to a false data/response injection attack. Their technique uses a high fidelity intrusion detection system (IDS) to detect the minute differences between the network packets sent by authenticated devices and those spoofed by the attacker. Although such a technique may detect naive attackers who are unaware of the physics-related response packets or lack proper equipment to craft accurately timed packets, this chapter shows that an advanced attacker may bypass the defense by launching a **mimicry attack** via a compromised PLC. Existing work have demonstrated the vulnerability of the PLC. For example, Garcia et al. showed that a PLC rootkit can be implemented to launch a stealthy attack in CPS [70]. Such exploits provide a basis for attackers to directly access and manipulate the commands for actuators and sensory data, as well as means to produce packets with accurate timing. **Hence, the defense proposed in [13] may be defeated under such circumstances.**

This work intends to study how an advanced attacker who is aware of the detection system can spoof packets that correspond to the actual device model and configuration (DMC) to avoid being detected by the techniques as described in [13]. More specifically, in this chapter, it is illustrated that an advanced attacker who can perform reconnaissance when attacking a targeted system and obtain the DMC in the CPSs can better evade detection, and hence calls for a more in-depth defense. Note that for the rest of this chapter, "device" refers to the "actuator" in the CPSs. To give a brief description of the method presented in this chapter, it starts with modeling different types of devices based on their construction and the physical process in their operation. Mathematical equations are derived to describe the operation of the devices and used to characterize the devices in the network domain. Using real testbeds that are built, it is demonstrated the process of inferring the models and configurations of the devices from their response traffic, which in turn are used to forge the responses of the same devices. With the results from the testbeds, it is shown that the

forged responses are much more difficult to be detected using device physics fingerprinting methods.

4.1.1 Observation

In CPSs, physical devices either take commands from the *cyber* side of the CPS and operate objects in the physical world (i.e., actuators), or provide digitized information of the physical objects to the controller or monitoring system (i.e., sensors). In [13], Formby et al. discussed how the actuators have to obey the laws of physics when executing commands sent by the controllers. During the experiments, it is found that these actuators exhibit different temporal features which are correlated with the device physics when carrying out the actions.

For example, two motors given the same command of *set to full speed* may take a different amount of time to accelerate to full speed, depending on the torque generated by the motor and the magnitude of its load. Likewise, a valve given a command of *close* or *open* can take an interval of time defined by its specification set according to the mechanical and electrical properties, which include the physical composition of the valve (e.g., torque characteristics of the motor, power rating, gear ratio, size of the fluid passage, etc.). In some cases, such responses may be recorded and used as replay attack. However, there are two problems that must be addressed for such an attack to succeed. First, the responses for certain types of devices are not constant over time, and may be a function of the run-time condition of the devices. Second, the accuracy required to reproduce such responses may exceed the capability of many embedded devices as shown in [13].

It can also be noticed that an actuator can generate feedback information during the execution or after the completion of a command, either actively or passively. The feedback signal carries the information about the actuator's physical attributes, which can then be used to infer the model and configuration of the device. Taking the valve again as an example, all valves used in the experiment are capable of outputting signals to indicate its

real-time status, such as open/closed or the percentage of opening. The response can be used as a fingerprint that uniquely identifies the device.

In this chapter, the device responses are leveraged to infer the knowledge of the devices in the CPS. Without loss of generality, a typical set of devices that adequately represent the common types of actuators in CPS are considered. It is found that the response packets contain useful information that reflects the model and configurations of the device. Finally, the obtained information is applied to forge response packets corresponding to the actual devices.

4.1.2 Challenges

As a relatively less studied area in security research, CPS security research poses several challenges, some of which are common to the other areas of in security research, while others are inherent to the multidisciplinary nature of the topic itself.

CPS research often involves knowledge in both the network security domain and other fields, such as process control, actuation technologies, sensor networks, physics modeling, etc. This requires researchers to understand and incorporate many technical fields that were not previously associated.

Unlike other data-driven research, such as network traffic analysis, web security, or social network privacy analysis, it is extremely difficult to obtain sufficiently large amount of data to analyze in CPS research. This is partially due to the fact that CPSs have not been nearly as popular as other information systems. Another reason is that for many CPSs, the critical nature of these systems means that their data has traditionally been kept private. Extensive research based on the data collected on these systems could be prohibitively costly, if not impossible, due to the system stability concerns, remote locations, and the sensitive information involved. Furthermore, real attack data is rare and even emulated attacks are rarely feasible on real systems, because such experiments could be cost prohibitive and even life threatening.

Using testbeds as substitutes for the real physical systems is a common practice in CPS research [71, 8, 9, 10]. However, small testbeds or software simulated models may have large differences compared to the real system in an industrial environment, while large testbeds can be very expensive and take a long time to build and collect data. A trade-off between the complexity of the system and the authenticity of data needs to be made in order to achieve a balance between the validity and practicability of CPS research.

4.1.3 Contributions

The contributions made in this chapter are summarized as follows:

- This work incorporates the physical domain of the CPSs, and extends the idea of reconnaissance in an attack down to the physical level of CPSs.
- The timing-based fingerprinting techniques are exploited to infer the models and configurations of the devices in CPSs. The method not only can defeat the device-physics based defenses (e.g., [13]), but also provides the information for the attacker to launch more targeted attacks against the physical processes.
- Testbeds are built to emulate common industrial systems and collect data from multiple types of real CPS devices to verify the effectiveness of the model and configuration inference technique. The results are promising and show that both the devices' models and configurations can be inferred through the response packets, and responses can be forged using the inferred values.
- Several possible defenses are discussed and a challenge-response based defense mechanism is proposed for the device physics aware response-spoofing attacks.

4.1.4 Attacks in CPS

Despite various attack vectors to penetrate the CPS and take control of one or more components, it would be a fair assumption that a potential goal for an attacker is to drive the

physical plant to an unsafe state, i.e., attacking either the actuator(s), sensor(s) or controller(s) serves to achieve this goal. One way to do this is spoofing control commands to the actuators or sending incorrect sensor values to the controller. For example, Gu et al. [19] demonstrated using real testbeds that such attack can be stopped by checking the timestamped responses (i.e., device fingerprint) from the actuator (that may have been given a tampered command) against previously known fingerprint of the authentic device. It may also be possible to sabotage the controller and execute malicious routines to send false commands to the actuator. Physical attacks against the plant is certainly another option. However, defending against direct physical access to the plant falls outside the realm of the cyber world and hence the discussion of this chapter.

4.2 Problem Description

This work can be motivated with a realistic attack scenario as depicted in Figure 4.1, where a centrifuge (the plant) and a small part of the SCADA system are shown. The objective of the system is to set the power of the centrifuge and monitor its safe operation from the supervisory host. The control command is sent from the supervisory host over the local area network (e.g., Ethernet) using industrial standard communication protocols, such as Modbus or Ethernet/IP. The PLC starts the motor which drives the centrifuge upon receiving the command from the host (e.g., set the centrifuge to run at 25% power). In the meantime, the motor sends back its real-time speeds which are timestamped and encapsulated by the PLC to the supervisory host. The PLC also receives various measurements (e.g., temperature, vibration, etc.) from the sensors connected to the centrifuge and adjusts the control output accordingly.

To bound the problem, this chapter focuses on attacking the system by spoofing the falsified commands to control the devices, while sending back responses to the original commands that can deceive the device fingerprint detection mechanism. The attacker's objectives are two-fold: a. to obtain the models and configurations of the devices (e.g.,

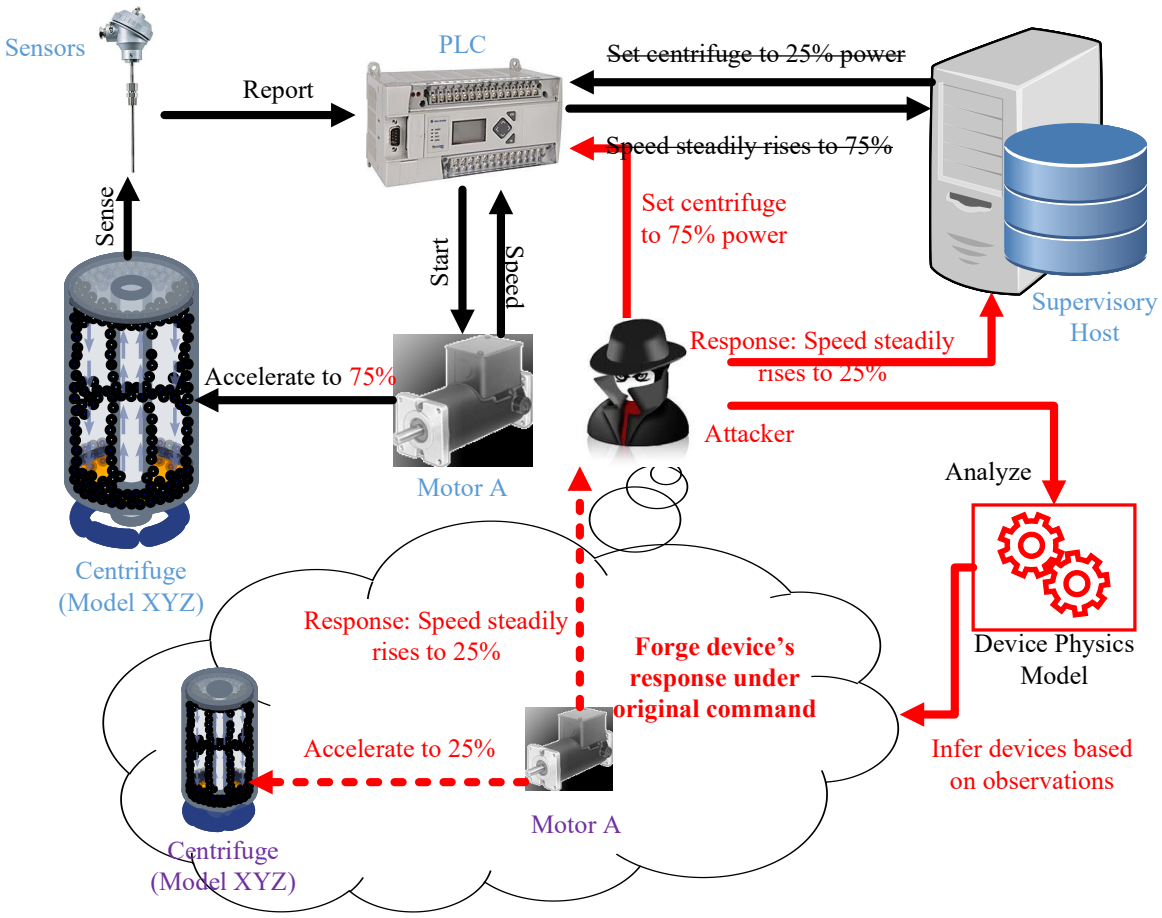


Figure 4.1: Attack model used in this chapter, where the attacker injects a false command to the PLC and a forged response to the supervisory host. The attacker needs to first observe the legitimate traffic to infer the actual devices' models and configurations before spoofing the response.

details related to the motors or centrifuges such as their weight and capacity); b. to forge responses based on such information. It is worthy noting that the information related to their physical properties is not explicitly transmitted in the network. Thus, the attacker can only learn such information through passive observation of the network traffic in order to minimize disturbance to the system before mounting the actual attack.

4.2.1 Attack Model

In this work, an attacker is assumed to have gained access to the corporate network and can spoof the control command sent by the PLC to the actuators. He can modify the network traffic between the PLC and the supervisory host by compromising the PLC firmware/program, but does not have access to the supervisory host. Such an assumption is reasonable, as the PLCs are usually not protected with mature defense mechanisms, while the supervisory host is usually a relatively powerful computer system that is equipped with modernized defense techniques. The attacker is also assumed to have only network access but no physical access to the target CPS, hence can only observe and modify the PLC firmware and the network traffic. Such network traffic does **not** include direct information of the models (e.g., manufacturer's name, model number) and configurations (e.g., load percentage) of the sensors and actuators, as these devices are controlled with electrical signals by the PLC, and do not directly transmit data/packets on the network. The attacker can carry out reconnaissance first to collect data on the general system architecture, but does not have information on the specific model and configuration of each device. For example, an attacker who targets a critical infrastructure such as a thermal power plant may be able to get information on its commission date and capacity [72]. Finally, the attacker also has access to the specifications for all models of the target device types, and can acquire specific models of the devices to perform experiment and build a catalog of their signatures.

The attacker's objective is to sabotage the physical process of the CPS by injecting a false command to the PLC. However, the attacker is also required to deceive the IDS by

injecting the forged response corresponding to the original command, as the IDS ensures the system is intact and normally operating by checking the response from the PLC against the expected response from the underlying physical devices. Note that a naive replay attack may result in a failure as mentioned in Section 4.1.1. Because of the imperfection in the timing of the replay attack, the threshold for mounting a successful replay attack is high [13]. Hence he must achieve enough precise timing control of the spoofed packets, whether the spoofed packets were pre-recorded or dynamically generated.

4.2.2 Formal Definition of the Device Response Mimicry Attack Problem

The primary goal of this work is to infer the models and configurations of the devices in CPSs. The assumptions used in this study include:

1. There exist $\mathcal{N}$ product models $\mathcal{D}_i$ ($i \in [1, \mathcal{N}]$) for a given device type. Each $\mathcal{D}_i$ has $\mathcal{M}$ configurations $\mathcal{D}_{i,j}$ ($j \in [1, \mathcal{M}]$).

2. The attacker initially does not have knowledge of the value for i and j. By observing the legitimate responses $\mathcal{R}_{i,j}$ sent by a device $\mathcal{D}$, the attacker classifies $\mathcal{D}$ into $i_a \in [1, \mathcal{N}]$ and $j_a \in [1, \mathcal{M}]$.

3. The attacker injects false command $\mathcal{C}_a$ to alter the original command $\mathcal{C}$ sent to the the device $\mathcal{D}_{i,j}$, while spoofing the finite-time response $\mathcal{R}_{i_a,j_a}$ of the device $\mathcal{D}_{i_a,j_a}$ under command $\mathcal{C}$.

The problem can be formally defined as a two parts. The first part is a classification task, where the product model i_a and configuration j_a needs to be determined based on the observation of $\mathcal{R}_{i,j}$ and $\mathcal{C}$. The second part is an attack on the binary classifier trained with $\mathcal{R}_{i,j}$, where goal is for the spoofed response $\mathcal{R}_{i_a,j_a}$ to be classified as legitimate.

Table 4.1: List of CPS devices and their physical properties

Device Type	Load Dependent	Input Type	Output Type	Motion Type
Relay	No	Binary	Binary	N/A
Valve	Slightly	Binary/Analog	Binary/Analog	Linear
Pump	Yes	Binary	Analog	N/A
Stepper Motor	Yes	Analog	Analog	Rotary
Solenoid	No	Binary	Binary	Linear
Electric Motor	Yes	Analog	Analog	Rotary
Hydraulic Cylinder	Yes	Analog	Analog	Linear

4.3 Methodology

As stated in Section 4.2, the proposed method focuses on inferring the information of a device in a CPS in two aspects: which specific **product model** a certain device corresponds to, and what **run-time configuration** it runs in. For example: a motor in the schematic of a CPS (e.g., a conveyor system in a factory controlled with SCADA) can be implemented with the product selected from various brands and model numbers. The specific **product model** number used during the construction of the system can be chosen from a variety of available products, as long as it satisfies the required constraints. The parameters of each model may also vary within a reasonable range, e.g., power rating at $500W$ versus $600W$. The **run-time configuration** refers to the configurable states with which the device directly interfaces in the CPS, e.g., the speed setting of the motor, and the load attached to the motor's output shaft.

4.3.1 Device Physics Modeling

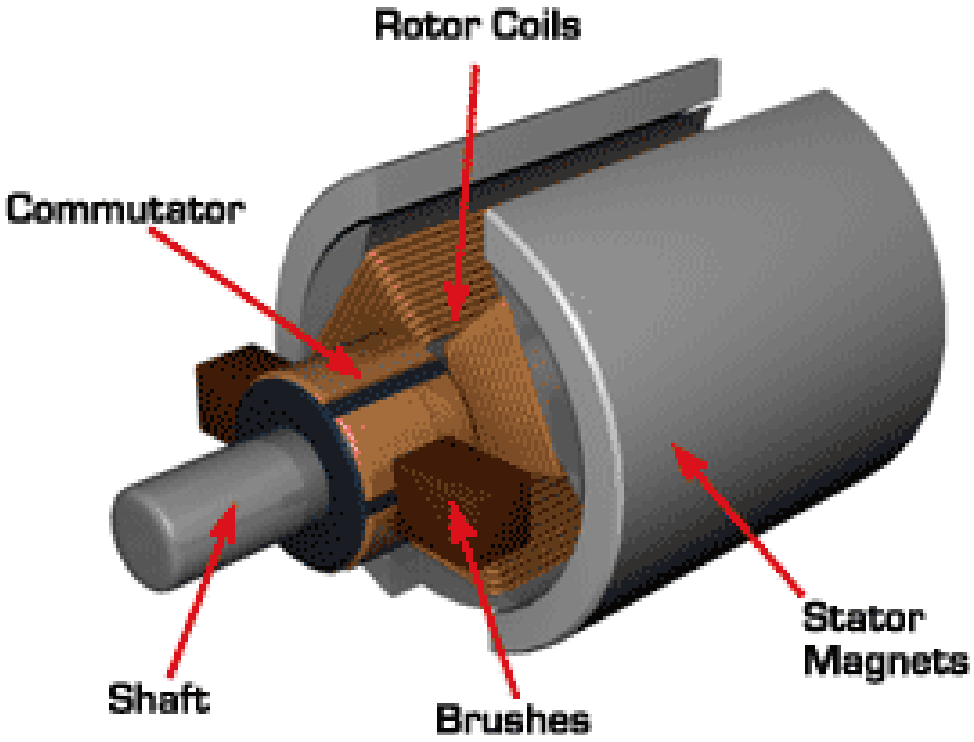
To have an understanding of how the DMC inference technique can be applied among different devices, it is necessary to first build the physical models of various devices. In this section, the model of one device is built as an illustration, namely the electric motor. Table 4.1 lists the comparison of seven common devices in a CPS and their physical properties. *Load Dependent* refers to whether the output behavior of the device depends on

its load. *Input/Output Type* means whether the device takes/generates binary (e.g., on/off, closed/open) or analog (e.g., continuously variable speed) signal/states. Note that some devices such as valves can have both binary and analog types of input/output, depending on its model and application. Without loss of generality, three types of devices are leveraged which cover the most variations in each property dimension to demonstrate the proposed method, namely electric motor, relay, and valve. The electric motor is taken as a running example and begin with the mathematical modeling of its device physics. The processes for building three other types of devices can be found in the appendix.

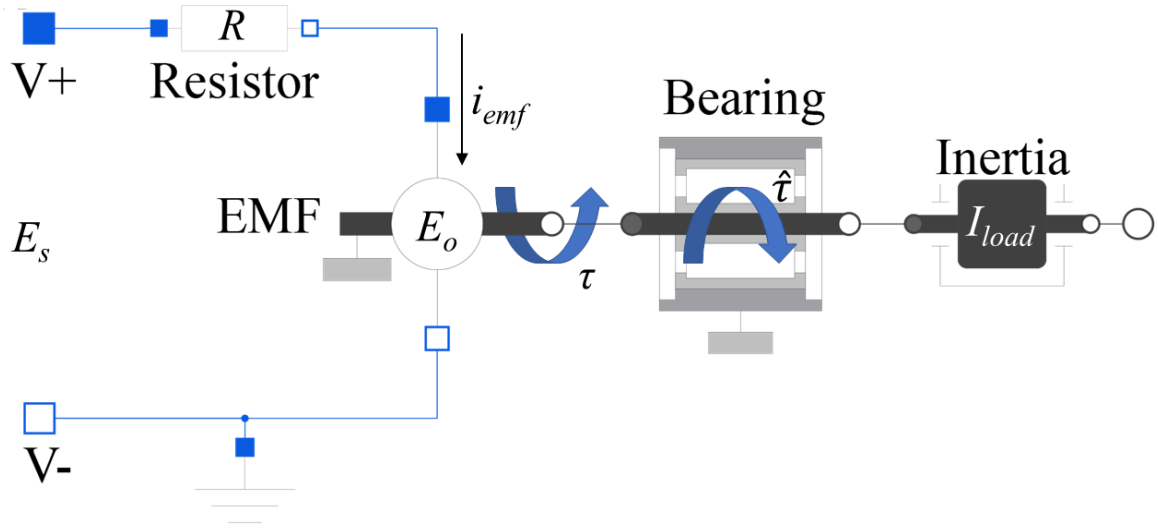
There are mainly two types of motors, namely direct current (DC) motor and alternating current (AC) motor. Each type is powered by its corresponding source of electricity. The motors can be further classified by the internal construction, application, type of motion output, and so on. For the purpose of modeling the physics of an electric motor, a specific type named *permanent magnet DC* (PMDC) motor is chosen.

A PMDC motor is like similar to other motors, which consists of a *stator* and a *rotor*, as shown in Figure 4.2a. In the case of a PMDC motor, the rotor is also the *armature*, which carries current in the coils winding on it. The rotor has an integral part, called a *commutator*, which can change the direction of current flowing in the coils. The commutator ensures that the magnetic field generated by the rotor coils always produces a "repelling" force which drives the rotor in the same direction. In contrast, the stator is the *field* and produces a constant magnetic field that interacts with the rotor's. The physical connection of voltage to the armature is done through *brushes*, via the metal plates on the commutator [74].

A physical model connecting the electrical domain to the mechanical domain is illustrated in Figure 4.2b. A circuit loop is formed between the positive and negative leads (i.e., brushes) connected by the rotor coils, where E_s is the source DC voltage, i_{emf} is the armature current, R is the armature resistance, and E_o is the induced *counter-electromotive force* (CEMF), as its polarity always acts against E_s. E_o is generated on the abstract component



(a) Construction of a PMDC motor. The brushes are connected to the positive and negative voltages [73].



(b) Model of the PMDC motor connecting the electrical and mechanical domains.

Figure 4.2: Physical construction and abstract model of permanent magnet DC motor.

EMF, which produces a torque τ on the shaft connected to the load through a bearing. The load element models the total *moment of inertia* (MOI) I_{load} of the rotor itself and the external object driven by the motor. In practice, a bearing is used to support the rotor while exerting a friction $\hat{\tau}$ on it, in the reversed direction of τ.

Using *Ohm's Law*,

$$i_{emf} = \frac{E_s - E_o}{R}. \tag{4.1}$$

The power P produced by *EMF* is defined as

$$P = E_o i_{emf}. \tag{4.2}$$

E_o is induced in the armature conductors as they cut the magnetic field produced by the permanent magnets with winding coefficient Z and flux per pole F, given by the equation

$$E_o = ZnF/60, \tag{4.3}$$

where n is the rotor's rotation speed in revolutions per minute (RPM). Using *Newton's second law for rotation*, the torques τ and $\hat{\tau}$, inertia I_{load} of the load and angular acceleration α of the rotor can be expressed as $\tau - \hat{\tau} = I_{load}\alpha$. The mechanical power P is given by the expression

$$P = \tau n \times \frac{2\pi}{60}. \tag{4.4}$$

Combine Equations 4.2 and 4.4 and plug in Equations 4.1 and 4.3,

A physical model connecting the electrical domain to the mechanical domain is illustrated in Figure 4.2b. A circuit loop is formed between the positive and negative leads (i.e., brushes) connected by the rotor coils, where E_s is the source DC voltage, i_{emf} is the armature current, R is the armature resistance, and E_o is the induced *counter-electromotive force* (CEMF), as its polarity always acts against E_s. E_o is generated on the abstract component *EMF*, which produces a torque τ on the shaft connected to the load through a bearing and

can be expressed as

$$E_o = ZnF/60, \tag{4.5}$$

where Z is the winding coefficient, n is the rotor's rotation speed, and F is the flux per pole. Using *Ohm's Law* and *Newton's second law for rotation*, the equation which governs the dynamics of the motor is thus

$$\tau = \frac{ZF(E_s - ZnF/60)}{2\pi R}. \tag{4.6}$$

Consider the case where the load is initially at reset and accelerated by applying a constant voltage E_s to the motor. The angular velocity ω over time t satisfies the equation

$$\omega(t) = \int \alpha dt = \int \frac{ZF(E_s - ZnF/60)}{2\pi R I_{load}} dt - \int \frac{\hat{\tau}}{I_{load}} dt, \tag{4.7}$$

under the boundary condition $\omega(0) = 0$. Solving the differential Equation 4.7 and substituting $n = \frac{30}{\pi}\omega$, an exponential decay function is obtained

$$\omega(t) = -Ae^{-t/B} + A, \tag{4.8}$$

where $A = (\frac{ZF}{2\pi R}E_s - \hat{\tau})\frac{4\pi^2 R}{Z^2F^2}$ and $B = \frac{4\pi^2 R}{Z^2F^2}I_{load}$. Recall that Z, F, R are constants determined by the specific construction of a motor, hence are **product model** related. $\hat{\tau}$ is independent of ω and can be assumed to be constant too. Therefore, A is linearly correlated to the source voltage E_s and B is proportional to I_{load}. A is **product model** related parameter, while B is a **configuration** related parameter given a fixed product model.

Recall that in Figure 1.1, the controller sends a command to the actuator and receives response when the actuator executes the command. In this case, the command can be *start motor*, and the response is the timestamped rotation speed of the motor. As network packets containing the timestamps and speed values are sent back from the controller to the other host(s), a time series can be extracted from the packets that correspond to Equation

4.8. Such time series satisfy the equation when the proper device physics parameters are plugged in.

4.3.2 Characterization

The output type of each device in Table 4.1 is classified as either binary or analog. Combining this with the physics models of each device, it can be found that each device has a deterministic response which can be observed and measured. Two types of response from a CPS device are defined as follows.

Operation Curve. For those that have an analog output, e.g., speed of a motor, position of a modulating valve, a time series data can be obtained by continuously sampling the output value with a timestamp. This type of data is referred to as the operation curve.

Operation Time. Devices with binary output, such as the open/closed states of a relay or a two-position valve can be characterized based on the time difference between the applied signal and the desired state change. This time difference is called the operation time

The operation curve is a direct result of the signal given to the device, as well as the specific physical construction of the device itself. Similar to the principle behind the operation curve, the operation time is also dependent on the physics process inside the device. Therefore, both the operation curve and the operation time are a function of the device physics, defined by its governing equations and the parameters. Whether the reverse is also true is the key to infer the device physics from its observable output, and will be discussed in the following sections.

4.3.3 Device Model and Configuration Inference

There are two methods to infer the DMC in CPS, namely forward model inference and backward model inference. The forward model inference method takes as input the mathematical equations relating the device's product model and run-time configuration to its

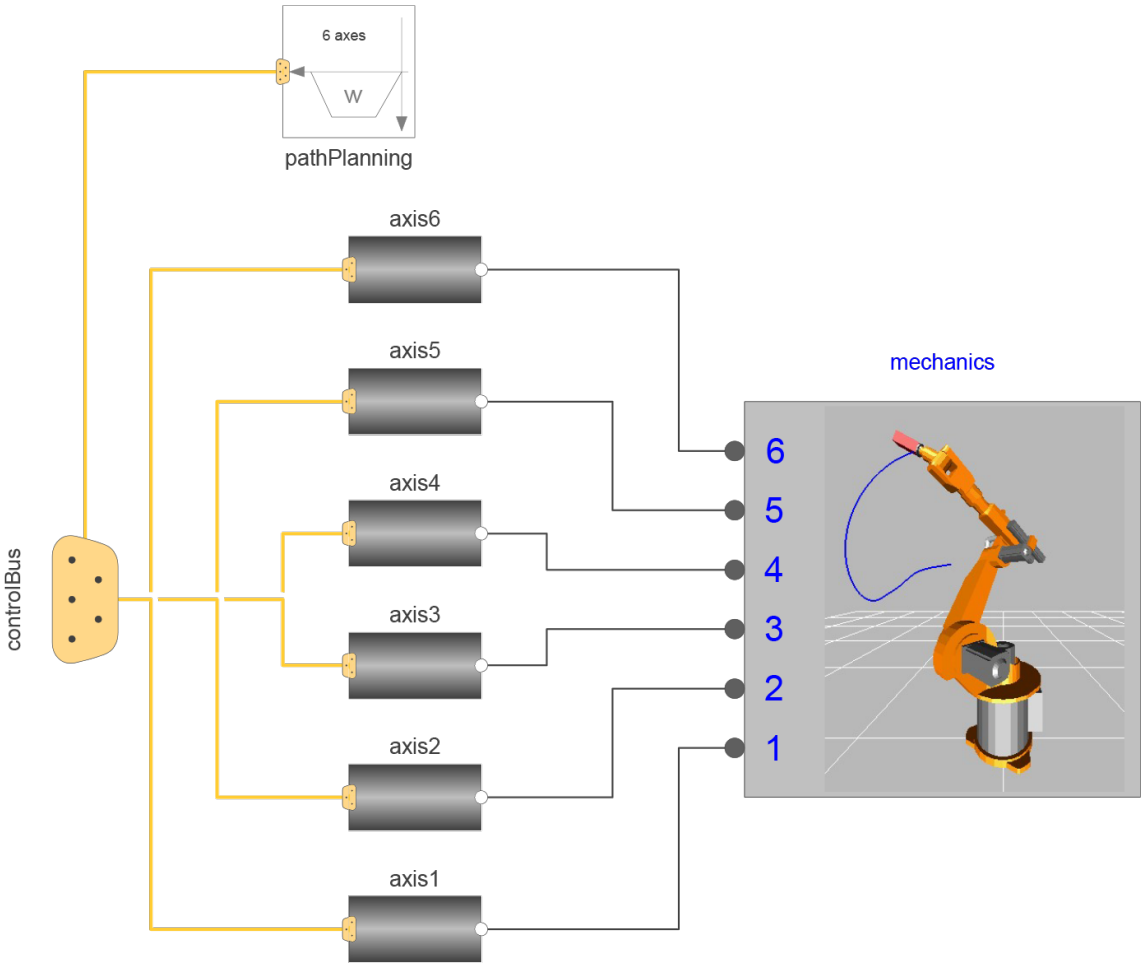
operation curve or operation time. It then simulate the model under all parameter combinations and compare the output with the observed response to find the closest one. The backward model inference method takes the same equations as well as the observed response as input, and reverse the process to compute the product model and run-time configuration related parameters.

Forward Model Inference

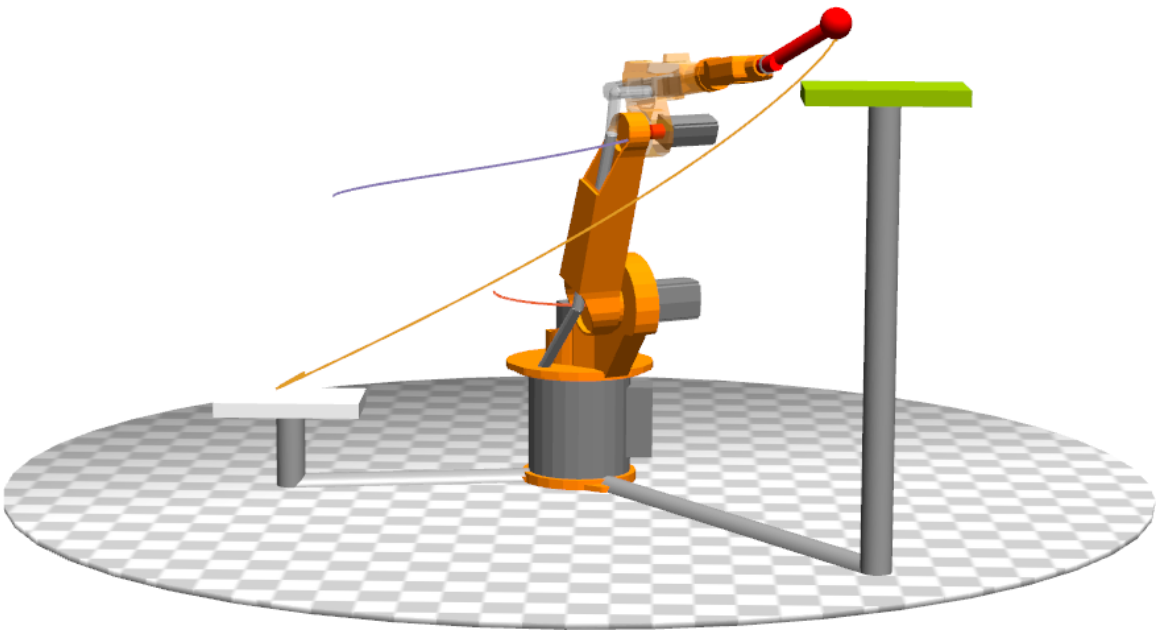
To classify the device feedback into the specific product and configuration, one way is to simulate the device's operation using the mathematical models derived in Section 4.3.1. A software specifically designed for this purpose called Wolfram SystemModeler [75] is used. Figure 4.3 shows an example an industrial robot simulation using the software. The system is modeled using a language called Modelica, which is an object-oriented, equation based language to conveniently model complex physical systems. Modelica can be used to model many CPS related domains, including mechanical, electrical, hydraulic, thermal, control, electric power or process-oriented sub-components. More importantly, the open source Modelica Standard Library contains thousands of model components and functions [76] which can be easily extended to model the devices in CPS.

For each type of device, a Modelica model $\mathcal{M}(\mathcal{P}, \mathcal{O})$ is constructed, which contains the equations of the physics process of the device. $\mathcal{P}$ is a vector containing all the parameters related to the product model and run-time configuration, and $\mathcal{O}$ is the observable output of the device. There are a total of n_p set of combinations of the values in $\mathcal{P}$. Figure 4.4 shows the Modelica model $\mathcal{M}_{PMDC}$ of the PMDC motor based on the equations in Section 4.3.1, where $\mathcal{P}_{PMDC}$ includes R, Z and F of the motor, as well as I_{load} and E_o from its configuration. The output $\mathcal{O}$ is a time series value $\omega(t)$.

