Detecting Code Injections in Noisy Environments Through EM Signal Analysis and SVD Denoising

Presented in Partial Fulfillment of the Requirements for the Degree of

Master of Science

with a Major in

Computer Science

in the

College of Graduate Studies

University of Idaho

by

Ekaterina Miller

Major Professor
Konstantinos Kolias, Ph.D.

Committee
Frederick T. Sheldon, Ph.D.
Jia Song, Ph.D.
Alex Vakanski, Ph.D.

Department Administrator
Terence Soule, Ph.D.

May 2022

Abstract

The widespread use of embedded devices in virtually all types of critical applications has rendered them a vulnerable target for attackers and evildoers. However, modifying traditional well-known perimeter protection mechanisms such as Intrusion Detection Systems to function in corresponding devices is not a trivial task. The main challenges revolve around Random Access Memory (RAM) and processing limitations. Recently, the analysis of electromagnetic emanations has gathered the interest of the research community. Thus, analogous protection systems have emerged as a viable solution for providing external, non-intrusive control-flow attestation for resource-constrained devices. Unfortunately, the majority of current work fails to account for the implications of real-life factors, predominantly the impact of environmental noise. In this work, we introduce a framework that integrates Singular Value Decomposition (SVD) along with outlier detection mechanisms for discovering malicious modifications in the execution of embedded software even under variable conditions of noise. Our proposed framework achieves near-perfect accuracy, i.e., above 99% Area Under the Curve (AUC) score of even minimal and unknown code injection attacks in moderately noisy environments, e.g., 0 Signal-to-Noise Ratio (SNR), and maintains high detection accuracy, i.e., above 93% AUC score for unseen attacks, even under extreme noise conditions, i.e., -10 SNR. To the best of our knowledge, this is the first time this realistic limiting factor, i.e., environmental noise, has been successfully addressed in the context of Electromagnetic (EM)-based anomaly detection for embedded devices.

Acknowledgements

This project would not have been possible without the incredible team's work led by Constantinos Kolias. Specifically, I would like to thank Georgios Michail Makrakis, Kurt Vedros, and Craig Rieger.

Dedication

I would like to thank my husband Pete and my sons Serafim and Eastin for putting up with me while I was working on this project.

Table of Contents

List of Tables

List of Figures

Acronyms

AUC Area Under the Curve. ii

CI Critical Infrastructure. 1

CPU Central Processing Unit. 1

DWT Discrete Wavelet Transform. 50

EM Electromagnetic. 1

ICS Industrial Control Systems. 1

LOF Local Outlier Factor. 2

LSTM Long Short-Term Memory. 47

ML Machine Learning. 2

PCA Principal Component Analysis. 42

PD Partial Discharge. 49

PSD Power Spectral Density. 48

RAM Random Access Memory. ii

RNN Recurrent Neural Network. 47

SNR Signal-to-Noise Ratio. ii, 2

STFT Short-Term Fourier Transforms. 45

STS Shot-Term Spectra. 45

SVD Singular Value Decomposition. ii

VRM Voltage Regulator Module. 43

WT Wavelet Transform. 50

CHAPTER 1

Introduction

Today, embedded devices have become an indispensable component for a wide range of heterogeneous applications. Besides home automation, transportation, and retail, such systems are increasingly deployed even within Industrial Control Systems (ICS). Such systems may even support mission-critical tasks that are part of Critical Infrastructure (CI). Therefore, it does not come as a surprise that embedded devices time and again become the targets of cyberattacks. Nonetheless, due to their limited on-chip processing capabilities and proprietary nature, installing traditional means of protection such as anti-malware or Host-based Intrusion Detection Systems (H-IDS) is generally considered impractical.

Because of this, the applicability of external means of detecting the execution of malware, e.g., via control-flow attestation, has attracted the interest of the research community. Among the considered solutions, monitoring systems based on side-channel analysis have gained traction. The term *side-channel* refers to analog signals that get emanated by devices during their normal operation involuntarily. Typically, such signals are disregarded as they do not correspond to those wireless signals transmitted by the devices purposefully to achieve communication. However, relevant bibliography Nazari *et al*. (2017),Han *et al*. (2017),Khan *et al*. (2019),Khan *et al*. (2018),Han *et al*. (2019), and Sehatbakhsh *et al*. (2020a), indicates that there is a correlation between these analog signals and the operations actively performed by a subject device.

Among the different types of side-channels (power Liu *et al*. (2016),Wei *et al*. (2019), acoustic Anton *et al*. (2019), thermal Islam *et al*. (2017)), the analysis of Electromagnetic (EM) signals is preferable because it provides high bandwidth and enables the monitoring of the Central Processing Unit (CPU) activity at high sampling rates Nazari *et al*. (2017),Han *et al*. (2017),Sehatbakhsh *et al*. (2018),Kolias *et al*. (2020),Kolias *et al*. (2019).

Previous work in the area Nazari *et al*. (2017),Han *et al*. (2017),Khan *et al*. (2019) has shown high accuracy for detecting *significant* violations in the execution flow of a monitored program under *controlled laboratory environments*. However, minimal code injections of just a few instructions may be more stealthy and challenging to detect. A *code injection* is an attack-type that consists of injecting malicious code that is, in turn, forcefully executed by the application. Detecting minimal code injections becomes particularly challenging when such protection systems are deployed in realistic environments and get affected by noise. Vedros *et al*. (2021) identified that even one-instruction injections could be identified reliably, given a high sampling rate, when operating at high SNR environments. The authors also noted the necessity of a noise elimination step, particularly for low SNR conditions. Finally, they highlighted that some noise filters might prove detrimental to the anomaly detection process due to their tendency to eliminate critical artifacts associated with the anomaly.

In this work, we propose a framework for reliably identifying code injection attacks of *arbitrary* lengths of code even under the influence of *variable* levels of environmental noise. At the core of the framework lies a non-destructive, noise-reduction technique that is based on the Signal-to-Noise Ratio (SNR) method. The proposed system relies on Machine Learning (ML) to effectively *describe* the normal operations based on examples of EM signals. The analysis process is not supervised, but rather it is based on an outlier detection strategy at the core of which lies the well-known Local Outlier Factor (LOF) Breunig *et al*. (2000) method. This design decision renders the system capable of detecting *unseen attacks* i.e., in this context, the injection of variable lengths of code and alternative types of instructions. These considerations are reasonable, particularly for deployments in industrial settings where we have virtually no control over the noise level.

The contributions of this work are summarized as follows:

- We propose a noise reduction strategy based on a modified version of SVD to reduce the environmental white noise from a signal without eliminating critical anomalous artifacts.

- We contribute equations derived from the empirical analysis to predict critical hyperparameters of the method.
- We propose an anomaly detection strategy that is based on the incorporation of the Local Outlier Factor in transduction for identifying even unseen anomalies.
- We propose an integrated framework for the reliable capturing and analysis of EM signals from the CPU of embedded devices that can achieve high performance even in highly noisy environments for the detection of code injection attacks.

CHAPTER 2

Technical Background & Definitions

This chapter describes fundamental technical terms and related previous work for a better understanding of the scientific foundations of the proposed framework.

2.1 SIDE CHANNEL ANALYSIS

The execution of instructions by the CPU of a given device results in changes in the flow of electric current inside the CPU's circuitry. This change produces a magnetic field that interacts with the electric field, resulting in an EM field. Moreover, the components of the printed circuit board act as antennas that unintentionally transmit the EM signals. These emanations can be captured by placing a probe near the source of the signal Han *et al*. (2017), Han *et al*. (2019). It is possible to identify the execution state of a program by analyzing these analog signals. The reader should notice that there are alternative types of side-channels, for example, cache and timing side-channels Yarom and Benger (2014), Hund *et al*. (2013). These are considered out of the scope of this work, as this research concentrates on EM side-channels.

EM signals obtained by the CPU are amplitude modulated according to corresponding instruction, with a carrier signal that has as base frequency the clock of the monitored CPU Khan *et al*. (2018). Therefore, different instructions can theoretically be distinguished by observing the amplitude of the signal across time.

Unfortunately, the presence of noise increases the difficulty in identifying sequences that correspond to potentially unseen instructions. Figure 2.1 presents an example of an EM signal that corresponds to the same execution path in clean (red) and noisy (blue) environments for different SNR levels. The reader may notice that as the levels of noise increase, the critical differences in amplitude become unrecognizable.

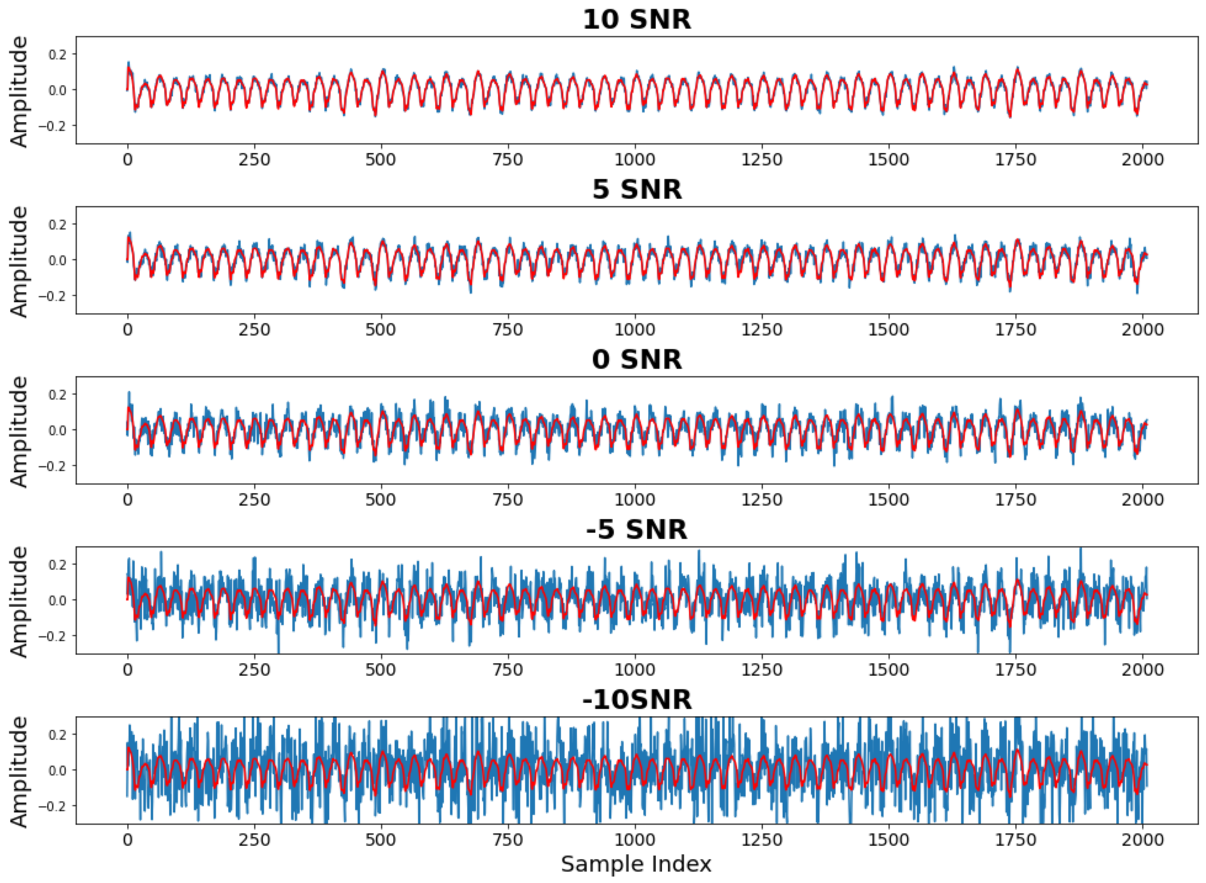

FIGURE 2.1: Comparison between a clean signal (red) and the noisy version of the same signal (blue) for various SNR levels.

2.2 CODE INJECTION ATTACKS

Code injection attacks start by exploring a target software for pre-existing vulnerabilities that allow an attacker to inject malicious code. During the exploitation step, the instructions that are injected change the original logic of the program or forge a new (malicious) execution path, thus altering the original control flow.

Typically, code injection attacks are possible due to the lack of proper sanitization of program input blocks. When such a vulnerability exists, an aggressor may insert directly executable code to effectively modify a program. Control flow attestation and control-flow integrity tools aim to detect or protect against such malicious practices. However, the majority of conventional practices that fall under this category rely on the instrumentation of the binary in collaboration with monitoring tools installed natively. That is not always possible.

To fill this gap, side-channel techniques aim to achieve the same, but in a totally external fashion, simply by relying on the EM signals emitted by the subject CPU. Previous work in the area has shown that code injection attacks can be reliably identified with this method. However, the majority of these works (a) consider only significant modifications of the logic of a program where an attacker does not wish to remain stealthy and (b) The majority of anomaly detection techniques proposed in the literature achieve high-quality results assuming ideal conditions of a laboratory setting. Nevertheless, in certain applications such as ICS, the devices are expected to be deployed in conditions of high levels of EM noise. Indeed, critical artifacts corresponding to the anomalous behavior are frequently concealed behind noise Vedros *et al*. (2021), Khan *et al*. (2018),Epps and Krivitzky (2019).

In previous work, the authors Vedros *et al*. (2021) have been able to detect as low as one-instruction injection with an average AUC rate of 98.7% for low noise environments (10 SNR) using EM-based techniques and a rudimentary analysis method. However, as the noise level increases, the detection rate decreases remarkably.

2.3 SVD DENOISING

SVD has been applied in diverse fields such as feature reduction in data analysis Abdi and Williams (2010), image processing Epps and Krivitzky (2019), Rajwade *et al*. (2012), fault diagnosis for machine Zeng and Chen (2019),Zhao and Jia (2017), to name just a few. Regardless of the application, SVD essentially discovers the most *important* components of a *signal*, (where the terms important and signal depends on the application).

A critical mass of works has applied SVD for signal denoising, but only a handful of these works have applied SVD for noise reduction of EM signals Jha and Yadava (2010), Zhao and Jia (2017), Zeng and Chen (2019). In this context, a received signal is considered as the composition of the original signal and the noise:

$$X_m = X_s + X_n \tag{2.1}$$

where X_m is a set of examples of a signal mixed with random noise (usually expressed as a 2-D matrix), X_s is a set of clean signals, and X_n is the white Gaussian noise added to each signal. Applying SVD in X_m will result in the decomposition of that matrix into three components $X_m = U\Sigma V^T$, where the columns of U are the left singular vectors of X_m, the columns of V are the right singular vectors of X_m, and Σ is a diagonal matrix where the values along the main diagonal correspond to singular values of A. These values are ordered $\sigma_1 \geq \sigma_2 \geq \sigma_3 ... > 0$ Golub and Van Loan (2013). In general, the first singular value contained in the Σ matrix will correspond to the most *important* features of the signal. U can be found as the eigenvector of $X_m X_m^T$, V is the eigenvector of $X_m^T X_m$, and singular values are the square roots of the eigenvalues of $X_m X_m^T$ or $X_m^T X_m$ (which are the same). From a mathematical point of view, to have real eigenvalues and eigenvectors, the matrix must be symmetric. Multiplying any rectangular matrix by its transpose will create a square symmetric matrix.