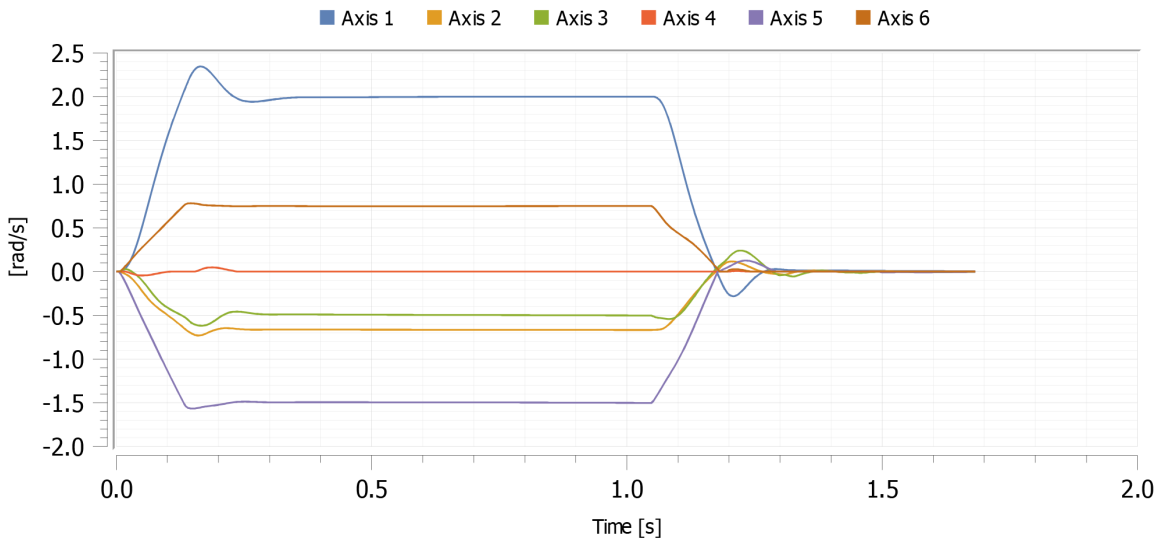
A simulation of $\mathcal{M}(\mathcal{P}, \mathcal{O})$ is then performed using Wolfram SystemModeler for each set of combination of the values $\mathcal{P}_i$ and a corresponding output $\mathcal{O}_i$ is generated. A mapping $\mathcal{N}$ exists between each $\mathcal{P}_i$ and $\mathcal{O}_i$, where $i \in [1, n_p]$. Finally, a distance $dist_i$ between the



(a) Graphical representation.



(b) 3D rendering showing the path of each component.



(c) Plot of the angular velocities for each axis.

Figure 4.3: An example showing the model and simulation results of an industrial robot using Wolfram SystemModeler.

observed device response $\mathcal{O}_{attack}$ and each $\mathcal{O}_i$ is calculated using the appropriate metric. In this work a number of different metrics are used to compare their performance, including Euclidean, Chebychev, Manhattan and cosine distance. Finally, all distances $dist_i, i \in [1, n_p]$ are ranked in the ascending order. The $\mathcal{O}_j$ with the top k distance $dist_j, j \in [1, k]$ are thus the closely matched set of device responses, and a simple lookup in the mapping $\mathcal{N}$ gives the k most likely set of DMCs $\mathcal{P}_j$.

Backward Model Inference

Another method which used to achieve the same goal, i.e., identify the device product model and configuration from its response is using non-linear least squares method. It is a form of least squares analysis used to fit a set of m observations with a model that is non-linear in n unknown parameters ($m > n$).

Consider device's response to be a set of m data points, $(t_1, \mathcal{V}_1)$, $(t_2, \mathcal{V}_2)$, ..., $(t_m, \mathcal{V}_m)$. Without loss of generality, $\mathcal{V}_i$ is a vector of all the values in the response at time t_i, where $i \in [1, m]$. Assume that the function mapping all the product model and run-time configuration

```
model DCMotorEMF "Electromotoric force (electric/mechanic
    transformer) of a permanent magnet DC motor"
  parameter Boolean useSupport = false "= true, if support
      flange enabled, otherwise implicitly grounded";
  parameter Modelica.SIunits.ElectricalTorqueConstant k "
      Transformation coefficient";
  Modelica.SIunits.Voltage E0(start = 0) "Induced voltage";
  Modelica.SIunits.Current i "Current flowing from positive to
      negative pin";
  Modelica.SIunits.Angle phi "Angle of shaft flange with respect
      to support (= flange.phi - support.phi)";
  Modelica.SIunits.AngularVelocity w "Angular velocity of flange
      relative to support";
  Modelica.Electrical.Analog.Interfaces.PositivePin p;
  Modelica.Electrical.Analog.Interfaces.NegativePin n;
  Modelica.Mechanics.Rotational.Interfaces.Flange_b flange;
  Modelica.Mechanics.Rotational.Interfaces.Support support if
      useSupport "Support/housing of emf shaft";
equation
  phi = flange.phi - internalSupport.phi;
  w = der(phi);
  E0 = p.v - n.v;
  E0 = k * w / (2 * Modelica.Constants.pi);
  0 = p.i + n.i;
  i = p.i;
  flange.tau = -k * i / (2 * Modelica.Constants.pi);
  connect(internalSupport.flange, support);
  connect(internalSupport.flange, fixed.flange);
protected
  Modelica.Mechanics.Rotational.Components.Fixed fixed if not
      useSupport;
  Modelica.Mechanics.Rotational.Interfaces.InternalSupport
      internalSupport(tau = -flange.tau);
end DCMotorEMF;
```

(a) EMF component of a PMDC motor.

```
model DCMotorFriction "DC Motor with bearing friction"
  parameter Real R;
  parameter Real J;
  parameter Real EMFk;
  parameter Real friction;
  Modelica.Electrical.Analog.Interfaces.PositivePin pin_p;
  Modelica.Electrical.Analog.Interfaces.NegativePin pin_n;
  Modelica.Mechanics.Rotational.Interfaces.Flange_b flange_b;
  Modelica.Electrical.Analog.Basic.Resistor resistor(R = R);
  Modelica.Electrical.Analog.Basic.Ground ground;
  Modelica.Mechanics.Rotational.Components.Inertia inertia(J = J
    );
  CPS.DCMotorEMF dCMotorEMF(k = EMFk);
  Modelica.Mechanics.Rotational.Components.BearingFriction
    bearingFriction(tau_pos = [0, friction; 1, friction], peak
    = 1);
equation
  connect (pin_p, resistor.p);
  connect (ground.p, pin_n);
  connect (resistor.n, dCMotorEMF.p);
  connect (dCMotorEMF.n, pin_n);
  connect (dCMotorEMF.flange, bearingFriction.flange_a);
  connect (bearingFriction.flange_b, inertia.flange_a);
  connect (inertia.flange_b, flange_b);
end DCMotorFriction;
```

(b) A PMDC motor with bearing friction and MOI.

Figure 4.4: Modelica models for a PMDC motor. Code used from Modelica Standard Library is not shown.

related parameters $\mathcal{P}$ to $\mathcal{V}$ is $\mathcal{V} = f(t, \mathcal{P})$, where $\mathcal{P} = (P_1, P_2, ..., P_n)$ $(m > n)$. The goal is to find the vector $\mathcal{P}$ such that the curve best fits the given data in the sense of the least sum of squares, i.e.,

$$S = \sum_{i=1}^{m} R_i \cdot R_i \tag{4.9}$$

is minimized, where the residuals $\mathcal{R}_i$ are given by

$$R_i = \mathcal{V} - f(t, \mathcal{P}). \tag{4.10}$$

The Gauss-Newton algorithm can be used to solve this optimization problem. Given an initial value of $\mathcal{P}_0$, an iterative search updates the parameters by

$$\mathcal{P}_{k+1} = \mathcal{P}_k + \triangle\mathcal{P}, \tag{4.11}$$

where k is the iteration number. The best-fitting $\mathcal{P}$ is found when the algorithm converges.

The final step is to map the values in $\mathcal{P}$ to the device's product model and run-time configuration. While the latter can be interpreted numerically, the former one can be found using the specification sheets of a set of candidate devices commonly found in CPSs.

The advantage of this method is that it does not require the assumption of a previously known device list, because it guarantees to find a set of parameters that makes the model output best fit the observed response.

4.3.4 Device Response Packets Synthesis

The last step is to synthesize the device response packets containing the timestamped measurements. The most straightforward method is to plug the parameter values found in Section 4.3.3 back into the device physics modeling equations in Section 4.3.1 using the values corresponding to the device models and configurations. However, the parameter values obtained using the non-linear least square method can vary across different observa-

tions of the device's responses, even when the DMC is constant. This may arise either due to the measurement errors or slight change in these device parameters themselves (e.g., the resistance varies when the temperature changes). Therefore, a sampling process is added to randomly choose an observed value for each parameter before plugging in.

4.4 Experiments

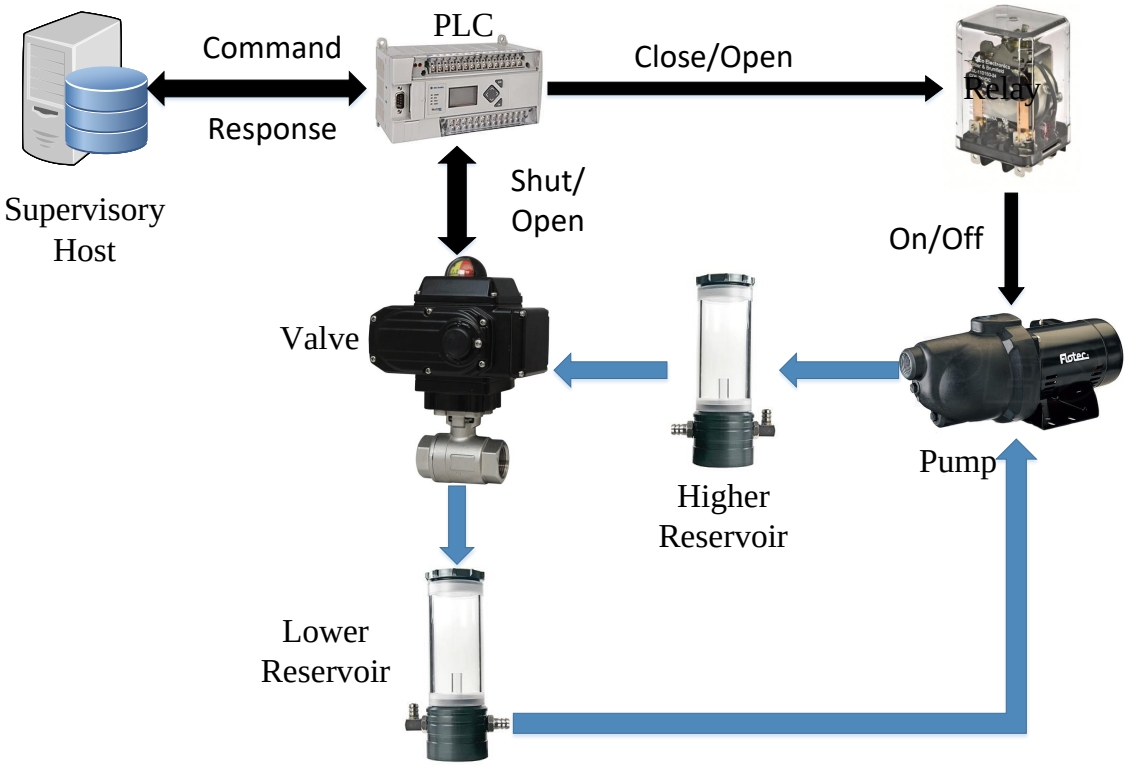


Figure 4.5: Block diagram of Testbed 1 setup. The valve and relay are of specific interest in this study.

It comes with extreme difficulty to obtain real data in CPS due to the cost associated with interfering the normal operation of a potentially valuable system. To address it, two testbeds are set up using real industrial standard devices that simulate corresponding systems that have realistic objectives, such as balancing the liquid level in a container or stirring materials during a chemical reaction in a chemical factory. Each of the testbeds could be a potentially valuable target for the attacker.

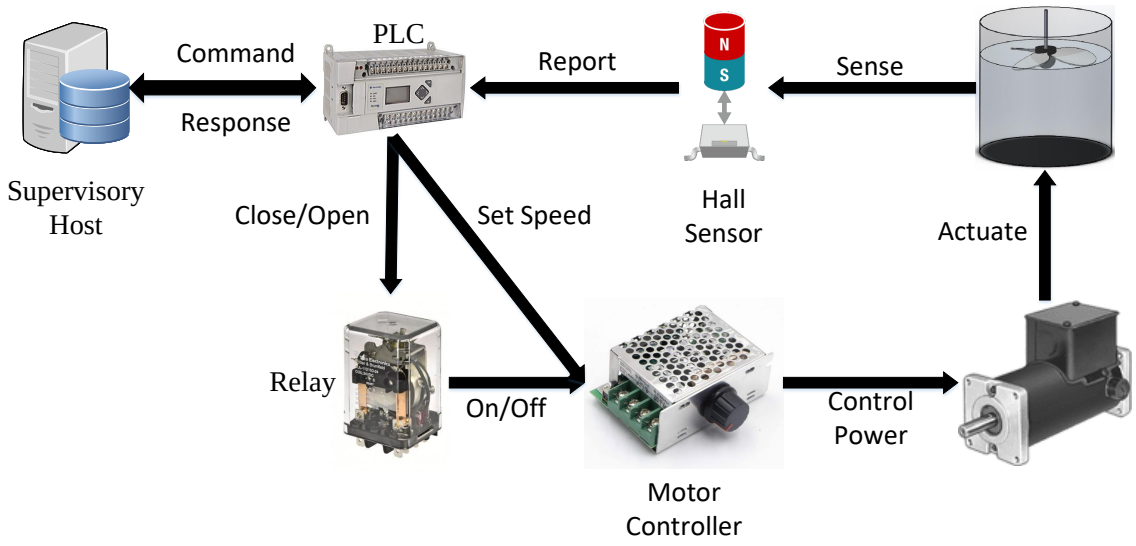


Figure 4.6: Block diagram of Testbed 2 setup. The motor and relay are of specific interest in this study.

Among the devices used to build the testbeds, three types of devices are focused on to infer the DMC of each of the devices, namely electric motor, relay, and valve. Leveraging the three types of devices, two physical testbeds were constructed that appropriately mirror a real-world CPS environment, as shown in Figure 4.5 and 4.6. In this section, how each type of chosen device operates in the testbed is explained, as well as the experiment procedures and parameter settings.

In general, the responses of each type of device are collected, and the operation curves or operation times of each device labeled with the actual model $i \in [1, \mathcal{N}]$ and configuration settings $j \in [1, \mathcal{M}]$ are extracted. These operation curves and operation times are then used as the input to the classifiers to generate the predicted model $i_a \in [1, \mathcal{N}]$ and configuration $j_a \in [1, \mathcal{M}]$.

In this section, the common setup across each testbed is first explained. Then the methodology used to attack each of the three types of devices used in the testbeds are discussed, namely electric motors, relays and valves, followed by explaining the technique used to craft precisely timed response packets. Finally, the results of attacking the three types of devices are presented and interpreted.

4.4.1 Timestamps and Protocols

As can be seen in Figure 3.4, a supervisory host sends high-level command to the controller, and the controller processes the command in the form of network packets and directly controls the field device (motor in this case) to perform the action. This setup corresponds to the Level 0 to 2 in a SCADA architecture. To closely mimic an industrial environment, a PLC is chosen as the controller among other available options (e.g., a PC or embedded platform). In addition to adding authenticity to the testbed, the PLC also comes with an important feature that may be absent from other controllers, i.e., a high precision clock that is used to *timestamp* the events. The PLC used is an Allen-Bradley Micrologix 1400 series. It has a 32-bit high speed clock which provides a timing resolution of $9.92063492\mu s$. Due to the simple program flow and predictable program scan cycles of a PLC, the timing for an event at the PLC can almost always occur immediately after it. In comparison, neither a PC nor an embedded controller such as a Raspberry Pi is able to achieve a constant timing resolution like a PLC, because of their non-real time operating system or much slower clock frequency. The timestamps can either be embedded natively in the packet if supported by the protocol (e.g., distributed network protocol (DNP3)), or taken by the program running on a PLC, if the protocol does not have native support for timestamps (e.g., Modbus). Without loss of generality, the latter method is chosen in all experiments. Note that the timestamps are transmitted to the supervisory host as network packets.

For the command and response sent between the supervisory host and the PLC, Modbus is used as the communication protocol, as it has become a de facto standard communication protocol and is now a commonly available means of connecting industrial devices [77]. The supervisory host runs a Python script leveraging the *modbus-tk* library and acts as an HMI running on the engineering workstation. It is responsible for initiating the communication and sending command to the PLC acting as a Modbus slave, as well as constantly checking for new responses. Each response contains an event, such as a new speed reading in the case of the motor testbed. The event is always accompanied with a timestamp taken by the

high speed clock on the PLC.

4.4.2 Electric Motor

The testbed setup of the electric motor can be seen in Figure 4.6. Note that although the command sent from the host contained both the power setting and a start command, only the power setting is applied at the motor controller as different power output, which is set prior to the start of the motor. This is intended to emulate the scenario where the motor is of a certain model that operates under the given power ratings. Thus, the power setting would be obscured from an attacker who can only observe the network traffic. The shaft of the motor is connected to a load with variable MOI as shown in Figure 3.2. Two hall sensors are placed to enable the PLC to calculate the angular speed of the load. A timestamp relative to the command received from the host is taken and read by the host together with the angular speed of the motor, which forms an operation curve.

Table 4.2: Experiment Settings of the Electric Motor Testbed

Parameter	Type	Range
EMF constant	Product model	Constant
Armature resistance	Product model	Constant
Power rating	Product model	5 values
Load MOI	Run-time configuration	16 values

Both the product model and the run-time configuration related parameters are variable in this testbed. For each start command sent from the host, the responses from the motor is collected while varying its product model and run-time configurations. In summary, 100 trials are performed for each of the 80 experiment settings, and all $8,000$ sets of data are collected. The experiment settings are shown in Table 4.2.

4.4.3 Relay

The relay testbed setup shared a similar structure as that of the electric motor, except that the focus is on different models of the relay and hence are swapped with the motors in

Table 4.3: Key Parameters of the Relays Taken from Their Specifications

Model	Close Time	Open Time
Schneider 785XBXCD-24D	20ms	20ms
Omron G2RV-SR500 DC24	20ms	20ms
TE K10P-11D15-24	10ms	13ms
TE KUEP-11D15-24	10ms	15ms
TE KUIP-14D15-24	15ms	20ms
TE KUL-11D15D-24	25ms	25ms
TE KUP-14D15-24	15ms	20ms
Omron MKS3PI DC24	20ms	30ms
TE MT221024	10ms	15ms
Schneider RSLZVA1	5ms	12ms

the testbed. Because the operation of the relay is hardly affected by the load it controls (i.e., run-time configuration), therefore no load was connected as was in the electric motor testbed.

In reality, the selection of the specific relay model at a single point in a system depends on the application requirements, e.g., the control circuit voltage, socket type, rated operating voltage and current, etc. To set up a realistic experiment, the set of relays are selected using common industrial settings, i.e., $24VDC$ control voltage and DIN rail mounted relays. A total of 10 different relays are found as listed in Table 4.3. The supervisory host sent either a *close* or *open* command to the PLC depending on the last status of the relay. The PLC records the time when the command is received, and either energizes or de-energizes the coil of the relay accordingly. It then polls its input pin connected to one of the contacts of the relay output, while the other contact is connected to a logic high voltage. When the PLC first detects a signal level change, it again records the command completion time. The difference of the two timestamps are calculated and stored with a flag, which can be read by the host as the operation time. For each relay, the close/open cycle is collected for $1,000$ operations.

Table 4.4: Key Parameters of the Valves Taken from Their Specifications

Model	Type	Open/Close Time
Dwyer WE01-CTD01-A	Two-Position	4s
Dwyer WE01-CMD01-A	Modulating	10s
Dwyer WE01-GTD02-A	Two-Position	20s

4.4.4 Valve

Similar to the relay testbed, the components from the supervisory host to the PLC are kept, while replacing the field device with three different valve models listed in Table 4.4.

This setup studies the correlation between the operation of each valve and its specification sheet. In each experiment, only a single valve is connected to the PLC, and different models are swapped. Without loss of generality, two types of valves are used in the experiment, namely the two-position valve and the modulating valve. The two-position valve operates in a binary manner, i.e., the PLC outputs a binary signal to fully *close* or *open* input to the valve. When the valve finished executing the action, a limit switch in the valve would be triggered, and thus the event could be detected by the PLC. A time difference is calculated between the reception of the command and detection of the completion, and is read by the host as an operation time. The modulating valve operates in an analog manner, i.e., the PLC outputs a $4-20mA$ current loop to the valve that linearly translates to a valve position. The valve compares the current loop input with its current position, and executes accordingly to adjust for the difference. Meanwhile, its physical position is continuously translated to another $4-20mA$ current loop, which can be read by the PLC. The PLC keeps polling the readings and stores in its memory with timestamps, which in turn is read by the host as an operation curve. To compare this operation curve of the valve with its open/close time listed in Table 4.4, the operation curve is converted to an operation time by defining a cutoff position that the valve reaches at the end of each actuation, and the difference between the command time and the time of the cutoff position is computed. Another reason which calls for the conversion is that the modulating valves are changing their positions at

a constant speed, unlike the electric motor. Hence converting their operation curve to the operation time resulted in negligible loss of information.

A second testbed is also built to study the impact of load on the valve. The load is defined as the the minimal subset of the process that directly interacts with the device. For a valve, its load is the fluid that it controls, which may be of various viscosity. A testbed is constructed with a pump, a valve and a reservoir, and are connected using tubes to form a closed sealing loop. Different types of fluids are poured into the reservoir in each experiment, including olive oil, canola oil and honey, sorted in ascending order of their viscosity. The pump is used to provide an adequate amount of pressure to keep the fluid flowing when the valve is open. For each fluid used in the loop, the same experiment procedure used for the two-position valve is carried out to measure the operation time of the same valve under different load.

4.4.5 Implementing Timestamped Forged Response Packets

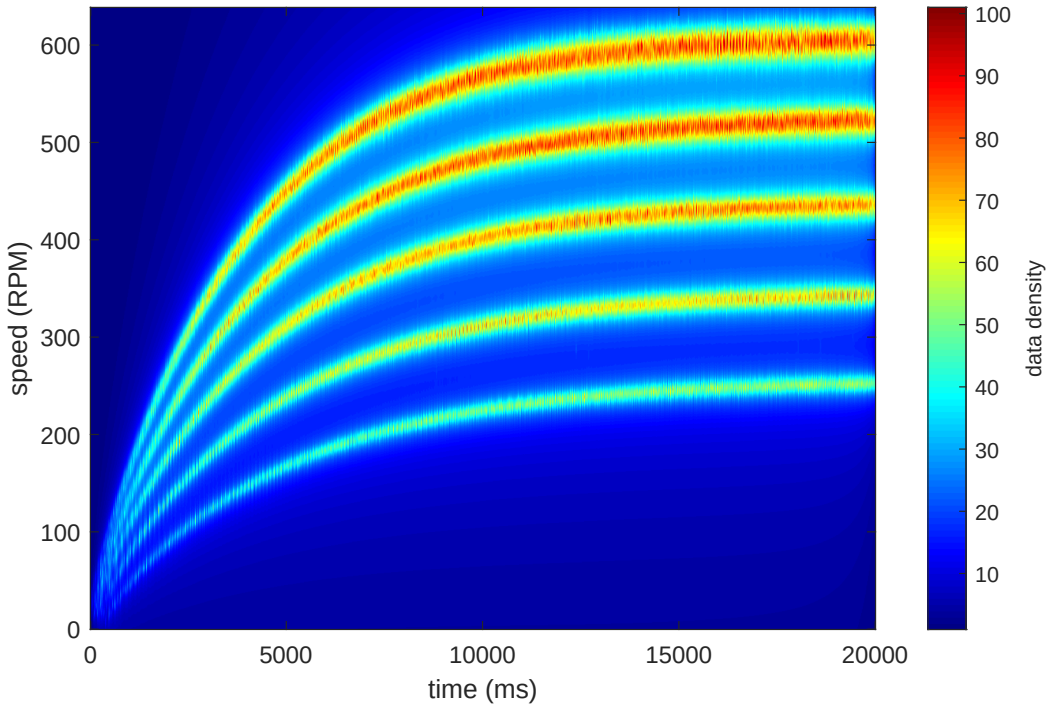
As mentioned in 4.1.1, an attacker with the forged response packets faces the challenge of sending them with accurate timing. In [13], the authors mentioned that an attacker who has a device with limited capability can be detected due to low clock precision and clock drifting. To overcome this issue, the PLC's real-time program execution feature is leveraged. During the experiment, the timestamps and measurement values in the generated responses are stored in the PLC's memory table and loaded by the ladder logic diagram sequentially. The algorithm can be seen in Algorithm 1. Specifically, starting from the first stored timestamp/measurement value pair in the table, the time t since a command has been received is taken using the high-speed clock's value in the PLC. A pointer index p is initialized to 0 following each command. The elapsed time is compared with the timestamp t_p stored in the table t_i. If $t \geq t_i$, the pth measurement value (if the device has an operation curve) or the flag value (if the device has an operation time) is copied to the Modbus memory for the supervisory host to read, and p is incremented by 1. The spoofing

process ends when all responses are sent.

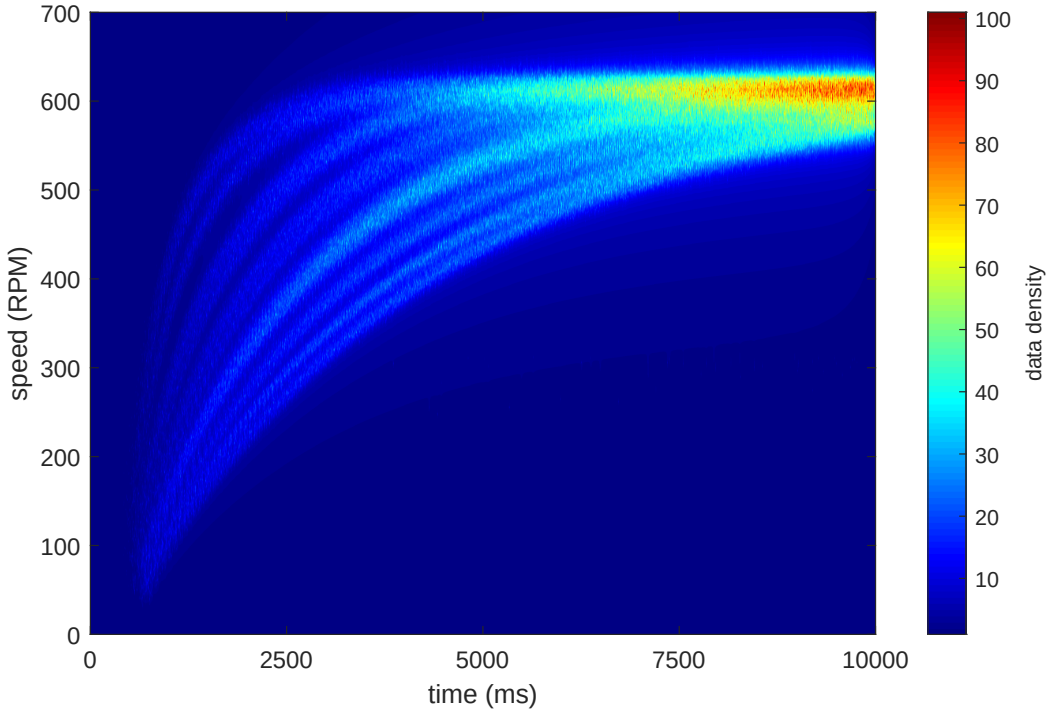
4.4.6 Results

This section shows the results of inferring each device's product model and configurations, as well as the attacker's forged responses. The performance of the inferring technique is measured with three standard metrics in classification tasks, namely accuracy, precision and recall. Let TP, TN, FP and FN be true positive, true negative, false positive and false negative, respectively. **Accuracy** is defined as $\frac{TP+TN}{TP+TN+FP+FN}$. **Precision** is defined as $\frac{TP}{TP+FP}$. **Recall** is defined as $\frac{TP}{TP+FN}$. Another metric is also introduced, namely **estimation error tolerance**, when evaluating the performance of device model and configuration inference. The estimation error tolerance is defined as the number of values between the estimated and the correct values, when all possible values for the given parameter are sorted.

Electric motor. The authentic responses from the electric motor were gathered by varying both its models and run-time configurations, namely the power ratings and the MOI of the load. For better visibility, the aggregated operation curves are shown as heat maps in Figure 4.7. Each heat map was generated by plotting the density of the 100 operation curves for every model or configuration setting. It can be seen that the operation curves are clearly distinguishable and are highly correlated with the parameter values. In the experiment, 5 different power ratings are used (to emulate five different models of electric motors, corresponding to the product model related parameter A from Section 4.3.1) as well as 16 different load settings (corresponding to the configuration related parameter B from Section 4.3.1). Thus there are 80 different combinations and $8,000$ operation curves in total. Each operation curve is taken as a response from this device and its model and configuration are inferred. The result is shown in Figure 4.8, which shows that the aforementioned method is able to correctly infer 98.6% of the power ratings (device model) within a tolerance of 1 value, and 99.9% of the load values (device configuration) within 3 values. The non-zero error tolerance values exist because the operation of physical devices could not be

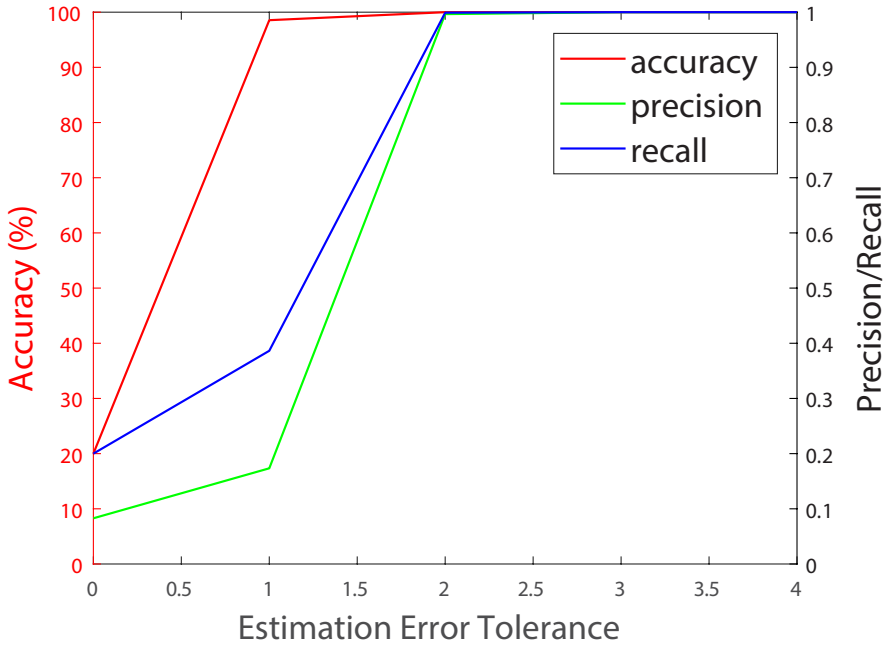


(a) 5 Different Power Ratings.

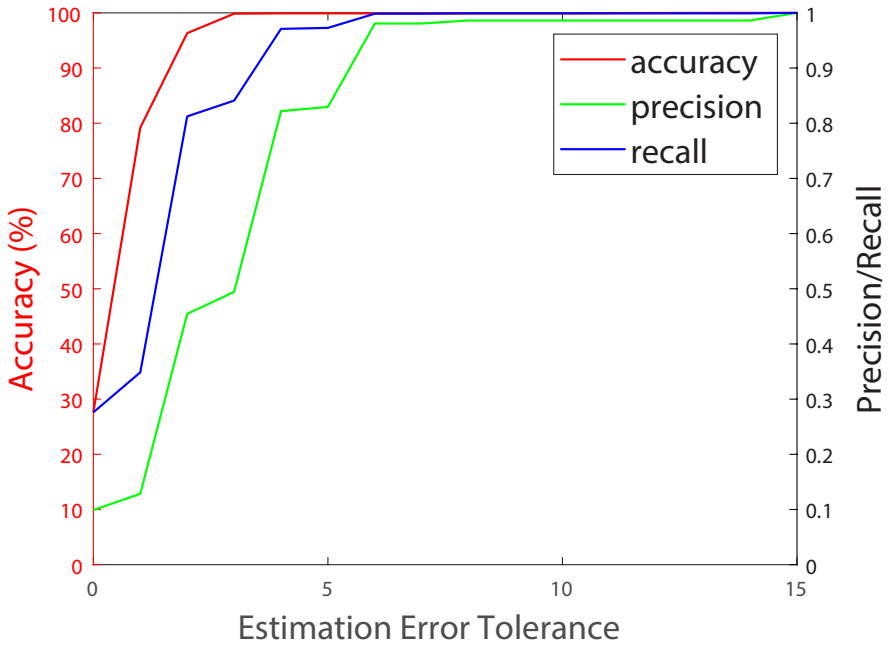


(b) 16 Different Load MOI.

Figure 4.7: Heat map plot of the electric motor's operation curves from different models and under various run-time configurations. Each curve is aggregated over 100 runs.



(a) Accuracy, Precision and Recall of Power Rating Inference.



(b) Accuracy, Precision and Recall of Load MOI Inference.

Figure 4.8: Performance of the run-time configuration inference of the electric motor. The estimation error tolerance is the allowed distance between the estimated value and the attacker's assumed value.

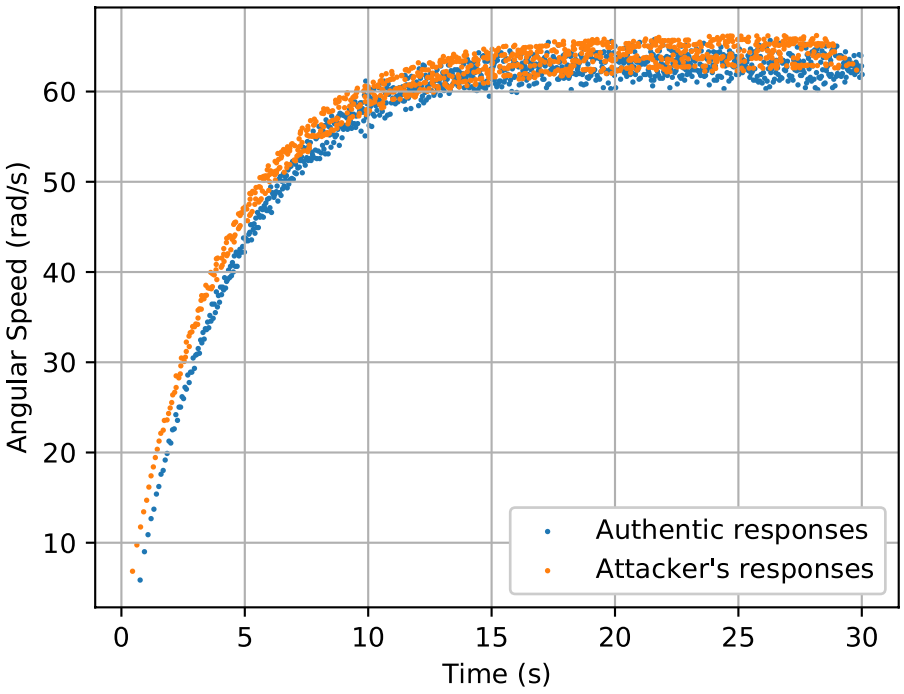


Figure 4.9: Comparison of the authentic and spoofed responses from an electric motor.

perfectly modeled with the equations. For example, there is always energy loss in the form of heat or vibration, which causes the inferred power ratings to be lower than the actual power consumed. Also note that under the same estimation error tolerance, an increase in the number of possible values for a parameter may adversely affect the performance of the inference results. This is expected from the method used, as a larger number of possible values for a parameter typically means these values are more densely distributed over a range. Nevertheless, the differences between the inferred values and the actual ones corresponding to the DMC are found to be systematic errors, which can trivially be calculated either by knowing at least one actual values or a prior experiment conducted by the attacker using testbeds. Moreover, when using these DMC parameter values to generate the forged responses, the difference with the authentic responses is negligible, as shown in Figure 4.9. The DMC inference method is applied to the forged responses and all of the responses correspond to the actual DMC are found. Hence the device physics fingerprinting

method in [13] cannot detect the forgery attack.