Each of these components can still be expressed as the composition of two signal subspaces: (a) that of the clean signal and (b) that of the noise as follows:

$$X_m = U\Sigma V^T = [U_s U_n][\Sigma_s \Sigma_n][V_s^T V_n^T] \tag{2.2}$$

In order to denoise the signal, all singular values that are below a certain threshold (from now on, this will be referred to as *cutting point*) are set to zero, thus creating a new approximation of the singular matrix, Σ_{new}. Figure 2.2 (right) illustrates this transformation. After this, a denoised version of the signal X_s can be obtained as the dot product of U, Σ_{new}, and V^T. More specifically, this can be expressed as in Equation 2.3.

$$\begin{aligned} X_s &= U\Sigma_{new}V^T \\ &= U \begin{bmatrix} \sigma_s & 0 \\ 0 & \sigma_n = 0 \end{bmatrix} V^T \end{aligned} \tag{2.3}$$

where X_s is the clean signal, U is the left singular matrix, σ_s are the singular values that correspond to the signal, σ_n are the singular values that correspond to noise, and V^T is the right singular matrix transposed.

One challenge with this approach is to accurately pinpoint the cutting point, i.e., the index of the singular value in the Σ matrix that demarcates the singular values that correspond to the signal from those corresponding to noise. The reader should remember that the σ values in the Σ matrix are ordered across the main diagonal. Hassanpour *et al*. (2012) identified this point as the one where the slope of the curve of the corresponding singular values changes drastically (Figure 2.2). This can be obtained as the maximum change between two consequent derivation values when calculating the derivation of the curve at each point.

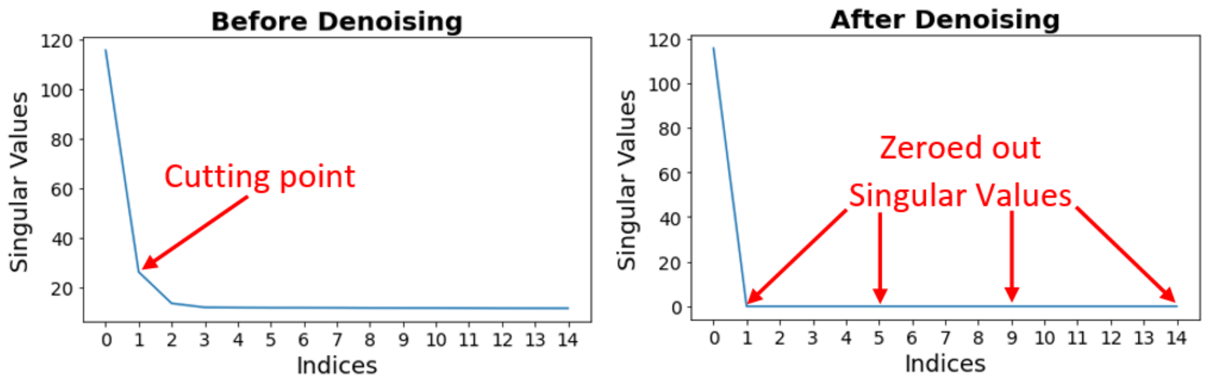

FIGURE 2.2: Position of the cutting point where the slope of the curve of the corresponding singular values changes drastically. Noise reduction can be achieved by zeroing out singular values after that point.

Another obvious problem with such methods has to do with the high computational complexity of the denoising process, which can be up to $O(m^2n + nm^2 + n^3)$ for m by n matrix or $O(min(m^2n, nm^2))$ if we only need the k largest singular values.

Finally, a significant shortcoming revolves around the applicability of the approach in streaming applications. Mainly because SVD cannot be directly applied to denoise a single signal (streaming mode) since, in that case, SVD always yields a Σ, which is a vector with only one non-zero singular value:

$$X_m = u \begin{bmatrix} \sigma_1 0 \ldots 0 \end{bmatrix} \begin{bmatrix} v_s^T & \cdots \\ \vdots & \vdots \\ v_n^T & \cdots \end{bmatrix} \tag{2.4}$$

where u is a scalar value, σ_1 is the only non-zero singular value of the Σ vector, and v_i are the right singular vectors, of which only some correspond to the actual signal. To tackle this problem, a common solution is to first transform the single observation X_m into a Hankel Matrix (HM) representation, apply the denoising, and then revert back to a one-dimensional signal. Let X_m represent a signal in time domain, i.e., $X = x_0, x_1, x_2, ..., x_n$. The Hankel Matrix (HM) of the signal can be written as

$$H = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_k \\ x_1 & x_2 & x_3 & \cdots & x_{k+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_l & x_{l+1} & x_{l+2} & \cdots & x_n \end{bmatrix} \tag{2.5}$$

where n is the number of timestamps, l is the number of rows in H, and $k = n - l + 1$ is the number of columns. Govindarajan *et al.* (2019) use the HM on the whole signal to remove the random noise from the Partial Discharge (PD) signal. Zhou *et al.* (2019) propose to create an HM for each sliding window. This way, they were able to prevent waveform distortion, a typical issue when SVD is performed for the whole signal. Generally, Hankel transformations result in structures that have high memory requirements, in combination with the long execution times of SVD. Therefore, the denoising time may easily become greater than the arrival rate of new observations. Because of this, in this work, instead of denoising each signal individually, we consider the use of a matrix created out of multiple synchronized signals and perform a grouped denoising. The theoretical idea is described in De Moor (1993) and implemented in Jha and Yadava (2010) for removing a "noise" in vapor samples.

CHAPTER 3
Problem statement & Threat Model

The main purpose of this work is to introduce an integrated framework for reliably identifying code injection attacks in embedded devices even in the presence of noise by relying only on totally external, non-intrusive means. To achieve this, we rely on EM signals, and we assume we have knowledge of what type of signals should be emitted when the program operates normally (without the code injection).

We treat this problem as an anomaly detection problem comprising of two steps (a) the fingerprinting and (b) the deployment stage. During the fingerprinting stage, several examples of the same operation are gathered and used to identify the most representative morphology of the signal. We assume that during this stage, the code executed corresponds to the normal operation; therefore, we have no knowledge of the anomalous operations and their corresponding signals. During the deployment stage, we obtain signals whose type is not yet known, but their morphology has been altered due to the impact of higher levels of noise.

The main assumptions regarding the attacker's behavior and their capabilities are the following:

- We assume that a vulnerability in the code of a protected program that allows a code injection exists, and the attacker is aware of this vulnerability.
- The attacker has the capability of injecting arbitrary lengths of malicious instructions.
- Even a minimal number of extraneous instructions can have a significant (yet sometimes stealthy) impact.
- The attacker can inject the malicious set of instructions anywhere in code.

CHAPTER 4
Proposed Framework

This study aims to detect one instruction code injection in a noisy environment using an unsupervised detection mechanism and SVD denoising techniques. We will discuss the challenges this denoising presents and the requirements for effective noise elimination without the loss of essential artifacts. In addition, we will show the possibility of training a detection algorithm in one noise domain and using this algorithm to detect code injection in a different (noisier) noise domain. The research will determine the choice of parameters for the SVD denoising using empirical and statistical analyses.

The proposed framework specifies practices for (a) obtaining the EM signals from the subject device, (b) performing signal pre-processing to reduce noise while maintaining important characteristics that correspond to anomalies, (c) fingerprinting the morphology of EM signals corresponding to normal operations (training) and finally, (d) quantifying the level of disparity of these signals versus the ones obtained on the field towards recognizing anomalous operations. Further details for each of these procedures are discussed below.

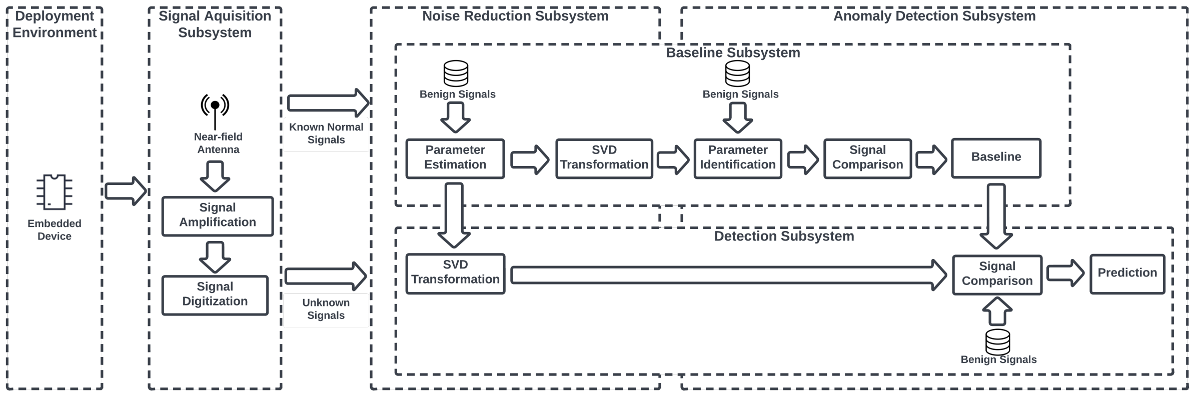

FIGURE 4.1: Data flow and basic procedures in the proposed framework.

4.1 SIGNAL ACQUISITION

As a first step, the signals are obtained from the CPU using a near-field antenna that is placed in close proximity to the target device. The actual distance depends on the size of the monitored device, the type of enclosure, the type of the antenna, the presence of other digital devices operating in the area, etc. Generally, one to five cm away from the target is expected to give good accuracy in most noise-free setups.

During the fingerprinting phase, an adequate number of signals must be captured to define the baseline. The number of signals varies and is analogous to the total duration of the operation-to-be-fingerprinted and the sampling rate. From empirical observations, one must collect a large number of samples that correspond to each potential execution path of instructions (e.g., same loop) to achieve efficient fingerprinting. Ideally, the number of samples of each execution path should be at least as many as the total number of timestamp samples that are used to describe those signals. During the deployment phase, to avoid the use of a Hankel matrix, multiple signals must also be obtained. Our experiments indicate that the number of observations should be double the number of samples in each observation (Chapter 7).

Due to the low power of the captured signals, the readings must first be amplified before they are fed to an oscilloscope. After this, signals are digitized and stored in a database in binary (preferable) or CSV format. According to Vedros *et al.* (2021) high sampling rates, i.e., around x16, the CPU clock speed (with higher sampling rates providing diminishing returns) are ideal for anomaly detection purposes. Moreover, such sampling rates are found to provide some level of robustness against noise and can reliably detect even small code injections.

4.2 NOISE REDUCTION & PRE-PROCESSING

Noise poses a significant challenge when the given task is anomaly detection. In this work, we apply a modification of the SVD denoising as a pre-processing step to reduce the noise level in the captured signals. The variations revolve around two axes. On the one hand, the method is applied to *multiple examples* (as opposed to a single) describing the same phenomenon, i.e., the same branch. This significantly speeds up the denoising process and alleviates the need for bringing individual observations to their HM representation, thus reserving memory. On the other hand, the signals are denoised by considering a *higher-than-usual values for cutting points*. Although this decision may appear counterintuitive, the reader should keep in mind that anomaly detection and not noise elimination is the main objective of this work. Therefore, the traditional cutting points defined in the literature based on the slope of the curve of the corresponding singular values Hassanpour *et al.* (2012) or some hard threshold Gavish and Donoho (2014) may be irrelevant in this context. In Chapter 7 we suggest the choice of the cutting point that will allow us to observe the maximum contrast between normal signals and signals with code injection. Since the only characteristic that distinguishes different instructions is the difference in amplitude, we must take special care in preserving those differences during the denoising process.

4.3 ANOMALY DETECTION

Our EM signal anomaly detection strategy is a modification of the transduction and hypothesis testing algorithm introduced by Barbara et al. in Barbara´*et al.* (2006). Transduction operates using the process of reasoning from specific, *baseline* cases to other specific *testing* cases. This is in contrast to the more popular method of *induction*, which reasons from specific cases to rules that can be applied to test other cases. In the context of this problem, the baseline represents the morphology of the benign signals and quantifies the variability in the population of the signals caused primarily by noise. The basic assumption is that benign instances are easy to obtain in real-life conditions. Meanwhile, malicious cases, while bearing differences compared to benign observations, are highly unpredictable regarding the exact location, the morphology, and the extent of the disparity. Thus, the anomalous cases that can be observed in real life may potentially be infinite. The latter is the primary reason why we chose to approach the problem at hand as an outlier detection problem rather than a classification problem.

Transduction is carried out by placing an unknown signal in a known sample distribution of data and then carrying out hypothesis testing; it determines whether that instance is a good fit, i.e., highly similar to the rest or not. To that end, a measure of *strangeness* i.e., *level of unfitness* is used for every signal with respect to every other signal in the distribution, including the one we are trying to fit. Strangeness is a function that measures the uniqueness or unfitness of that point. The authors of Proedru *et al.* (2002) coined the term, and the term has been used in machine learning algorithms based on transduction ever since.

The process described above is given in Algorithm 1. In further detail, the proposed approach requires that the following steps are executed:

Step 1: Collect a population of normal signals. Here, we assume that during the fingerprinting step, all examples correspond to a benign operation, and the device is not infected. This will be the *baseline* of signals.

Step 2: Apply the LOF algorithm for each example in the baseline to quantify the level of the strangeness of that signal with the rest. The reader may recall that LOF has been proven to have superior discrimination power over simpler methods like K-Nearest Neighbors (KNN) due to the fact that LOF takes into consideration the density of the neighboring points relative to the density of the point in consideration. For this reason, LOF has become our method of choice. This will be the *baseline strangeness distribution.*

Step 3: During the deployment phase, obtain a signal we wish to test, and compute its *unfitness score* with respect to the baseline constructed in Step 1 by using LOF. Then, transduce that value against the baseline strangeness distribution that was constructed in the previous step.

Step 4: As a result of the previous step, one obtains a fraction of the number of signals whose unfitness score is greater or equal to the unfitness score of the test point to the total signals considered. This fraction can be considered a *p-value* for a statistical hypothesis test, where the *null hypothesis* is this point belongs to the baseline distribution (considered normal), while the alternative hypothesis is that this point does not fit the distribution, and hence, it needs to be deemed an anomaly.

Advantages: The proposed approach can detect anomalies without knowing (training examples) their morphology or characteristics. Moreover, this method is guided by a user's confidence level. This is contrary to commonly used anomaly detection algorithms that produce a ranking of outliers, leaving the user to select the desired number.

Algorithm 1 Anomaly Detection Algorithm

```
1: function DETECT(benign dataset $X$, test observation $q$)
2:     $status_q \leftarrow 0$
3:     for $\forall i \in X$ do
4:         $s_{x_i} \leftarrow LOF(X, x_i)$
5:         $S_i \leftarrow s_{x_i}$
6:     end for
7:     $s_q \leftarrow LOF(X, q)$
8:     for $\forall s_{xi} \in S$ do
9:         if $s_{x_i} \geq s_q$ then
10:            $n \leftarrow n + 1$
11:        end if
12:    end for
13:    $p_q \leftarrow \frac{n+1}{|x|+1}$
14:    $\tau \leftarrow 1 - confidence$
15:    if $p_{max} <= \tau$ then
16:        $status_q \leftarrow 1$
17:    end if
        return $status_q$
18: end function
```

CHAPTER 5
Experimental Setup & Data Gathering

To evaluate the proposed framework, we have created an experimental setup consisting exclusively of low-cost, off-the-shelve components. A simple control process emulating a *tank filling system* is used as the software-to-be-protected against malicious modifications. Two alternative versions of that software are compiled, each one corresponding to the version of the control logic after the malicious injection of a single instruction. During the fingerprinting stage, EM signals are obtained from the benign version of the software. These signals are used to create the baseline and train the detection engine. At the deployment stage, we collected signals while the device was operating under normal as well as anomalous states (i.e., after the injection). In an offline step, we add noise of various levels to the signals synthetically to simulate the change in environmental conditions. The predictive accuracy of our system is evaluated. Details regarding the process described above are given in this section.