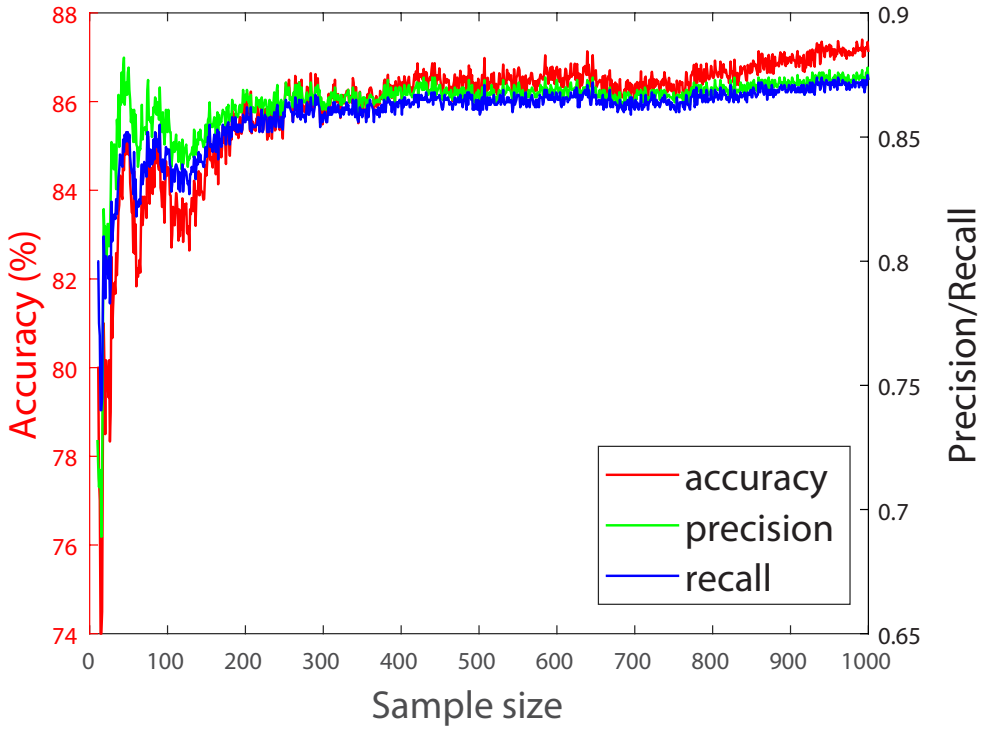
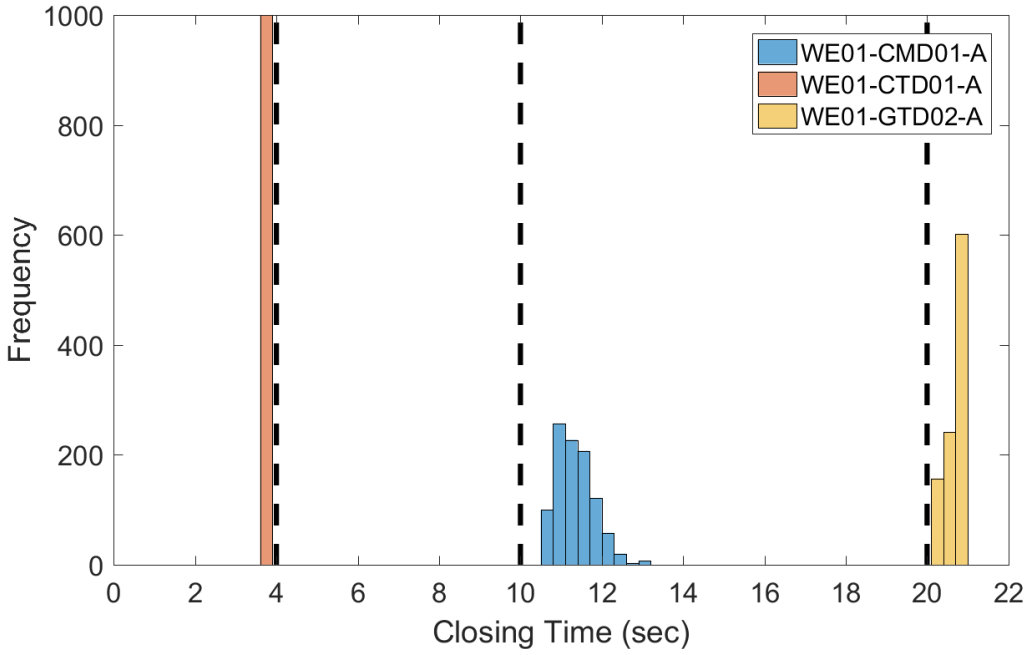


Figure 4.10: Classification performance using relays' operation time.

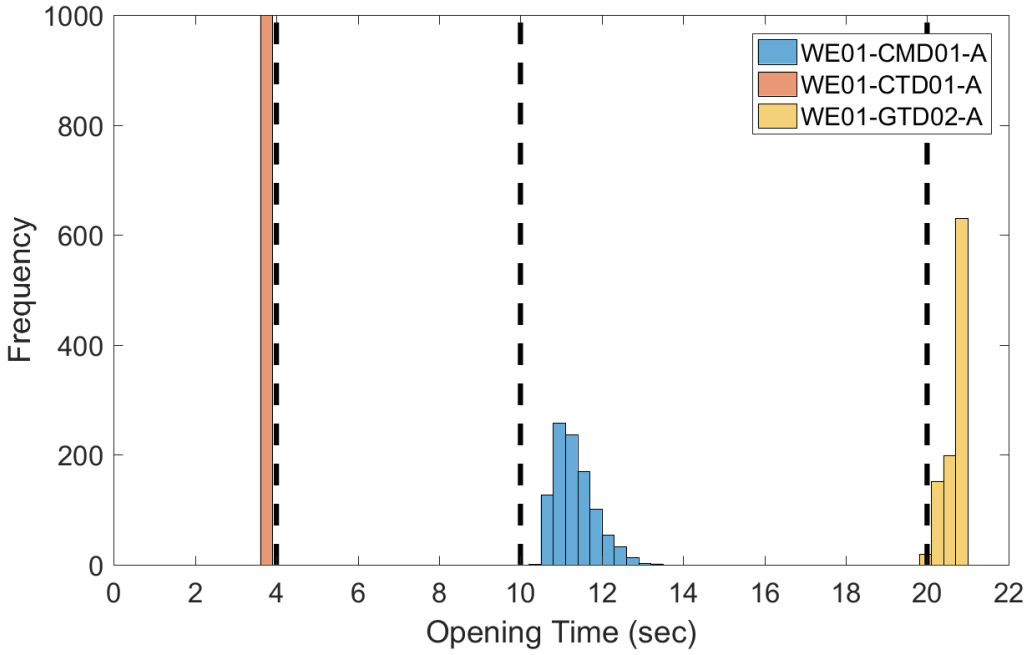
Relay. Similarly, the authentic responses of relays were collected first. Each relay model exhibited a densely distributed closing/opening time around a mean value, with some overlap among several models. Because the operation time of each relay has been explicitly noted in its specifications (which is an important factor in choosing the right relay for any time-critical application for safety), the logical next step was to test the correlation between the relays' specified operation time values with their experimental values. However, only 0.54 and 0.58 correlation coefficients are found for closing and opening time, respectively, using product-moment correlation coefficient (PPMCC). Thus, each model is focused on and ranked according to the distance between its experimental operation time and the specification values of all 10 models. Since the operation time is a 2-dimensional vector, a number of distancing metrics are used, including Euclidean, Chebychev and Manhattan. It is found that the Euclidean metric performs the best among all. In summary, when using

the Euclidean distance metric, six relays rank top 3, which means that when inferring the model of the relays, the attacker is guaranteed to find the correct model within three trials (which can be compared with the estimation error tolerance in the electric motor's results). However, if the reference operation time of each candidate relay model can be experimentally measured (the attacker may acquire every model of a device) instead of being taken from their specifications, the classification performance can be greatly improved as shown in Figure 4.10. Because the cost of data collection increases with the number of times the measurements are taken, the sample size is varied from 10 to $1,000$ to show the relationship between the classification performance and the resourcefulness of the attackers. The classification is carried out by training a Nearest Neighbor classifier in 10-Fold under each sample size. Because the forged responses are crafted based on the inferred model of relay, the accuracy of classifying them into the DMC (a correct classification means a successful attack) is not shown as it followed a similar curve as the accuracy curve in Figure 4.10.

Valve. In contrary to the relays, the valves' actual operation time values were very close to their specification values, and thus can all be correctly inferred without separate measurements. As shown in Figure 4.11, the distribution of the closing and opening time of each valve was adjacent to the corresponding values listed in Table 4.4. The only misalignment is the modulating valve, with model number WE01-CMD01-A, which has a slightly larger operation time than expected. Since it did not have a binary output as the other two valves did, it was not obvious to define an exact position of the valve as fully closed or open. As explained in Section 4.4.4, heuristic values were used to derive the operation time from the its operation curves, which lead to the minor offset. Nevertheless, this does not influence the performance of the proposed inference method, as all three valves' operation time was clearly distinguishable. For all three valves used in the experiment, the accuracy of inference is 100% and both precision and recall remain 1.0. All of the forged responses were also identified as the intended model of valve by the device physics fingerprinting method in [13].

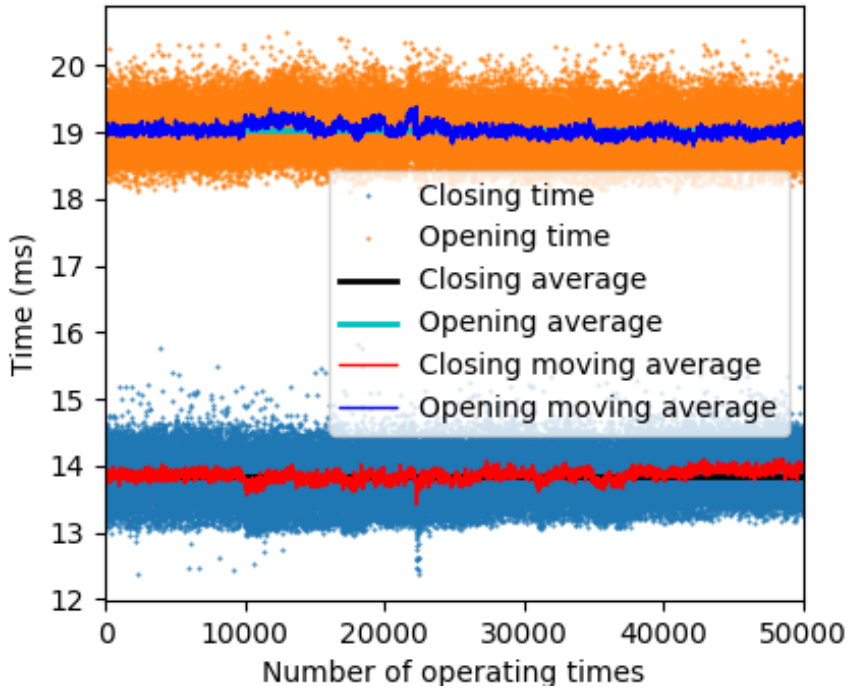


(a) Closing Operation Time.

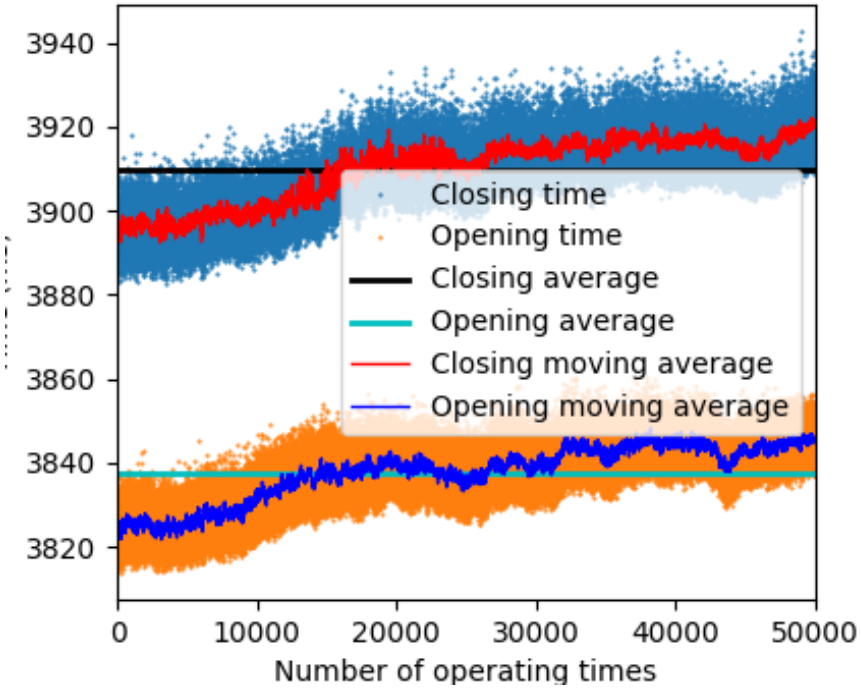


(b) Opening Operation Time.

Figure 4.11: Histogram of the valves' operation time. Dashed lines indicates the specification values.



(a) Relay's operation time over $50,000$ operations.



(b) Valve's operation time over $50,000$ operations.

Figure 4.12: Wearing and aging test of relay and valve.

Wearing and Aging. The physical nature of the devices means that they may suffer from wearing and aging effects in the long term, especially in an industrial environment where the devices are used frequently. In such case, the parameters of the device physics model could deviate from its ideal values. For example, the operation time of a mechanical relay depends partially on the electromagnetic force generated and the force in its spring. As the number of operations increases, the spring may suffer from fatigue, which causes the deviation of the relay's operation time. Similarly, a operation time of a valve may deviate as it keeps operating throughout its lifetime. To have a brief understanding of how this affects the aforementioned technique, extended tests are performed with both testbeds used in the experiments, and different changes in their operation curves or operation time are observed. Briefly, only the valve shows a deviation in its operation time. For example, Figure 4.12a shows the operation time recorded over $50,000$ times of open/close operation of a relay (Omron MKS3PI DC24), which is rated to be operated at $18,000 ops./hour$ and with a mechanical endurance of 5×10^6 operations. As it is challenging to perform measurements over its entire lifetime, its maximum operating frequency is used to accelerate the wearing and aging effects. Moving average is used as the metric, with the window size set to 50. However, no evidence is found that shows the operation time changing during the test. On the other hand, the valve used in the wearing and aging test (Dwyer WE01-CTD01-A) does exhibit a gradual increase in its operation times as shown in Figure 4.12b. Namely, the moving average of its closing time increases from $3883ms$ to $3943ms$, and that of its opening time increase from $3814ms$ to $3860ms$. The increase over the $50,000$ operations is 1.5% and 1.2%, respectively. At this rate, it would take approximately 3×10^6 operations before its operation time becomes indistinguishable from the other models used in the experiments. The experiments show that within reasonable amount of device operation frequency, the device physics maintains relatively constant. Hence, as a first look, the performance of the proposed DMC inference technique is insignificantly affected by the effect of wearing and aging of devices. More extensive experiments are planned to find out

the long-term impact on the performance of the aforementioned technique in future works.

4.5 Discussion

4.5.1 Applicability to Other Field Protocols

In the experiment,Modbus is employed as the communication protocol used between the PLC and the host. In reality, other protocols may be used depending on the specific application, such as DNP3, Common Industrial Protocol (CIP), BACNet, or ProfiNET. Most of the protocols are designed without security features and transmit application layer data (e.g., the timestamped values sent by the actuators) in clear text. However, some protocols have now been modified or designed with security in mind (albeit rarely used in practice), and use encryption to protect data privacy and data integrity, such as secure DNP3. In such cases, the system still may not be exempt from an attacker who has access to the PLC's firmware or program. As have been discussed in Section 4.4.6, the application layer data is decrypted in the PLC if 1) the protocol natively supports timestamping or 2) timestamping function is added to the PLC program. In the last case where timestamps can only be obtained at the tapping point, an extra step need to be employed to correlate the timestamp information at the tapping point with the value-only data decrypted at the PLC, in order to produce the accurate time series values generate by the actuators. The attack's performance can be affected if the attacker's source of time is different from the one used at the tapping point.

4.5.2 Applicability to Other Device Types

Although the methodology is only demonstrated using three types of CPS devices, it can be generalized to many other devices that share similar properties. Because the majority of actuators in CPSs are made of mechanical or electro-mechanical components that obey the laws of physics, and rarely include programmable components, their operations can be very stable in every actuation. Additionally, given a specific command of actuation, these

actuators' response signals in the temporal domain are tightly correlated with their models and configurations. The motion of most actuators (such as electric motor, relay, valve, solenoid, and stepper motor, etc.) can be described with first- or second-order differential equations, which enables the mathematical model inversion that leads to the inference of the DMC. The method can be applied as long as these devices can be modeled with equations, which are fitted with the operation curves or operation time of the devices. Depending on the type of device, the aforementioned method may be able to infer either its model, configuration or both. The cost associated with the method comes mostly from modeling the devices. However, some device types may take more effort to model, such as the turbine in a thermal power plant, due to the complicated computations involved with thermal and fluid dynamics.

4.5.3 Defending Against CPS Mimicry Attacks

Most of the existing defenses against mimicry attacks [78] focus on the IT domain. For example, some researchers proposed to use system call trace to enforce the correct program execution [79, 80]. Another paper used control-flow integrity (CFI) checks to prevent attacks from arbitrarily controlling program behavior [81]. However, such solutions can hardly be applied in the CPS environment, due to the difficulty of instrumenting code execution in the PLC. Moreover, the attack can be implemented on the PLC program level, which does not change the control flow of the firmware. A related work [55] attempted to design a PLC-compatible CFI mechanism. However, the authors mentioned that their framework could not be implemented on a real PLC and used an open source software instead. Therefore, a new defense technique is proposed.

It is worth noting that although Modbus - a communication protocol with no encryption - has been used in this experiment, however, even when using a protocol which supports encryption, the proposed mimicry attack will still be successful. This is because the encryption only exists **between** the PLC and the supervisory host, and the data has to be plain

text in PLC's memory. An attacker who compromises the PLC's firmware or program can access the PLC's memory regardless of the encryption used in the communication protocol.

An intuitive countermeasure is to inject noise into the operation curve or operation time when sending the response packets from the PLC, or at the sensor which is measuring the physical signal. While this may prevent an attacker from obtaining an accurate DMC and hence generate the correct response, there are two drawbacks of this approach. First, the control algorithm of certain types of devices may leverage these sensory signal inputs, then adjust the output signal to the actuator to achieve optimal operation of the device. Noise in the sensory input may interfere with such control algorithm and degrade the safe and smooth operation of the CPS. Second, the noise-injected responses may interfere with the device fingerprint based defense system, and inadvertently increase the false positive rate. Therefore, adding noise is not a feasible option.

Based on the observation in this study, it can be found that in order for the responses to be generated and spoofed in time when a command is received, the attacker needs to pre-compute the timestamps and measurements of a device and store the data in PLC's memory for fast access. Storing such data over another network device may not meet the real-time performance requirement of sending the responses. Also, unlike a computer program which uses the stack and heap which are dynamic in memory, a PLC program accesses its memory via assigned blocks arranged in table files, which almost always uses a known amount of memory in fixed locations. Therefore, a challenge-response defense framework is proposed as shown in Figure 4.13. The steps are described as follows:

1. In the initial setup, the (potentially large) unused memory in a PLC is filled with pseudo-random data known to the supervisory host. Essentially, the supervisory host keeps a copy of all unused memory of every PLC in its network. Typically, the user memory in PLC is less than 100MB (e.g., [82, 83], which leaves even less space after the control program has been loaded.

2. During normal operation, the supervisory host periodically sends a request to the

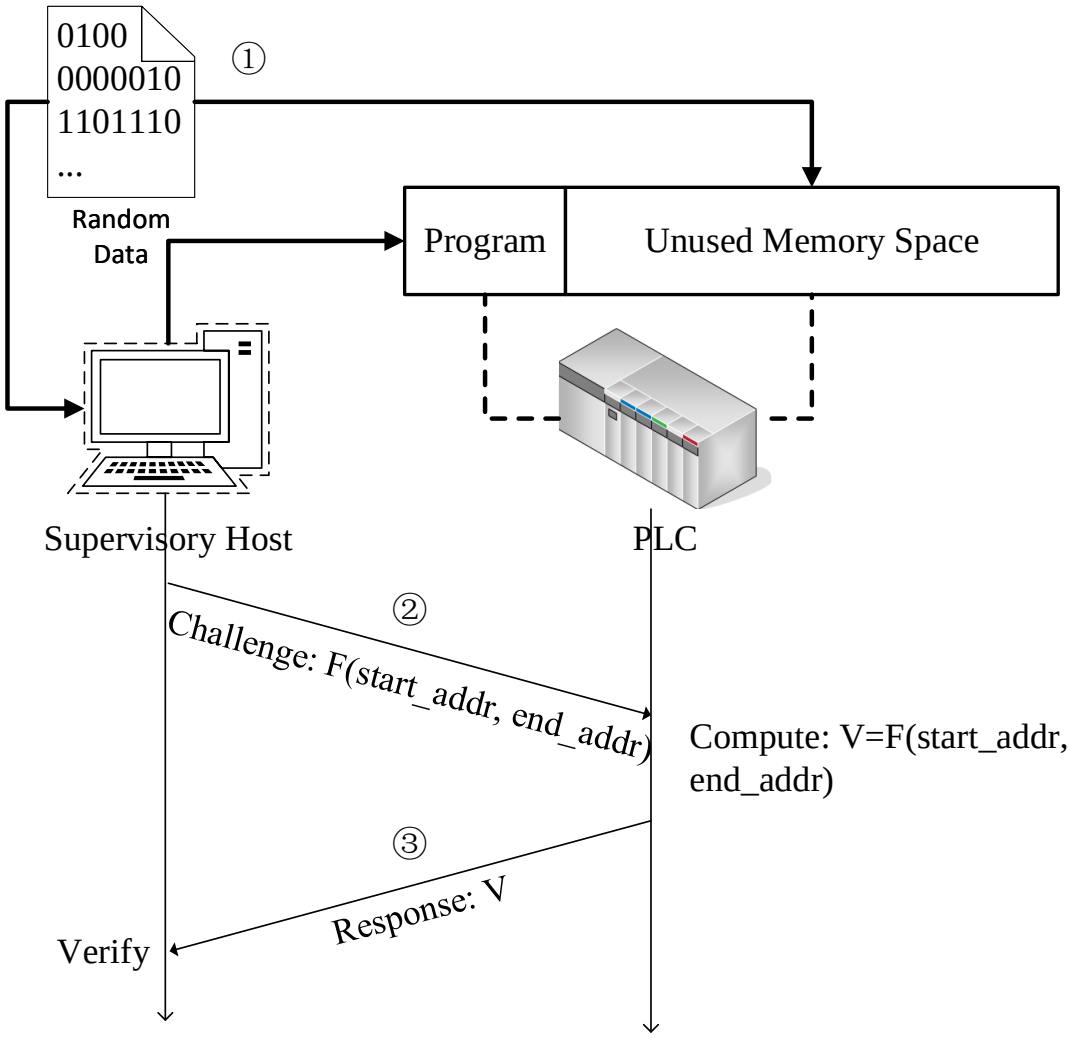


Figure 4.13: Challenge-response framework to defend against device physics mimicry attacks.

PLC to apply a function F over the data in a randomly chosen region of its unused memory $[start\_addr, end\_addr]$. Such function can be XOR, hash, etc.

3. The PLC computes and sends the result $V = F(start\_addr, end\_addr)$ back to the supervisory host. If there is a mismatch or a certain time threshold is exceeded, an alarm can be raised.

The attacker may choose to minimize memory usage and dynamically generate the responses from a few parameters instead of storing a large amount of pre-computed data in the PLC's memory. However, this will add a substantial amount of delay to the response packets.

4.5.4 Limitations

In this section, several limitations of this work and their influences are discussed.

Modeling Accuracy. Because the accuracy of the proposed DMC inference technique is highly dependent on the estimation of the parameters in the physics model of the device, such estimation can be less accurate when there is a mismatch between the speculated model and the real device. Thus, a comprehensive analysis may be required to understand the anatomy of the device. The model (and parameters) may also deviate from its original values due to wearing and aging of the physical components in the long term, although the extent to which this may affect the performance of the proposed technique varies among different types of devices. In such case, the operation time/curve may be re-collected and machine learning models need to be re-trained.

Signal Availability. The proposed method assumes that the device must be able to send either corresponding responses upon certain events (e.g., completion of a command), or contain observable state variables to recover the state of the device. Such assumption may often be valid because a closed control is often used in an industrial environment to ensure stability.

System level information. In this chapter, inferring the information of the *devices* in CPSs are being focused on. An advanced attacker may step up and attempt to infer the system level information of CPSs. This problem and the corresponding defense technique study is intended to be left as a future work.

4.6 Conclusion

In this chapter, the problem of launching a device response spoofing attack in Cyber-Physical Systems is studied. A novel technique is proposed which bridges the gap between the physics of the CPS devices and the responses from the devices. Several testbeds are built and used to benchmark the methodology with real devices used in CPSs, and high accuracy is achieved in inferring the device model and configuration information, as well as forging responses that are indistinguishable from the authentic devices' responses in most cases. Then the impact on the performance of the method stemmed from various factors is discussed, including the amount of available data and chronological wearing of the CPS devices. Finally, it is also proposed to use a challenge-response method to defend against such attacks.

CHAPTER 5

IDENTIFYING THE PROCESS FROM ITS CONTROL PROGRAMS

When facing potentially malicious programs in programmable logic controllers (PLCs), existing methods for securing PLC-based systems mostly fall into two categories: run-time monitoring of the system dynamics, or statically analyzing the source code of the PLC program. However, none of the existing methods is effective against a malicious PLC program if the attacker hides the malicious behavior (e.g., using logic bomb). The major challenge in analyzing a PLC program comes from combining the universal applicability without relying on the source code, and circumventing the intentional hiding of the attack. In this chapter, the aim is to address these shortcomings by providing a framework for fuzzing PLC program binaries to obtain the complete behavioral model of the program. A framework called LogicFuzzer is proposed, which consists of two stages: the first stage determines whether a suspicious control program is written for a given physical process (e.g., nuclear power plant, automatic production line, etc.). The second stage searches for malicious states that the program can run into, and finds the conditions to trigger such states (e.g., a counter with very large preset value). Through extensive evaluation on a large corpus of real PLC programs, it is shown that 99.1% accuracy can be achieved for classifying the program into the corresponding process, and 98.9% accuracy can be achieved for detecting whether the program is malicious. It is also shown that LogicFuzzer is agnostic to large timer or counter values set by the attacker to hide the malicious code segment, and can be used even as the complexity of the program grows.

5.1 Introduction

Threats against cyber-physical systems (CPSs) are increasingly prominent, especially in critical infrastructures. Targeted attacks which can damage the physical processes have

become more frequent due to the high value of these systems. Meanwhile, their control systems, known as industrial control system (ICSs) are still vulnerable. Compared to the versatile defensive techniques in the traditional information technology (IT) domain, the ICS, which falls in the operational technology (OT) domain, lack proper control software analysis tools [11].

Studies in the past few years have proposed various methods for securing systems based on programmable logic controllers (PLCs), which are predominantly used as the controller in the ICS. These studies can be divided into two categories. The first category focuses on the dynamics of the system physics to ensure that they do not deviate from the control objective[58, 59]. The second category analyzes the control software, i.e., PLC program, to ensure the code conforms to certain rules[50, 51]. However, the existing methods have their deficiencies in protecting the systems. The monitoring techniques are passive defenses. They can only be deployed within and tailored to certain systems, waiting for the anomalies to appear. In an orchestrated attack, the malicious code which causes the anomalies may be buried deep in the form of a logic bomb, which will only be set off when certain conditions are met. In theory, the conditions can be designed as difficult to meet as the attacker desires. Even a simple timer that takes long enough (e.g., six months) to trigger the malicious piece of code can convince the system administrator that the system is normal during a lengthy (e.g., three months) test stage.

Although the second type of techniques which examine the source code of the PLC program may detect the injected malicious code, in most situations, the source code is almost always unavailable for analysis. Such techniques work the best during an insider attack, where the source code can be cross referenced with the program running on the PLC. A sophisticated attacker can pre-compile the source code into binary files before sending them as payload to the target (e.g., Stuxnet[49]), rendering the source code-based analysis methods ineffective. As a result, these binary files found during propagation can be hard to analyze, especially because it is impossible to define *what should be considered malicious*

without knowing which physical process the program is intended for.

To address the aforementioned issues, a proactive framework called LogicFuzzer is proposed to protect the PLC-based systems. The goal of this framework is to scan a binary PLC program found "in the wild" and perform the analysis **without using the source code**. In fact, the analysis of computer program binaries running on popular OSs (e.g., Windows or Linux) have been studied for a long time and already commercialized into well-known software or services such as VirusTotal. However, there is no method or service for scanning PLC program binaries to check for malicious behavior. The main challenge comes from the lack of a universally applicable method to extract the complete behavioral model of the PLC program binary. Hence, LogicFuzzer is designed to meet this goal. Similar to VirusTotal, users can upload suspicious PLC programs to LogicFuzzer and provide minimal contextual information about the physical process before commencing automated analysis of the program's behavior. Note that while VirusTotal looks for malicious signatures of illegitimate code which exploits the IT-domain vulnerabilities (e.g., buffer overflow), LogicFuzzer checks the behavior of the program which can cause damage in the OT domain, such as driving the motor to a dangerously high speed or overflowing the tank. Compared to the existing methods, LogicFuzzer has the advantage that it is **agnostic to the length of wait time** to trigger the malicious logic in the program. Moreover, it can be used even as the complexity of the program grows, as it does not have the path explosion problem which is typically encountered in symbolic execution methods.

This work is based on the scenario that is likely to be encountered by a system administrator (SA) in an ICS environment. In the likely event that a PLC program is captured, or a suspicious program is found in the PLC, the system administrator would be interested to know the answers to the following two questions about the program:

1. Is this control program written for the SA's ICS?

2. Is this program potentially dangerous? In other words, can this program cause the corresponding physical process to run into unsafe states?

These questions can be answered with LogicFuzzer, combining binary analysis, fuzzing, as well as automaton theory. As shown in Figure 5.1, LogicFuzzer first parses the binary PLC program and extracts the elements required for the next stage of analysis. The parser translates the binary into the high-level data structure, and functions as an emulator for the code execution environment. Next, the fuzzer generates a complete behavioral model of the PLC program in the form of an automaton, leveraging the parser as an interface. Finally, a classifier predicts which process the automaton corresponds to, and a detector checks the automaton for unsafe states as well as the path leading to these states. To validate the proposed framework, 650 PLC programs are collected, which are the publicly known largest dataset, written for four ICS scenarios designed according to the real settings in an industrial environment. The result shows that LogicFuzzer can accurately classify the binary programs into the corresponding processes, and detect whether it is malicious with the triggering conditions.

The rest of this chapter is structured as follows. First, Section 5.2 explains the scenario where LogicFuzzer is used with several assumptions. Then, Section 5.3 performs an anatomy of the PLC program binary to help understand its structure. Based on the knowledge in Section 5.3, Section 5.4 walks through the methods used in building the automaton from the binary program. Subsequently, in Section 5.5, the methods used to collect PLC programs are explained in details which can evaluate the framework in Section 5.6. Finally, a discussion of this work is provided in Section 5.7 and Section 5.8 concludes this chapter.

5.2 Application Scenario

This work is based on a scenario where the system administrator of an ICS is interested in analyzing a suspicious PLC program, which is a compiled binary either captured in network traffic or found inside a PLC. The system administrator has the knowledge about the physical process of the ICS. He/she knows a set of rules which define the expected behavior of the process, and what states the process should be in under certain sensor inputs.

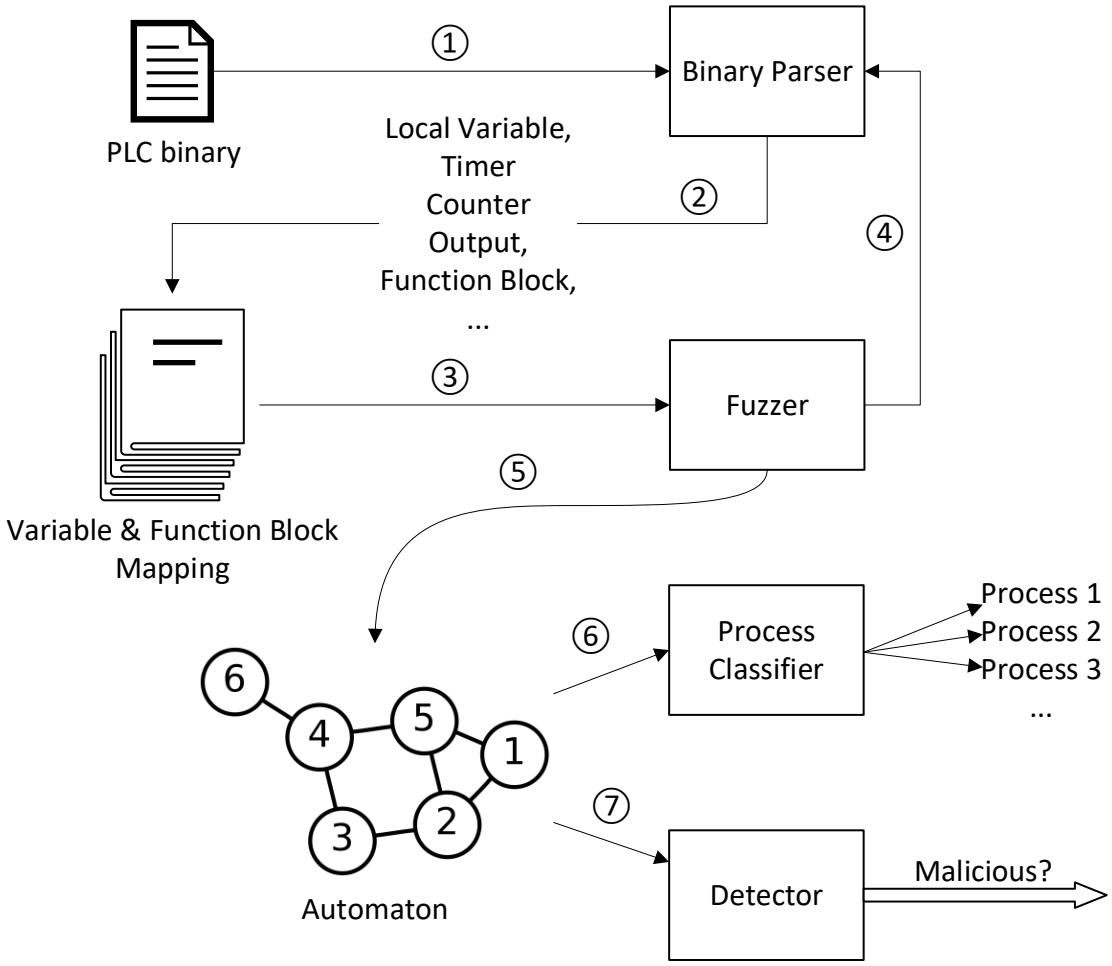


Figure 5.1: Overall system diagram of LogicFuzzer, which fuzzes the PLC program binary and generates the automaton.

The set of rules can either be given directly by the system administrator, or synthesized from a reference program which is originally used to control the ICS. Such rules are used for two purposes: 1) to classify whether the PLC program being analyzed is written for the underlying process; 2) to determine which states and transitions are valid. For example, the program may intend to target an assembly line, while the process owned by the system administrator is a water treatment plant.

To establish the ground for this work in analyzing the PLC program binaries, it is assumed that the model of the PLC which the program is written for is known or given by the system administrator. To make LogicFuzzer more applicable to realistic situations, only the compiled binary of the PLC program will be available, i.e., **no source code is used** throughout the analysis.

5.3 Building the Structure of PLC Program Binary

Before analyzing the PLC program binary, it is first need to understand its structure and parse it. As shown in Figure 5.1, the PLC binary is parsed with the *binary parser* module before the *fuzzer* can interact with it. The ARM926EJ-S based Schneider Modicon M241 PLC is used as an example in walking through the anatomy of the PLC program binary. As the reverse engineering of different other PLCs has been studied, such as Schneider Modicon M221[60], WAGO PLCs[56, 55], and Allen Bradley PLCs[70, 84, 61], It would be the most beneficial to the research community if the reverse engineering work is based on a less studied PLC model. Note that LogicFuzzer is agnostic to the specific PLC model which the program is written for. In this section, it is shown how to recover the essential information of the PLC program structure from bottom to top, i.e., from the binary to the subroutine, and eventually to the variables and function blocks.

5.3.1 Understanding the Binary Structure

As each PLC model is manufactured with various CPU architectures and runs different firmware, the PLC program written in the same high-level IEC 61131-3 representation can be compiled into the binary very differently as well. Unlike the x86-based platform, there is no standard format such as the Executable and Linkable Format (ELF) and hence no tool is readily available to parse the PLC program binary. Therefore, preliminary work is performed in order to understand the general structure of the binary used in the M241.

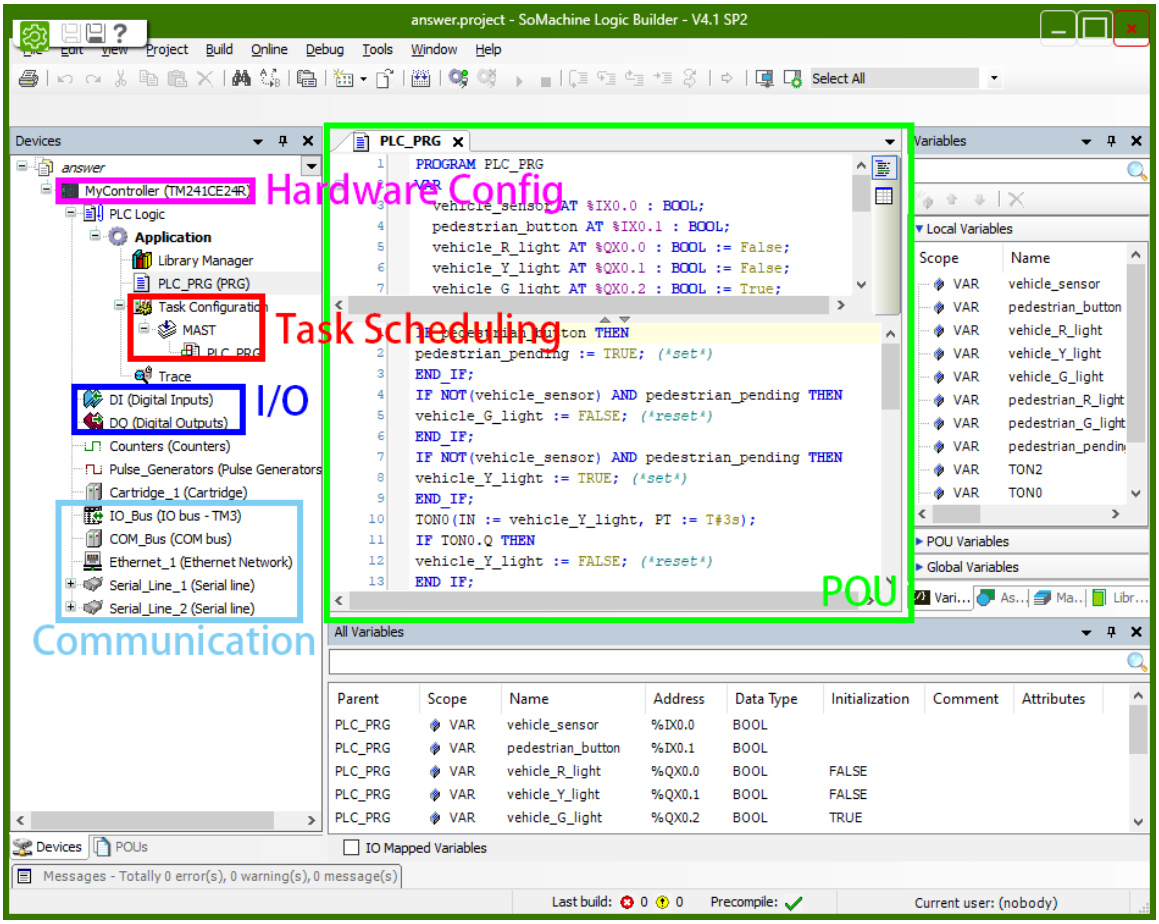


Figure 5.2: Typical structure of a PLC project.

As shown in Figure 5.2 A PLC program is usually generated from a complex-structured project consisting of many types of elements, such as PLC hardware configuration, communication interface, task schedule, Program Organization Unit (POU) etc. Because the

purpose of the work is to study the behavioral model of the program, this work focuses on recovering the POU from the binary. To do so, a batch of "empty" projects are created, which only contained the minimal project files and replaced the main POU with simple Structured Text (ST) instructions. Then these projects are compiled into binaries and *diff* the binaries to find how the differences in their source code translate into the differences in their binaries. To mitigate the noise introduced due to the metadata, such as time information that is added to the binary, the same project is compiled 10 times and *XOR*ed all of them to find the locations of bytes that change over each compilation, i.e., irrelevant to the actual program logic. These irrelevant bytes were then used as a mask that is *OR*ed (using *AND* is equivalent) with each binary, which allowed us to better find the differences between binaries due to the program logic.

A Python library called Capstone[85] is used to disassemble the binaries. In the case of ST code in Figure 5.3a and Figure 5.4a, the major difference was found to be an insertion of a single *exclusive or* instruction eor in the ARM instruction set architecture (ISA), as shown in Figure 5.5. The disassemblies of the complete program in each ST code are shown in Figure 5.3b and Figure 5.4b. Similarly, AND and OR logical operations in the ST code can find their counterparts in the disassembly as well, as shown in Figure 5.8. The ST code and disassembly of the program that each contain logical AND and OR operations are shown in Figure 5.6 and 5.7.

One challenge in disassembling the binary is that the code sections and data sections are mixed together. While reversing engineering the binary, it is found that the program section is actually wrapped inside a set of subroutine entry and exit instructions, just as every other subroutine in the disassembly does. Each subroutine begins with **push {sl, lr}**, **mov sl, sp** and **push {r4, r5, ...}** which pushes a list of registers that used in this subroutine to the stack. At the end of the subroutine is **pop {r4, r5, ...}** which pops the same list of registers that were pushed into the stack at the beginning of the subroutine, followed by **pop {sl, pc}**, which restores the value of register **sl** but loads the value of **lr** into **pc**. **lr** is a

```
PROGRAM PLC_PRG
VAR
    I_1 : BOOL;
    O_1 : BOOL;
END_VAR

O_1 := I_1
```

(a) ST code

```
00002000  e5db4000  ldrb r4, [fp] ; load I_1
00002004  e5cb4001  strb r4, [fp, #1] ; store O_1
```

(b) Disassembly

Figure 5.3: Program with value assignment

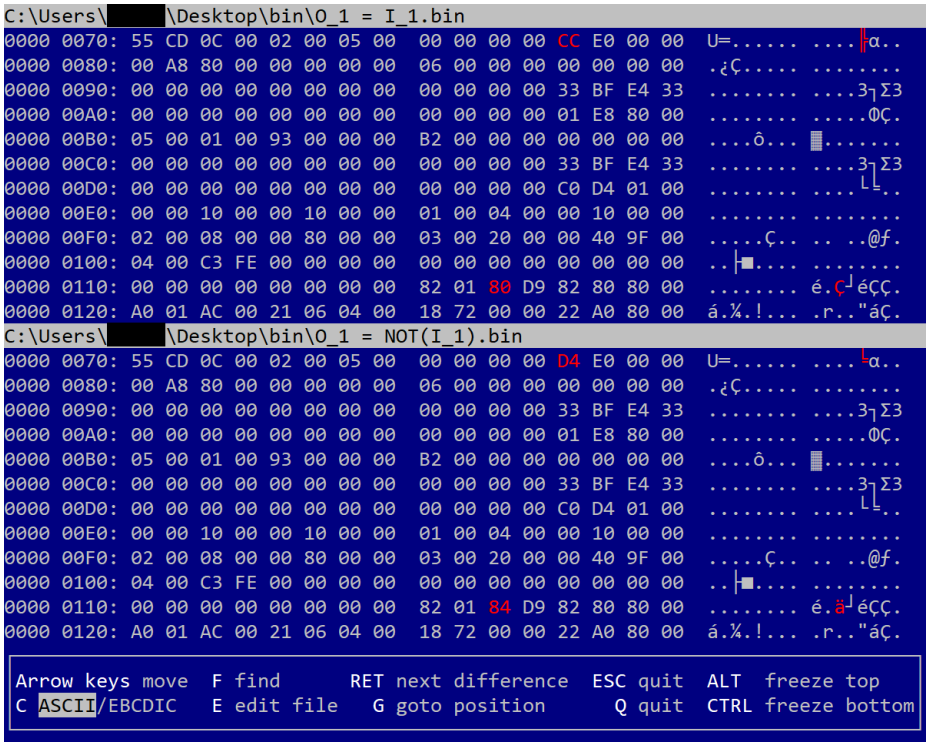
```
PROGRAM PLC_PRG
VAR
  I_1 : BOOL;
  O_1 : BOOL;
END_VAR

O_1 := NOT(I_1)
```

(a) ST code

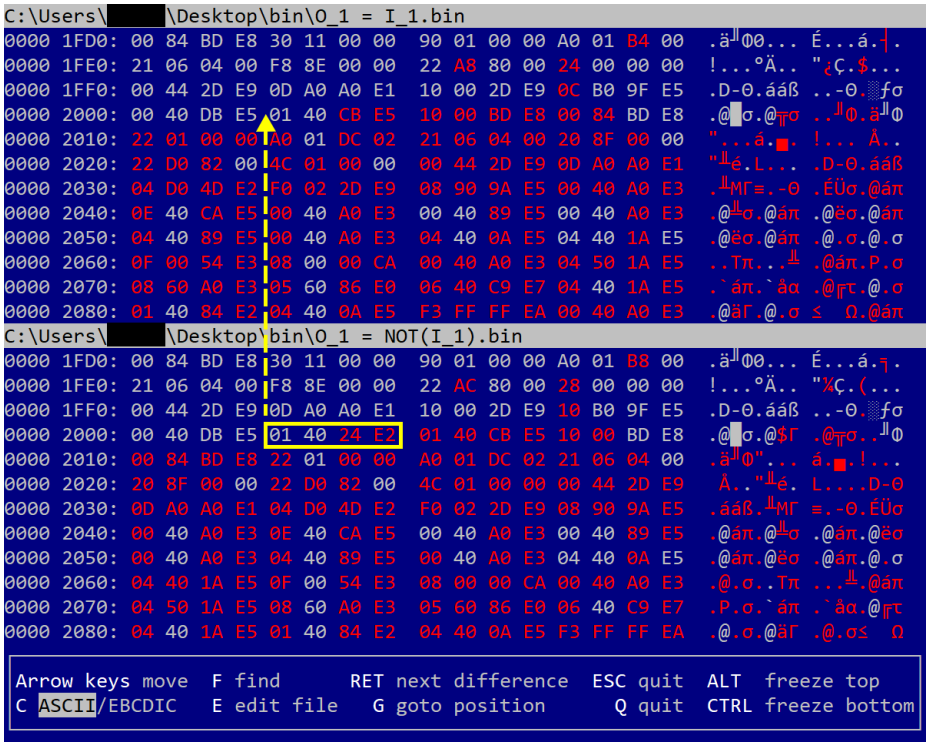
```
00002000  e5db4000  ldrb r4, [fp] ; load I_1
00002004  e2244001  eor r4, r4, #1 ; NOT(I_1)
00002008  e5cb4001  strb r4, [fp, #1] ; store O_1
```

(b) Disassembly

Figure 5.4: Program with value assignment and logical NOT



(a) The headers only have minor difference caused by different length and content of the logical expressions.



(b) The addition of NOT corresponds to an insertion of a single instruction.

Figure 5.5: Binary diff between value assignment and logical NOT operation.

```
PROGRAM PLC_PRG
VAR
   I_1  : BOOL;
   I_2  : BOOL;
   O_1  : BOOL;
END_VAR

O_1 := I_1 AND I_2
```

(a) ST code

```
00002000   e5db4000   ldrb r4, [fp] ; load I_1
00002004   e5db5001   ldrb r5, [fp, #1] ; load I_2
00002008   e0044005   and r4, r4, r5 ; I_1 AND I_2
0000200C   e5cb43dc   strb r4, [fp, #0x3dc] ; store O_1
```

(b) Disassembly

Figure 5.6: Program with value assignment and logical AND

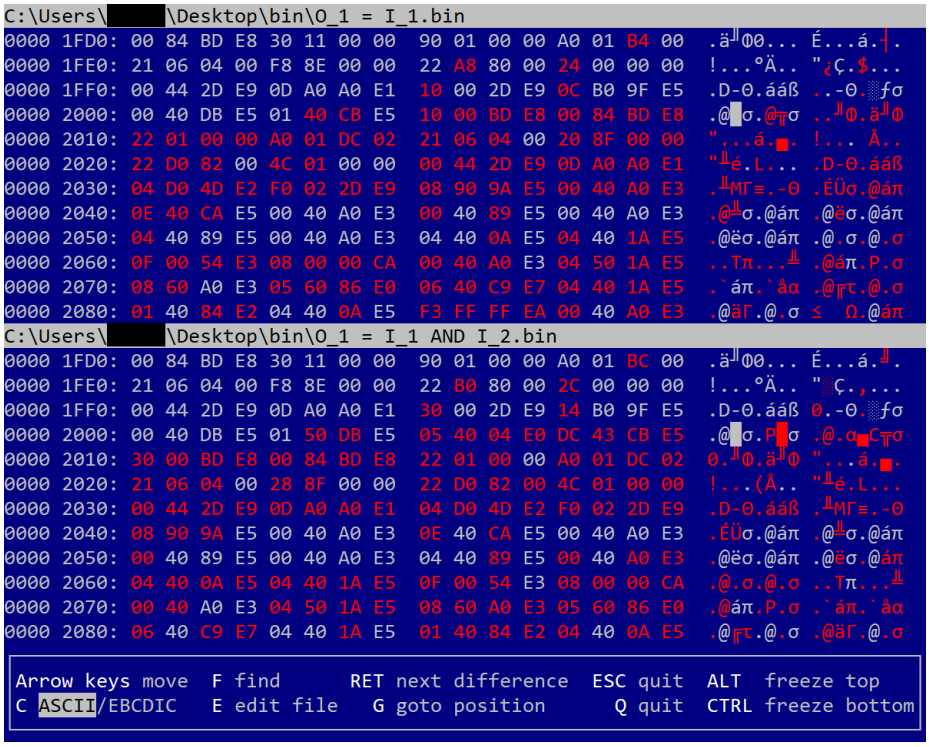
```
PROGRAM PLC_PRG
VAR
   I_1  : BOOL;
   I_2  : BOOL;
   O_1  : BOOL;
END_VAR

O_1 := I_1 OR I_2
```

(a) ST code

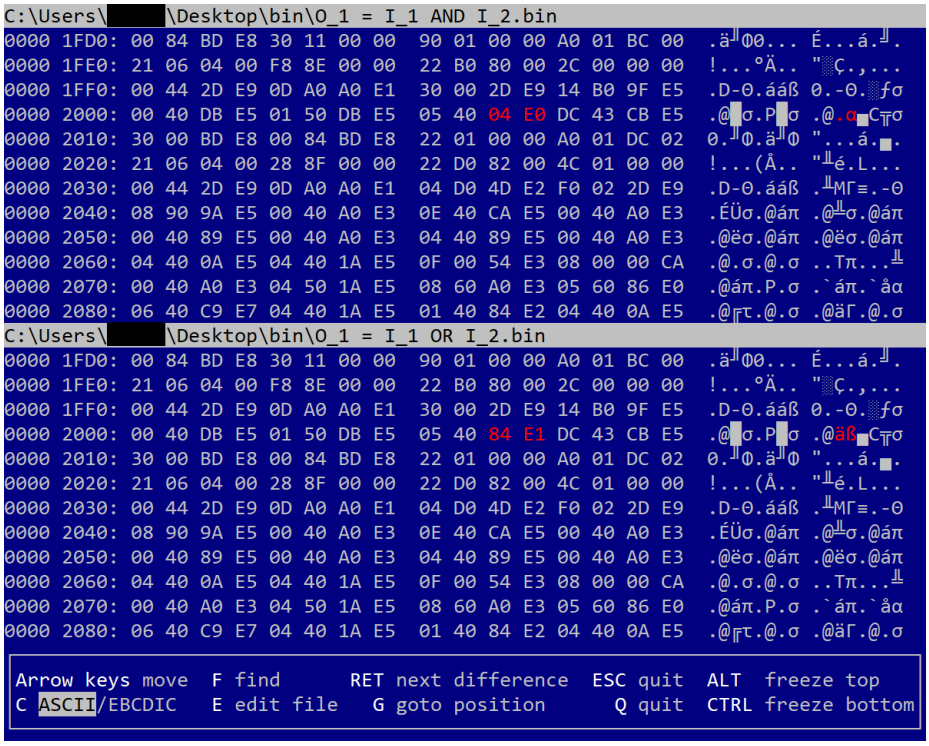
```
00002000   e5db4000   ldrb r4, [fp] ; load I_1
00002004   e5db5001   ldrb r5, [fp, #1] ; load I_2
00002008   e1844005   orr r4, r4, r5 ; I_1 OR I_2
0000200C   e5cb43dc   strb r4, [fp, #0x3dc] ; store O_1
```

(b) Disassembly

Figure 5.7: Program with value assignment and logical OR



(a) The AND logical operation in Figure 5.6 differs by reading of a second input variable and performing logical AND, compared to the program in Figure 5.3.



(b) The OR logical operation in Figure 5.7 differs only by performing OR instead of AND logical operation, compared to the program in Figure 5.6.

Figure 5.8: Binary diff showing AND and OR logical operations.

special-purpose register in ARM called the link register, which holds the address to return to when a function call completes. Hence the subroutine returns to its caller. By tracing all the memory accessing instructions such as **ldr** and **str**, it is further discovered that most directly accessed addresses from the subroutine are immediately after the code section of the subroutine. After excluding all the code and data sections found with this technique, the binary is left with a 20 byte header before the code section of each subroutine, as well as some meta data at the beginning and end of the binary.

The general structure of the disassembly can be seen in Figure 5.9, where the program subroutine is one of the subroutines contained in the binary, followed immediately by the function block subroutines that are called in the program, which will be further explained in Section 5.3.3. The initialization subroutine initializes all the constants and default value of the variables used in the program, including the output. The jumping table subroutine builds a jumping table that maps the referenced address used by a caller to the actual address where the callee subroutine is loaded into the memory.

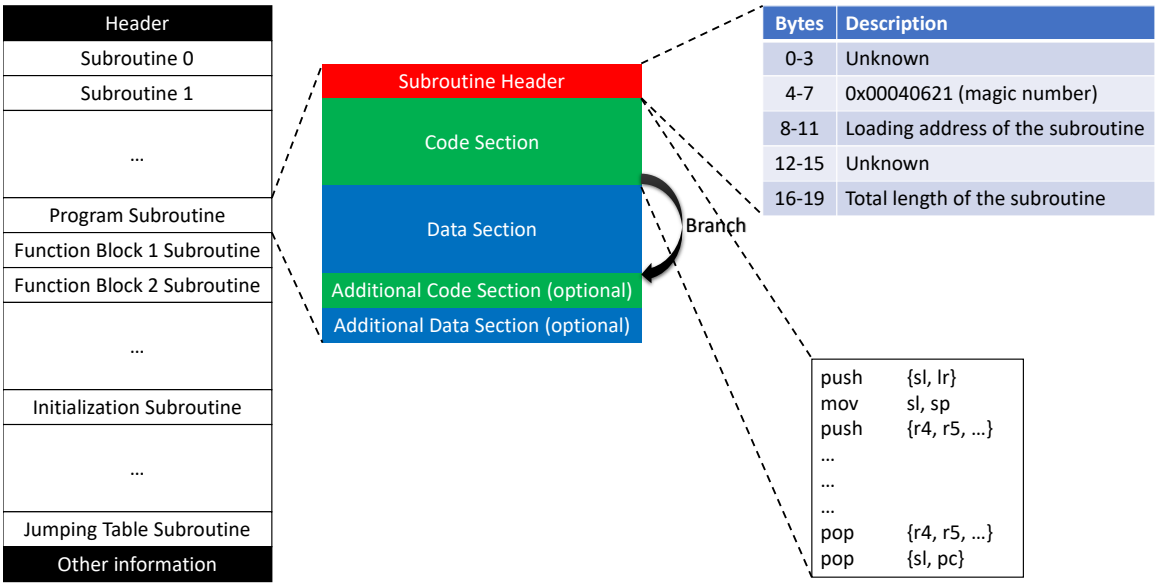


Figure 5.9: Structure of the disassembly of the PLC program

5.3.2 Input, Output and Internal Variables

To record the program's state, it is needed to identify the input, output and the internal variables used in the programs and track their values. The input and output (I/O) variables are directly mapped to the physical inputs and outputs on the PLC. During each scan cycle, the input variables are first updated by scanning the physical inputs. At the end of the scan cycle when all the program logic has been evaluated and the output variables have been updated, the circuit in the PLC drives the physical outputs to the states corresponding to the output variables. The internal variables are used to store temporary computation results and only reside in the memory. Although they are not mapped to any physical pin on the PLC, the values of the internal variables are usually persistent through the scan cycle.

Manual tracing of the execution of a few programs was performed, as well as comparing the disassemblies with the source code to understand how the input, output and internal variables are represented in the binary. For example, in the program snippet shown in Figure A.3a in the Appendix, two input variables ($LevelHigh$ and $LevelEmpty$), one output variable ($InValve2$) and two internal variables ($NOT3\_OUT$ and $AND34\_OUT$) are defined. The disassembly has been matched with the corresponding ST code in Figure A.3b. Note that the value of **pc** in ARM is 8 bytes larger than the current instruction's address. By tracing the memory access through **fp**, the frame pointer register, the address stored in the data section can be matched with the variable. The disassembly of every assignment instructions in the ST program ends with a *store* instruction, such as **strb**.

Each internal variable is stored in a unique address, while the input and output variables are referenced with their index (e.g., 0 in %QX0). This creates a confusion between the I/O variables as both %QX0 and %IX0 are accessed with 0x00000000. To solve this issue, the entire code section of the program subroutine can be scanned, as well as whether the I/O index stored in an address is used for *read*/*write* or both. Because it is not allowed to write to an input, any *write* to the index stored in an address means that the address is holding the index of an **output** variable. Hence, the rest of the indices that are only *read* from

belong to the **input** variables. Although it is syntactically legal to only *read* from an output throughout the entire program without once *writing* to it, it is meaningless and unlikely to appear in a PLC program.

5.3.3 Function Blocks

In addition to the logical operations involving I/O and internal variables, PLC programs also have a special programming element called the **function blocks** (FBs). A function block is similar to a function, except that it may contain internal states which causes it to behave differently when being invoked multiple times with the same input. Some FBs are stateless such as MOV. However, there are nine standard FBs defined in the IEC 61131-3 standard[14] that are stateful. For example, $SR3$ is a Set/Reset (**SR**) function block in the sample program shown in Figure A.3, which shows a sample PLC program disassembly. The names and description of the nine standard FBs are shown in Table 5.1.

In the program subroutine disassembly, each function block starts with **sub sp, sp, #4** and ends with **add sp, sp, #4**, because the parameters inside the parenthesis of the FB need to be stored in a contiguous data structure, whose address is pushed into the stack. The program then calls the subroutine corresponding to the FB with **move pc, r4**, which performs the operation on the data structure and returns to the main program. All internal values and results are contained in the same data structure, residing in the same region of memory as other variables.

The key to recovering the information of FBs is to match the address of the subroutine being called with the type of FB. This can be done in two steps. First, the address being called needs to be converted to the actual memory address according to the jumping table. In the second step, the memory address is traced to one of the subroutines following the main program by checking their headers. As mentioned in Section 5.3.1, these subroutines are the FBs used in the program. 20 programs collected in Section 5.5 is sampled, which cover all the standard FBs and manually compared the code section of all the subroutines

Table 5.1: Stateful standard function blocks

Name	Description
R_TRIG	Rising edge detector. It will activate the *Q* output when a rising edge is detected on the *CLK* input.
F_TRIG	Falling edge detector. It will activate the *Q* output when a falling edge is detected on the *CLK* input.
SR	Set/Reset flip flop. A leading edge on the *S1* input activates the *Q* output. A leading edge on the *R* input deactivates the *Q* output. *S1* has priority over *R*.
RS	Reset/Set flip flop. A leading edge on the *S* input activates the *Q* output. A leading edge on the *R1* input deactivates the *Q* output. *R1* has priority over *S*.
TP	A pulse timer that enables the *Q* output for a preset *PT* amount of time after the *IN* input is enabled.
TON	ON delay timer that enables the *Q* output after the *IN* input is enabled for a preset *PT* amount of time.
TOF	OFF delay timer that disables the *Q* output after the *IN* input is disabled for a preset *PT* amount of time.
CTU	Up counter. It increments the *CV* value by one on each rising edge of the *CU* input. When the *CV* reaches the preset value *PV*, the output *Q* is enabled. A rising edge is on the *R* input resets *CV* to 0.
CTD	Down counter. It decrements the *CV* value by one on each rising edge of the *CD* input. When the *CV* reaches the preset value *PV*, the output *Q* is enabled. A rising edge is on the *LD* input loads *PV* to *CV*.

corresponding to the same FB. The result showed that every FB has a subroutine with constant code section. The data section can differ due to the changing memory allocation in each program. Hence, these subroutines are collected as signatures of FBs and a mapping is created between the them. Therefore, the type of each FB for the subroutine calling encountered in the program is recovered.

5.4 Building the Automaton

With sufficient understanding of the binary program being executed by the PLC, it is only needed to find a way to interact with the I/O of the program and monitor its state. Normally, the PLC running the program is a black-box: only the input and output can be accessed via the physical I/O of the PLC, while the internal variables and FBs are inaccessible to the user. Two methods are designed to address this issue. The first method is based on the direct connection to the JTAG port, a debugging interface on the PCB of the PLC. The second one is based on the offline execution of the binary using a customized emulator that can simulate the execution of the assembly. Both methods can be used as an interface to the PLC program and facilitate the fuzzing process which generates the automaton representing of the program's behavioral model. In this section, the method of using the offline execution method to generate automaton is discussed, which corresponds to the *fuzzer* in Figure 5.1. The JTAG-based method will be explained in Appendix A.

5.4.1 Binary Execution Emulation

One way to interact with the PLC program is to *emulate* the execution of the binary. Obtaining the binary file can be done via an interception of malicious payload, or extraction from the PLC itself. Depending on the model of the PLC, the latter may be implemented differently. The Schneider M241 PLC used in the experiment supports copying the entire content of the flash memory including the binary program to an external SD card. A more generic alternative is to dump the flash memory data via its digital interface such as SPI.

The next step is to execute the binary in a software emulator. Despite that there have been several software for emulating the CPU execution, such as QEMU[86] or Unicorn[87], they were not designed for executing PLC programs. For example, every **ldr** and **str** instruction requires the corresponding memory address to be accessible. This could be difficult to handle when the confusion rises due to the I/O variable addressing problem described in Section 5.3.2. Hence it is better to implement a novel and more adaptive framework to facilitate access to the register and memory, which can result in a better control over the program's execution. This emulation framework addresses this issue by tracking the memory access history and label the address of the I/O index stored in the data section with the corresponding variable type. Another advantage of this emulation framework is dynamically recognizing the FB structure and FB subroutine invocation using the signatures collected in Section 5.3.3. Hence, special procedures can be carried out to handle temporal-related FBs, which will be explained in Section 5.4.2.

5.4.2 Timers and Counters

Timer and counter FBs are treated differently from their original behavior for optimization. Originally, timers depend on temporal changes. However, this is not only difficult to implement, but also unnecessarily costly as temporal values have no minimum unit (i.e., it is a continuous value) and can theoretically take an infinite number of values. It is worth noting that the essence of a timer lies in whether it is *not activated*, *running* or has *expired* (i.e., reached the PT). In other words, as long as less than PT time has passed since a timer is activated, the output of the timer would remain unchanged, hence the state of the entire program remains unchanged despite the increasing timer value. Therefore, the automaton can be simplified by discretizing the timers: if a timer is activated and starts running, its PT is recorded as the transition and the timer forced to expire in the next scan cycle, simulating PT amount of time has passed.

Although a counter takes discrete values, it would still be unnecessary to test all values

as the output of the counter does not change until it reaches the preset value. Therefore, similar to the timers, a counter can be simplified and discretized into three states: *not counting*, *counting*, and *expired* (i.e., reached the PV for CTU, or 0 for CTD). Whenever a counter is triggered and starts counting, it is forced to expire in the next scan cycle, simulating that PV times have been counted. The PV will be recorded as the transition.

With the aforementioned optimizations, the automaton can be properly generated, including the temporal information. It can also reduce the space and time complexity in certain conditions, e.g., a counter with extremely large PV to trigger a deeply buried logic bomb.

5.4.3 Fuzzing

With either JTAG or binary execution emulation as an interface to interact with the program, fuzzing can be used to build the automaton representation of the PLC program. A PLC program essentially defines a set of rules which update its output based on its current state and the input given. As random numbers are rarely useful in the control system involving PLC, the input variables defined by the program almost always become the only source of input to the program. Additionally, a PLC program operates on the input deterministically with logic expressions and FBs, and can only have limited number of storage elements, which means that it can be represented as a deterministic finite-state machine (FSM), also known as the deterministic finite-state automaton (DFSA). Without loss of generality, the automaton A for a program P is formally defined as a quintuple $(S, \Delta, \mathcal{T}, s_0, \mathcal{F})$, consisting of

1. a finite set of states S,
2. a finite set of input Δ,
3. a transition function $\mathcal{T} : S \times \Delta \to S$,
4. an initial state or start state $s_0 \in S$,

5. a set of accept states $\mathcal{F} \subseteq S$.

State. Each state $s \in S$ can be a combination of the output variables (e.g., %QX0), denoted as O, the internal variables V and the FBs' states F. The state of each of the nine FBs in Section 5.3.3 is defined such that the output (Q) of the FB is dependent and only dependent on the input to the FB and its state. The definition of the state of each FB is summarized in Table B.1 in Appendix B.

Input. Each input $i \in \Delta$ can be a combination of the input variables (e.g., %IX0), denoted as I, the timers T and the counters C. Because in each scan cycle of the PLC program, the input values are always updated, any input $\delta \in \Delta$ must at least contain I. I_{max} is also defined based on the number of inputs in the program, e.g., a 12-bit input means $I_{max} =$ 0xFFF. T and C are optional depending on whether any timer or counter FB is activated.

The fuzzing process is depicted as a flow chart in Figure 5.10. The algorithm begins with the following initialization (①):

- reset all input variables
- deactivate all timers and counters, reset all timers' *elapsed time* (ET) and counters' *accumulator* (Acc)
- initialize all internal variables and output variables to the values specified in the initialization subroutine

The fuzzing algorithm in general is carried out similar to the execution of PLC programs, i.e., a loop of actions consisting of reading the input, executing scan cycle, and updating the output. The initial state s_0 is referred to as the current state, s_α (②). After a transition $\tau \in \mathcal{T}$ is applied (④ and ⑦), as will be explained in this section, a single scan cycle is executed on the PLC program (⑨), updating all its internal variables, FBs, output variables, etc. The new state is referred to as the next state, s_β (⑩). This process iterates until the entire automaton is generated (⑱).

The automaton that will be generated is equivalent to a **directed multigraph**, with each state in the automaton as the node, and each transition as the edge. Starting from s_0, the maximum I_p that has been tested on each state s_i is recorded in a mapping $\mathcal{M}\colon s_i \rightarrow I_p$, where $s_i \in S$ and $0 \leq I_p \leq I_{max}$. If no timer or counter is activated in the current state, the fuzzing is in **normal mode** (③), otherwise it is in **timer/counter mode** (⑤).

Normal mode. In normal mode, the objective is to traverses through the entire automaton with depth-first search (DFS), by enumerating every I_p value in every state s_i, while updating the mapping $\mathcal{M}$ with $s_i \rightarrow I_p + 1$. If a new state s_j is discovered (⑪), $\mathcal{M}$ is updated with $s_j \rightarrow 0$. Otherwise the algorithm goes into the next iteration (⑬). If $\mathcal{M}(s_i) > I_{max}$ (⑭), the DFS reaches a node with no more outwards edge which has not been traversed. In this case, the automaton needs to be restored to a state s_k (if there is any) for which $\mathcal{M}(s_k) \leq I_{max}$ (⑮). Otherwise the algorithm goes into the next iteration (⑯). For any input I_x that causes the state of the automaton change from s_α to s_β, a transition τ will be recorded iff $s_\alpha \neq s_\beta$ (⑫).

Timer/counter mode. In timer/counter mode, at least one timer or counter is activated in the current state. Since this is a transient state, the objective is to reach a stable state (i.e., no timer or counter) and resume normal mode. Hence the input will be kept unchanged from the last transition (⑥). During this mode, new states will not be added to $\mathcal{M}$. All transitions will be recorded in the automaton. However, if a loop of states is detected in the timer/counter mode (⑧), the automaton needs to be restored to a state s_k for which $\mathcal{M}(s_k) \leq I_{max}$ before going into the next iteration (⑰).

The fuzzing completes when $\mathcal{M}(s_i) > I_{max}, \forall s_i \in S$.