5.1 CONTROL LOGIC PROCESS

For the inspected control process, we devised a simple tank filling system. According to the considered scenario, fluid is inserted into a tank by a pump. At the point where the fluid reaches critical levels, a valve opens, and the fluid is removed from the container. Two sensors indicate the levels of the fluid that are placed at two different levels (high and low). In the unlikely event that the sensors provide erroneous readings (e.g., high is enabled but low is not), an alarm is triggered, and the system automatically disables both the pump and the valve. For reasons of simplicity, both the pump and the valve can be in an *open* or *closed* state, and the sensors can provide an indication or not depending on whether the fluid has reached the sensor. The pump keeps pushing fluid to the tank until both the high-level and low-level sensors provide readings. When that happens, the pump switches to a closed state, and the valve switches to an open state. Once the fluid is removed from the tank, the process

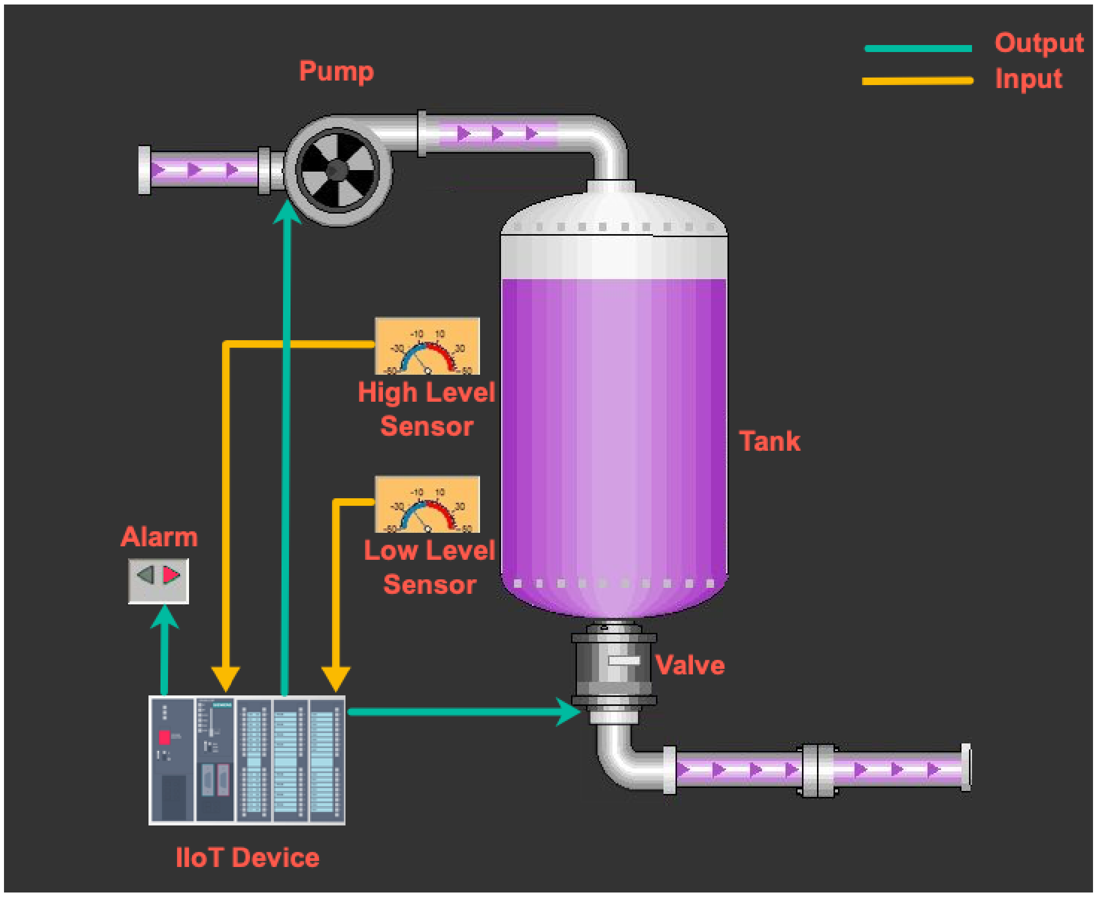

FIGURE 5.1: A graphical representation of described tank filling scenario.

starts again from the beginning (loop). A sample setup supporting the described control process is illustrated in Figure 5.1 and the corresponding pseudocode can be found in Algorithm 2

The above control logic is implemented in the AVR assembly language and installed in an Arduino Mega with an ATmega2560 CPU clocked at 16MHz. The choice of language is made to have better control over the actual instructions being executed at the CPU at every instance. The demonstrated results could have been achieved using C language, which is the de facto language for the platform of choice, without loss of generality. However, the C compiler tends to add or remove numerous instructions that are not directly relevant to the objective, e.g., for purposes of optimization. The assembly code that corresponds to the described control logic can be found in Listing 5.1.

```
.....
loop:
  sbi pinb, 7      ; toggle pinb 7 every loop (as sync artifact)

  ; just a NOP sequence
  nop
  nop
  nop
  nop
  nop
  nop

  ; read the lowLevel input (arduino pin 11)
  clr   r20
  sbic  pinb, 3
  ldi   r20,  1

  ; read the highLevel input (arduino pin 12)
  clr   r21
  sbic  pinb, 4
  ldi   r21,  1
  jmp low_0_high_lowerEq_1
  rjmp  loop
low_0_high_lowerEq_1:

  ; if lowLevel is not set, go to the next case
  ldi r22, 0
  cp  r20, r22
  brne  low_1_high_lowerEq_1

  ; if highLevel is not set, go to the next case
  ldi r22, 0
  cp  r21, r22
  brne  low_1_high_lowerEq_1

  ; set only the pump
  sbi portb, 2
  cbi portb, 1
  cbi portb, 5
  rjmp  loop
.....
```

Listing 5.1: The normal assembly program for the tank filling process.

Algorithm 2 Tank Filling process

```
1: $LowLevel \leftarrow readLowLevelSensor()$
2: $HighLevel \leftarrow readHighLevelSensor()$

3: if $LowLevel = 0$ and $HighLevel = 0$ then
4:     $pump \leftarrow 1$
5:     $valve \leftarrow 0$
6:     $alarm \leftarrow 0$
7: else if $LowLevel = 1$ and $HighLevel = 1$ then
8:     $pump \leftarrow 0$
9:     $valve \leftarrow 1$
10:    $alarm \leftarrow 0$
11: else if $LowLevel <= 1$ and $HighLevel > 0$ then
12:    $pump \leftarrow 0$
13:    $valve \leftarrow 0$
14:    $alarm \leftarrow 1$
15: else//outputs unchanged
16: end if
```

The two adversarial cases considered in this work include the injection of ADD and JMP instructions, respectively. In other words, both adversarial cases consider the injection of a single instruction. However, both cases can be harmful and can alter the program execution in a meaningful way, e.g., alter the value of an important variable (ADD) or even skip entire blocks of code (JMP). Regardless, the malicious injection is expected to be much more extensive in real-life scenarios. These two modifications are presented in Listings 5.2 and 5.3.

```

.....
  nop
  nop
  nop

  add r2, r0

  ; read the lowLevel input (arduino pin 11)
  clr    r20
  sbic   pinb, 3
  ldi    r20,  1

.....
```

Listing 5.2: The assembly program for the injection of ADD instruction in the tank filling process.

```

.....
  nop
  nop
  nop

  jmp label1

  ; read the lowLevel input (arduino pin 11)
  clr   r20
  sbic  pinb, 3
  ldi   r20,  1
.....
```

Listing 5.3: The assembly program for the injection of JMP instruction in the tank filling process.

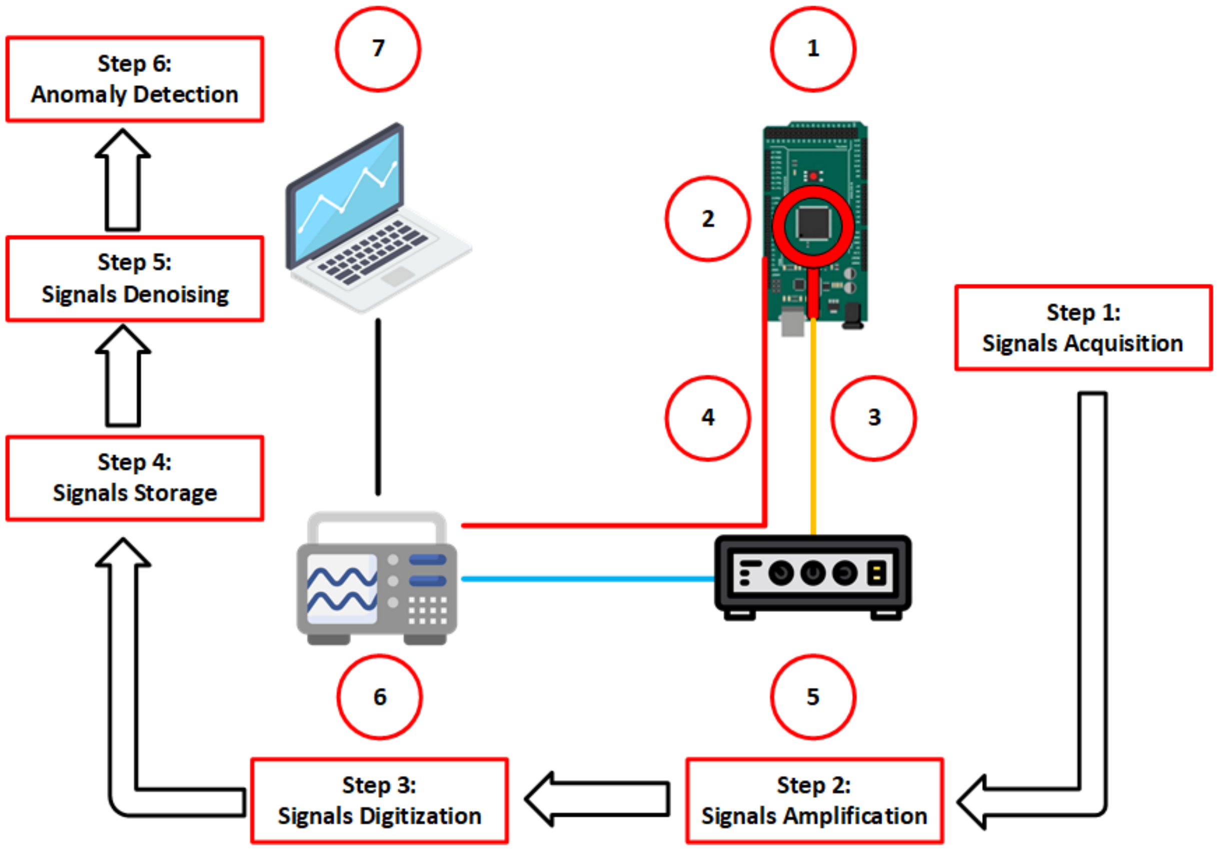

Figure 5.2: The proposed framework along with the major components used in our experimental setup.

5.2 EXPERIMENTAL SETUP

An illustration of our setup, along with the relevant placement of the mentioned components, is given in Figure 5.2. For all considered experiments, the subject device was the Arduino Mega ①. While the framework assumes any type of near-field antenna placed at a distance of a few cm away from the device, in our experiments, we used a near-field probe placed directly on top of the CPU; namely, an EMRSS RF Explorer H-Loop ②. This was done to obtain EM readings that are virtually noise-free. Later on, synthetic random noise having a Gaussian distribution was added to each signal. We considered noise levels of 10dB, 5dB, 0dB, -5dB, and -10dB SNR. The noise was added using Python and a version of the Additive Gaussian White Noise function adapted from Viswanathan (2019). Since such signals are emitted involuntarily using no dedicated antennas they are typical of extremely low amplitude. Therefore, each signal captured ③ was first amplified using a Beehive 150A EMC probe amplifier ⑤. The captured signals were saved in a digital format with the use of a PicoScope 3403D oscilloscope ⑥ connected to a laptop ⑦. For all experiments, the sampling rate was set to 250MS/sec, i.e., a sampling interval of 4ns. Notice that this sampling rate is almost 15 times more than the CPU clock of the chosen platform. Using an external signal from a specific I/O pin (④ in Figure 5.2), we were able to identify the beginning and end of the program's execution loop by toggling between high/low values on the I/O pin. This was done to maintain a strong synchronization among all the signals.

5.3 OBSERVATIONS ON THE CAPTURED SIGNALS

The injection of a malicious instruction occurs right after a sequence of NOP instructions. An ADD instruction consumes one cycle while a JMP instruction takes three cycles. Naturally, the injection of these malicious instructions causes a displacement of one and three cycles to the right for the following benign instruction, namely the CLR. This is presented in Figure 5.4 which contains a zoomed-in version of the area of interest in Figure 5.3.

By comparing the amplitude of the signals, we can verify that there is a difference between the normal (expected CLR) and malicious instructions (ADD and JMP). More specifically, The CLR has a much higher amplitude than either the ADD or JMP. A zoomed-in version of the area of interest is given in Figure 5.5. A comparison solely based on the differences in EM amplitudes for specific cycles indicates that the injection took place around the indexes 400 to 450.

5.4 EVALUATION METHOD

The method of ten-fold cross-validation was used to evaluate the detection accuracy of the proposed system. More specifically, 90% of the normal dataset was withheld and used for training, and 10% of the remaining normal dataset, along with the same amount of anomalous signals, was used for testing purposes in each of the folds.

The average from the folds has been calculated and reported. The predictive accuracy is evaluated using the Area Under the Curve (AUC) score of the corresponding Receiver Operator Characteristic (ROC) curve. A ROC is a popular metric for describing the efficiency of anomaly detection systems. It is essentially a graph illustrating the True-Positive Rate (TPR) vs. the False Positive Rate (FPR) for various thresholds. This graph is usually a curve, and since the shape of the curve is arbitrary, the most common method for comparing two ROC curves is by measuring the AUC.