5.5 Data Collection

Unlike traditional computer programs and software, PLC programs are much harder to collect in a large scale. Because most PLC programs are written for specific physical systems, it is unlikely that more than one implementation will be available for each system. More-

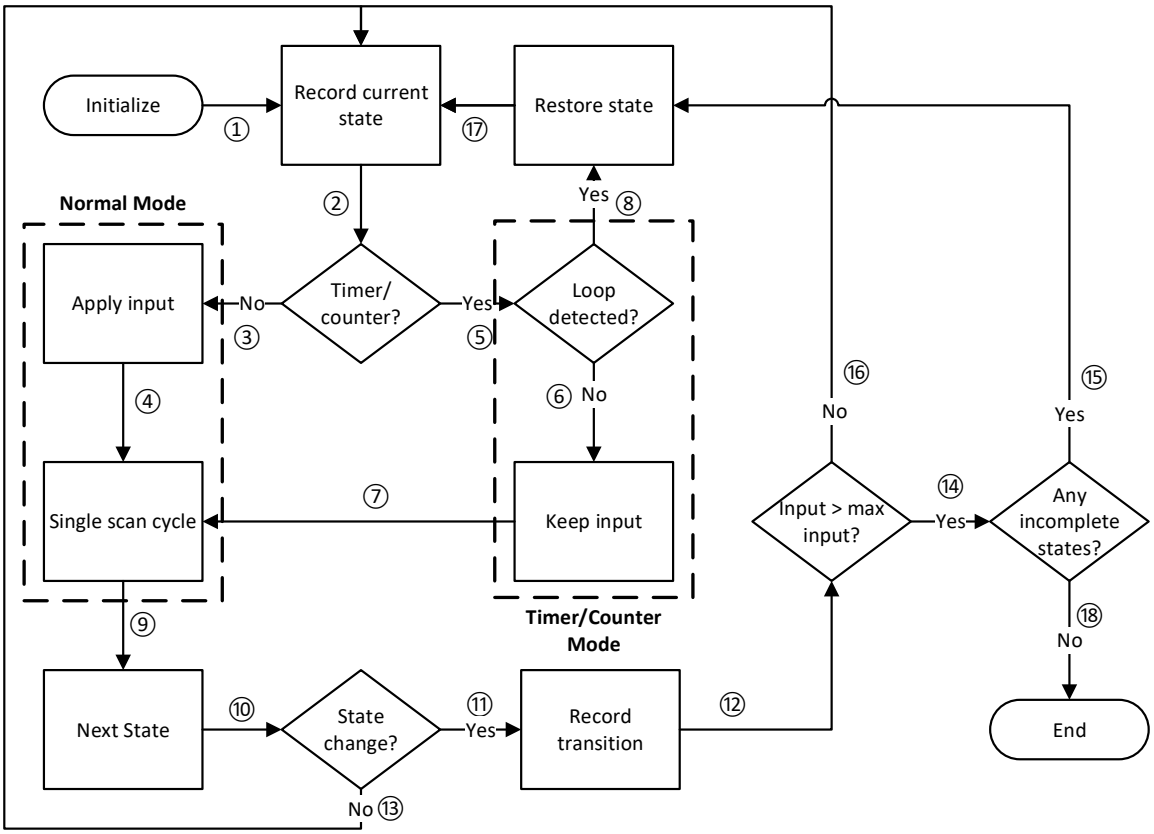


Figure 5.10: Flow chart of the fuzzer which generates the automaton

over, programs written for existing ICS may not be willingly shared, as it may interrupt the normal operation of the system and leak the confidential information. This may lead to economic loss or even cause targeted attacks. To make data collection even more difficult, various models of PLCs may be used in the industrial environment. Hence it either requires more effort in reverse engineering or collecting more data to increase the number of programs for a specific PLC.

To address these issues, a highly-rated cyber security class taken by graduate students in the author's university was leveraged. Four realistic scenarios were designed, namely *TankBalancer*, *StirringSystem*, *RobotPath* and *TrafficLight*. Each scenario has a physical system controlled by a PLC. Then the scenarios were distributed as project assignment to the students in the class, which were well trained with the knowledge background in ICS security and PLC programming. They were asked to write PLC programs that satisfy the requirements in each scenario. For two of the scenarios, the students were also asked to implement an *attack* version of the program, which contains a hidden "logic bomb" that is only activated after certain conditions are met. To accommodate the size of the class and facilitate the students' implementation, OpenPLC[88] was used as the PLC device in each scenario. As shown in Figure 5.11, the Object Linking and Embedding (OLE) for Process Control (OPC) protocol[89], more specifically OPC Unified Architecture (OPC UA) was added to the original OpenPLC design. It can exchange I/O data with a process simulator program and a human-machine interface (HMI) software. The students can leverage the OpenPLC editor to write and test the PLC program in any IEC 61131-3 programming language, which is similar to the experience in writing a program for physical PLCs. To expand the size of the data and make the programs more generalized, the submissions from the students were collected **across multiple semesters**. After the students' submissions were graded, those which satisfied all the requirements and earned full credits in each scenario were collected. After being anonymized, these collected programs were converted to project files for a real PLC, i.e., Schneider M241, before being compiled to binaries. In

total, 650 PLC programs have been collected. The details of each scenario and its requirements are described in Appendix B.1.

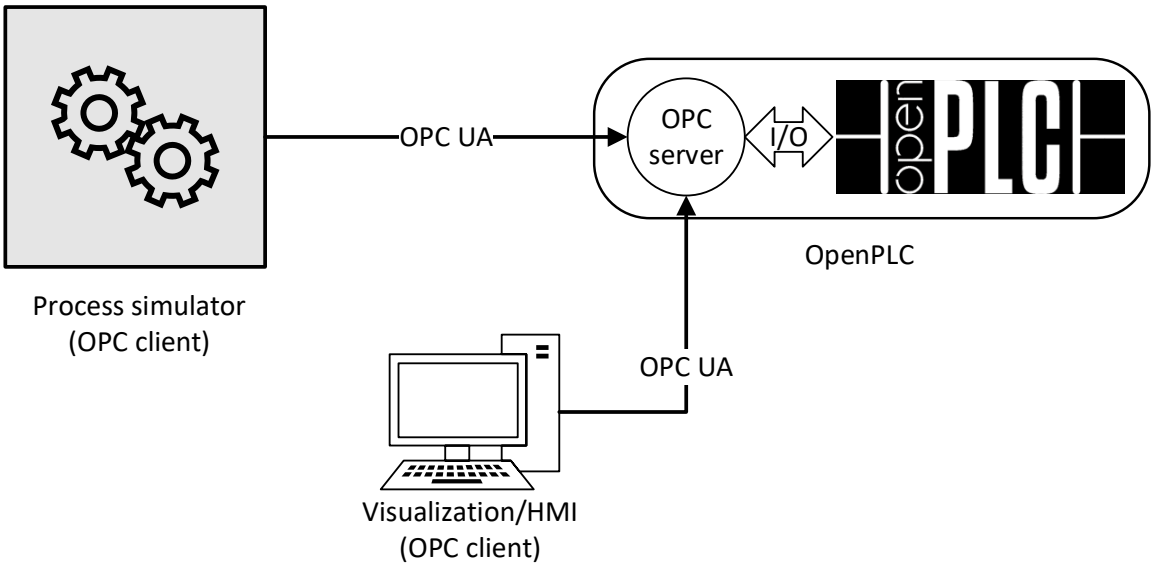


Figure 5.11: Process and PLC simulator architecture.

Ethics. As data was being collected from human subjects which may involve personal data, cautious procedures were taken and the IRB in the author's institute was consulted with. After detailing the research plan to the IRB, the author was told that IRB review was not required as no personal data about the students would be used. The programs have also been anonymized so that no data can be identified to the individual student.

Data Validation. To ensure the data is valid and representative, several measures are adopted throughout the each stage in the data collection process. The class was only offered to skillful graduate students, who have solid understanding of the design principle in PLC programming. The aforementioned scenarios were designed to implement critical functionalities in real ICSs (e.g., water treatment plant), in addition to providing educational values. The students were also provided with graphical HMI software (which is a common practice in industrial environment) specifically designed to help them better understand each process in an immersive environment. After collecting the students' submissions, the PLC programs written by the students are manually checked and ensured that they fully

Table 5.2: Number of automata in the four categories

Category	Number of automata
TankBalancer	224
TankBalancer_Attack	117
StirringSystem_v1	54
StirringSystem_v2	113
StirringSystem_attack	33
RobotPath	58
TrafficLight	51

met each process's requirements. The data collected for each process is shown in Table 5.2. These programs are **open-sourced** to help future studies in this domain.

5.6 Evaluation

LogicFuzzer consists of two stages: **classifier**, which determines whether a suspicious control program is written for a given physical process; and **detector**, which searches for malicious states that the program can run into and finds the conditions to trigger such states. As introduced in Section 5.4 and Section 5.5, the dataset consists of the automata extracted from PLC programs. These automata are then used as the input to the classifier and the detector.

This work is evaluated by answering the following questions:

- Q1. Can the classifier correctly predict which process an automaton corresponds to?
- Q2. Can the detector determine whether a program is malicious reliably?

5.6.1 Classifier

Features

To classify these PLC programs efficiently, several features that can be extracted from their corresponding automata are analyzed:

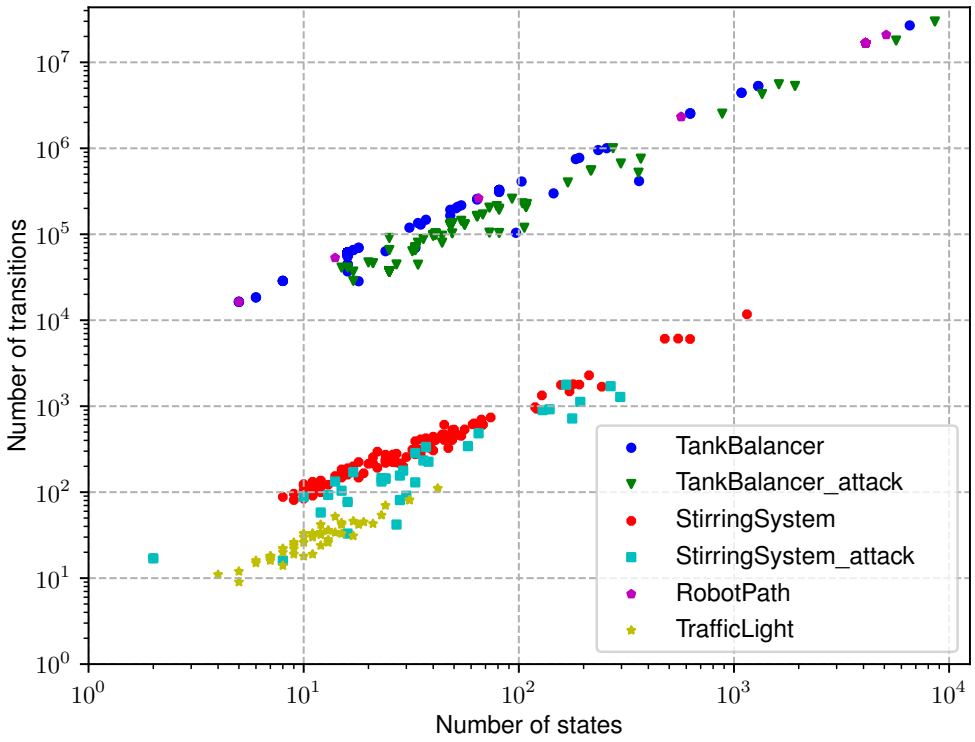


Figure 5.12: States versus transitions.

Table 5.3: The average of *Number of States, Average Degree, Degree Variance* and *Triggers* in all categories of data.

Category	Number of States	Average Degree	Degree Variance	Number of Triggers
StirringSystem_v1	108.800	9.539	5.781	5.296
StirringSystem_v2	63.531	9.913	7.082	4.929
StirringSystem_attack	61.393	6.072	4.697	8.424
TrafficLight	12.686	2.551	2.586	6.471
TankBalancer	54.982	3749.799	2889.723	0.509
TankBalancer_attack	152.922	2179.156	1483.343	2.034
RobotPath	3782.172	4074.586	385.385	0.034

- **Number of States.** The number of an automaton's states is a very basic feature. Based on the different approaches of implementation, the number of states of PLC programs which corresponds to the same process may vary greatly. However, it can be observed that the number of states of programs that correspond to the same process is within a range. The number of states is used instead of that of transitions because it is usually more representative of the process.

- **Average Degree.** In graph theory, the degree of a node indicates the number of edges which end with or start from it. One edge in a graph is associated with two types of degrees, one for the start vertex and one for the target. Therefore, the average degree can be calculated by num_edges/num_nodes. The average degree of different programs is compared and it is found to be highly correlated with the type of the process the program is associated with. Figure 5.12 shows the comparison between the number of transitions versus the number of states of all the processes. It is observed that the scattered plot corresponding to each process approximates a straight line, indicating a strong correlation between the number of transitions and the number of states. Additionally, different processes can be visually separated in the plot, suggesting that the average degree can be an important feature in the classification.

- **Degree Variance.** Figure 5.13 shows the visualization of four automata, two of which are from *Stirring System* programs and the other two from *Traffic Light* programs. It can be observed that there are many differences among the nodes. Hence, three types of nodes are defined: **border-node**, **pass-node** and **pivot-node**. Border-node refers to the node with a relatively small degree, and most of its edges are pointing to instead of starting from it; pass-node refers to the node with only one edge in and one edge out; pivot-node refers to the node with a relatively larger degree compared to the average. These types of vertices are usually sufficient to describe a PLC automaton. For instance, the implementation of a function can be made of a sequence

of states, and there are many functions of which the states can only transform from the previous to the next, i.e, if a function is made of n states $s_1, s_2, ..., s_n$, then s_i is only linked to s_{i+1}. That means each vertex in this graph has one in-degree and one out-degree, which is outlined as *pass-node*.

Instead of directly using the proportion of these three types of nodes as a feature, the variance of degree is considered a better choice. With a larger variance of degree, a graph is usually much more complicated, with more types of nodes in addition to border-node, pass-node and pivot-node. Meanwhile, with a smaller variance, a graph tends to be simpler.

- **Triggers.** As introduced in Section 5.3, a PLC program may contain several function blocks such as *F_TRIG* or *TON*. These triggers are designed to implement specific functionalities. Therefore, the types of triggers and the number of triggers a program has used can be useful information to infer which category of process the program belongs to.

Note that if both *Number of States* and *Average Degree* are chosen as features, it implies that the number of transitions is also considered. The average values of concluded features (*Number of States, Average Degree, Degree Variance* and *Triggers*) are shown in Table 5.3.

Feature Generalization

Four features are chosen to classify PLC programs' automata. **Number of States** and **Triggers** are fundamental elements that are associated with the function of a PLC program. Here it is discussed how the other two features can be generally used in PLC program classification.

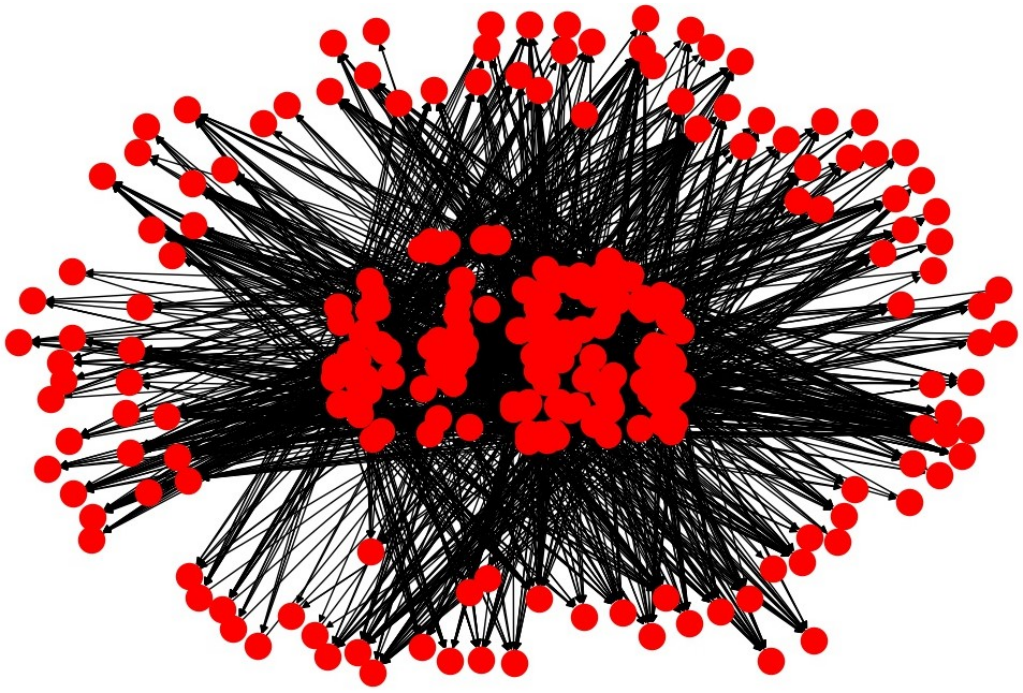
Average Degree. Programs which implement the same functionality can have different automata. Starting with the minimal version, a variant of the most minimal automaton could be thought as new transitions which transform the original states to redundant states,

denoted as $s \rightarrow r(s)$. Thus there should also be new transitions $r(s_i) \rightarrow r(s_j)$ corresponding to the original transitions $s_i \rightarrow s_j$. Suppose the origin graph has n nodes and m edges. In the extended graph, there are $n + n = 2n$ nodes and $m + n + m$ edges. Further, there would be $k * n$ nodes and $k * m + (k - 1) * n$ edges with k redundant extensions. The ratio of edges and nodes is $m/n + (k - 1)/k$, which is very close to m/n. Therefore, it is sufficient to use average degree as a feature.

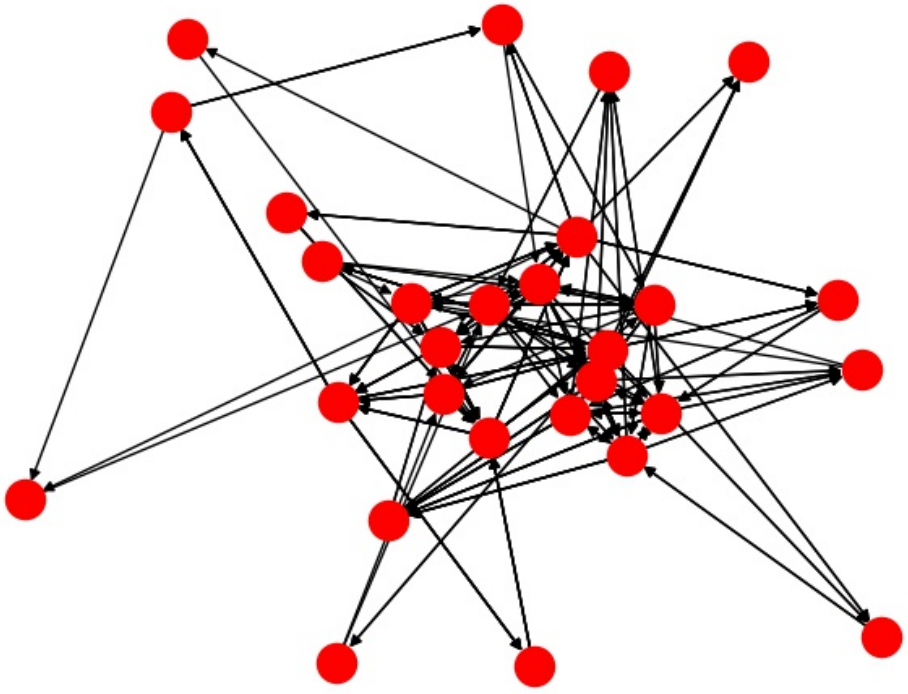
Degree Variance. As described earlier, summarized three types of nodes are summarized, namely *border-node*, *pass-node* and *pivot-node*. However, there are much more different varieties of nodes in other PLC processes. Hence the best way to use this kind of feature is using the proportions of summarized special nodes if the classifier knows the samples in advance. However, in many cases the classifier has no idea what processes the PLC programs are from. To address this issue, the feature **Degree Variance** is used, which to some extent implies the proportions of different types of nodes. The experiments have shown that this feature contributes greatly.

SVM-based Classification

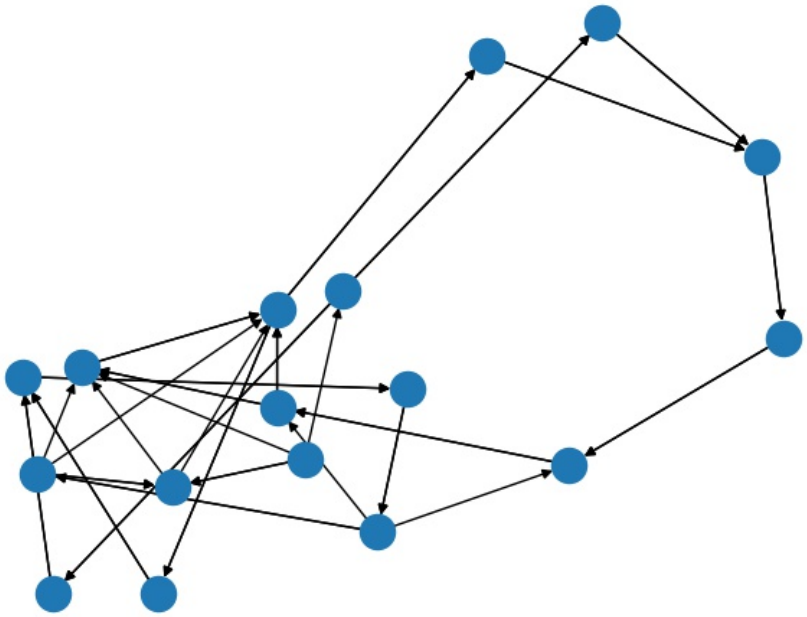
A classification method is implemented based on support vector machine (SVM). There are already four features introduced earlier: *Number of States, Average Degree, Degree Variance* and *Triggers*. Therefore, these features can be directly used to test the effectiveness. In addition, the features are adjusted to make it more specific based on SVM. The *Triggers* feature is further divided into four specific features: (1) R_TRIG and F_TRIG; (2) SR and RS; (3) TP, TON and TOF; (4) CTU and CTD, based on their functionalities. For example, CTU and CTD are both counters. The data is split into training set $(2/3)$ and testing set $(1/3)$. With the SVM classification method, LogicFuzzer achieved an average accuracy of 98.9%.



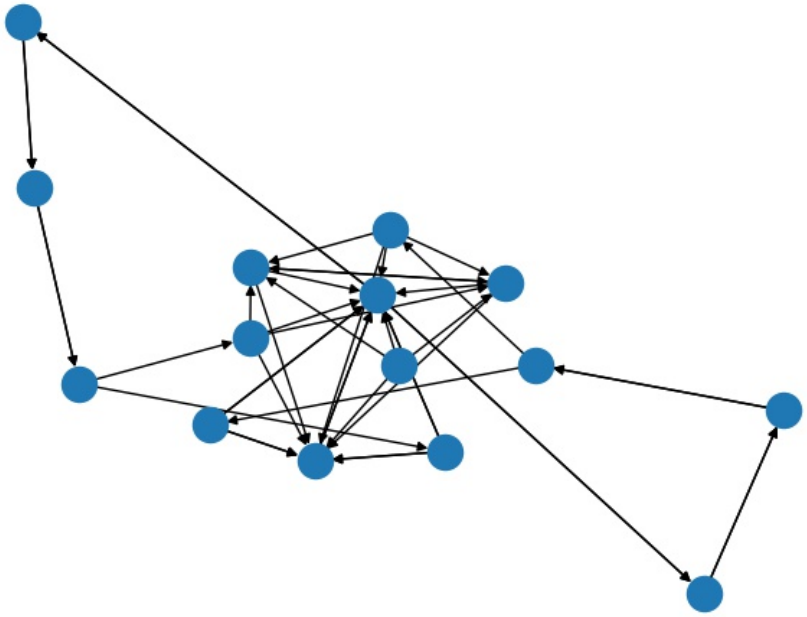
(a) Stirring System a.



(b) Stirring System b.



(c) Traffic Light a.



(d) Traffic Light b.

Figure 5.13: Visualization of some automata. There are various types of vertices in a graph. E.g., in Stirring System's automata, there are more *pivot-nodes* while in Traffic Light's automata there are more *pass-nodes*. The proportion of different types of vertices can be a effective feature to distinguish graphs.

Rule-based Classification

When used in real scenarios, LogicFuzzer usually faces a binary classification problem rather than a multi-class one, i.e., the output will be whether the target program is associated with a certain physical process. In this case, training a machine learning model is infeasible due to lack of large samples of data. Hence the rule-based method can be used. The rule-based method classifies programs based on rules decided by administrators.

By observing the features given in Figure 5.3, it can be found that based on *Average Degree*, programs can be divided into two classes: 1. StirringSystem and TrafficLight, which have small degrees; 2. TankBalancer and RobotPath, which have quite large average degrees (more than one thousand). Furthermore, StirringSystem programs can be distinguished from TrafficLight programs, or TankBalancer programs from RobotPath by checking their *Number of States* and *Degree Variance*. The rules are applied in the following two steps: in the first step, the programs are classified into which of the two big classes they belong to by their *Average Degree*; in the second step, a variant of the sigmoid function is used to calculate the similarity between the program and the other two categories.

Finally, the rule-based classification reached an accuracy of 95.2%. Among those programs which are mis-classified, more than 50% are attack version programs. Therefore, the attack version of a program does have difference with the normal version programs, which can help us using detector to detect whether a program is malicious.

Similarity-based Classification

In the situations where manually providing rules for classification is infeasible or inconvenient, an alternative method can be used by LogicFuzzer. The user may provide the original control program written for the physical process for reference, and compare the similarity between the target program and the reference program. The similarity comparison is based on the graph structure of the automata generated from the two programs. Intuitively, this can be computed by counting the number of states and transitions that are

common in both automata, then divided by the size of the automata (i.e., total number of states and transitions) to normalize the result. However, there are several challenges that make this process difficult to implement. Recall that the state is defined as a combination of the output variables, the internal variables and the FBs' states. While the use of output variables are constrained by the connections to the physical output of the PLC, the use of the internal variables and FBs are less restrictive as long as the program's behavior (i.e., output) meets the requirement. In some cases, unused output may even be mapped to the internal variables to further complicate the situation. Hence the states in two automata cannot be directly compared for equality check. Similarly, transitions cannot be directly compared because the use of timers and counters is at each programmer's discretion. To address these issues, the automata are processed with the following steps.

1. Sanitize the state by masking the output, leaving only the connected output in the PLC.

2. Group the adjacent states which become the same from the sanitization into a "super node".

3. Remove the transitions inside the "super node" from the automaton.

After the processing, the similarity score can be obtained by computing the maximum ratio of connected graph in the reference automaton which is also contained in the target automaton. Four standard programs are used as the reference programs. The standard programs were written by the course developer as example programs. As shown in Figure 5.14, each program collected in the four categories is tested with all four reference programs. The target program leads to a high similarity score when it is in the same category as the reference program. Meanwhile, the similarity score is close to 0 if the category is different. More interestingly, as can be seen in Figure 5.14a and Figure 5.14b, programs written for the same physical process but with different control objectives show slightly lower similarity, which is still much higher compared to the programs written for different processes.

Moreover, the attack version of a program also shows a slightly lower similarity, as can be seen in Figure 5.14c and Figure 5.14d. This result indicates that the similarity-based classification is effective in measuring the likelihood which the target program belongs to a given category of physical process.

Summary

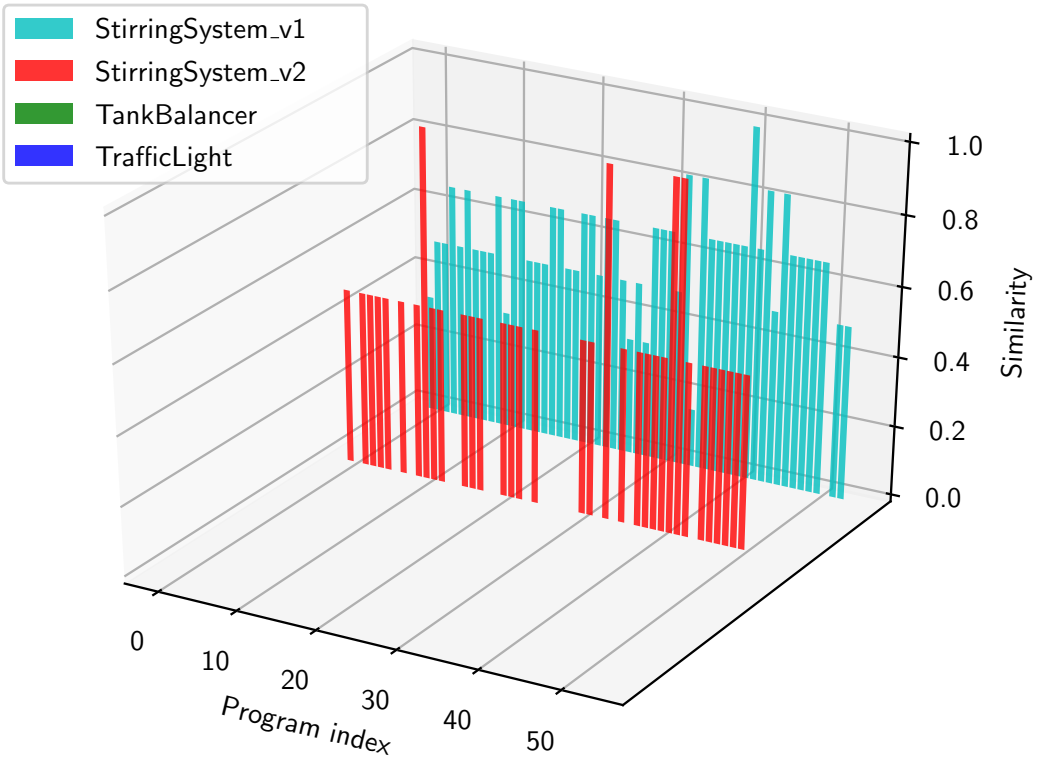
Q1 is answered in this subsection. The classification result is summarized in Table 5.4. The SVM based classification has a higher accuracy while it needs more time to extract features from data and train on them. The rule-based method can be a sufficient choice if the time for training is too costly or when there is a lack of sufficient samples to train with. Additionally, the similarity-based technique can provide a convenient method for the user as an alternative.

Table 5.4: Results of classification. Training Time contains both feature generating and training. Predict Time is the average time spent on reading and predicting one single sample.

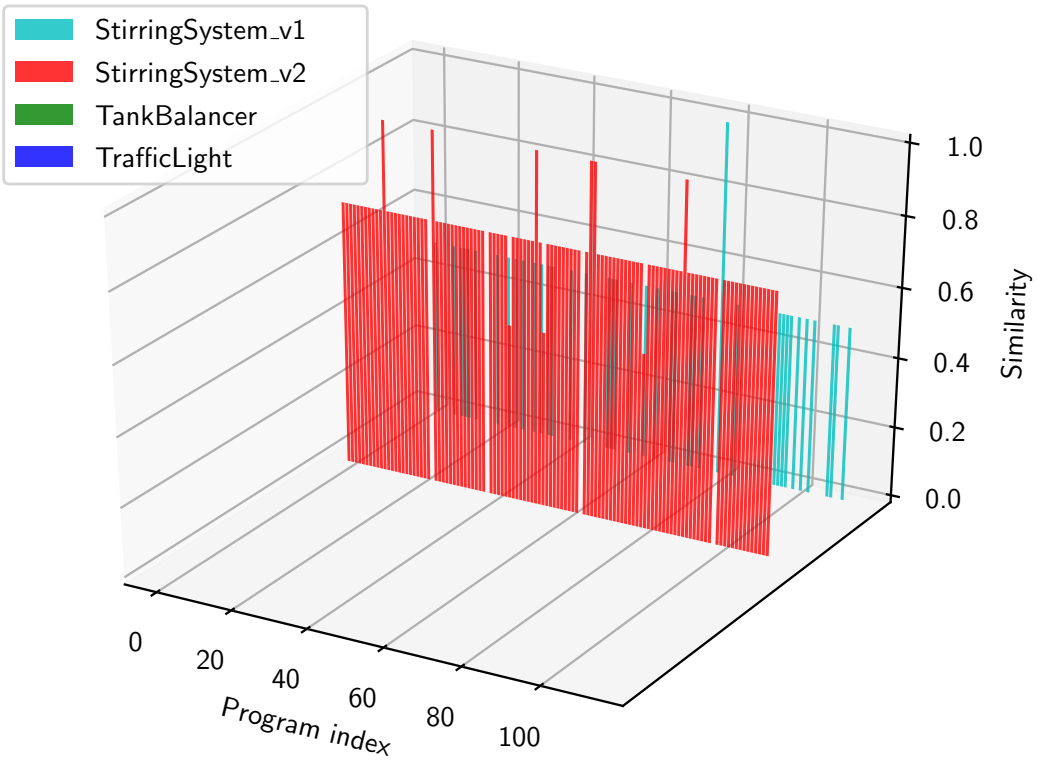
Method	Training Time(s)	Predict Time(s)	Accuracy
Rule	-	11.1	95.2%
SVM	8111	12.5	98.9%
Similarity	-	29.5	96.7%

5.6.2 Detector

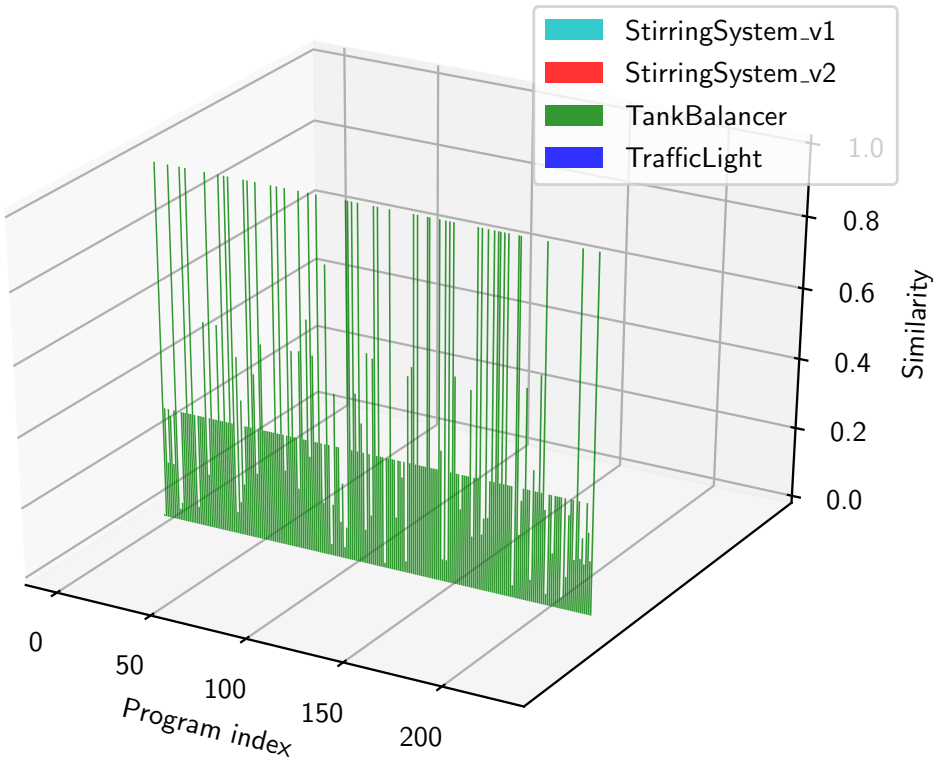
As shown in Table 5.3, in the *attack* version of the programs, there is always an increase for *Triggers* and a decrease for *Average Degree*. Increased usage of triggers is easy to understand: attackers usually need a timer or a counter for a conditional execution of the malicious logic. Figure 5.15 shows the distribution of *Number of States* in both benign and malicious programs. The *Number of States* in the *attack* version is usually no less than that in the normal version, which indicates that there are fewer transitions detected in the *attack* version of the programs. The programs and their automata are analyzed and it



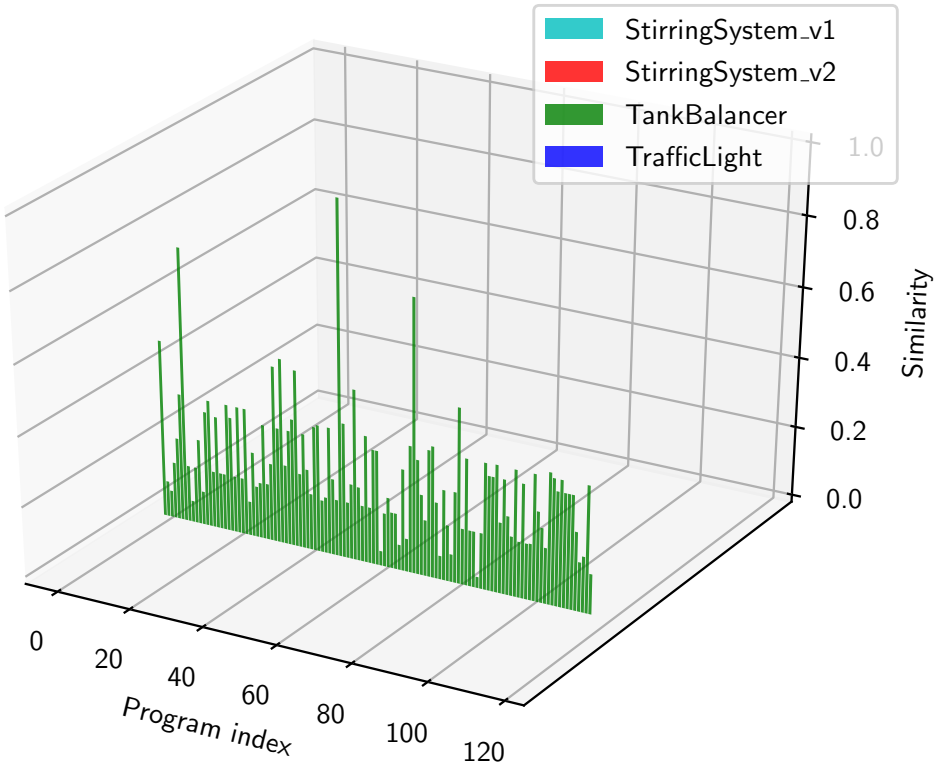
(a) StirringSystem_v1.



(b) StirringSystem_v2.



(c) TankBalancer.



(d) TankBalancer_attack.

Figure 5.14: Similarity score of between each program and the reference programs.

Table 5.5: Result of detection. Training Time contains both feature generating and training; Predict Time is the average time spent on reading and predicting one single sample.

Method	Training Time(s)	Predict Time(s)	Accuracy
SVM	479	1.5	83.8%
Rule	-	0.3	98.9%

is found that malicious programs would hijack the control flow of a benign program and lead it to run the malicious logic, which bypassed some normal states and introduced the unsafe states. Besides, the malicious logic is usually simple and not as complicated as the intended functionality, which explains the decrease in the *Average Degree* and *Degree Variance*. Therefore, it is feasible to determine whether a program is malicious or not based on the result obtained by a classifier.

In addition, determining whether a program is malicious can rely on the rules formulated by system administrators. As explained earlier, an intention of a malicious logic is to lead the program to the unsafe states. These unsafe states can either be obtained from the system administrator or by monitoring the legitimate process. For instance, in Tank Balancer System as described in Appendix B.1.1, the attack version programs will cause Tank 2 to overflow after 60 seconds, which means after 60 seconds, the output value of Tank 2's "OUT" will always be False. Thus, the administrator can make a rule such as *"Any tank's 'OUT' should change periodically."*. The detector can check the automaton for whether the program would go into a loop where some of the tank's "OUT" is always the same.

Q2 is answered in Table 5.5. The SVM based detector resulted in an accuracy of around 85%, which is not satisfactory. However, the rule-based detector achieved the accuracy of 98.9% with higher efficiency. The *precision* and *recall* of both methods are also listed in Table 5.6. Therefore, using the rule-based malicious detector is more efficient and accurate.

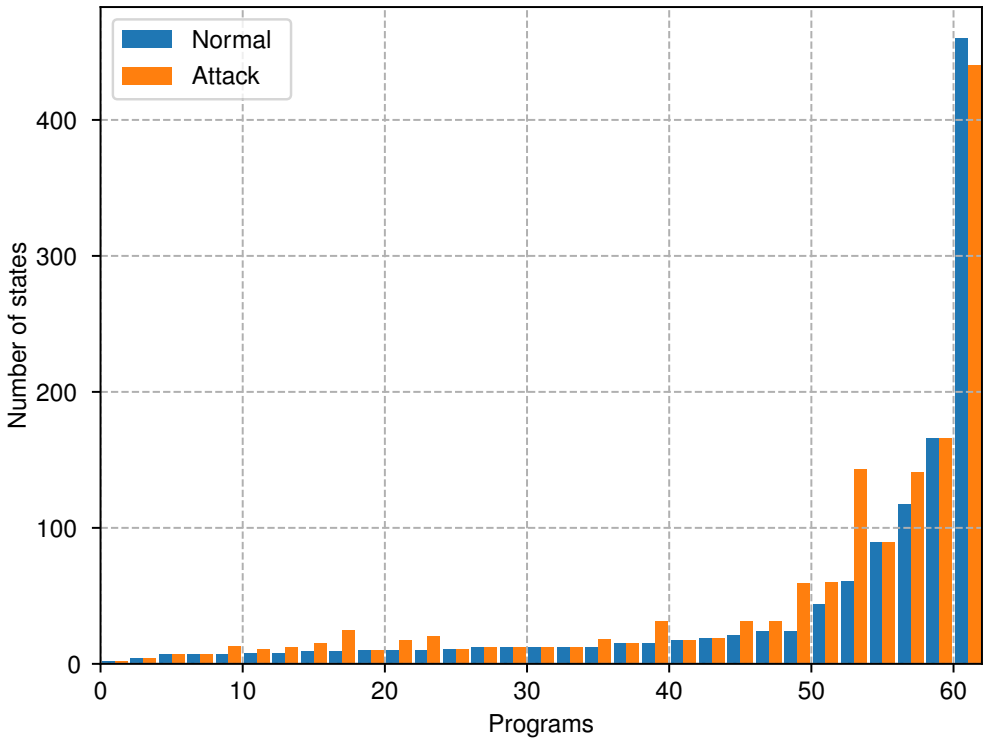


Figure 5.15: Number of states: normal versus attack. The number of states in the attack version is usually no less than that in normal version.

Table 5.6: Precesion and recall of detection methods, where precesion is $TP/(TP+FP)$, and recall is $TP/(TP+FN)$.

Method	Precesion	Recall
SVM	94.2%	88.0%
Rule	99.6%	99.2%

5.6.3 Summary

In this section, the fuzzer, classifier and detector are evaluated. The fuzzer can obtain automata from PLC programs within a reasonable amount of time. For the classifier, SVM performs slightly better because it has a higher accuracy, and does not cost significantly more time than the rule-based method. The detector based on rules which are formulated by administrators has an accuracy close to the SVM method and immediate response.

5.7 Discussion

In this section, how LogicFuzzer can be generalized and the limitations of this work are discussed.

5.7.1 Generalization

A common question that should be asked in every solution to a security-related problem is how well it can be extended to a broader scale. As a framework designed based on the characteristics of the PLC, LogicFuzzer is agnostic to the large timer or counter values set by the attacker to hide the malicious code segment from regular checking. Whenever it detects the activation of such FBs, it will directly read the preset time/value from the data structure and override the FB to trigger the event being hidden. A useful application that can further extend LogicFuzzer is that it not only detects the malicious states, but also shows the attack path which the program takes to enter the malicious state. This can be done by a search for all paths between two states in the automaton. The first state can be the initial/reset state of the program, and the second state is the malicious state. For

Table 5.7: Time to generate the automaton. n is the total number of states. p is the number of inputs. σ is the time to perform a single cycle scan.

Process	Average n	p	σ (ms)	Average time (s)
TrafficLight	13	2	3.5	1.9
StirringSystem	109	4	2.7	2.8
TankBalancer	55	12	1.2	148.9
RobotPath	3782	12	2.3	18122.6

example, such path is found in 28 out of the 33 programs in the *StirringSystem_attack* category. The path contains the information required to reproduce the malicious behavior in a physical PLC in ICS, such as the inputs and time to wait for.

Another important feature of LogicFuzzer is that it is free of the *path explosion* problem which is often encountered in symbolic execution techniques. This is because LogicFuzzer does not rely on analyzing the feasible paths a program can take, which would result in infinite number of paths in case of programs with unbounded loop iterations. Instead, it runs the program and traverses through the automaton of the program based on concrete observation of the path which the program *actually* takes. As a result, the time it takes for LogicFuzzer to generate the automaton of a program is largely dependent on the size of the automaton. Because LogicFuzzer performs an exhaustive search of all possible inputs on each state, the time complexity would be $O(n * 2^p * s)$, where n is the number of states, p is the number of inputs and σ is the time to perform a single cycle scan. Although there is an exponential term 2^p in the big O notation, in reality, p is usually limited by the number of physical inputs on the PLC. The time it takes to generate the automaton is measured and it is found that the experimental results agree with the analysis as shown in Table 5.7. This means that LogicFuzzer performs well even as the complexity of the program grows.

5.7.2 Limitations

One limitation of LogicFuzzer is in the binary parsing stage of the system, which requires understanding of the ISA and the file structure for the specific model of the PLC. Re-

searchers not affiliated with the manufacturer of a PLC may be left with the time-consuming option of reverse engineering, although it is only one-time effort for each PLC model.

The other limitation is the cost associated with the generation of the automaton. As discussed in Section 5.7.1, the time complexity of the automaton generation largely depend on the number of states and the number of inputs (the time to perform a single cycle scan is directly proportional to the length of the program). Although the number of inputs is limited by the physical PLC, the number of states can vary greatly due to the change in the program. How this can affect the effectiveness of LogicFuzzer is discussed. Moreover, how to handle the situation where an attacker has advanced knowledge of LogicFuzzer is also discussed.

Currently, LogicFuzzer is designed to exhaustively search for the entire state space of the program to discover all its behaviors, due to the well defined input in the PLC. In contrast, the fuzzer for a general computer program needs to "guess" which input values can be most likely used to trigger a program's certain behavior, since most computer programs can in theory take infinite number of inputs (e.g., a string input can be arbitrarily long). However, this may not always work well for LogicFuzzer. As the state is composed of the outputs, the internal variables and the FBs' states, the number of total states in the automaton of the program grows exponentially with respect to the sum of the number of these elements. Similarly, the search space for transitions, consisting of inputs, also grows exponentially to the number of inputs. In a normal program, variables are usually used when necessary. Moreover, from the observation of the collected programs, many variables are dependent on each other. For example, a program may have three variables to denote the state of the mixing blade at different places in the *Stirring System* process. When being fuzzed, these three variables will always have the same value and hence does not multiply the total number of states of the program by 2^3.

However, if an attacker has advanced knowledge of LogicFuzzer, he may try to find a relatively easy way to add dummy variables into the program to significantly increase the

search space of LogicFuzzer. He may then exploit such asymmetry in order to make it more difficult to analyze the malicious program. Assume the existing variables are labeled v_1, v_2, ..., etc. Generally, there are three ways to add dummy variables:

1. Dummy variables only appear on the left hand side (LHS) of the instructions, e.g., $v_{d1} = v_1$, $v_{d2} = v_2\ AND\ v_3$

2. In addition to 1), dummy variables only appear on the right hand side (RHS) of the instructions with dummy variables on the LHS, e.g., $v_{d1} = v_1\ XOR\ v_{d1}$

3. In addition to 2), dummy variables appear on the RHS of the instructions with existing variables on the LHS, e.g., $v_{d1} = NOT(v_1)$, $v_2 = v_{d1}$

For 1), the state space of the program does not increase as mentioned previously. For 2), the dummy variables can be found and excluded from the state with a data dependency analysis. Because the no instruction with existing variables on the LHS is changed, the values of the added dummy variables do not affect the outputs of the program. For 3), although data dependency analysis cannot rule out the dummy variables, adding dummy variables to the RHS of the instructions with existing variables on the LHS means the functionality of the program will be changed. For example, only a TON and a state variable are added into the *Stirring System* program so that the instruction with outlet valve on the LHS now has the added TON on the RHS. Upon simulating the execution of the program, it is found that the mixture level changed differently compared to the requirements as shown in Figure 5.16. This would cause a premature exposure of the malicious intention of the program and result in an immediate discovery by the system administrator in the initial testing stage. Hence, the attacker needs to trade-off the increase in the state space of the program with the stealthiness of the attack.

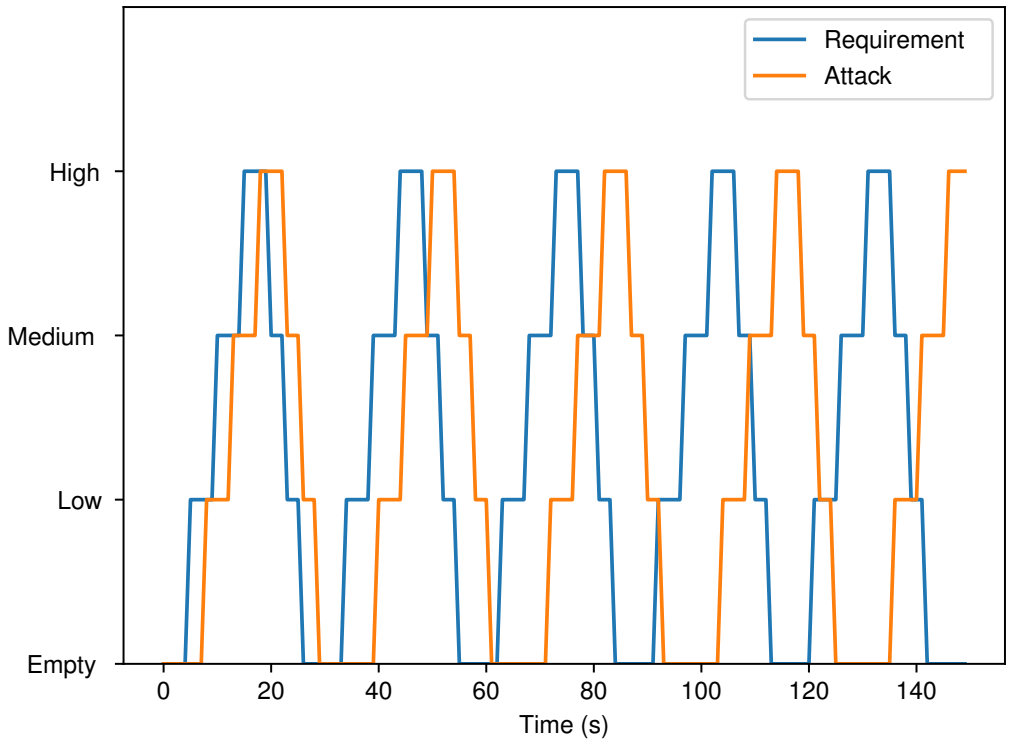


Figure 5.16: Mixture level of the tank in the Stirring System controlled with the malicious program, compared with the original requirements.

5.8 Conclusion

In this chapter, a framework called LogicFuzzer is presented, which generates the behavioral model of PLC program binary via fuzzing. Automaton representation is used to model the program's behavior and developed analysis methods based on the automaton to classify the program into specific physical process and detect whether the program is malicious. Program data is collected to validate this framework and achieved high accuracy in both the classification and detection. Overall, LogicFuzzer can be considered as an effective framework which can be used to secure the PLC-based systems.

CHAPTER 6

IDENTIFYING THE PROCESS PARAMETERS USING SIDE-CHANNEL INFORMATION

When securing a cyber-physical system (CPS), the most commonly used methods focus CPS itself, including both the information technology (IT) and operation technology (OT) domains. While such domains are most tightly associated with the underlying systems and thus can block most of the active and passive attack vectors, physical side channel has inevitably become an important source of information leakage, which can be a form of passive attack or even a pre-sequel of an active and orchestrated attack. The use of physical side channels to infer information about a (presumably secure) system has been demonstrated to be effective in many areas, such as reconstructing the object being printed with 3D printers through the sound emitted [90], or detect the leaking information about the underlying cryptographic computation in a CMOS from its electromagnetic emanations [91]. In this research, audio channel information is leveraged as side channel information of an operating CPS to study the feasibility of identifying the process parameters using the side channel information. More specifically, the types of devices, their operation status and their locations in space are inferred from the audio recorded using microphones. Convolutional neural network (CNN) is employed to learn and predict these parameters based on the transformed audio data. The result demonstrates that with only small amount of training data, the CNN can correctly predict the operation status of individual devices in a realistic water treatment testbed with approximately 100% accuracy.

6.1 Introduction

6.1.1 Audio Side Channel in CPSs

The use of audio side channel as sources of information to a physical system has been widely studied in the past few years. For example, researchers have reconstructed the objects being printed by additive manufacturing systems, also known as 3D printers, using the information carried by the sound emitted while the objects is being created [92]. The correlation between the audio emitted and the object being printed is established beginning with the G-code, a commonly used computer numerical control programming language generated from the model file of the object. The G-code instructions are translated to the actuation of stepping motors which control the movement of the printer's nozzle (or equivalently the printing platform), including their speeds and directions. As pulse width modulated (PWM) signals are fed to the stepper motors, high frequency sounds are generated. Therefore the audio carries information of the speed and axis information of each stepper motor, which can be used to recover the movement of nozzle in the space as well as filament extrusion speed.

Similarly, in industrial control systems (ICSs), the audio side channel can carry significant amount of information about the operation status of the ICSs. The most commonly used actuators in ICSs include motors (ranging from synchronous/asynchronous motors, to stepper motors), pumps, relays, etc. Most of these devices produce audible sounds when operating. Furthermore, the frequency of the sounds produced is often correlated to the operating parameters of the device. For example, as the rotation speed of a motor increases, it produces higher pitch sound. Such correlation means the operating parameters of the devices can be inferred based on audio side channel emitted by these ICS devices. If such parameters can be accurately inferred, it could bring significant implications for CPS security applications. For system administrators whose main objectives include ensuring the safety and integrity of the CPSs, the audio side channel can provide an additional layer of

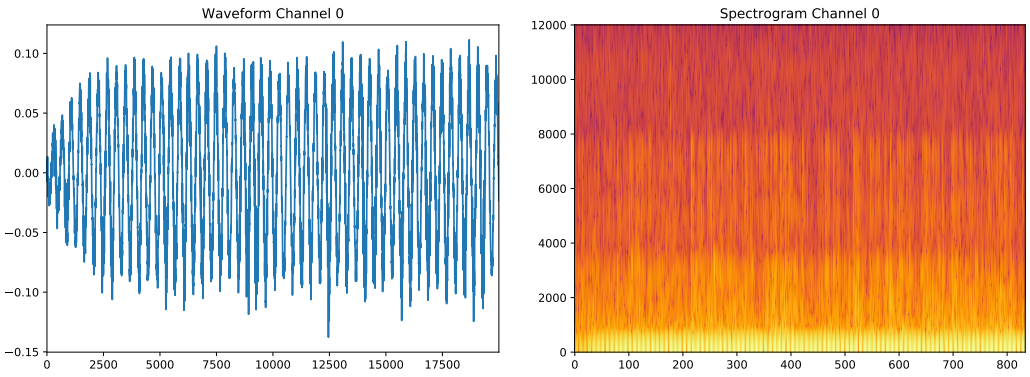
assurance for the underlying physical system by comparing the parameters inferred from the audio side channel with the expected values. On the other hand, attackers may use such information to steal confidential and sensitive intellectual property information about the CPSs, such as process control design information in chemical engineering sectors which are often trade secrets. More sophisticated attackers may even steal such information as a prior espionage step for launching a more stealthy and damaging attack.

6.1.2 Analyzing Audio with Deep Learning

As the deep learning methods advance in the past few decades, they have been applied in many domains in the research academia, and achieved astonishing results. Most modern deep learning models are based on artificial neural networks, specifically, convolutional neural networks (CNNs). Other deep learning methods include multilayer perceptron, recurrent neural networks (RNNs), modular neural network etc. CNNs have been been most widely and successfully applied in the field of computer vision, such as facial recognition, autonomous vehicles, etc. CNNs are used for image classification and recognition because of its high accuracy. It was proposed by computer scientist Yann LeCun[93] in the late 90s, when he was inspired from the human visual perception of recognizing things. CNNs leverage spatial information, and they are therefore well suited for classifying images.

Traditionally, audio processing tasks such as audio event detection (AED) have been addressed with features such as mel frequency cepstral coefficients (MFCCs), and classifiers based on Gaussian mixture models (GMMs) [94], hidden Markov models (HMMs) [95, 94, 96], non-negative matrix factorization (NMF) [97], or support vector machines (SVMs) [94, 96]. However, more recently, researchers have also started to explore the use of CNN in analyzing audio data [98, 99, 100]. This can be achieved by first convert the one-dimensional (1-D) raw data (i.e., timestamped sampling of audio signals) to the two-dimensional (2-D) features using a technique called spectrogram, which performs Fourier transform on the windowed raw data. As shown in Figure 6.1, the horizontal axis

is timestamps, while the vertical axis is the frequency, both of which are discrete rather than continuous. Each coordinate has a value which represents the intensity of the audio signal at the specific frequency during the specific time interval. Then the 2-D spectrogram features can be transformed to be more suitable for deep learning. This method enables the processing of the audio signal using CNNs by essentially converting it to an image.



(a) Waveform (b) Spectrogram

Figure 6.1: Waveform and spectrogram representation of the single channel audio signal. The horizontal axis of the spectrogram is in units of "windows".

Although it is possible to process audio signals with CNNs, the results may not be nearly as satisfying as they have been for visual images [101]. One major difference between the visual images and the spectrograms generated from audio signals is the composition of different "objects" in the images. Most visual objects are opaque and usually do not overlap with each other in computer vision tasks. This makes classification of individual objects easier due to fewer sources of noises. On the other hand, the spectrograms generated from audio signals can be said to be "transparent", as sounds generated from different objects (likely far from each other in space) can overlap and superimpose in the image, as long as they overlap in the time domain. An immediate implication is that a pixel of certain color in a visual image can often be attributed to a single object. Meanwhile, the color of a single pixel in the spectrograms does not necessarily correspond to the frequency of sound generated by a single object, but can also be produced by multiple accumulated sound sources or even by complex interactions between sound waves such as phase cancellation.

This "transparent" nature of audio spectrograms makes it difficult to separate simultaneous sounds using CNNs.

Another major difficulty arises from from the fact that the two axes in spectrograms carry different meanings, contrary to that in visual images. An object in visual images can be translated in either horizontal or vertical direction, or even rotated, without changing its ontology. While the spectrogram generated by a certain sound can move along the horizontal (time) axis without changing its meaning, the same cannot be said if it moves along the vertical axis. For example, an adult's voice moved upwards in the spectrogram can become very similar to the voice of a child. The translation invariance property in CNNs may become a problem in this case.

There are many other challenges involved in analyzing audios with CNNs such as the more temporal property of sound waves makes them harder to be scanned multiple times for analysis. However, they are not relevant in the context of this study and hence are not further discussed.

6.1.3 Attack Scenario

Although the techniques described in this study can be applied by both system administrators (referred to as defenders) and attackers, the former would have far better access to the side channel information as well as the computational resources. In addition, understanding how much information an attacker can extract from CPSs through audio side channel has rarely been studied before. Hence, this section describes the attack scenario used throughout this chapter.

The basic assumption for the attacker is being stealthy and passive in order to capture as much information as possible without interfering the operation of the CPSs or being detected. One key aspect of the attack is the placement of the sensory devices (e.g., microphones) around the physical systems. One option would be to place miniature embedded sensors directly on or near the devices being monitored, with the apparent benefit of high

signal-to-noise (SNR) ratio. However, this method also has the following downsides:

1. It is difficult to place and setup such devices in hidden locations, which usually requires human intervention on site, e.g., collusion with an insider. The same problem may arise when the espionage operation is terminated and the attackers may want to eliminate physical evidence of such attack.

2. The data captured may be in large amount and few communication channels exist that can transmit the data outside of the target facility. Common methods either require a transceiver near the location of the embedded sensors (e.g., outside the facility) as a relay, or a cellular modem. However, the antenna and communication modules can often be large relative to the size of the sensors themselves. This is against the stealth requirement in the basic assumption.

3. The sensors can almost never have access to reliable power sources to support sufficiently long term operations.

Another viable option for the attacker to achieve the goal under the aforementioned constraints is to leverage the already widely used smart phone devices. They are usually equipped with high quality microphones, with more phones nowadays having multiple microphones (or microphone arrays) for better calling quality. They are constantly upgraded with state of the art communication protocols (e.g., Wi-Fi or 4G LTE) that support high speed data transfer, and increasingly larger local storage to cache data even when network is temporarily unavailable. Meanwhile, power sources are almost never an issue for smart phones. The attack can even be carried out with minimal human intervention due to the fact that most people carry their smart phones with them throughout the day, therefore the smart phone of any legitimate personnel near the target CPSs can be a potential target to exploit through the apps installed on them. The variety of sensors on board a smart phone also provides even more information for the attacker to collect data with rich context. For example, by collecting geo-location, accelerometer, gyroscope and magnetic sensors' data,

the phone's location and orientation can be accurately determined. Such information can be combined with the audio signals collected by the microphone arrays to locate the source of the sound from individual devices, and even map the devices in the 3D space.

6.2 Audio Side Channel in Electric Motors

To understand how the sound generated by a device can be used to infer the knowledge of the operation status of the device, it can be useful to understand how such sound is generated when the device is operating. In this section, electric motors are used as an example of such device due to their prevalence in the CPSs, joining the cyber domain with physical domain through electromagnetic forces.

Electric motors generally have three noise sources [102, 103, 104]:

1. airflow (e.g., ventilating fans);
2. electromagnetic sources;
3. mechanical (e.g., bearings).

The noise from ventilating system commonly comes from the airflow produced by the fans, which can generate several frequencies of noises due to siren type effects associated with the blades. Frequency analysis of such noise shows that it has a broad band spectrum [103]. The discrete spectrum noise is created from turbulent airflow near fan blade and the airway entrance, as well as the tip of the blades, while the continuous spectrum noise arises from the siren effect. The airflow noise can become significant when the rotation speed of the motor is high.

The magnetic noise is generated due to the rotating magnetic field. Resonance can occur when one of the various exciting frequencies in induction motors coincides with the natural frequency of either the stator teeth or core, with the double-line frequency (e.g., 120Hz in United States) being always present. The noise is generated due to the vibration

of motor body or other parts of the machine under the electromagnetic forces (EMF). As a result, the frequency of the noise is proportional to the frequency of the supply power, which is in turn proportional to the rotation speed of the motor [103].

Rolling element bearings, ball or sleeve bearings which are worn or overheated can all produce noises when the motor is operating. The imbalance, eccentricity, misalignment and tolerance of motor structure during manufacturing or usage can all contribute to the mechanical noises [104].

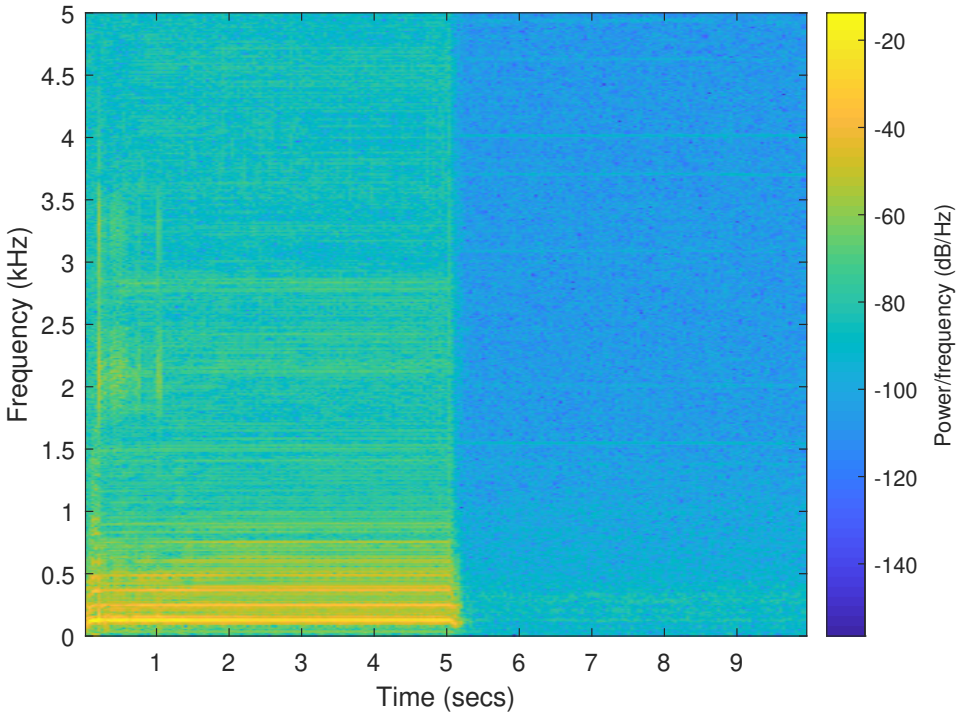
The exact contribution of each noise source varies depending on the type of motors. For example, the electromagnetic circuit in 6- and greater pole motors are usually more significant than that in 2- and 4-pole motors [102]. It also depends upon the design of the motors. For example, motors which use water cooling instead of air ventilation, and sleeve bearings to minimize noise can attribute most of its noises to magnetic noise.

It is worth noting that although the noise of electric motor can be reduced by enclosing it in sound absorbent materials, this is usually not done in practice due to the stronger thermal dissipation requirement than the need to reduce noise in CPS environment.

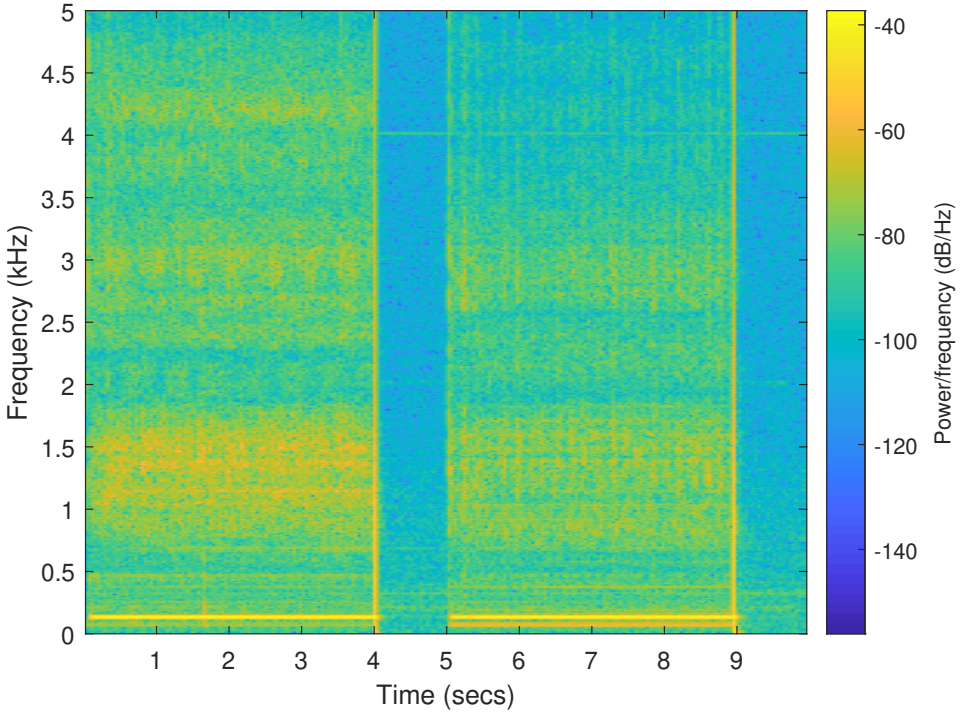
6.3 Pilot Study: Water Loop Testbed

During the study, a water loop is used to setup the experiment to perform the pilot study, consisting of a pump and a binary valve individually controlled by a PLC. Mono-channel directional microphone is used to collect the audio data from the process when the actuators are operating. Figure 6.2 shows the spectrogram of the audio signals under three different sets of operations: pump on/off, valve close/open, and valve close/open when pump is turned on. Note that the pump is set to maintain a certain water pressure. Therefore when the pump is left turned on and valve shuts off after the outlet of the pump, the pressure in the pipe will build up and eventually reach the preset value, causing the pump to stop operating.

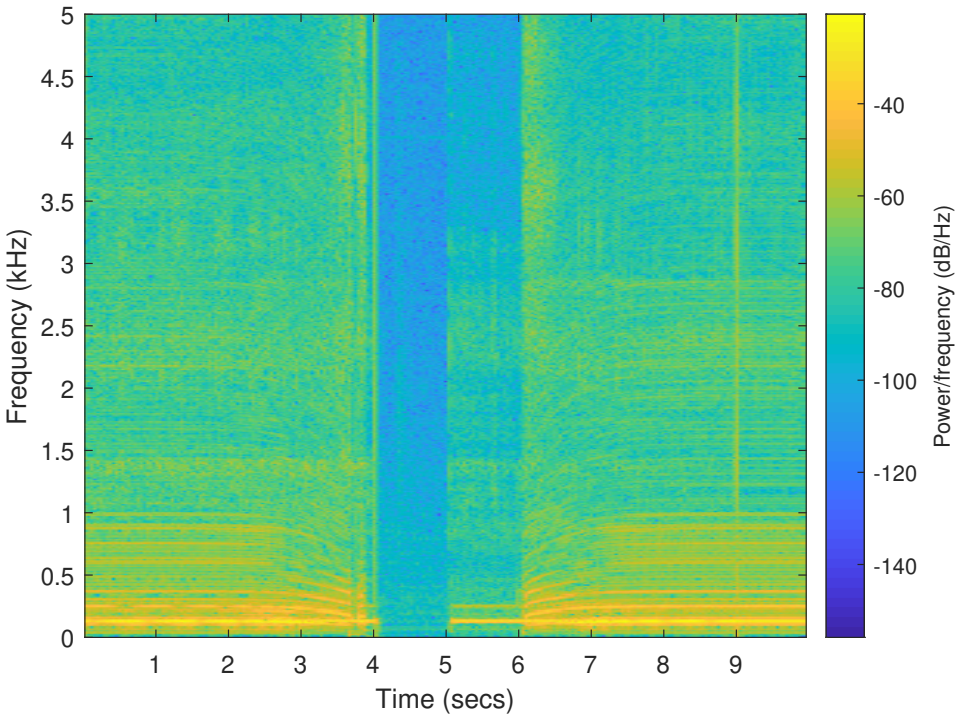
In each case, the two opposite commands are given to the actuators, and it can be seen



(a) Pump on/off.