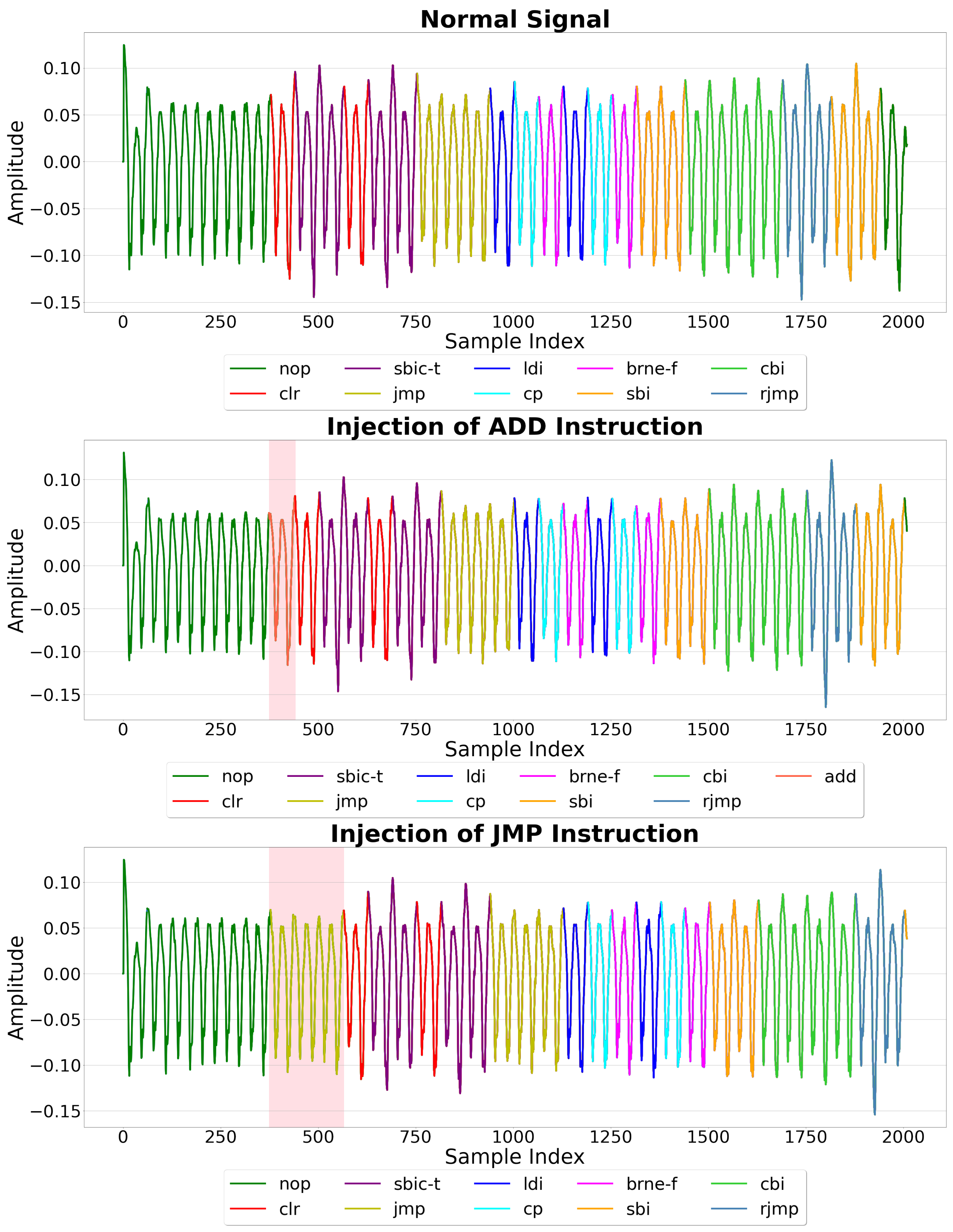

Figure 5.3: Examples of benign and malicious signals with the injected code are highlighted in red. It is clear that the injection of malicious instructions causes an analogous displacement.

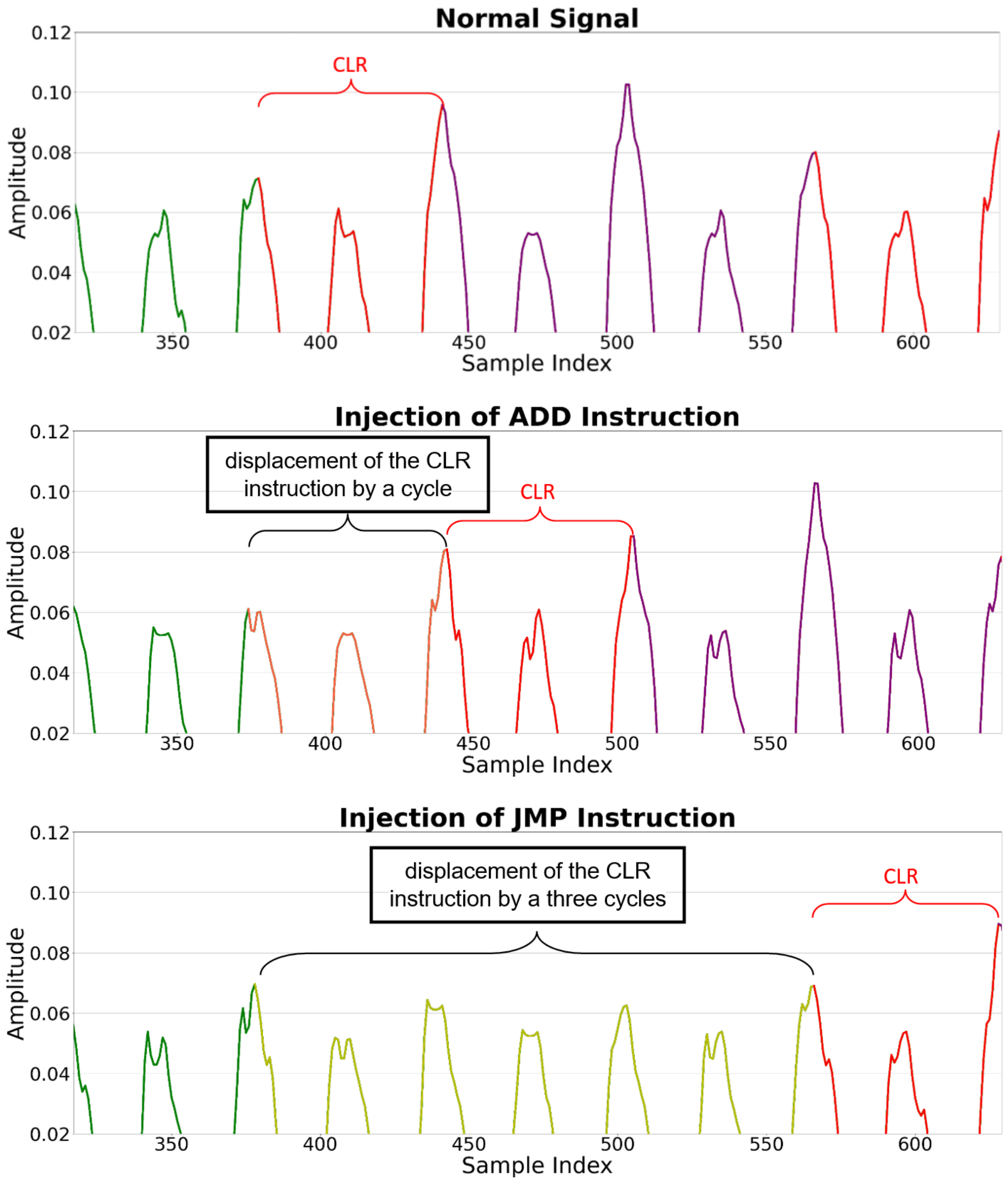

FIGURE 5.4: Zoom-in at the critical section where the injection occurred. Notice the displacement of the normal instruction (CLR) due to the injection of malicious instructions.

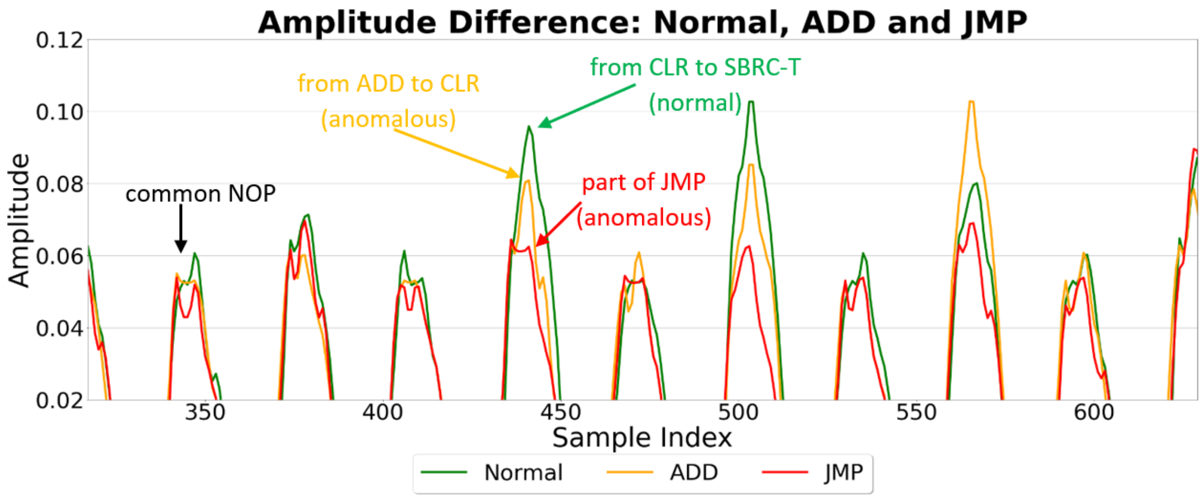

FIGURE 5.5: Zoom-in at the critical section where the injection occurred. The reader should compare the difference amplitude for each instruction that was injected vs. the normal instruction.

CHAPTER 6
Experimental Evaluation

We have conducted four sets of experiments to evaluate the efficiency of our framework. The reader should keep in mind that all experiments consider injecting a minimum number of instructions, i.e., one. Therefore, all results presented in this section define a lower bound (minimum) of predictive accuracy.

6.1 CONSIDERING NO PRE-PROCESSING FOR NOISE REDUCTION

Table 6.1: Performance of the detection algorithm without applying noise reduction as pre-processing. In parentheses is the number of neighbors provided as input to the anomaly detection algorithm.

Injection of ADD						
test SNR		10	5	0	-5	-10
training SNR	10	99.23% (3)	23.1% (3)	0.00% (3)	0.00% (3)	0.00% (3)
	5	-	98.74% (3)	0.00% (3)	0.00% (3)	0.00% (3)
	0	-	-	61.93% (3)	0.00% (3)	0.00%(3)
	-5	-	-	-	31.72% (3)	0.00% (3)
	-10	-	-	-	-	21.9% (3)
Injection of JMP						
test SNR		10	5	0	-5	-10
training SNR	10	99.40% (3)	22.03% (3)	0.00% (3)	0.00% (3)	0.00% (3)
	5	-	99.10% (3)	0.00% (3)	0.00% (3)	0.00% (3)
	0	-	-	39.36% (3)	0.00% (3)	0.00% (3)
	-5	-	-	-	15.3% (3)	0.00% (3)
	-10	-	-	-	-	9.97% (3)

In the first experiment, we wanted to identify the performance of the anomaly detection algorithm when no noise elimination was applied. More specifically, we have posed two questions: (a) what are the limits of our framework for various noise levels assuming that noise remained the same during the fingerprinting and deployment phases? (b) how does the predictive accuracy get impacted if the noise intensity increases from the fingerprinting to the deployment phase?

Table 6.1 contains the results (AUC scores) of the experiments. The highlighted values at the main diagonal of the table directly provide an answer to the first experiment. The rest of the values correspond to the second experiment. The results indicate that the AUC score is near-perfect when the system is deployed in a relatively clean environment, i.e., 10 to 5 SNR. However, the predictive accuracy rapidly drops to very poor levels (below 70%) when considering SNR below 5dB. For example, for -10dB, the score achieved is only 21.9% for identifying the injection of ADD instruction and 9.97% for identifying the injection of JMP instruction. Regarding the second experiment, the best results are obtained when the noise level during the deployment stage remains close to those applied during the fingerprinting process. Regardless of whether the noise levels are variable, the results indicate that higher noise levels have a clear negative impact.

Conclusion: *A noise reduction step is necessary, especially when the system is expected to operate at the SNR levels below 5 dB.*

6.2 CONSIDERING SVD WITH TRADITIONAL CUTTING POINTS FOR NOISE REDUCTION

In the next experiment, we introduce a pre-processing step aiming at noise reduction. The denoising method of choice is based on SVD. In this experiment, the parameter *cutting point* is chosen according to traditional criteria, i.e., where the slope of the curve in the corresponding singular values graph changes drastically. The value of the parameter is chosen based on the noise conditions of the fingerprinting stage solely, assuming no knowledge of the noise levels in the deployment stage. Notice that based on the chosen criteria in all of our experiments, the value of the cutting point is dictated to be one.

The results of this experiment are given in the Table 6.2. According to the values in the table, the application of SVD with traditional cutting points drastically degrades the anomaly detection accuracy for all cases. For example, the AUC score drops from near-perfect 99.23% to 4.49% even for the cleanest environment, i.e., 10dB for detection of the ADD instruction. Such an aggressive noise reduction procedure eliminates important characteristics that indicate the anomalies.

Conclusion: *The application of SVD for noise reduction as a pre-processing step, when using traditional cutting points, has a negative impact on the anomaly detection process. Therefore, new criteria for choosing optimal cutting points should be identified.*

Table 6.2: Performance of the detection algorithm with applying SVD using cutting point 1 and number of neighbors for the LOF algorithm equal 3

Injection of ADD					
SNR	**10**	**5**	**0**	**-5**	**-10**
	4.49% (1,1,3)	5.06% (1,1,3)	5.49% (1,1,3)	4.91% (1,1,3)	5.38% (1,1,3)
Injection of JMP					
SNR	**10**	**5**	**0**	**-5**	**-10**
	4.46% (1,1,3)	4.98% (1,1,3)	5.49% (1,1,3)	4.87% (1,1,3)	5.60% (1,1,3)

6.3 STATISTICALLY INFERRING CUTTING POINTS FOR ANOMALY DETECTION

For the next set of experiments, we have considered the simple case where the noise levels remain the same across training and deployment environments. More specifically, the experiments aim to answer the following questions: (a) Can we statistically infer alternative cutting points that are optimized for anomaly detection? (b) Are these parameter values applicable for the detection of alternative instructions?

From our observations, we noticed that as noise levels increase, the number of singular values that need to be retained decreases. This is in accordance with the dominant approaches in the bibliography. However, unlike common practices, we opt to retain more than the usual number of singular values than the perceived ideal. Let us consider the case for the case of -10dB (extremely high levels of noise). Traditional approaches typically dictate the retention of just one singular value. By retaining four singular values instead, we achieve, on average, a better TPR to FPR

Table 6.3: Average AUC score achieved for the detection of two malicious injections when the same noise conditions apply during training and testing phases. In parentheses, the value for the cutting point for the training set, the cutting point for the testing set, number of neighbors for the LOF algorithm.

Injection of ADD					
SNR	**10**	**5**	**0**	**-5**	**-10**
	99.54% (25,25,3)	99.46% (15,15,3)	98.84% (10,10,3)	97.45% (6,6,5)	92.20% (4,4,51)
Injection of JMP					
SNR	**10**	**5**	**0**	**-5**	**-10**
	99.84% (25,25,3)	99.51% (15,15,3)	99.00% (10,10,3)	97.47% (6,6,5)	93.07% (4,4,51)

rate, despite the denoised signals retaining a significant amount of noise. Yet, a dichotomy exists. Discarding many singular values may lead to the elimination of important characteristics indicative of anomalies, while retaining too many many singular values may result in retaining noise that may conceal or distort the artifacts of interest.

Therefore, across all considered noise environments, by increasing the value of the cutting point, we are able to achieve a significant increase in predictive accuracy. More specifically, a near-perfect accuracy is achieved, i.e., close or above to 99% AUC score for environments with low to moderate noise levels, i.e., from 10dB to 0dB. For more noisy environments, e.g., -5dB up to extremely high levels of noise, i.e., -10dB, the predictive accuracy drops at a slow pace but remains above 90% levels. Detailed results for each experiment, along with the chosen hyperparameters, are given in Table 6.3.

Interestingly, the values for the cutting points and the number of neighbors are the same for the two types of injections. This indicates that the identified parameters for a given type of injection at a specific noise level are potentially *transferable* across different types of injections.