(b) Valve close/open.



(c) Valve and pump.

Figure 6.2: Spectrogram of the audio collected from an operating water loop system.

that stable frequencies are generated during each command execution. Specifically, Figure 6.2c shows that even when multiple devices are operating simultaneously, their individual operation status are still recoverable given the fingerprints of their standalone operation. Because the pump automatically shuts off when reaching a preset pressure, i.e., when the valve is about to close and when it just opens, the pump generates a gradually decreasing and gradually increasing chirp sound, respectively. Such phenomenon can be used to determine even the detailed pressure value inside the water loop.

6.4 Case Study: Water Treatment Testbed

In order to make the experiment as close to the industrial environment as possible, a testbed called Secure Water Treatment (SWaT) at Singapore University of Technology and Design is leveraged as the target CPS. SWaT consists of a modern six-stage process [105]. The process begins by taking in raw water, adding necessary chemicals to it, filtering it via an Ultrafiltration (UF) system, de-chlorinating it using UV lamps, and then feeding it to a Reverse Osmosis (RO) system. A backwash process cleans the membranes in UF using the water produced by RO. The cyber portion of SWaT consists of a layered communications network, Programmable Logic Controllers (PLCs), Human Machine Interfaces (HMIs), Supervisory Control and Data Acquisition (SCADA) workstation, and a Historian. Data from sensors is available to the SCADA system and recorded by the Historian for subsequent analysis. The engineering schematics of SWaT can be seen from its human-machine interface (HMI) in Figure C.1 in Appendix C. The physical dimensions of the testbed is shown in Figure C.2 in Appendix C.

The sound recording device used in the experiment are two AKG P170 small-diaphragm condenser microphones. They are placed on a tripod fixed with the adapters to ensure constant relative angle and distance between the two microphones. This setup is a generic representation for most smart phone's microphone array setup.

There are three variables to be adjusted in the experiment, namely 1) the **angle** between

the microphone's direction and the sound-generating device; 2) the **horizontal-level** difference between the microphone and the device; and 3) the **distance** between the microphone and the device. Each variable is adjusted as described below.

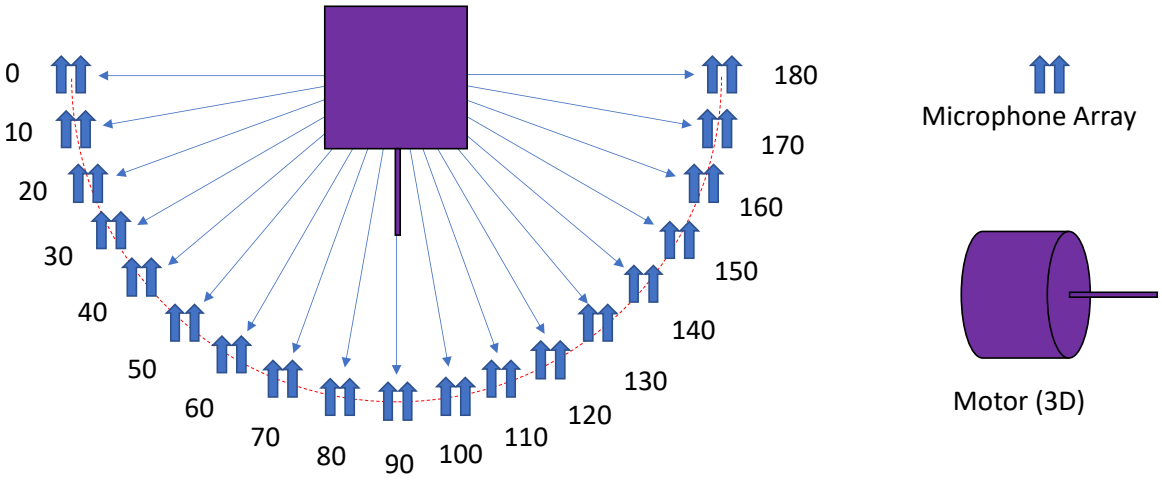


Figure 6.3: Top view of the microphone array placement at different angle along the arc with device at the center.

Angle. The sound-generating device is placed at different angles to the direction of the microphone(s). The audio is recorded from 0 to 180 degrees, increasing at 10 degrees. The other two variables are maintained during the angle change, resulting in the microphone(s) being moved along an arc with the device at the center as shown in Figure 6.3.

Horizontal Levels. The microphone array is placed on different horizontal levels relative to the sound-generating device. Starting at the same level, the microphone is moved upwards and downwards respectively, $10cm$ at a time for up to $20cm$. This results in a matrix of measurement points combined with the varied angles as shown in Figure 6.4.

Distance. The distance between the microphone array and the sound-generating device is first set at a minimum distance, in this case $50cm$. Then the distance is multiplied integer times (e.g., $1\times$, $2\times$, $3\times$ ...). This results in a tensor of measurement points in the space as shown in Figure 6.5.

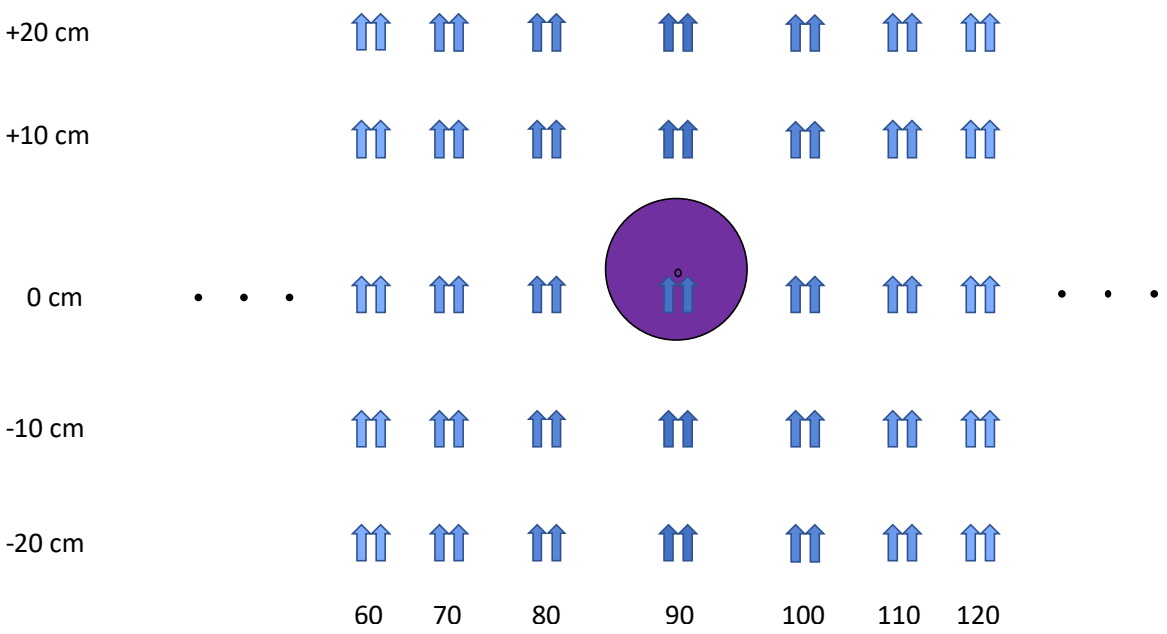


Figure 6.4: Front view of the microphone array placement matrix showing different angle and horizontal levels relative to the device.

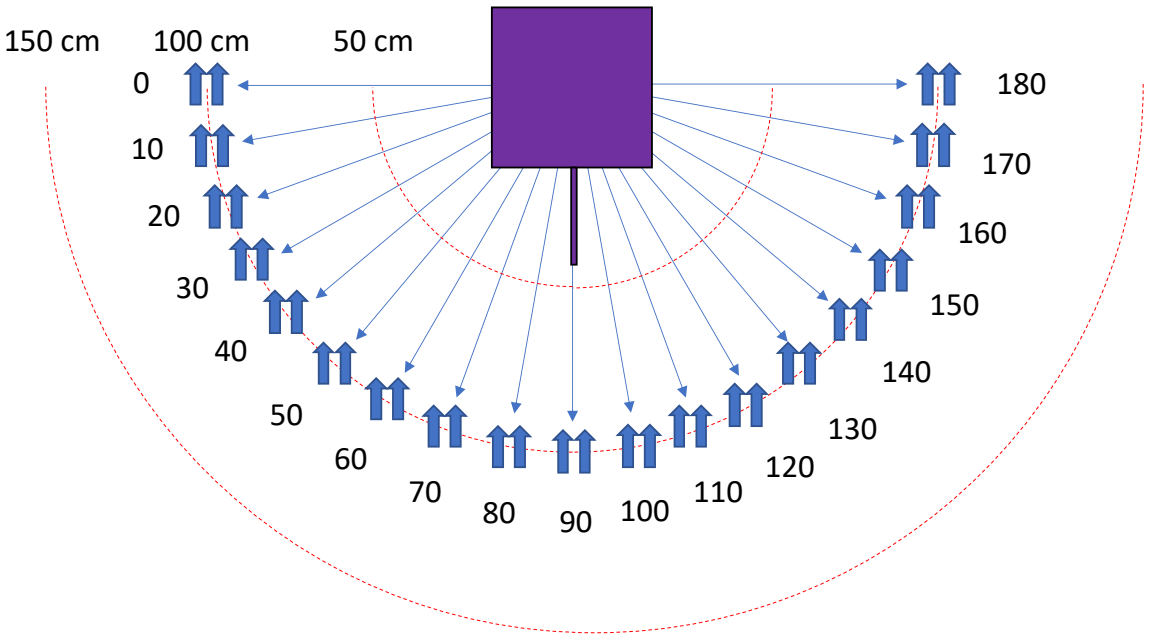


Figure 6.5: Top view of the microphone array placement at different distance to the device.

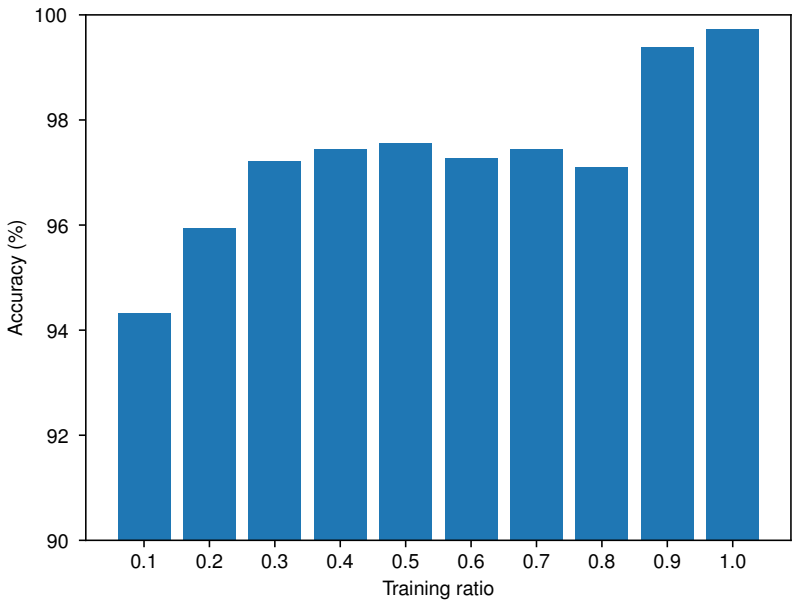


Figure 6.6: CNN's prediction accuracy of the devices' operation status in stage 1 of SWaT under different training size ratio.

6.5 Results

The audio collected from the first stage of the SWaT testbed is used as the dataset for study. The dataset is labeled with the operation status of each device involved every second (i.e., pump on/off). It is split into training and testing dataset using $10 - fold$ cross-validation to ensure the result is generalized. The CNN is implemented using PyTorch framework [106].

The spectrogram is pre-processed to be suitable for input to the CNN. First, it is clipped to upper frequency of 800 Hz, as a balance between the information preserved and the computational complexity. The number is derived from the observation in the pilot study in Section 6.3, due to negligible sound power present in the spectrogram above this frequency. Then the spectrogram is segmented into 200 windows per frame. The primary rationale for segmenting the spectrogram is to normalize the input data to the CNN, while maintaining a proper level of computational complexity. Finally, the 800×200 matrix is flattened to a 1-D feature vector due to efficiency in the computation.

The CNN is designed with three layers, two of which are the hidden layer. Each layer uses a max pooling and downsamples the features, eventually into $1,568$ neurons that are mapped to the number of classes, which can be the number of states the device being analyzed can be in, or the types of devices. The cross entropy loss function is used with stochastic gradient descent (SGD) optimizer.

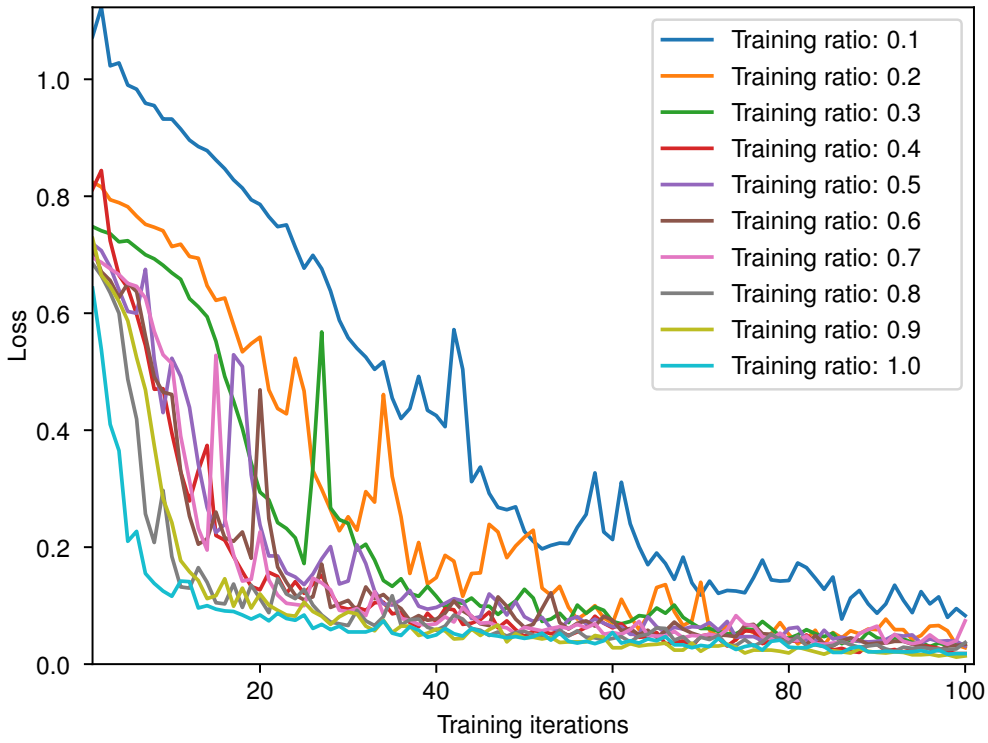


Figure 6.7: CNN's loss versus number of training iterations.

As shown in Figure 6.7, the accuracy of CNN's prediction of the devices' operation status is high even when few training data is used. As the training size increases, the accuracy approaches 100%. The output loss of the CNN decreases rapidly and stabilizes within 100 training iterations under most training ratio, as shown in Figure 6.7.

6.6 Discussion

As shown in Section 6.5, the operation status of the individual devices in the CPSs can be accurately recovered using audio side channel information collected by the microphones. In a real attack, the attacker can leverage the smart phone near the physical process inside the target facility, then record the sound for either an online analysis (running on the phone) or offline analysis (uploading to the attacker's remote server) using CNN. With the dual (or even triple) microphone array becoming available in more recent smart phones, the direction of the source of these sounds can be identified using an effect called the Binaural hearing, which is based on the difference of the audio's timing, loudness, phase and frequency response received at two (or more) receivers (e.g., ears, microphones). Such information can be further combined with the rich location and microelectromechanical systems (MEMS) attitude sensors onboard the smart phones, such as accelerometer, gyroscope and magnetometer (also known as digital compass), to reconstruct a map of the audio sources (i.e., CPS devices) in the space. The resulting information is an essential artifact to the reverse engineering of the process control information, which can often be confidential and/or trade secrets.

6.7 Conclusion

In this chapter, audio side-channel is used to infer the process parameter information (i.e., devices' operation status) in CPSs. The methodology is experimentally demonstrated using a realistic water treatment testbed, which shows high accuracy in the results predicted by the CNN.

CHAPTER 7

CONCLUSION

The traditional IT-based techniques are problematic in practice due to their lack of consideration for securing the physical aspects of the CPSs. In this research, novel techniques are studied which tightly integrate with the physical domain of the CPSs to better secure them. This research is stratified into three layers. Starting from the bottom layer, the individual devices are fingerprinted based on their operation time determined by the physical models and configurations. In the second layer, the process structural information is identified using both static and dynamic analysis of the process control programs. Finally, on the top layer, the process parameters are identified using the physical side-channel information (e.g., sound). Overall, the methodologies used in this research provide an insight for the CPS security research community to leverage the physical nature of the CPSs in order to better secure them.

Appendices

APPENDIX A

JTAG

The JTAG interface, named after the Joint Test Action Group which codified it, is an industry standard for debugging, verifying designs and testing PCBs after manufacture. Several studies have leveraged the JTAG as an attack vector to the PLC[61, 70] by reading from and writing to the flash memory containing the firmware. We discovered that the JTAG port can also be used to control the program execution using its debugging capability.

The JTAG port can often be found by examining the PCB with the CPU inside the PLC. The pin configuration can then be identified using deductive reasoning[107], or commercial off-the-shelf (COTS) tools such as the JTAGulator[108]. For example, the JTAG port of the Schneider M241 PLC used in our study can be seen in Figure A.1, with each JTAG pin identified and labeled. Then a debugger is attached to the JTAG port on the PLC, providing an interface between the host PC and target device. We used a COTS product called the J-Link Pro from Segger[109] as the debugger. Alternative solutions such as the Open On-Chip (OpenOCD) debugger are also available. Figure A.2 shows that we were able to step through individual instructions and control the program execution. We were also able to set the PC to arbitrary value, as well as access and modify the registers and RAM of the PLC as the program executes. This allows us to interact with the PLC program, perform fuzzing and build the automaton of the program.

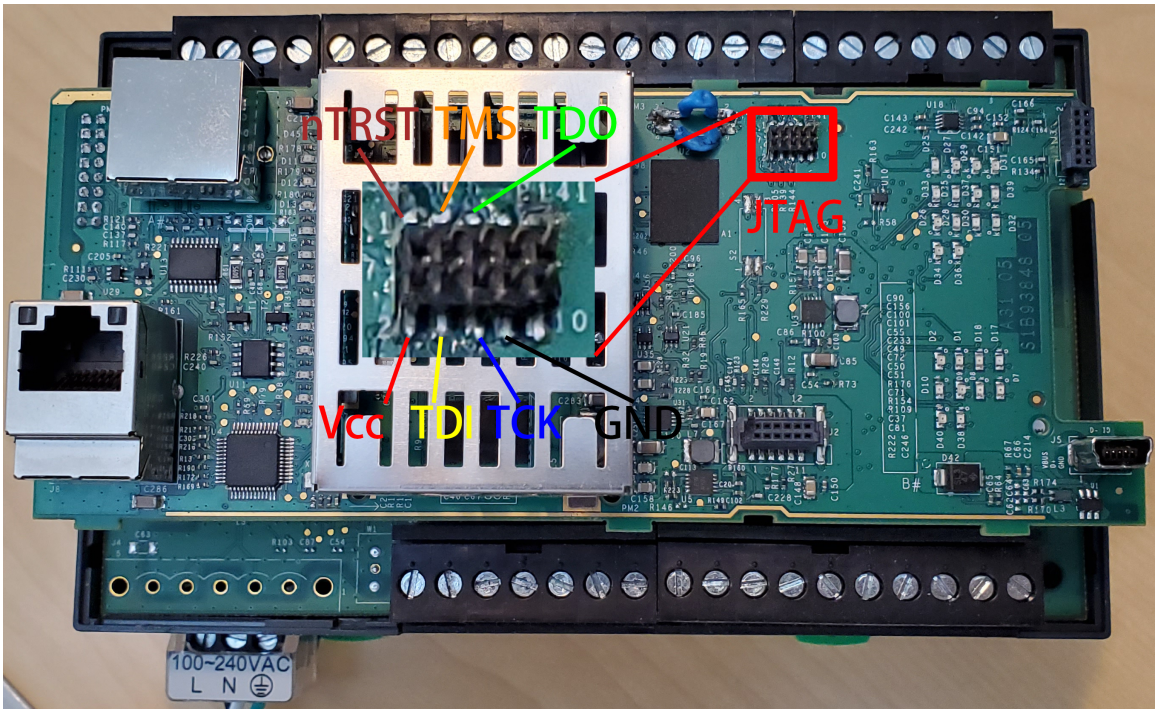


Figure A.1: PCB of Schneider M241 PLC with JTAG debugging port

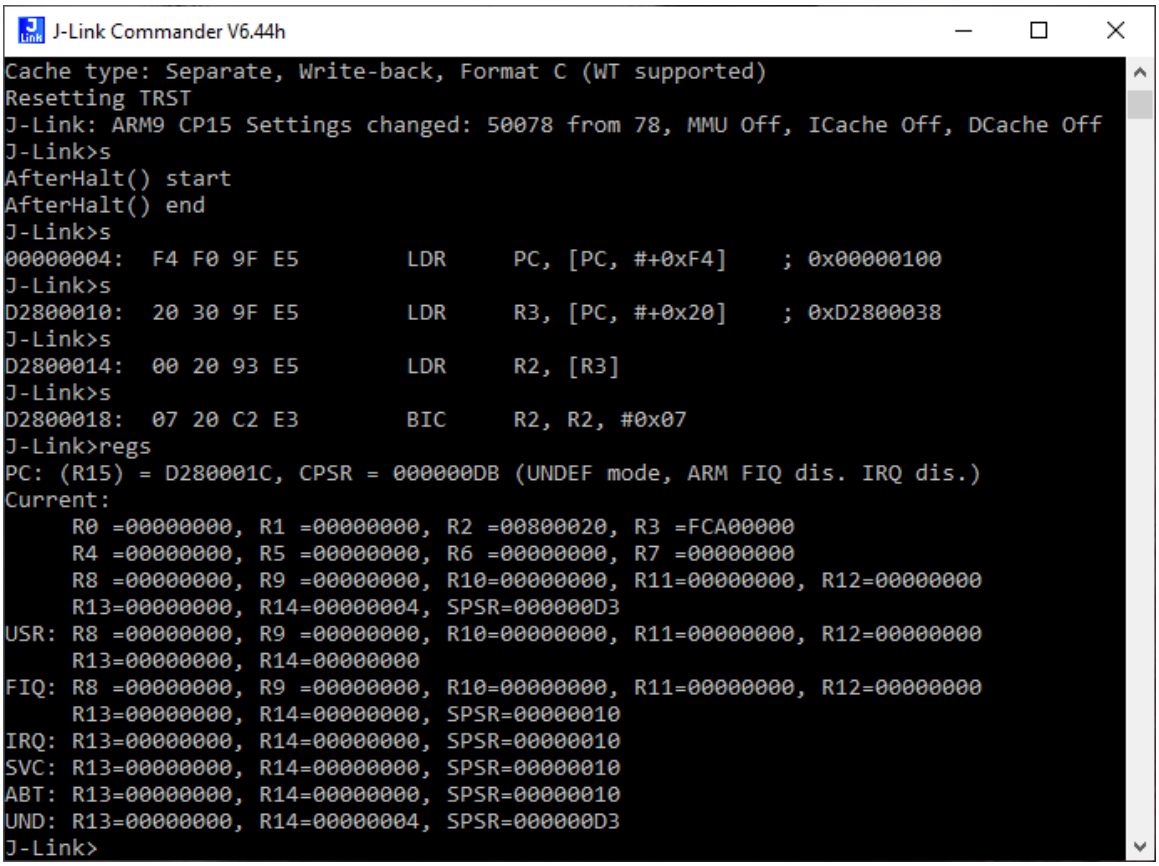


Figure A.2: Using J-Link debugger to step through the program and accessing the registers and memory of the PLC

```
PROGRAM PLC_PRG
VAR
   LevelHigh AT %IX1.3 : BOOL;
   LevelEmpty AT %IX1.0 : BOOL;
   InValve2 AT %QX0.1 : BOOL;
   NOT3_OUT : BOOL;
   AND34_OUT : BOOL;
   SR3 : SR;
   ...
END_VAR

NOT3_OUT := NOT(LevelEmpty);
SR3(SET1 := LevelHigh, RESET := NOT3_OUT);
InValve2 := AND34_OUT;
...
```

(a) ST code

```
;code section
;NOT3_OUT := NOT(LevelEmpty);
00001FA8  bcb49fe5  ldr fp, [pc, #0x4bc]
00001FAC  0040dbe5  ldrb r4, [fp]
00001FB0  014004e2  and r4, r4, #1
00001FB4  014024e2  eor r4, r4, #1
00001FB8  a8b49fe5  ldr fp, [pc, #0x4a8]
00001FBC  0040cbe5  strb r4, [fp]
;SR3(SET1 := LevelHigh, RESET := NOT3_OUT);
00001FC0  04d04de2  sub sp, sp, #4
00001FC4  a0b49fe5  ldr fp, [pc, #0x4a0]
00001FC8  0040dbe5  ldrb r4, [fp]
00001FCC  084004e2  and r4, r4, #8
00001FD0  a441a0e1  lsr r4, r4, #3
00001FD4  88b49fe5  ldr fp, [pc, #0x488]
00001FD8  0040cbe5  strb r4, [fp]
00001FDC  7440dbe5  ldrb r4, [fp, #0x74]
00001FE0  0140cbe5  strb r4, [fp, #1]
00001FE4  74549fe5  ldr r5, [pc, #0x474]
00001FE8  05408be0  add r4, fp, r5
00001FEC  00408de5  str r4, [sp]
00001FF0  64b49fe5  ldr fp, [pc, #0x464]
00001FF4  00409be5  ldr r4, [fp]
00001FF8  0fe0a0e1  mov lr, pc
00001FFC  04f0a0e1  mov pc, r4
00002000  04d08de2  add sp, sp, #4
...
;InValve2 := AND34_OUT;
00002348  b4b09fe5  ldr fp, [pc, #0xb4]
0000234C  0040dbe5  ldrb r4, [fp]
00002350  b0b09fe5  ldr fp, [pc, #0xb0]
00002354  0050dbe5  ldrb r5, [fp]
00002358  000054e3  cmp r4, #0
0000235C  0100000a  beq #0x2368
00002360  025085e3  orr r5, r5, #2
00002364  000000ea  b #0x236c
00002368  0250c5e3  bic r5, r5, #2
0000236C  94b09fe5  ldr fp, [pc, #0x94]
00002370  0050cbe5  strb r5, [fp]
...
;data section
00002404  0x00000163  ;AND34_OUT
00002408  0x00000000  ;%QX0
...
00002468  0x00000158  ;NOT3_OUT
0000246C  0x00000001  ;%IX1
```

(b) Disassembly

Figure A.3: Sample program to demonstrate the input, output and internal variable representations.

APPENDIX B

STATE DEFINITION OF THE STANDARD FUNCTION BLOCKS

The state of each function block (FB) is defined such that the output of the FB is dependent and only dependent on the input to the FB and its state. The definition is shown in Table B.1.

B.1 Realistic Physical Systems

In this section, we describe the settings and requirements of each scenario containing physical systems controlled by PLCs.

B.1.1 Tank Balancer

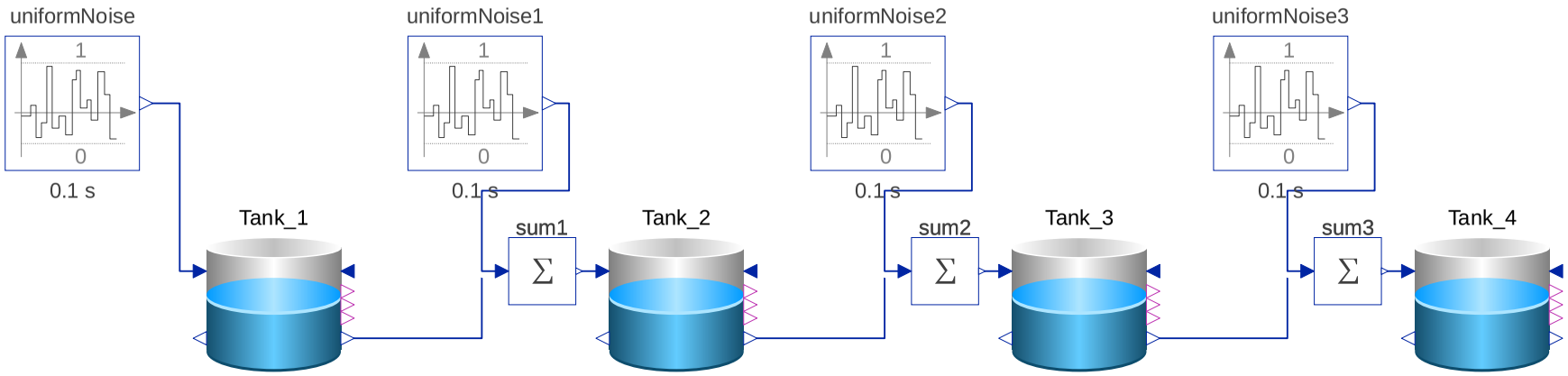
Description. The tank balancer process is a system consisting of four tanks, as shown in Figure B.1a. Each tank has one water inlet and one water outlet, as well as three water level sensors. The outlet of Tank 1, 2 and 3 are sequentially connected to the next tank's inlet. Water flowing out of tank 4's outlet is discarded. Additionally, each tank's inlet is also connected to an independent water source, which provides random water flow into the corresponding tank. The water sources are not controlled by the PLC. Three water level sensors on each tank are activated when the water level in the tank is above certain thresholds (i.e., low, medium, high).

Requirements. Write a program which takes the 12 sensor readings as inputs, and outputs control signals to the actuators of the 4 tanks' outlet valves. The goal is to maintain the water level in each tank between "low" and "high" levels exclusively by adjusting the water flow among the tanks.