By plotting those values 6.1, we can statistically derive the equation 6.1 that estimates the value of the optimal cutting point based on the active noise level. More specifically, the formula is given as:

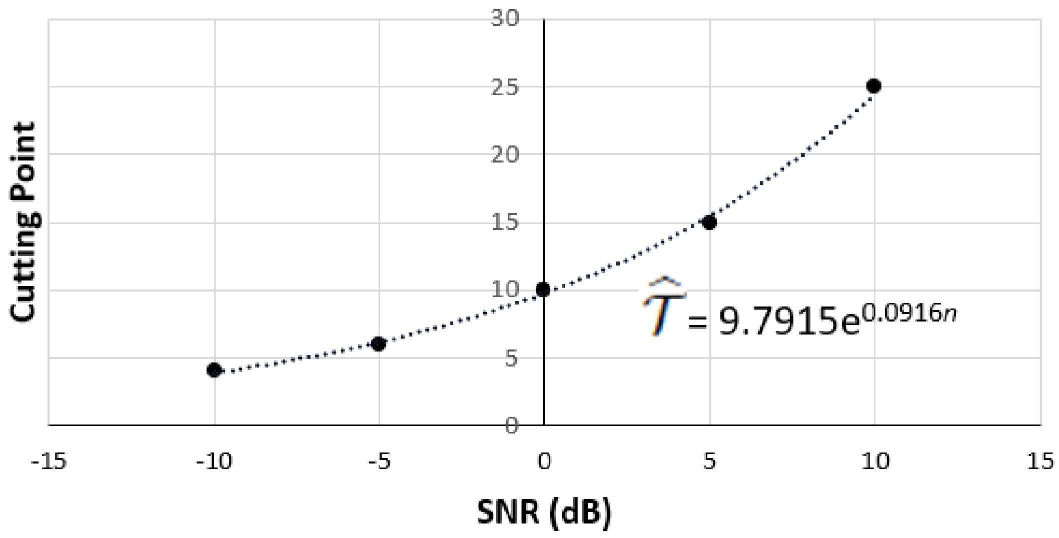

FIGURE 6.1: Statistical analysis of cutting points based on the noise level.

$$\widehat{\mathcal{T}} = \alpha \cdot e^{\beta \cdot n} \tag{6.1}$$

where α and β are constants specific to the environment. In our experiments, the values for these parameters are $\alpha = 9.7915$ and $\beta = 0.0916$; n is the noise level in the training and deployment environments; $\widehat{\mathcal{T}}$ is the value for the cutting points that need to be applied to reduce noise in the training and deployment environments. In other words, $\widehat{\mathcal{T}}$ is a number of singular values that will be retained during denoising. **Conclusions:** *(a) SVD denoising with cutting points that lead to the retention of more singular values provides high predictive accuracy even for minimal code injections and for extremely high levels of environmental noise; (b) It is possible to statistically infer a formula for identifying (near-optimal) cutting points for anomaly detection, regardless of the type of injection.*

6.4 EVALUATING CONSIDERING VARIABLE NOISE LEVELS

Our next set of experiments focused on quantifying the accuracy of our system in variable noise domains; more specifically, we considered the case where we trained our baseline at a specific noise level, but the noise level increased during the deployment stage. As in the previous experiment, we relied on the equation 6.1 to identify optimal cutting points for fingerprinting and deployment environments separately. The results of this experiment are provided in table 6.4. The reader can see that the AUC score decreases as the difference between the noise level between the two environments increases.

Subsequently, we posed the question of whether these results are optimal we can achieve. Based on exhaustive experimentation regarding the values of the cutting points (brute force), we obtained alternative values for the hyperparameters. The optimal results are reported in table 6.4. Intuitively, one may recognize that the choice of optimal cutting points may not only depend upon the active noise levels but possibly on the difference in noise levels between fingerprinting and deployment.

This observation, allowed us to derive a new formula that takes into account both the original and the new noise levels. Specifically, the formula is defined as follows:

$$\mathcal{T} = \lambda \cdot e^{\gamma \Delta_{SNR}} \cdot \widehat{\mathcal{T}} \tag{6.2}$$

where $\mathcal{T}$ and $\widehat{\mathcal{T}}$ are the values for the cutting points that need to be applied to reduce noise in the training and deployment environments respectfully; λ and γ are constants specific to the environment. In our experiments, the values for these parameters are $\lambda = 2.255$ and $\gamma = 0.1637$; Δ_{SNR} is the difference between the levels of noise in the training and deployment environments.

With this new formula, we repeated the experiments by utilizing cutting points as they were dictated by the new formula. To test our findings, we run the experiment with a cutting point during fingerprinting calculated by 6.2 and a cutting point for deployment based on equation 6.1 with $\alpha = 8.913$ and $\beta = 0.1068$. The result of this experiment is presented in table 6.6. The reader can clearly see a significant increase in predictive accuracy across all noise levels that reaches near-optimal levels.

Figure 6.2 shows the comparison between AUC scores achieved when cutting points are identified through exhaustive search (red), predicted by equation 6.1 (green), and AUC based on parameters predicted with the utilization of equation 6.2 (blue). This image compares performance when fingerprinting is done in different noise

Table 6.4: AUC scores across different noise levels. In parentheses, the parameters provided as input for anomaly detection algorithm: cutting points for training and testing determined by formula 6.1 and number of neighbors for the LOF algorithm.

Injection of ADD						
test SNR		**10**	**5**	**0**	**-5**	**-10**
training SNR	**10**	99.54% (25,25,3)	98.91% (25,15,3)	97.92% (25,10,3)	95.71% (25,6,3)	86.25% (25,4,25)
	5	-	99.46% (15,15,3)	98.02% (15,10,3)	96.67% (15,6,5)	86.88% (15,4,25)
	0	-	-	98.84% (10,10, 5)	86.89% (10,6,5)	87.17% (10,4,25)
	-5	-	-	-	97.45% (6,6,5)	87.28% (6,4,51)
	-10	-	-	-	-	92.20% (4,4,5)
Injection of JMP						
test SNR		**10**	**5**	**0**	**-5**	**-10**
training SNR	**10**	99.84% (25,25,3)	98.97% (25,15,3)	98.12% (25,10,3)	96.20% (25,6,3)	88.37% (25,4,25)
	5	-	99.51% (15,15,3)	98.33% (15,10,3)	96.94% (15,6,5)	88.72% (15,4,25)
	0	-	-	99.00% (10,10, 3)	96.99% (10,6,5)	88.90% (10,4,25)
	-5	-	-	-	97.47% (6,6,5)	88.94% (6,4,51)
	-10	-	-	-	-	93.07% (4,4,5)

domains and deployment is done at -10 SNR. As can be seen, the new formula can choose denoising parameters that approximate near optimal levels.

Table 6.5: AUC scores across different noise levels with cutting points selected through brute force. In parentheses, the parameters provided as input for anomaly detection algorithm: cutting points for training and testing and number of neighbors for the LOF algorithm.

Injection of ADD						
test SNR		**10**	**5**	**0**	**-5**	**-10**
training SNR	**10**	99.84% (61,25,3)	99.85% (54,14,3)	99.59% (100,10,5)	98.78% (200,6,9)	90.01% (200,3,20)
	5	-	99.83% (38,15,3)	99.66% (50,10,5)	98.81% (50,5,25)	90.12% (76,3,25)
	0	-	-	99.77% (25,10, 5)	98.84% (25,5,25)	90.41% (29,3,25)
	-5	-	-	-	99.43% (13,5,5)	91.20% (15,3,11)
	-10	-	-	-	-	94.20% (7,3,11)

Injection of JMP						
test SNR		**10**	**5**	**0**	**-5**	**-10**
training SNR	**10**	99.95% (61,25,3)	99.92% (55,15,3)	99.63% (100,10,5)	98.86% (200,6,9)	92.20% (200,3,20)
	5	-	99.94% (37,15,3)	99.70% (50,10,5)	98.90% (50,4,25)	92.29% (75,3,25)
	0	-	-	99.79% (21,7, 5)	98.94% (25,5,25)	92.31% (29,3,25)
	-5	-	-	-	99.47% (14,5,5)	93.15% (15,3,11)
	-10	-	-	-	-	95.33% (7,3,11)

Table 6.6: AUC scores are based on cutting points predicted using formula 6.2

Injection of ADD						
test SNR		10	5	0	-5	-10
training SNR	10	99.84% (59,26,3)	99.75% (77,15,3)	99.12% (104,9,5)	97.46% (131,5,5)	89.47% (179,3,11)
	5	-	99.88% (34,15,3)	99.66% (46,9,5)	97.69% (58,5,5)	89.70% (79,3,11)
	0	-	-	99.77% (26,9, 5)	98.42% (25,5,5)	90.03% (35,3,11)
	-5	-	-	-	99.34% (11,5,5)	91.20% (15,3,11)
	-10	-	-	-	-	94.20% (7,3,11)
Injection of JMP						
test SNR		10	5	0	-5	-10
training SNR	10	99.95% (59,26,3)	99.87% (77,15,3)	99.17% (104,9,5)	97.64% (131,5,5)	91.31% (179,3,11)
	5	-	99.94% (34,15,3)	99.73% (46,9,5)	97.84% (58,5,5)	91.41% (79,3,11)
	0	-	-	99.62% (26,9, 5)	98.56% (25,5,5)	91.61% (35,3,11)
	-5	-	-	-	99.34% (11,5,5)	91.20% (15,3,11)
	-10	-	-	-	-	95.33% (7,3,11)

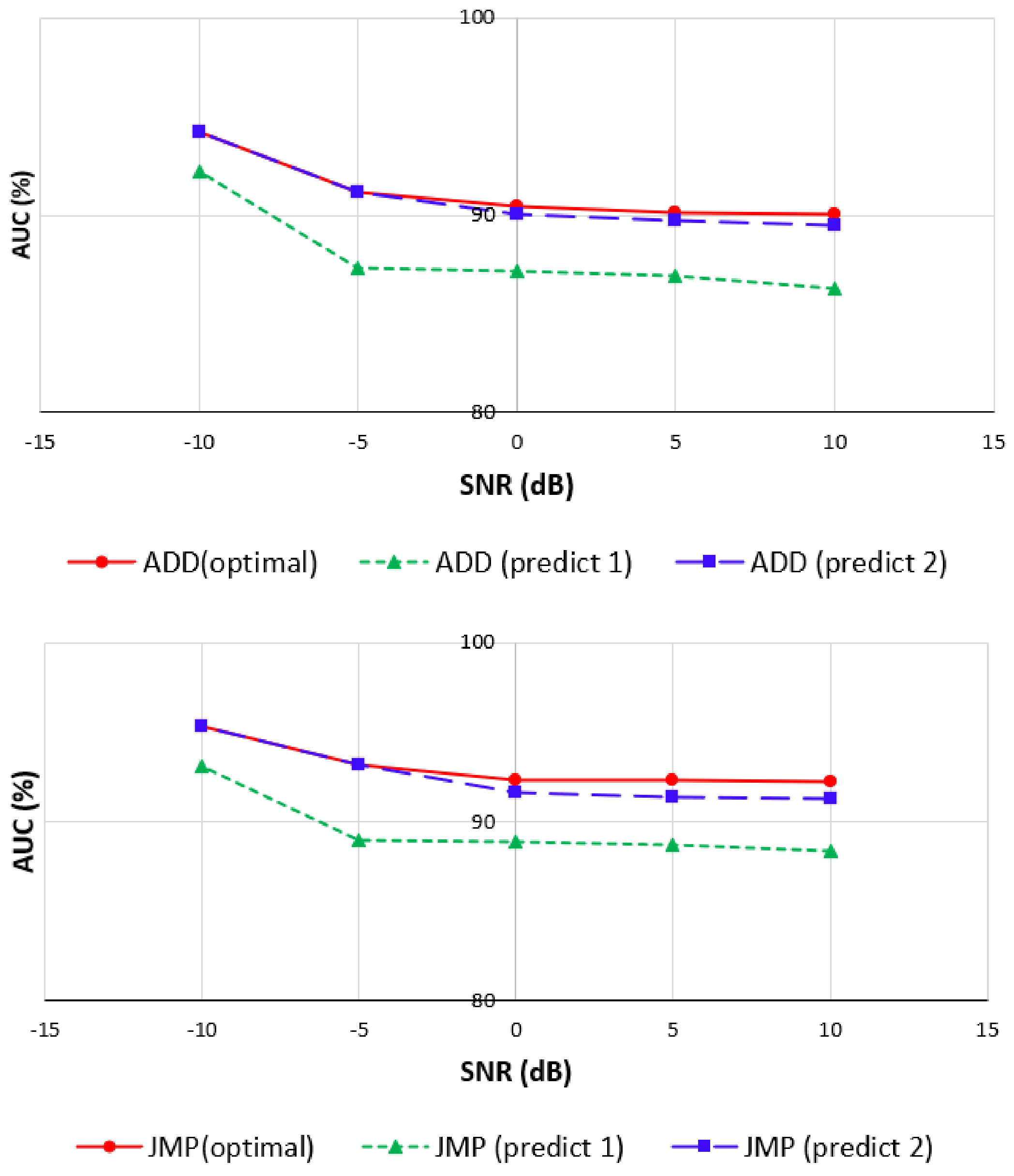

Figure 6.2: Comparison between optimal AUC, AUC based on parameters predicted by equation 6.1 only, and AUC based on parameters predicted by equation 6.2 and equation 6.1.

CHAPTER 7

Discussion

This chapter intends to give a deeper understanding of the evaluation numbers we saw in Chapter 6. It explains why denoising is a necessary step and how the matrix size and cutting points affect denoising. If we compare normal signals with abnormal ones, as in Figure 7.1, we can see that they have the same frequencies and are only slightly deferred by the amplitude of some "peaks."

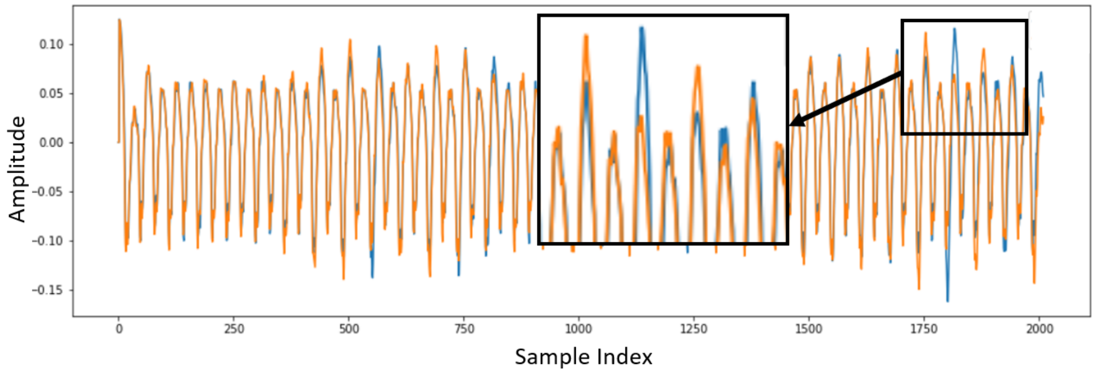

Figure 7.1: Normal signal(blue) vs. anomalous with ADD injection (orange) in the noise-free dataset.

By analyzing the clean signals, we can see that the difference in amplitude is consistent through all signals. In Figure 7.2 top, the reader can see the boxplots for the height of the "peaks" before (crest 3) and after (crest 32) injection. When there is no noise, this difference in amplitude is enough for our evaluation model to distinguish between normal and abnormal signals. However, as noise increases, the difference between the "peaks" heights is impossible to see (Figure 7.2 middle). Finally, if we denoise the dataset, the distance between hights is visible again (Figure 7.2 bottom) and can be used for anomaly detection.

Our anomaly detection algorithm is described in section 4.3, uses k nearest neighbors distance from the signal collected during the deployment to the baseline determined during the training phase. Figure 7.3 shows the average three nearest neighbor distances (3-NN) for the individual signals from the testing dataset to the training dataset. The training dataset (green) contains only normal observations and is used as

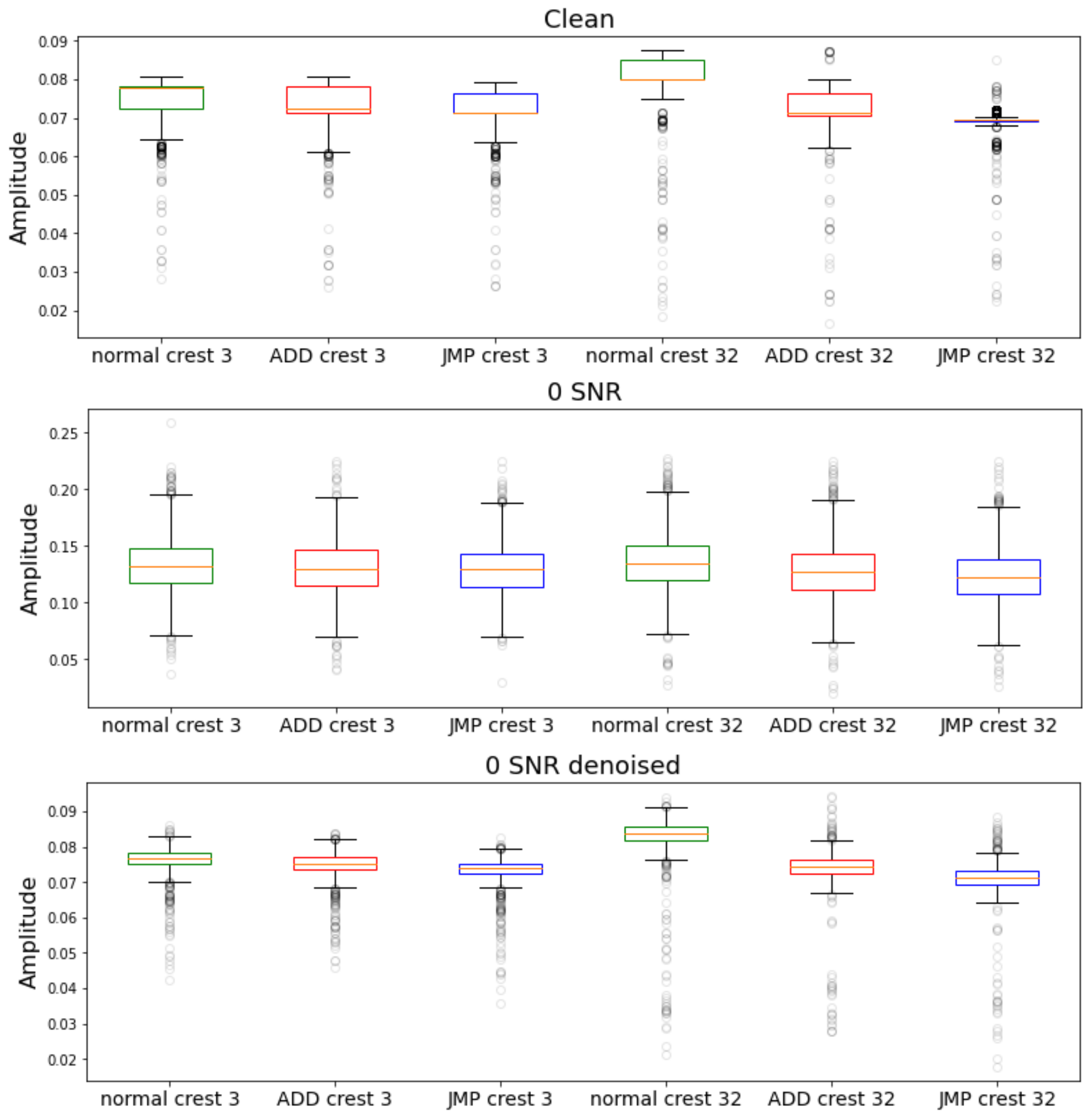

FIGURE 7.2: The distribution of the crests ("peaks") heights in the dataset normal vs. abnormal (with ADD injection) and abnormal (with JMP injection) before injection (crest 3) and after injection (crests 32).

a baseline. The testing dataset contains normal (blue) and abnormal signals (red) with ADD injection. It is easy to see that, with the exception of a few outliers, both normal and anomalous signals concentrate within specific distance ranges that are clearly separable when the signal is noise-free. However, if noise is present, the separation between normal and abnormal signals is impossible to distinguish.

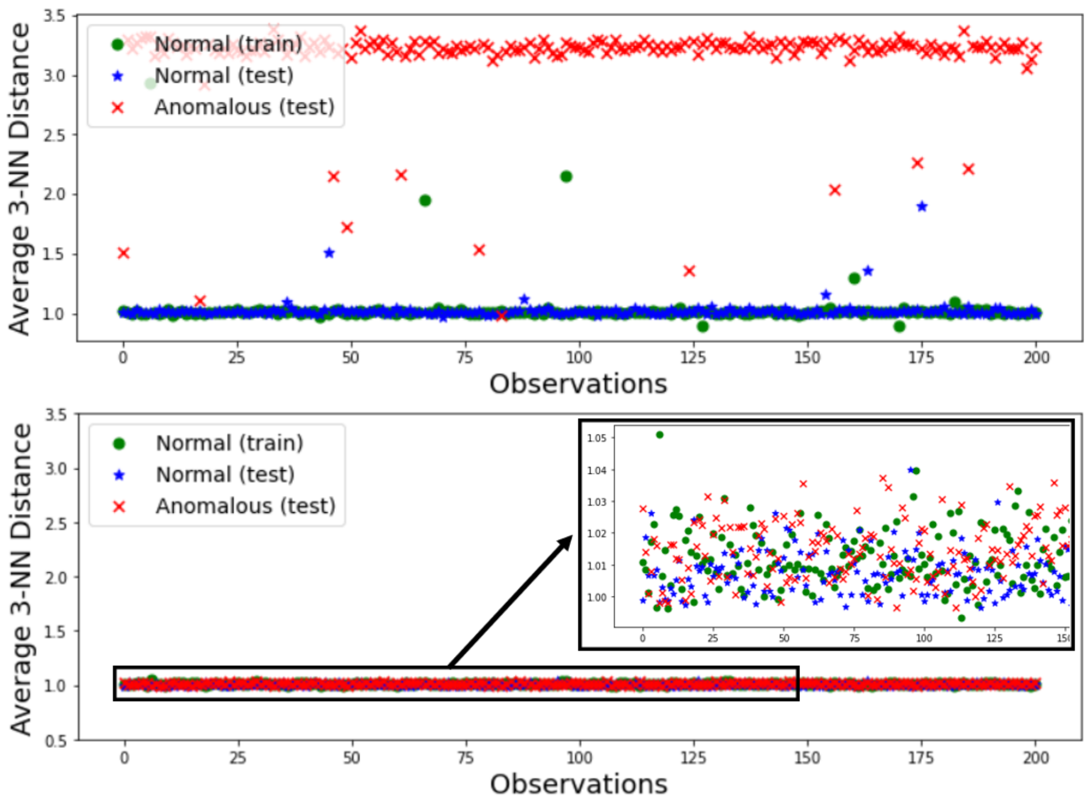

FIGURE 7.3: The average distance of the three closest neighbors of each individual signal (testing normal and testing anomalous) in the dataset to the totality of normal signals (training) that are considered as the baseline in a clean dataset (top) vs. a noisy dataset (bottom).

The reader can see that a noise reduction must be applied to the dataset. Since the given task is anomaly detection, the chosen methodology should be as non-destructive as possible and retain the maximum number of artifacts specific to the anomaly.

7.1 CONSIDERATIONS REGARDING THE NUMBER OF EXAMPLES

First, we investigate the effect of the matrix size on the quality of denoising. We denoise train(all signals are normal) datasets where the number of examples is around 20%, 30%, 50%, 100% of the number of samples in each signal. Figure 7.4, shows that as the number of signals gets closer to the number of observations in each signal, the denoising effect improves.

We have observed the same result in the testing dataset; as the number of examples gets closer to the number of observations in each signal, there is a lower

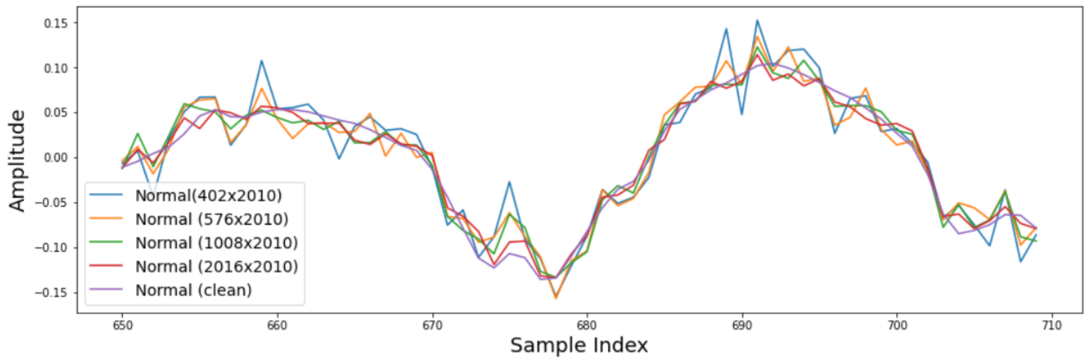

FIGURE 7.4: Signals in the -10 SNR train dataset after denoising using cutting point 4 for the matrixes size 402x2010, 576x2010, 2016x20010.

deviation in the denoised signals (Figure 7.5). By increasing the number of signals, we are able to achieve better denoising results without losing important artifacts. A similar observation was made by Jha and Yadava (2010). They stated that when the number of observations(in their case vapors) is more than the number of samples in each observation (in their case sensors), the noise variable of "vapor origins" is more effectively eliminated than the noise variable of "sensor origins."

Size of the Testing Dataset	**Average AUC**
402 x 2010	50.17%
576 x 2010	59.34%
1008 x 2010	73.60%
2016 x 2010	84.57%
4032 x 2010	92.20%

TABLE 7.1: The AUC score is based on the size of the matrix for denoising the testing dataset (normal + ADD)

The same result is confirmed by the experiment. In the Table 7.1, we denoised the training dataset using a matrix where the number of signals is equal to the number of observations in each signal. Then we created train datasets, where the number of signals is around 20%, 30%, 50%, 100%, and 200% of the number of observations in each example. The reader can see that AUC is increased as the number of rows in the matrix increase.

From observations of numerous test cases, we concluded that the number of observations that is around twice the number of samples of the signal is the best to achieve a high-quality noise reduction effect without negatively impacting the anomalous artifacts. However, keep in mind that a trade-off exists. By collecting more signals, more time is required to elapse between code injection and detection. This might be an issue for some applications with real-time restrictions.

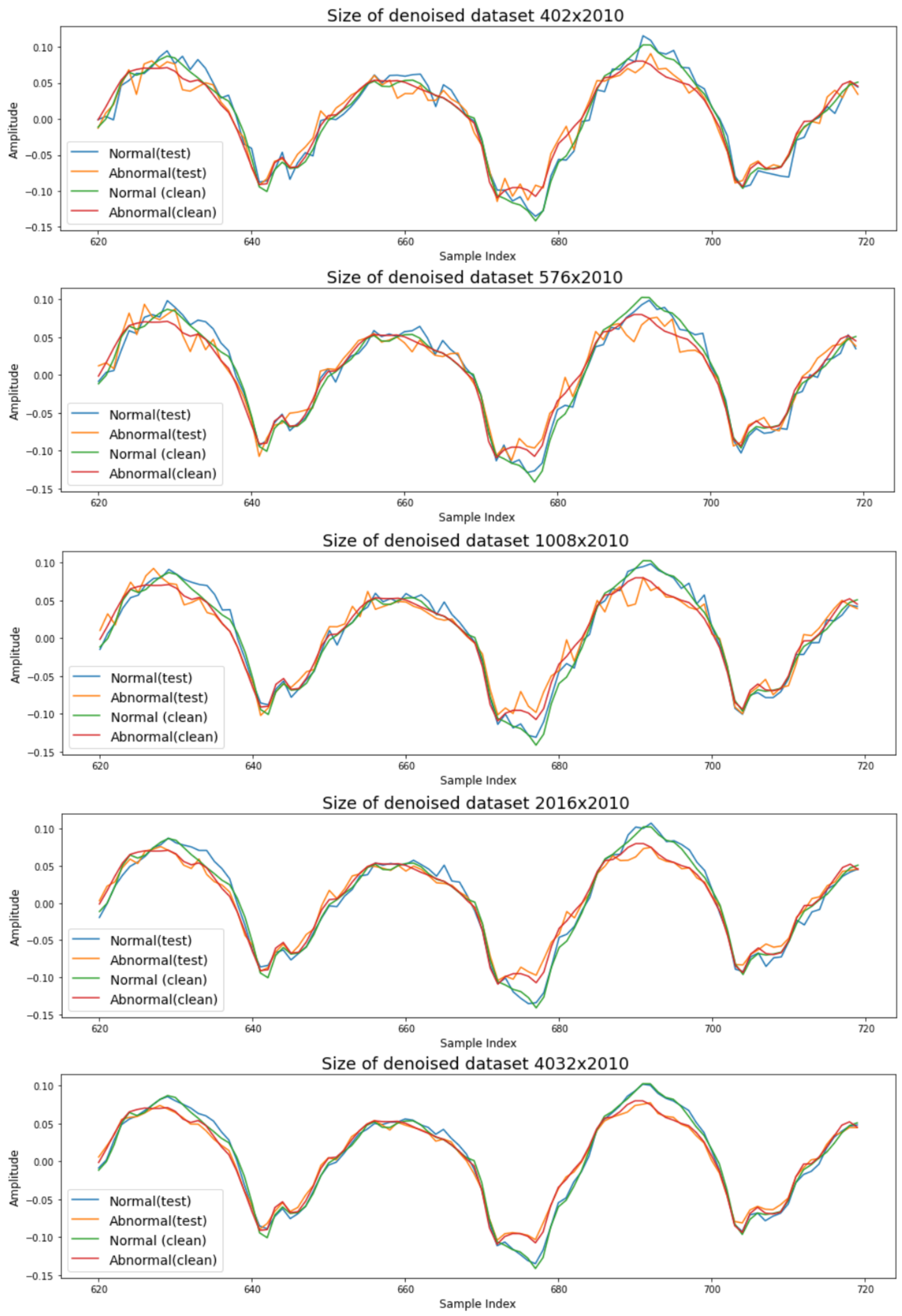

FIGURE 7.5: The effect of the number of observations vs. the number of samples in each observation on the quality of denoising.

CHAPTER 8
Related Work

The embedded devices are growing in popularity and can be found in numerous sectors, including energy, medical, security, etc. As a result, the malicious attacks on those devices increase as well. As an alternative to on-device detection, there is a growing interest in developing remote malware detectors that leverage physical side channels.

Han *et al*. (2019) presents a good overview of the side channel-based code execution monitoring systems. The paper mainly concentrates on power consumption and EM emanation. It shows that power consumption has very few variations between different runs of the same instruction than EM signal, which is more susceptible to environmental noise and interference. In addition, the authors go over how the Hidden Markov Model, a neural network, and Principal Component Analysis (PCA) can assist in detecting anomalies in embedded devices during code execution. They give a basic idea of how to construct a monitoring system using a side channel. This system is based on two phases: profiling and deployment. During the profiling, a signal is collected, preprocessed, and a profiling model is established. During the deployment phase, the model is used to evaluate the collected signal and report anomalies. Finally, the authors discuss the challenges this monitoring system present: determine the window length for each code segment, the effect of the size of the loops in the program, and the processor specifications the program is running on. The paper does not address the white noise problem.