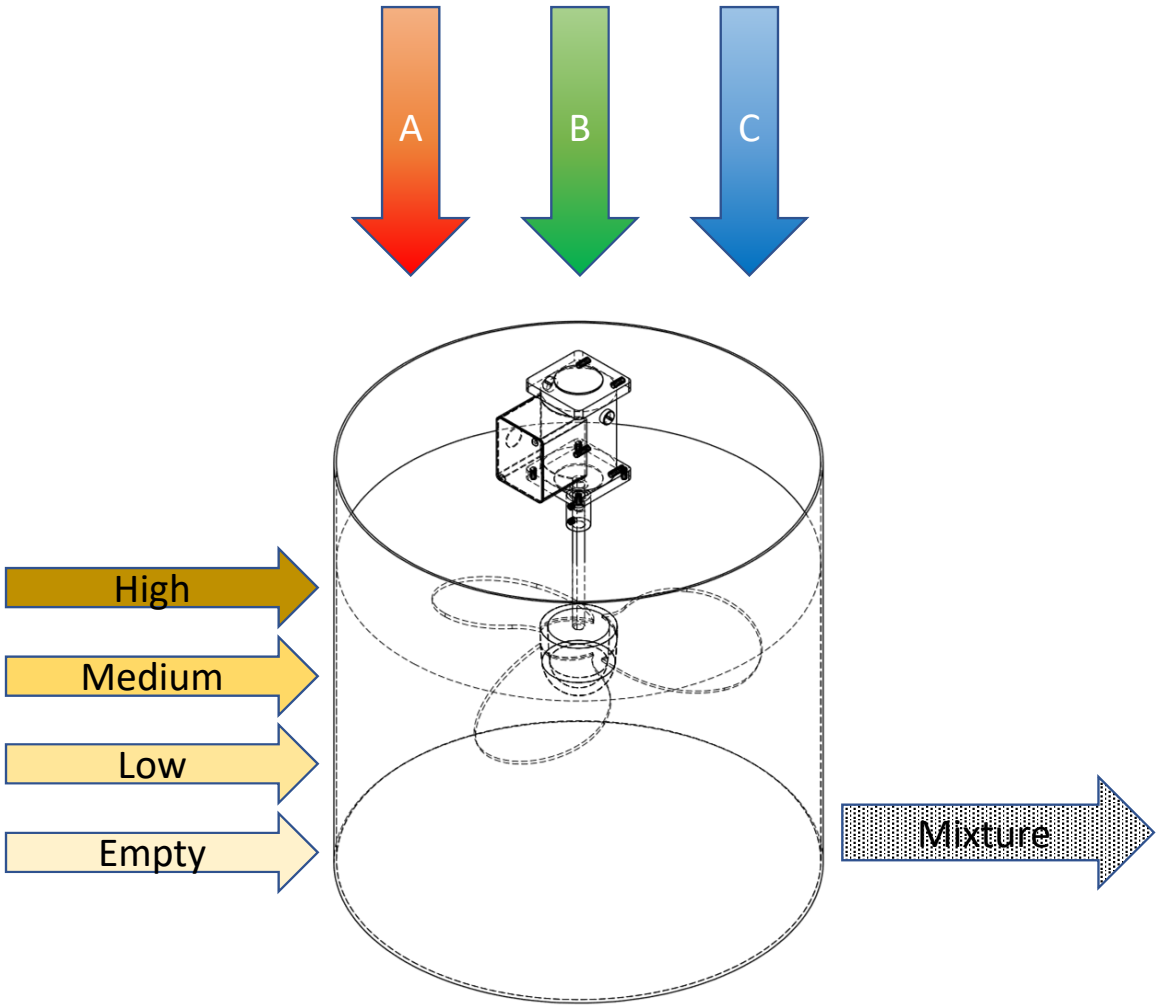
Attack. In the attack version, the goal of the program is to remain *stealth* during the beginning "test run" of the program and only start the sabotage afterwards. Between 60 and

Table B.1: State definition of the standard function blocks

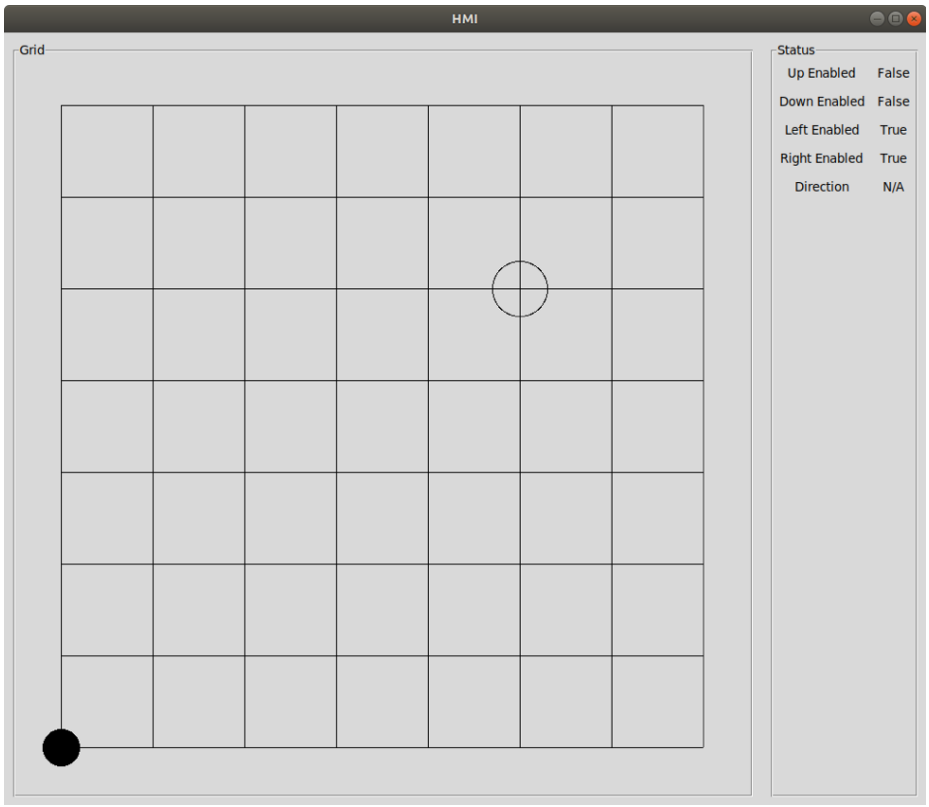
Name	State	Function
R_TRIG	last *CLK*	$Q = \neg state \wedge CLK$
F_TRIG	last *CLK*	$Q = state \wedge \neg CLK$
SR	last *Q*	$Q = S1 \vee state \wedge \neg R$
RS	last *Q*	$Q = (S1 \vee state) \wedge \neg R$
TP	(last *IN*, last *Q*)	$Q = \neg state[0] \wedge IN$
TON	(last *IN*, last *Q*)	$Q = \begin{cases} True & \text{if } IN \wedge state[0] \\ False & \text{if } \neg IN \\ \text{unchanged} & \text{otherwise} \end{cases}$
TOF	(last *IN*, last *Q*)	$Q = \begin{cases} True & \text{if } IN \\ False & \text{if } \neg IN \wedge \neg state[0] \\ \text{unchanged} & \text{otherwise} \end{cases}$
CTU	last Q	$Q = \begin{cases} False & \text{if } R \\ True & \text{if } \neg state[0] \wedge CU \\ \text{unchanged} & \text{otherwise} \end{cases}$
CTD	last Q	$Q = \begin{cases} False & \text{if } R \\ True & \text{if } \neg state[0] \wedge CD \\ \text{unchanged} & \text{otherwise} \end{cases}$



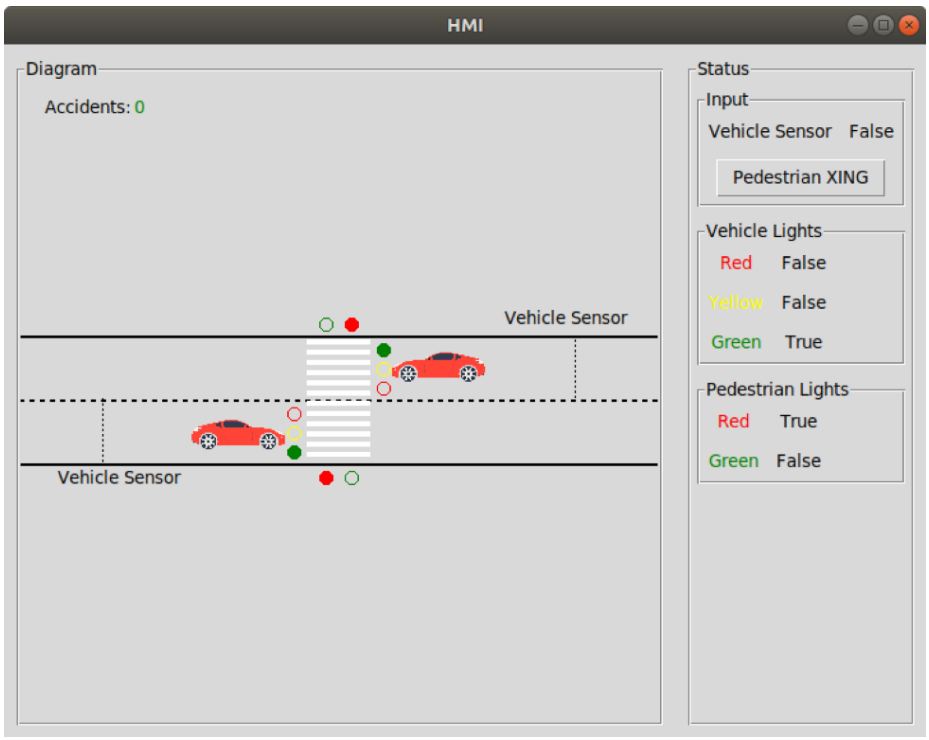
(a) Tank Balancer.



(b) Stirring System.



(c) Robot Path.



(d) Traffic Light.

Figure B.1: Scenarios with different physical systems controlled by PLCs

90 seconds, **only tank 2** will start to overflow. All other tanks should remain the originally benign behavior.

B.1.2 Stirring System

Description. The mixing and stirring system is a single tank with three inlets for different materials (A, B, and C, respectively), a mixing blade for stirring the mixture, and an outlet for exhausting the mixture, as shown in Figure B.1b. This tank also has four level sensors, namely the empty, low, medium, and high sensors, installed at different heights inside the tank. Each sensor will be activated when the mixture level is above it.

Requirements. Write a program which takes the four level sensors as inputs, and controls the valves for the three inlets, the mixing blade, and the valve for the outlet. It should follow this specific sequence in order to properly mix the materials:

1. The tank starts in an empty state. All inlet and outlet valves are closed, and mixing blade is turned off.

2. Only open the valves for inlet 1 to add A into the tank, until the "low" level sensor is activated. Turn off the valve for inlet 1.

3. Repeat step 2 for inlet 2 and 3, with "medium" and "high" level sensors being used as the stopping point.

4. Turn on the mixing blade. After the mixing blade has run for 5 seconds, open the outlet valve to drain the mixture and keep the mixing blade running.

5. As soon as all mixture has been drained, turn off the mixing blade. Wait for 3 seconds before closing the outlet valve.

6. Start over from step 1.

Variant. A variant of the Stirring System program was collected with slight modification to the requirements. In the original version (denoted as *StirringSystem_v1*, the valves

of inlet 1, 2 and 3 are opened sequentially. However, in *StirringSystem_v2*, inlet 2 stays open during the entire filling process. The purpose of the variant is to examine how well the classifier can resist the noise introduced by a benign change in the program behavior.

Attack. In the attack version, the goal of the program is to remain *stealthy* during the beginning "test run" of the malicious program and only start the sabotage afterwards. After looping through the process three times, it will invert the behavior of the mixing blade. In other words, the mixing blade needs to be turned off in step 5 and on during the other time.

B.1.3 Robot Path

Description. The robot path process models a robot (shown as a black solid circle in Figure B.1c) looking for its path on an 8×8 grid. The robot can only travel along the wires and stop at the intersections. It starts at the lowest and leftmost position in the grid, denoted as $(0, 0)$. The horizontal direction is the x-axis, and the vertical direction is the y-axis. The maximum coordinates in this grid is $(7, 7)$. The hollow circle is the target position.

Requirements. Write a program that takes the current position and the target position of the robot, and controls the direction of the robot so that it will arrive at the target. The robot can take only one direction at a time. Any viable path can be chosen, as long as it always gets closer to the target after each move. The position and target coordinates are encoded in the binary form, mapped to the sensors in the little-endian format.

B.1.4 Traffic Light

Description. The traffic light process models a traffic light system at a pedestrian crossing, as shown in Figure B.1d. There are vehicles traveling easterly and westerly on the road, and pedestrians who wish to cross the road. A red/yellow/green traffic light is used for the vehicles. A red/green traffic light is used for the pedestrians. Normally, red light is displayed to the pedestrians and green light is displayed to the vehicles.

Requirements. A vehicle congestion sensor is used to detect the amount of traffic. If

the traffic becomes congested, the sensor will be activated, and vice versa. A button is available for the pedestrian to indicate that they request to cross the road. The request shall be granted immediately if the vehicle congestion sensor is not active (i.e., no traffic jam). Write a program that executes the following sequence:

1. Switch off the vehicle green light and switch on the vehicle yellow light for 3 seconds.
2. After the yellow light is on for 3 seconds, switch vehicle yellow light off, vehicle red light on, pedestrian red light off, pedestrian green light on.
3. Let the pedestrian green light be on for 15 seconds. Then switch pedestrian green light off, pedestrian red light on.
4. After 1 second, switch vehicle red light off, vehicle green light on.

If there is a traffic jam (vehicle sensor activated) while the pedestrian's button is pressed, the above sequence shall not be executed until the traffic jam is cleared. If the button is pressed again before the above sequence finishes, it should be ignored.

APPENDIX C

SECURE WATER TREATMENT (SWAT)

This appendix includes relevant figures of the SWaT testbed at Singapore University of Technology and Design, used to collect audio side channel data.

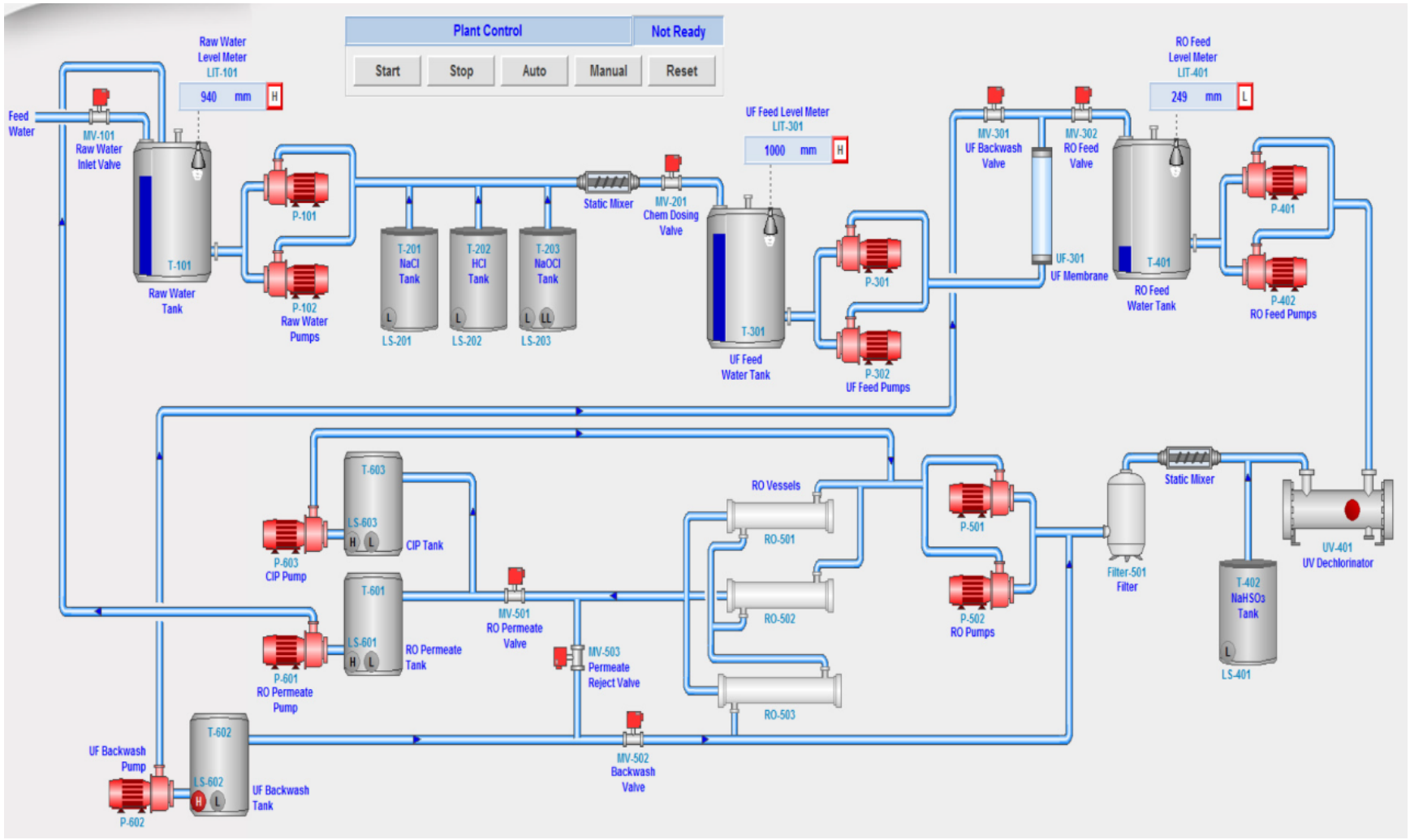


Figure C.1: Human-machine interface (HMI) of the SWaT testbed showing its engineering schematics.

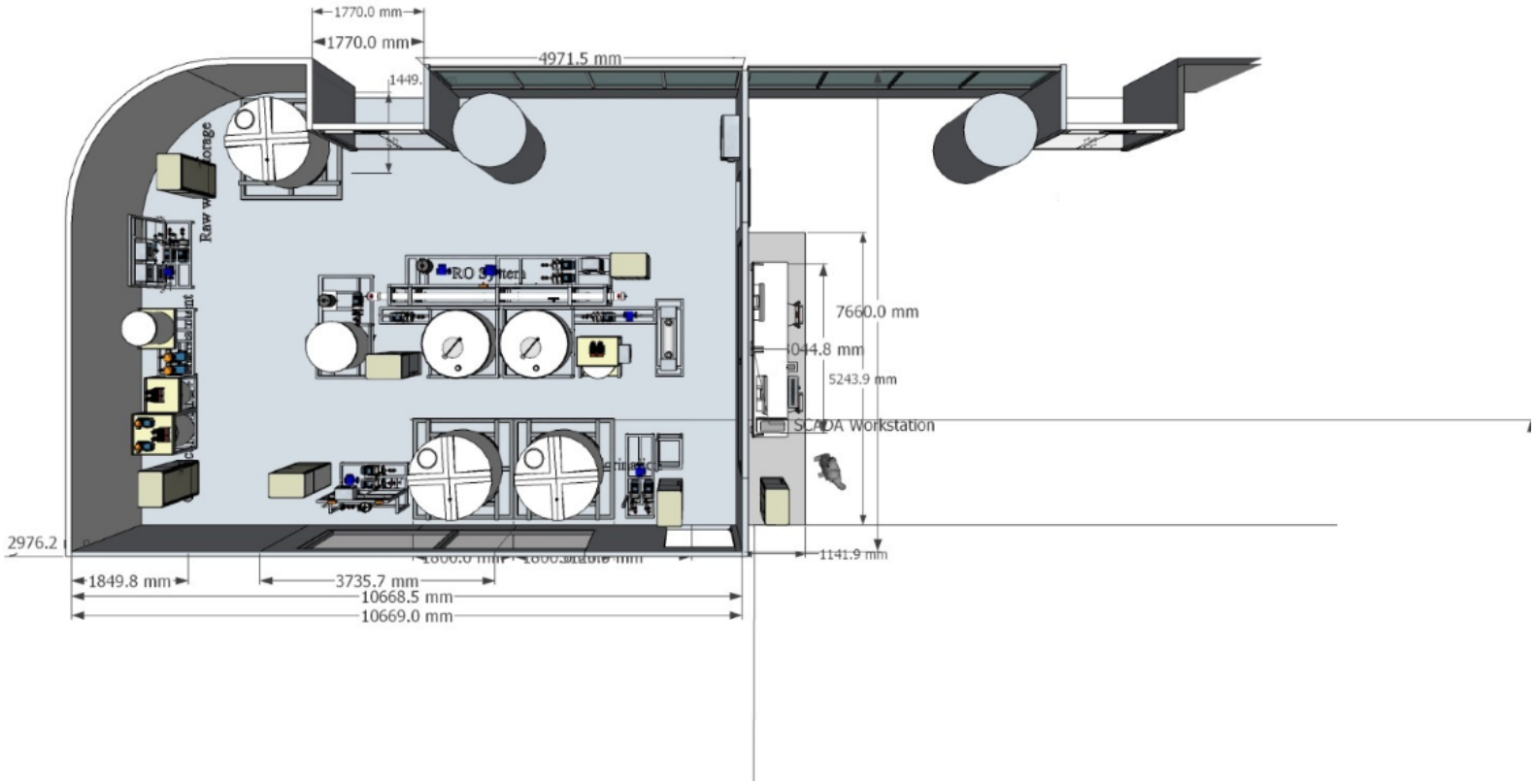


Figure C.2: Floor plan of the SWaT testbed showing its physical dimensions.

REFERENCES

[1] *Why 2017 will finally be the year of the smart home: Consumers figure it out*, 2018.

[2] *Global Industrial Controls System Market to Grow at CAGR of 4.9% from 2015 to 2021*, 2018.

[3] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet Dossier," *Symantec-Security Response*, no. February 2011, pp. 1–69, 2011.

[4] J. Slay and M. Miller, *Lessons learned from the Marchoory Water Breach*. Springer, 2008, pp. 73–82, ISBN: 978-0-387-75462-8.

[5] *Crash Override Malware Took Down Ukraine's Power Grid Last December | WIRED*, 2018.

[6] D. Albright and F. Pabian, "It Fits! Qom Site Layout," Tech. Rep., 2018.

[7] *Ukraine's power outage was a cyber attack: Ukrenergo*, 2018.

[8] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: Risk Assessment, Detection, and Response," in *ASIACCS*, New York, New York, USA: ACM Press, 2011, p. 355, ISBN: 9781450305648.

[9] C. McParland, S. Peisert, and A. Scaglione, "Monitoring Security of Networked Control Systems: It's the Physics," *IEEE Security & Privacy*, vol. 12, no. 6, pp. 32–39, Nov. 2014.

[10] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the Impact of Stealthy Attacks on Industrial Control Systems," in *CCS*, New York, New York, USA: ACM Press, 2016, pp. 1092–1105, ISBN: 9781450341394.

[11] A. A. Cárdenas, S. Amin, and S. Sastry, "Research challenges for the security of control systems," *Proceeding HOTSEC'08 Proceedings of the 3rd conference on Hot topics in securit*, p. 6, 2008.

[12] A. A. Cardenas, S. Amin, and S. Sastry, "Secure Control: Towards Survivable Cyber-Physical Systems," in *2008 The 28th International Conference on Distributed Computing Systems Workshops*, IEEE, Jun. 2008, pp. 495–500.

[13] D. Formby, P. Srinivasan, A. Leonard, J. Rogers, and R. Beyah, "Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems," in *NDSS*, San Diego, CA, USA: Internet Society, 2016.

[14] *Standard function blocks*, `https://www.fernhillsoftware.com/help/iec-61131/common-elements/standard-function-blocks/index.html`, Accessed: 2019-11-07.

[15] C. Neilson, "Securing a Control Systems Network," *ASHRAE*, 2013.

[16] W. He, M. Golla, R.-u. Bochum, R. Padhi, J. Ofek, M. Dürmuth, W. He, M. Golla, R. Padhi, J. Ofek, and D Markus, "Rethinking Access Control and Authentication for the Home Internet of Things (IoT)," *Proceedings of the 27th USENIX Conference on Security Symposium*, pp. 255–272, 2018.

[17] R. Schuster, V. Shmatikov, and E. Tromer, "Situational Access Control in the Internet of Things," *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*, pp. 1056–1073, 2018.

[18] A. A. Cárdenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, and S. Sastry, "Challenges for Securing Cyber Physical Systems," in *Workshop on Future Directions in Cyber-physical Systems Security*, DHS, 2009.

[19] Q. Gu, D. Formby, S. Ji, H. C. Cam, and R. Beyah, "Fingerprinting for Cyber Physical System Security: Device Physics Matters Too," *IEEE Security & Privacy*, 2018.

[20] J. Cordy, T. Dean, A. Malton, and K. Schneider, "Source transformation in software engineering using the TXL transformation system," *Information and Software Technology*, vol. 44, no. 13, pp. 827–837, 2002.

[21] *The C++ Front End*, 2006.

[22] C. Fischer and R. Leblanc. Ben- jamin/Cummings Publishing Company, 1988.

[23] R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and K. Zadeck, "Efficiently computing static single assignment form and the control dependence graph," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 13, no. 4, pp. 451–490, 2002.

[24] R. Gupta, M. Harrold, and M. Soffa, "An Approach to Regression Testing using Slicing," *Proceedings Conference on Software Maintenance*, 1992.

[25] D. Weise, R. Crew, M. Ernst, and B. Steensgaard, "Value dependence graphs: representation without taxation," *Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 297–310, 1994.

[26] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "SmartAuth: User-Centered Authorization for the Internet of Things," *Usenix*, pp. 361–378, 2017.

[27] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2014, pp. 55–60. arXiv: `arXiv:1011.1669v3`.

[28] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and Logging in the Internet of Things," *Proceedings 2018 Network and Distributed System Security Symposium*, no. February, 2018.

[29] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A Systematic Survey of Program Comprehension through Dynamic Analysis," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 684–702, 2009.

[30] A. W. Biermann, "On the inference of Turing machines from sample computations," *Artificial Intelligence*, vol. 3, no. C, pp. 181–198, 1972.

[31] M. F. Kleyn and P. C. Gingrich, "GraphTrace—understanding object-oriented systems using concurrently animated views," *ACM SIGPLAN Notices*, vol. 23, no. 11, pp. 191–205, 1988.

[32] W. De Pauw, R. Helm, D. Kimelman, and J. Vlissides, "Visualizing the behavior of object-oriented systems," *ACM SIGPLAN Notices*, vol. 28, no. 10, pp. 326–337, 1993.

[33] W. De Pauw, D. Kimelman, and J. Vlissides, "Modeling object-oriented program execution," in *Object-Oriented Programming*, vol. 821, Springer, Berlin, Heidelberg, 1994, pp. 163–182.

[34] W. De Pauw, D. Lorenz, J. Vlissides, and M. Wegman, "Execution Patterns in Object-Oriented Visualization," 1980.

[35] I. Jacobson, P. Jonsson, and G. Övergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Pub, 1992, p. 54 435, ISBN: 0201544350.

[36] R. J. Walker, G. C. Murphy, B. Freeman-Benson, D. Wright, D. Swanson, and J. Isaak, "Visualizing dynamic software system information through high-level models," *ACM SIGPLAN Notices*, vol. 33, no. 10, pp. 271–283, 1998.

[37] T. Bell, "The concept of dynamic analysis," *ACM SIGSOFT Software Engineering Notes*, vol. 24, no. 6, pp. 216–234, 1999.

[38] D. Heuzeroth, T. Holl, and W. Lowe, "Combining Static and Dynamic Analyses to Detect Interaction Patterns,"

[39] D. Heuzeroth, T. Holl, G. Hogstrom, and W. Lowe, "Automatic design pattern detection," *MHS2003. Proceedings of 2003 International Symposium on Micromechatronics and Human Science (IEEE Cat. No.03TH8717)*, pp. 94–103, 2003.

[40] B. Schmerl, J. Aldrich, D. Garlan, R. Kazman, and H. Yan, "Discovering architectures from running systems," *IEEE Transactions on Software Engineering*, vol. 32, no. 7, pp. 454–466, 2006.

[41] H. Y. H. Yan, D. Garlan, B. Schmerl, J. Aldrich, and R. Kazman, "DiscoTect: a system for discovering architectures from running systems," *Proceedings. 26th International Conference on Software Engineering*, no. May, pp. 470–479, 2004.

[42] J. Koskinen, M. Kettunen, and T. Syst?? "Profile-based approach to support comprehension of software behavior," *IEEE International Conference on Program Comprehension*, vol. 2006, pp. 212–221, 2006.

[43] G. Hassapis, I. Kotini, and Z. Doulgeri, "Validation of a SFC Software Specification by Using Hybrid Automata," *IFAC Proceedings Volumes*, vol. 31, no. 15, pp. 107–112, Jun. 1998.

[44] T. Mertke and T. Menzel, "Methods and tools to the verification of safety-related control software," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, IEEE, 2000, pp. 2455–2457.

[45] G. Canet, S. Couffin, J. J. Lesage, A. Petit, and P. Schnoebelen, "Towards the automatic verification of PLC programs written in Instruction List," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, pp. 2449–2454, 2000.

[46] S. Kowalewski and J. Preuβig, "Verification of Sequential Controllers with Timing Functions for Chemical Processes," *IFAC Proceedings Volumes*, vol. 29, no. 1, pp. 4843–4848, Jun. 1996.

[47] N. Völker and B. J. Krämer, "Modular Verification of Function Block Based Industrial Control Systems," *IFAC Proceedings Volumes*, vol. 32, no. 1, pp. 159–164, May 1999.

[48] F. Jiménez-Fraustro and É Rutten, "A synchronous model of IEC 61131 PLC languages in SIGNAL," in *Proceedings - Euromicro Conference on Real-Time Systems*, 2001, pp. 135–142, ISBN: 0769512216.

[49] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet Dossier," *Symantec-Security Response*, no. February 2011, pp. 1–69, 2011.

[50] S. McLaughlin, "On dynamic malware payloads aimed at programmable logic controllers," *Proceedings of the 6th USENIX conference on Hot topics in security. HotSec*, vol. 11, p. 10, 2011.

[51] S. Mclaughlin and P. Mcdaniel, "SABOT : Specification-based Payload Generation for Programmable Logic Controllers," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ACM, 2012, pp. 439–449, ISBN: 9781450316514.

[52] D. Bohlender, H. Simon, N. Friedrich, S. Kowalewski, and S. Hauck-Stattelmann, "Concolic test generation for PLC programs using coverage metrics," in *2016 13th International Workshop on Discrete Event Systems, WODES 2016*, IEEE, May 2016, pp. 432–437, ISBN: 9781509041909.

[53] S. Guo, M. Wu, and C. Wang, "Symbolic execution of programmable logic controller code," 2017, pp. 326–336, ISBN: 9781450351058.

[54] L. Cheng, K. Tian, and D. Yao, "Orpheus : Enforcing Cyber-Physical Execution Semantics to Defend Against Data-Oriented Attacks," *ACSAC - Annual Computer Security Applications Conference*, pp. 315–326, 2017.

[55] A. Abbasi, T. Holz, S. Etalle, and E. Zambon, "ECFI: Asynchronous Control Flow Integrity for Programmable Logic Con-trollers," vol. 12, 2017.

[56] A. Keliris and M. Maniatakos, "ICSREF: A Framework for Automated Reverse Engineering of Industrial Control Systems Binaries," in *NDSS*, 2019, ISBN: 189156255X. arXiv: 1812.03478.

[57] M. Zhang, C.-y. Chen, B.-c. Kao, Y. Qamsane, Y. Shao, and Y. Lin, "Towards Automated Safety Vetting of PLC Code in Real-World Plants," in *IEEE Security & Privacy*, 2019, pp. 560–576.

[58] H. Prähofer, R. Schatz, and C. Wirth, "Detection of high-level execution patterns in reactive behavior of control programs," 2010, pp. 14–19, ISBN: 4373224687132.

[59] H. Prahofer, R. Schatz, and A. Grimmer, "Behavioral model synthesis of PLC programs from execution traces," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, IEEE, Sep. 2014, pp. 1–5, ISBN: 978-1-4799-4845-1.

[60] S. Kalle, N. Ameen, H. Yoo, and I. Ahmed, "CLIK on PLCs! Attacking Control Logic with Decompilation and Virtual PLC," *Workshop on Binary Analysis Research (BAR) 2019 NDSS 19*, no. March, 2019.

[61] C. Schuett, J. Butts, and S. Dunlap, "An evaluation of modification attacks on programmable logic controllers," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 1, pp. 61–68, 2014.

[62] C. O'Flynn and Z. Chen, "Side Channel Power Analysis of an AES-256 Bootloader," in *CCECE*, 2015.

[63] J. Park and A. Tyagi, "Using Power Clues to Hack IoT Devices: The power side channel provides for instruction-level disassembly.," *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 92–102, 2017.

[64] J. Longo, E. Mulder, D. Page, and M. Tunstall, "SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip," in *Proceedings of the 17th International Workshop on Cryptographic Hardware and Embedded Systems*, 2015, pp. 620–640.

[65] M. A. Al Faruque, S. R. Chhetri, A. Canedo, and J. Wan, "Acoustic Side-Channel Attacks on Additive Manufacturing Systems," in *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, IEEE, 2016, pp. 1–10.

[66] *Mixing Equipment - Industrial Equipment - Shopping Cart by INDCO, New Albany, Indiana*.

[67] T. Holleczek, V. Venus, and S. Naegele-Jackson, "Statistical Analysis of IP Delay Measurements as a Basis for Network Alert Systems," in *2009 IEEE International Conference on Communications*, IEEE, Jun. 2009, pp. 1–6.

[68] D. Kushner, "The real story of stuxnet," vol. 50, no. 3, 48–53, 2013.

[69] P Srinivasan, "Fingerprinting Cyber Physical Systems: A Physics-Based Approach," Master's thesis, 2015.

[70] L. A. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. Mohammed, and S. A. Zonouz, "Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit," *NDSS*, no. March 2017, 2017.

[71] P. Liu, W. Zang, and M. Yu, "Incentive-based modeling and inference of attacker intent, objectives, and strategies," *ACM Transactions on Information and System Security*, vol. 8, no. 1, pp. 78–118, 2005.

[72] *Hacking Critical Infrastructure | OSINT Soup*, 2018.

[73] *Permanent Magnet DC Motor or PMDC Motor | Working Principle Construction*, 2018.

[74] D. Polka, *Motor and Drive Control*. ISA, 2006, pp. 133–152, ISBN: 978-1-55617-984-6.

[75] *Wolfram SystemModeler: Modeling, Simulation & Analysis*, 2018.

[76] *Modelica Libraries — Modelica Association*, 2018.

[77] B. Drury, *Interfaces, communications and PC tools*. Institution of Engineering and Technology, 2010, pp. 485–531, ISBN: 978-1-84919-013-8.

[78] D. Wagner and P. Soto, "Mimicry Attacks on Host-Based Intrusion Detection Systems *," Tech. Rep., 2002.

[79] N. Provos, "Improving host security with system call policies," in *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, ser. SSYM'03, Washington, DC: USENIX Association, 2003, pp. 18–18.

[80] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and Weibo Gong, "Anomaly detection using call stack information," in *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, IEEE Comput. Soc, pp. 62–75, ISBN: 0-7695-1940-7.

[81] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-Flow Integrity Principles, Implementations, and Applications," in *12th ACM Conference on Computer and Communication Security*, 2005.

[82] S. Electric, "Modicon M241 Logic Controller - Hardware Guide - 04/2014," Tech. Rep., 2014.

[83] *ControlLogix 5570 Controllers*.

[84] C. D. Schuett, "Programmable Logic Controller Modification Attacks for use in Detection Analysis," p. 118, 2014.

[85] *Programming with python language - capstone*, `https://www.capstone-engine.org/lang_python.html`, Accessed: 2019-11-10.

[86] *Qemu*, https://www.qemu.org/, Accessed: 2019-11-10.

[87] *Unicorn - the ultimate cpu emulator*, https://www.unicorn-engine.org/, Accessed: 2019-11-10.

[88] T. R. Alves, M. Buratto, F. M. De Souza, and T. V. Rodrigues, "OpenPLC: An open source alternative to automation," in *Proceedings of the 4th IEEE Global Humanitarian Technology Conference, GHTC 2014*, Institute of Electrical and Electronics Engineers Inc., Dec. 2014, pp. 585–589, ISBN: 9781479971930.

[89] *Opc foundation*, https://opcfoundation.org/, Accessed: 2019-11-10.

[90] M. A. Al Faruque, S. R. Chhetri, A. Canedo, and J. Wan, "Acoustic side-channel attacks on additive manufacturing systems," in *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, 2016, pp. 1–10.

[91] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side—channel(s)," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, B. S. Kaliski, ç. K. Koç, and C. Paar, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 29–45, ISBN: 978-3-540-36400-9.

[92] S. Chhetri and M. A. Al Faruque, "Poster: Attacks on Confidentiality of Additive Manufacturing Systems using Acoustic Side-Channel," *International Conference on Micro-manufacturing (ICOMM)*, 2016.

[93] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[94] X. Zhuang, X. Zhou, M. A. Hasegawa-Johnson, and T. S. Huang, "Real-world acoustic event detection," *Pattern Recognition Letters*, vol. 31, no. 12, pp. 1543–1551, Sep. 2010.

[95] A. Mesaros, T. Heittola, A. Eronen, and T. Virtanen, "Acoustic event detection in real life recordings," in *2010 18th European Signal Processing Conference*, 2010, pp. 1267–1271.

[96] A. Temko, R. Malkin, C. Zieger, D. Macho, C. Nadeu, and M. Omologo, "Clear evaluation of acoustic event detection and classification systems," in *Multimodal Technologies for Perception of Humans*, R. Stiefelhagen and J. Garofolo, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 311–322, ISBN: 978-3-540-69568-4.

[97] J. F. Gemmeke, L. Vuegen, P. Karsmakers, B. Vanrumste, and H. Van hamme, "An exemplar-based nmf approach to audio event detection," in *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2013, pp. 1–4.

[98] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss, and K. Wilson, "CNN architectures for large-scale audio classification," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 131–135, 2017. arXiv: 1609.09430.

[99] Y. Sakashita and M. Aono, "Acoustic Scene Classification by Ensemble of Spectrograms based on Adaptive Temporal Divisions," Tech. Rep. 1, Mar. 2018, pp. 29–29.

[100] J. Pons and X. Serra, "Randomly Weighted CNNs for (Music) Audio Classification," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2019, pp. 336–340, ISBN: 9781538646588.

[101] L. Wyse, "Audio Spectrogram Representations for Processing with Convolutional Neural Networks," 2017. arXiv: 1706.09559.

[102] "Acoustic noise in induction motors: causes and solutions," in *Record of Conference Papers - Annual Petroleum and Chemical Industry Conference*, 2000, pp. 253–260.

[103] Y. Javadzadeh and S. Hamedeyaz, "Noise of Induction Machines," *Trends in Helicobacter pylori Infection*, vol. i, no. tourism, p. 13, 2014.

[104] T. Bertolini and T. Fuchs, *Vibrations and Noises in Small Electric Motors*. 2012, ISBN: 9783862360352.

[105] A. P. Mathur and N. O. Tippenhauer, "Swat: A water treatment testbed for research and training on ics security," in *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, 2016, pp. 31–36.

[106] *PyTorch*.

[107] T. Weber, *HAXPO: Reverse Engineering Custom ASICs by Exploiting Potential Supply-Chain Leaks*, 2019.

[108] *Jtagulator*, http://www.grandideastudio.com/jtagulator/, Accessed: 2019-11-10.

[109] *J-link pro | segger*, https://www.segger.com/products/debug-probes/j-link/models/j-link-pro/, Accessed: 2019-11-10.