8.1 EM ANALYSIS AS AN ATTACK VECTOR

Several papers use EM emanation as an attack vector. In essence, extracting information from the device for malicious purposes has strong similarities to extracting information for monitoring/protection monitoring purposes. Because of this, we find the following papers relevant to our work.

Sehatbakhsh *et al*. (2020b) describe a new micro-architectural vulnerability that is created by power management units of modern computers and can be exploited through EM emanation. Modern computers are using multiple power-management techniques to save battery life. One of them is voltage-frequency scaling, where the processor's clock frequency can be adjusted dynamically depending on the required level of performance. The other technique is placing unused units within the processor into a low-power state. The module that supplies power to the core is called Voltage Regulator Module (VRM). The power module contains a capacitor that VRM tries to keep filled to the desired voltage level. Every time a capacitor gets charged (typically every 1-4 microseconds, called the switching period), the charge creates a burst of EM current with a powerful component at a frequency that has a very high amplitude compared to other electrical current flows in a computer system. To improve the power consumption, the process called phase shedding skips the capacitor charging if the load is low. As a result, the "spikes" in an EM signal are prominent under a high load and much weaker when the load is low. Those "spikes" can be detected with a lower-cost radio receiver or a loop antenna. The authors create a covert channel, a simple program that reads the file on the victim's computer. If a bit is 1, it runs a loop for LOOP_PERIOD and then sleeps for SLEEP_PERIOD. If a bit is 0, the program sleeps for two SLEEP_PERIOD. By measuring EM emanation, an attacker can read the files on a victim's computer. Through experiments, the authors find that SLEEP_PERIOD and LOOP_PERIOD should be no less than 10μs. Specifically, they use 100μs for UNIX-based machines and 1ms for Windows-based machines. The authors conclude that all other background activities produce a small burst of amplitude that is usually shorter than SLEEP and LOOP PERIODs and can easily be distinguished. In the experimental setup, the authors have been able to detect signals from up to 2.5m away and from 1.5m away behind a 35cm office wall with a working printer and refrigerator as a sort of additional EM emanation. In addition, the authors demonstrate that by measuring EM emanation on modern laptops, the attacker could identify keystrokes and distinguish words.

Khan *et al*. (2018) use EM emanation to infer a cryptographic key from a device running an OpenSSL. To achieve it, the authors determine the instruction that the CPU executed. They start with AM demodulating EM signals at the processor clock frequency and low-pass filtering. They create a dictionary of all possible EM signatures corresponding to different branch decisions during the training phase by executing encryption with known keys. They use a 1-NN algorithm with Euclidean distance to match the capture signal with the dictionary during the detection phase. They achieve a >90% accuracy rate for minimal differences in program execution (one instruction) even when the sampling rate is only 4% of the monitoring system's clock cycle rate and an SNR of 20dB or better. They observe that accuracy increases as an SNR improves and the sampling rate increases. In addition, they show that large differences (a tenth of instructions) can be identified with an accuracy rate >99% even when the sampling rate is only 2% of the processor's clock cycle rate, and SNR is only 5dB.

Camurati *et al*. (2018) presented a new channel attack named *screaming channel* that affects mixed-signal chips used in wireless communication protocols, such as Bluetooth and WiFi. Mixed-signal chips include digital and analog circuits on the same silicon die in close physical proximity. Because of this proximity, noise from the digital circuit leaks to the analog circuit. This noise corresponds to processes running by digital logic. As a result, sensitive information from the digital domain gets propagated to the radio transmission, which broadcasts it at a much larger distance than normal EM emanation. The authors investigate this effect on several popular devices and demonstrate a completely cryptographic key recovery from AES-128 implementation in tinyAES at a distance of 10m in a controlled environment using template attacks. The template attacks are based on aligned traces of the signal, each corresponding to a single execution of the computation under attack. To capture radio emission, they used software-defined radio (SDR) tuned to $f_{chan} + 2 \cdot f_{clock}$, where f_{chan} is the center frequency of the Bluetooth or WiFi channel and f_{clock} is the frequency of the microprocessor's clock. They split the signal into individual traces and aligned the traces. The authors use the point-wise mean of all aligned traces as a measurement for a new alignment. Averaging aligned traces removes random noise, making key recovery easy. To recover the cryptographic key, the authors used around 70000 traces with 500 timestamps to create a template. This allowed them to use only

428 traces during the attack in a controlled environment. For the attack to succeed in an uncontrolled environment (office), 52589 traces need to be collected. In addition, the number of traces increases as the distance to the target increase. The authors note that the full range of devices affected by this attack still needs to be determined.

8.2 EM BASED MONITORING SYSTEMS

Apart from utilizing the analysis of EM signals as an attack vector, several works have attempted to apply such techniques for protection purposes e.g., to achieve external control flow attestation.

Nazari *et al*. (2017) describes an anomaly detection framework based on the analysis of EM signals, namely EDDIE. It creates a model of normal program execution using Short-Term Fourier Transforms (STFT). Computing FT in each window will show frequencies for this time period instead of frequencies for the whole signal. Looking at the different windows demonstrates how frequencies change over time. The authors call the frequency domain for each window Shot-Term Spectra (STS). EDDIE gets trained on sequences of STSs obtained for each loop nest and each loop-to-loop transition. The training phase starts with adding instrumentation during compilation. EDDIE then runs the application multiple times with different inputs. It is necessary to obtain signals to the regions that are not executed all the time. Since there are pieces of code that get rarely executed, collecting signals for those pieces requires specially crafted input data. The signal collected during training is then divided into windows. Finally, for each region in the program, EDDIE collects the set of sample windows that belongs to that region. Due to noise, "random" variation in program behavior, or micro-architectural events like cache misses, different STSs that belong to the same code region are not exactly the same. To compare the observation during the testing phase to the STSs distribution, EDDIE uses a nonparametric statistical test. The authors have discovered that Kolmogorov-Smirnov (K-S) nonparametric test works better for their purpose. Kolmogorov-Smirnov (K-S) test determines the differences in the shapes of two sample distributions. It uses the critical value of D that is found in the K-S table, where is the level of significance. The Null Hypothesis is that two samples are drawn from populations with the same distribution. The Alternative Hypothesis is that two samples are drawn from different populations. The maximum absolute difference between the two cumulative distribution functions is calculated. If this value is less than the critical value, then Null Hypothesis is accepted. Otherwise, the Null Hypothesis is rejected, and an Alternative Hypothesis is accepted. Another challenge the authors have encountered has been choosing the number of monitoring-observed STSs that would be tested in each K-S test. This number should be large enough to provide a near-zero false positive rate but not too large because it will affect detection latency. The problem is that this number is different for each region. The authors apply the K-S test to the train STSs for each region and choose the smallest number that provides the minimum false positive rate to overcome this problem. The authors test EDDIE on a few instructions (2,4,6,8) injected inside the loop, and significantly more instructions (100K, 187K, 218K, 315K, 400K, 500K) injected outside the loop. They have been able to achieve accuracy from 93% to 100%.

Sehatbakhsh *et al.* (2018) extend the framework presented in EDDIE. Their new system is called SYNDROME - Spectral analYsis for aNomaly Detection On Medical and Embedded devices. The authors evaluate Syndrome on real word medical device called a Syringe Pump that is designed to dispense or withdraw a precise amount of fluid and is used at hospitals to administrate a small amount of medication to the patient at frequent intervals. The authors execute an exploit that performs buffer-overflow and hijacks the control flow of the syringe pump application. Similar to EDDIE, the training phase of the SYNDROME starts with a statistical analysis of the application to determine the loops and their possible ordering. Then they perform train runs with knowing input to collect EM signal. This signal gets converted into a sequence of sample spectra, where each spectral sample corresponds to a 1 ms window of time and overlaps 75% with the previous sample. Those samples get categorized based on the region of the program they belong to. After categorizing samples, the features (spikes in frequency domain corresponding to loops) get extracted. During the detection(monitoring) phase, the EM signal gets converted into spectral samples. The features from those samples get extracted and compared to train samples using K-S statistical test. The samples of the monitoring signal get tested against the train time samples for the first region. If the test fails, that means that the malicious activity has begun or the code has switched to another region. In this case, all possible regions get tested. If all of those possibilities are rejected, the Syndrome reports malicious execution. One of the main challenges authors run into was handling the operation system's interrupts. They found out that the spectral features of the interrupt activities are very similar to each other, which allows them to create a training-time model for interrupts. When the K-S test rejects the sample, the sample gets tested against training-time samples of interrupt-related activity. In summary, it took SYNDROME lees then 2ms to detect every occurrence (100% true positive rate without false positives) of an actual attack on an embedded medical system.

Han *et al*. (2017) have proposed Zeus, a contactless monitoring system for embedded devices that uses electromagnetic emission. Zeus runs the program on the PLC for each execution path and collects the electromagnetic emanations using an external sensor. The collected signal traces, along with their labels (corresponding control lows), are fed to a sequence neural network classifier for training. Since the execution time of the individual instructions is not fixed, Zeus looks at frequency representations of signals within a local sliding window. Since electromagnetic signals suffer from random perturbation, 1000 signal traces are collected for each instruction type for the classifier training and the validation testing. To construct the classification model with sequential inputs, the authors use a Long Short-Term Memory (LSTM) network layer. LSTM is a variation of the Recurrent Neural Network (RNN) used for modeling sequential data. A stacked two-layer LSTM network with 100 nodes on both layers is used to fingerprint the execution behavior of each program. The authors train the network using Stochastic Gradient Descent (SGD). An average of 50 epochs (iterations) are required for the network to converge on the tested programs. The authors obtain 100 traces for every feasible control flow to train the model. 2-fold cross-validation is performed ten times to stabilize the result. After deployment, Zeus feeds the spectrum sequences of the collected signal into the trained model. The model computes a probability distribution for each sequence over all the classes. The class with the highest score is compared to a predefined threshold. If the threshold check fails, the system indicates a mismatch between the query signal trace and the trained model. Consequently, it triggers an alert about an illegitimate control low. To evaluate Zeus's performance, the authors use ten real PLC programs from different application domains. The area under curve (AUC) evaluation technique shows the worth performance at 96% and the best at 100%.

Chawla *et al*. (2021) presents an EM signal processing approach that uses machine learning (ML) to detect a malicious application running on the device and classify the application into a malware family. Instead of being trained on trusted execution behavior, the proposed model learns characteristics of different malware families from their EM side-channel signatures by running multiple malware samples. The collected EM signal is segmented into short windows, Short-time Fourier Transform (STFT) is computed for each window, and a spectrogram is generated. The spectrogram shows Power Spectral Density (PSD) variations of the EM side-channel trace over time. Then, a two-dimensional Discrete Wavelet Transform (2D-DWT) is used the resulting coefficients are used as features. After this, the authors apply Principal Component Analysis (PCA) to reduce the feature dimensions. Finally, the ML model gets to train on the resulting reduced dimensional feature vector. Specifically, Random Forest (RF) Classifier and Support Vector Machines (SVM) are used to learn unique futures for each malware family based on the malware samples. During the testing process, the above steps are repeated. The feature vector is tested on the trained ML model. The authors achieve an F1 accuracy score of 99% with the SVM model and 97% using RF. In addition, the authors demonstrate the software update capability of the proposed framework to detect unknown applications or malware families and re-training model on new features.

Khan *et al*. (2019) introduces another intrusion detection framework called IDEA based on an EM side-channel signal. Unlike the approaches described above, it does not use instrumentation and works in the time domain. During the training phase, IDEA collects the EM signal of the trusted program during multiple executions. The collected EM signal is split into multiple overlapping windows that are recorded as "words." Then mean subtraction and scale normalization are applied to each word.

This step is necessary to ensure that matching is based on the shape of the signal rather than the actual amplitude. There is a chance that testing and training signals will have different amplitudes due to the position of the antenna. After this, words get separated into clusters. Since the number of clusters is unknown, popular clustering algorithms are not feasible for this work. Instead, threshold-based clustering is used. In summary, the random word is chosen as a cluster centroid. Every word in the dictionary whose Euclidian difference to this centroid is less than the threshold gets assigned to the cluster. After this, the cluster centroid gets updated by averaging all cluster members, and Euclidian distance is recalculated. The process repeats until assignments no longer change. After this, a new cluster centroid is selected from words that have not been assigned to any cluster. After all words have been assigned to the clusters, cluster centroids create a dictionary of words. During the monitoring phase, the EM signal is split into windows and matched against the words in the dictionary using the 1-Nearest Neighbor algorithm, with Euclidian distance as a distance metric. The signal is then reconstructed by using these words. The reconstructed signal is compared with the monitoring signal, and the square difference is reported. The result is compared to the threshold; when this threshold is breached, the intrusion is reported. The IDEA is able to detect different attacks with AUC >99.5% for high SNR (10SNR and above).

8.3 SVD DENOISING

As seen from the work mentioned above, we can achieve good anomaly detection results using EM emanation as long as we collect signals at a low noise level. When the noise level is significant, additional preprocessing techniques should be applied. In this work, we choose SVD denoising because it shows better results than other technics based on the work presented below.

Govindarajan *et al*. (2019) use SVD denoising to remove the random noise from the Partial Discharge (PD) signal. In electrical engineering, PD is a localized nonperiodic electrical discharge (impulse) that occurs for a duration of 10^{-9} to 10^{-7}sec. PD occurs whenever there is a stressed region due to some cavity inside the insulation under high voltage stress. PD indicates the insulation weakness at the beginning of destruction. Timely indication helps to prevent catastrophic failure. The authors apply the Hankel Matrix (HM)-based Fast Singular Value Decomposition (H-FSVD) technique to one signal at a time. The authors use HM on the whole signal and then reduce the HM to a tridiagonal matrix using the Lanczos method. Then, they calculate the SVD of the tridiagonal matrix using the twisted factorization method. Next, they use Singular Spectral Analysis to zero out singular values corresponding to noise and reconstruct the noise-free signal. They test their method on three different data sets (synthetic and two real with added synthetic noise -10dB and 10 dB). Their result shows that this method is superior to the Wavelet Transform (WT) based method and Empirical Mode Decomposition (EMD) method with respect to noise removal. This paper concentrates on noise removal and not anomaly detection.

Zhou *et al*. (2019) propose the Adaptive Short-Time Singular Value Decomposition (ASTSVD) to remove a white noise in Partial Discharge (PD) signals. The idea is to use signal segments extracted by a sliding window moving along the time axis of the PD signal. Instead of calculating HM for the whole signal, they create a Hankel matrix for each snapshot. The authors use a number of columns in Hankel matrix $K = T_W - L + 1$, where T_W is the length of the snapshot, and the number of rows $L = T_W/3$. In addition, they propose using the minimum description length (MDL) criterion to determine the number of singular values (cutting point) for SVD denoising. The sliding window width is determined empirically as half of the PD pulse. One of the techniques to identify the pulse mentioned in this work is to calculate the SVD of each sliding window, choose the same cutting point r for each window, and plot them. The graph will have a peak. The beginning of the peak will correspond to the beginning of the PD pulse. The end of the peak will correspond to the end of the PD pulse. The authors have tested the technique for the 0 SNR and have been able to reconstruct the original PD signal without waveform distortion, a typical effect when SVD is performed for the whole signal. They show that their results are better than results obtained using Discrete Wavelet Transform (DWT) and using a technique proposed in Govindarajan *et al*. (2019). The authors note that despite good denoising results, there are still some limitations, including the determination of the optimal sliding window and computation time that is still higher than DWT-based methods.

Zhang *et al*. (2020) takes two-division recursion singular value decomposition and extends it to three- and four-division recursion singular value decomposition and evaluates them. Based on the evaluation, the authors concluded that two-division recursion singular value decomposition is optimal for noise reduction. The idea of two-division recursion SVD is for one-dimensional signal $s = s_1, s_2, \ldots, s_n$ construct a HM with two rows and perform SVD of this matrix $A = \sigma_{11}u_{11}v_{11}^T + \sigma_{12}u_{12}{v_{12}}^T$, where u_{11} and u_{12} are the vectors formed by the first two rows of the first decomposition of matrix U, σ_{11} and σ_{12} are the two singular values obtained by the first decomposition, and v_{11} and v_{12} are the vectors formed by the first two rows of the first decomposition of matrix V. $\sigma_{11}u_{11}v_{11}^T = A_1$ is the approximated signal and $\sigma_{12}u_{12}{v_{12}}^T = D_1$ authors call the detailed signal. This signal gets discarded as noise. The above process gets repeated on A_1 creating A_2 and D_2, etc. As a result, $A = A_M + \Sigma_{i=1}^{M} D_i$, where M is the number of decomposition layers and A_M is the denoised signal. The M is determined experimentally by plotting SNR and Mean Square Error (MSE) for different M and finding the peak in performance. The choice of M will depend on the type of signal. The authors apply the noise reduction model to the diagnosis of faulty bearings. They have been able to successfully remove noise from the signal with noise levels from 20SNR to -5SNR.

As can be seen from a few works mentioned above, SVD denoising usually gets performed on one signal at a time. Examples of matrix denoising usually relate to image denoising that is not related to our study. However, a theoretical base for denoising multiple signals at once exists.

De Moor (1993) discuss why and when the SVD is successful in using geometrical, algebraic, and statistical arguments. The authors have used a noisy matrix of measurement m-by-n, where m is the number of measurement channels and n is the number of measurements in each channel. Specifically, when $m << n$, they call the column space long space and row space short space. Authors give the theoretical justifications why under certain assumptions, the short space of the noise-free matrix can be estimated consistently while the long space cannot be recovered. This work explains why in our experiment, we have been able to achieve a better denoising effect when the number of observations is significantly larger than the number of samples in each observation.

To the best of our knowledge, Jha and Yadava (2010) is the only paper we have come across that uses SVD denoising to remove noise in a matrix of observation with real-life applications. In this work, the authors analyze the work of the SVD in denoising sensor array data of electronic nose systems. They show that this method effectively reduces noise components arising from the order sampling, delivery system, and sensors electronics. Using a neural network classifier, the authors use denoised data in the classification problem. After denoising data, they use PCA to reduce the separation between samples of the same class and increase the separation between classes. The authors use formula $\epsilon \leq \sigma\sqrt{mn}$ to calculate threshold for singular values, where σ is a standard deviation of the noise in m-by-n matrix. Since, in practice, σ is unknown, they use empirical analysis to estimate σ. Authors argue that in sensor array data where $m >> n$, SVD is much more effective in denoising than PCA. They show that for good denoising, the number of rows should be significantly greater than the number of columns. In our work, we have come to the same conclusion.

Gavish and Donoho (2014) have proposed one more approach to calculating the threshold for singular values. Like with all traditional denoising technics, every singular value less than this threshold gets set to 0. Doing so allows the authors to recover a low-rank matrix (denoised matrix in our application) from noisy data. They evaluate their approach using Asymptotic Mean Square Error (AMSE). The authors start their development process by analyzing the "elbow" approach to detect the optimal cutting point. The "elbow" is where the slope of singular values changes dramatically when plotted in descending order. In addition, they look at the alternative approach; in a case where the "elbow" is not detectable (the matrix is exactly or approximately low-rank), the class histogram of the singular values should be plotted. In this case, a quarter-circle "bulk" will be formed on the left side of the graph. The edge of this bulk can be used to determine the cutting point for denoising. Finally, they evaluate the approach in which the threshold is calculated using $\tau = 2.02\sqrt{n}\sigma$, where n is the size of the matrix and σ is the noise level. Based on their analysis they propose an optimal threshold $\tau^* = \frac{4}{\sqrt{3}}\sqrt{n}\sigma \approx 2.309\sqrt{n}\sigma$ for the n-by-n matrix with white noise of level σ. For the non-square m-by-n matrix, they propose $\tau^* = \lambda(\beta)\sqrt{n}\sigma$, where $\beta = m/n$ and $\lambda(\beta) = \sqrt{2(\beta+1) + \frac{8\beta}{\beta+1+\sqrt{\beta^2+14\beta+1}}}$. In the case of an unknown noise level, the threshold can be calculated using $\tau^* = 2.858_{med}$ for n-by-n matrix, where y_{med} is the median of the singular values. For m-by-n matrix, with unknowing noise level the hard threshold can be estimated by formula $\tau^* = \omega(\beta)_{med}$, where $\beta = m/n$ and $\omega(\beta) \approx 0.56\beta^3 - 0.95\beta^2 + 1.82\beta + 1.43$. Using AMSE, the authors show that their approach produces better denoising than the above-mentioned methods. Unfortunately, this technique produces a cutting point similar to the "elbow" approach in our experiment. This cutting point was not sufficient for anomaly detection.

CHAPTER 9
Conclusion

In this work, we proposed an EM-based anomaly detection framework for remotely identifying code injection attacks against embedded devices that operate in ICS environments. We experimentally proved that systems adopting our framework could effectively perform anomaly detection even in environments of dynamically altering noise conditions or extremely high noise levels.

Typically, analogous systems have the inherent advantage of being non-intrusive compared to traditional malware detection systems. The latter family of solutions needs to be natively installed in the target device, which may not always be possible due to the resource-constrained nature of embedded devices. Unfortunately, due to their remote operation, EM-based systems may be negatively impacted by environmental noise. Despite the severity of this issue, most relevant works in the area consider only experiments in controlled laboratory environments with ideal noise conditions. However, in several real-life applications, this assumption is not realistic. For example, in ICS environments, it is natural to assume that the target device will reside in locations where numerous digital devices operate in parallel. Therefore the target device may be subject to environmental white noise and interference.

In this work, we introduced an EM-based anomaly detection framework that is robust against environmental noise. Our framework relies on a modified version of SVD denoising applied as a preprocessing step. Our main contribution to this step revolves around the way we infer the highly sensitive hyperparameters used internally in this step. More specifically, after a battery of experiments, we statistically inferred generic formulas for identifying the optimal value of the parameter *SVD cutting point* that is internally used to define the intensity of noise elimination. Our formulas take into account noise levels of the training and deployment environments, which are trivial to infer. In our case, the task at hand is primarily to achieve optimal anomaly detection; therefore, the criteria for choosing the value of this hyperparameter are different from those dictated in the bibliography. Our approach manages to achieve the necessary

level of noise reduction without losing important artifacts that differentiate normal from abnormal signals. Moreover, through several experiments, we identified that to achieve optimal noise elimination without losing important abnormality features, the number of example observations should be significantly more than the number of samples in each observation. Through experimental evaluations, we demonstrate that the proposed framework can yield a near-perfect accuracy rate (above 99% AUC score) in moderately noisy (above 0 SNR) environments and can maintain a high detection rate, e.g., above 94% AUC score in rather boisterous environments, (-10 SNR).
To summarize, the main **contributions** of this work can be outlined as follows:

- Our proposed framework can be used in highly noisy environments (up to -10 SNR).
- The proposed methodology is highly robust against noise changes.
- The framework does not require any stochastic analysis for the identification of optimal (or near-optimal) hyperparameters. By relying on contributed formulas, such parameters can be identified automatically.
- The methodology proposed can identify never seen before alterations of code without requiring any examples of anomalous signals. Our system gets trained using only normal signals.

9.1 Challenges & Future Work

Despite the advantages of our approach, there are several aspects that still remain challenging. These can be summarised as follows:

- Our framework assumes that the signals analyzed are synchronized. However, in order to collect synchronized signals, we must add an instrumentation code. Adding the instrumentation might not always be desirable or even possible in certain applications. The use of alternative metrics that involve a sliding window, e.g., *convolution* might be beneficial in such situations.

- So far, we have identified that the number of signals during the fingerprinting stage, ideally, must be twice the number of observations in each signal. Unfortunately, for large examples of code that involve hundreds of thousands of samples per observation, collecting a buffer of this size may compromise the real-time requirements of certain applications. Towards this end, we plan to evaluate a method that fragments given signals in a fixed size of windows, e.g., of 100 samples.

Besides providing solutions to these challenges, in the near future, we plan to conduct high-scale experiments to gain insight into how the location of the code injection affects the accuracy of our approach. Moreover, our long-term goals include expanding our framework to detect code injection attacks in the presence of internal or external interference. More specifically, we would like to identify how the system interrupts and the parallel execution of processes affects the precision of our methods. Finally, we plan to test the robustness of the proposed solution under adversarial attacks that aim to trigger misclassifications through the dissemination of well-crafted, malicious noise.

Bibliography

Abdi H. and Williams L.J. 2010. Principal component analysis. Wiley interdisciplinary reviews: computational statistics 2:433–459.

Anton S.D.D., Lohfink A.P., and Schotten H.D. 2019. Discussing the feasibility of acoustic sensors for side channel-aided industrial intrusion detection: An essay. *In* Proceedings of the Third Central European Cybersecurity Conference, pages 1–4.

BarbaraĎ., Domeniconi C., and Rogers J. 2006. Detecting outliers using transduction and statistical testing. *In* Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, Philadelphia.

Breunig M.M., Kriegel H.P., Ng R.T., and Sander J. 2000. Lof: identifying density-based local outliers. *In* Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pages 93–104.

Camurati G., Poeplau S., Muench M., Hayes T., and Francillon A. 2018. Screaming channels: When electromagnetic side channels meet radio transceivers. *In* Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 163–177.

Chawla N., Kumar H., and Mukhopadhyay S. 2021. Machine learning in wavelet domain for electromagnetic emission based malware analysis. IEEE Transactions on Information Forensics and Security 16:3426–3441.

De Moor B. 1993. The singular value decomposition and long and short spaces of noisy matrices. IEEE transactions on signal processing 41:2826–2838.

Epps B.P. and Krivitzky E.M. 2019. Singular value decomposition of noisy data: noise filtering. Experiments in Fluids 60:1–23.

Gavish M. and Donoho D.L. 2014. The optimal hard threshold for singular values is $4/\sqrt{3}$. IEEE Transactions on Information Theory 60:5040–5053.

Golub G.H. and Van Loan C.F. 2013. Matrix computations. JHU press.

Govindarajan S., Subbaiah J., Cavallini A., Krithivasan K., and Jayakumar J. 2019. Partial discharge random noise removal using hankel matrix-based fast singular value decomposition. IEEE Transactions on Instrumentation and Measurement 69:4093–4102.

Han Y., Christoudis I., Diamantaras K.I., Zonouz S., and Petropulu A. 2019. Side-channel-based code-execution monitoring systems: a survey. IEEE Signal Processing Magazine 36:22–35.

Han Y., Etigowni S., Liu H., Zonouz S., and Petropulu A. 2017. Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. *In* Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pages 1095–1108.

Hassanpour H., Zehtabian A., and Sadati S. 2012. Time domain signal enhancement based on an optimized singular vector denoising algorithm. Digital Signal Processing 22:786–794.

Hund R., Willems C., and Holz T. 2013. Practical timing side channel attacks against kernel space aslr. *In* 2013 IEEE Symposium on Security and Privacy, pages 191–205, IEEE.

Islam M.A., Ren S., and Wierman A. 2017. Exploiting a thermal side channel for power attacks in multi-tenant data centers. *In* Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 1079–1094.

Jha S.K. and Yadava R. 2010. Denoising by singular value decomposition and its application to electronic nose data processing. IEEE Sensors Journal 11:35–44.

Khan H.A., Alam M., Zajic A., and Prvulovic M. 2018. Detailed tracking of program control flow using analog side-channel signals: a promise for iot malware detection and a threat for many cryptographic implementations. *In* Cyber Sensing 2018, vol. 10630, page 1063005, International Society for Optics and Photonics.

Khan H.A., Sehatbakhsh N., Nguyen L.N., Callan R.L., Yeredor A., Prvulovic M., and Zajić A. 2019. Idea: Intrusion detection through electromagnetic-signal analysis for critical embedded and cyber-physical systems. IEEE Transactions on Dependable and Secure Computing 18:1150–1163.

Kolias C., Barbará D., Rieger C., and Ulrich J. 2020. Em fingerprints: Towards identifying unauthorized hardware substitutions in the supply chain jungle. *In* 2020 IEEE Security and Privacy Workshops (SPW), pages 144–151, IEEE.

Kolias C., Borrelli R., Barbara D., and Stavrou A. 2019. Malware detection in critical infrastructures using the electromagnetic emissions of plcs. Transactions 121:519–522.

Liu Y., Wei L., Zhou Z., Zhang K., Xu W., and Xu Q. 2016. On code execution tracking via power side-channel. *In* Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pages 1019–1031.

Nazari A., Sehatbakhsh N., Alam M., Zajic A., and Prvulovic M. 2017. Eddie: Em-based detection of deviations in program execution. *In* Proceedings of the 44th Annual International Symposium on Computer Architecture, pages 333–346.

Proedru V., Nouretdinov I., Vovk V., and Gammerman A. 2002. Transductive confidence machine for pattern recognition. *In* Proceedings of the 13th European Conference in Machine Learning.

Rajwade A., Rangarajan A., and Banerjee A. 2012. Image denoising using the higher order singular value decomposition. IEEE Transactions on Pattern Analysis and Machine Intelligence 35:849–862.

Sehatbakhsh N., Alam M., Nazari A., Zajic A., and Prvulovic M. 2018. Syndrome: Spectral analysis for anomaly detection on medical iot and embedded devices. *In* 2018 IEEE international symposium on hardware oriented security and trust (HOST), pages 1–8, IEEE.

Sehatbakhsh N., Yilmaz B.B., Zajic A., and Prvulovic M. 2020a. Emsim: A microarchitecture-level simulation tool for modeling electromagnetic side-channel signals. *In* 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), pages 71–85, IEEE.

Sehatbakhsh N., Yilmaz B.B., Zajic A., and Prvulovic M. 2020b. A new side-channel vulnerability on modern computers by exploiting electromagnetic emanations from the power management unit. *In* 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), pages 123–138, IEEE.

Vedros K., Makrakis G.M., Kolias C., Xian M., Barbara D., and Rieger C. 2021. On the limits of em based detection of control logic injection attacks in noisy environments. *In* 2021 Resilience Week (RWS), pages 1–9, IEEE.

Viswanathan M. 2019. Digital Modulations using Python. Independently published.

Wei S., Aysu A., Orshansky M., Gerstlauer A., and Tiwari M. 2019. Using power-anomalies to counter evasive micro-architectural attacks in embedded systems. *In* 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pages 111–120, IEEE.

Yarom Y. and Benger N. 2014. Recovering openssl ecdsa nonces using the flush+ reload cache side-channel attack. Cryptology ePrint Archive .

Zeng M. and Chen Z. 2019. Soso boosting of the k-svd denoising algorithm for enhancing fault-induced impulse responses of rolling element bearings. IEEE Transactions on Industrial Electronics 67:1282–1292.

Zhang G., Xu B., Zhang K., Hou J., Xie T., Li X., and Liu F. 2020. Research on a noise reduction method based on multi-resolution singular value decomposition. Applied Sciences 10:1409.

Zhao M. and Jia X. 2017. A novel strategy for signal denoising using reweighted svd and its applications to weak fault feature enhancement of rotating machinery. Mechanical Systems and Signal Processing 94:129–147.

Zhou K., Li M., Li Y., Xie M., and Huang Y. 2019. An improved denoising method for partial discharge signals contaminated by white noise based on adaptive short-time singular value decomposition. Energies 12:3465.