# MACHINE LEARNING METHODOLOGIES FOR LOW-LEVEL HARDWARE-BASED MALWARE DETECTION

A Dissertation
Presented to
The Academic Faculty

By

Nikhil Chawla

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

Georgia Institute of Technology

December 2021

# MACHINE LEARNING METHODOLOGIES FOR LOW-LEVEL HARDWARE-BASED MALWARE DETECTION

Thesis committee:

Dr. Saibal Mukhopadhyay
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Dr. Shimeng Yu
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Dr. Abhijit Chatterjee
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Dr. Santosh Pande
School of Computer Science
*Georgia Institute of Technology*

Dr. Arijit Raychowdhury
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Date approved: Nov 8, 2021

*To my Mom and Dad*

## ACKNOWLEDGMENTS

I would like to thank and acknowledge people who have contributed to my journey towards attaining the Ph.D. degree. I consider myself fortunate to experience a high-quality research environment, being amidst smart, hardworking, and intellectual academic professionals at Georgia Tech.

I would express gratitude to my thesis advisor, Dr. Saibal Mukhopadhyay for his mentorship, guidance, support, inspiration and encouragement. I am grateful to him for allowing me to work on this research project. Among my many great experiences, I would specifically like to highlight the conference trips to Europe and US locations which gave me a platform to present the proposed research and interact with academics and other researchers. Also, I am grateful and thankful for the experience of an industry-collaborative workshop on Energy Secure Systems Architecture (ESSA) organized by him. I am also genuinely thankful for his support during challenging times during the course of the Ph.D.

I would like to thank dissertation defense committee members, Dr. Shimeng Yu, Dr. Abhijit Chatterjee, Dr. Santosh Pande and Dr. Arijit Raychowdhury for evaluating this research and contributing vital inputs to improve this thesis work.

I am thankful to all excellent faculty members in the School of Electrical and Computer Engineering and Computer Science. Over the period, I have taken many exciting and intensive courses, which were critical in strengthening my foundation in various diverse topics related to computer hardware design and computer security. The academic courses' learning helped me comprehend and describe the novelty in my research work. I would also convey special thanks to Dr. Daniela Staiculescu, Tasha Torrence, and other academic advisors in the ECE Graduate Affairs office for administrative support and guidance towards timely progress towards the degree.

I would express thanks to my mentors and colleagues from GREEN Lab, Georgia Tech for their guidance, support and friendship. I am thankful to my mentors, Dr. Monodeep

Kar and Dr. Arvind Singh, who introduced and guided me on side-channel security research. Their excellent work, devotion, and leadership inspired me to pursue research in this domain. I am thankful to other colleagues from the lab, Harshit Kumar, Nael Mizanur Rahman, with whom I have worked closely and learned during our collaborations. I want to thank all GREEN lab members for important discussions and friendly talks; Dr. Jong Hwan Ko, Dr. Taesik Na, Dr. Faisal Amir, Dr. Yun Long, Dr. Burhan Ahmad Mudassar, Dr. Chaitanya Krishna, Sudarshan Sharma, Priyabrata Saha, Minah Lee, Xueyuan She, Daehyun Kim, Mandovi Mukherjee, Saurabh Dash, Biswadeep Chakraborty and all other members of GREEN Lab.

The path wouldn't have been easier without company of close-by friends from outside of work. I have cherished many memorable moments, exciting conversations and celebrations in their company. Finally, I would like to thank my parents for their financial and continuous moral support. I thank them for giving me opportunity to pursue higher education from a premier institute in United States. They have inculcated hardwork, discipline, and persistence principles, which were valuable in this pursuit and in life.

**TABLE OF CONTENTS**

# LIST OF TABLES

# LIST OF FIGURES

# SUMMARY

Malicious software continues to be a pertinent threat to the security of critical infrastructures harboring sensitive information. The abundance in malware samples and the disclosure of newer vulnerability paths for exploitation necessitates intelligent machine learning techniques for effective and efficient malware detection and analysis. Software-based methods are suitable for in-depth forensic analysis, but their on-device implementations are slower and resource hungry. Alternatively, hardware-based approaches are emerging as an alternative approach against malware threats because of their trustworthiness, difficult evasion, and lower implementation costs. Modern processors have numerous hardware events such as power domains, voltage, frequency, accessible through software interfaces for performance monitoring and debugging. But, information leakage from these events are not explored for defenses against malware threats. This thesis demonstrates approach towards malware detection and analysis by leveraging low-level hardware signatures.

The proposed research aims to develop machine learning methodology for detecting malware applications, classifying malware family and detecting shellcode exploits from low-level power signatures and electromagnetic emissions. This includes 1) developing a signature based detector by extracting features from DVFS states and using ML model to distinguish malware application from benign. 2) developing ML model operating on frequency and wavelet features to classify malware behaviors using EM emissions. 3) developing an Restricted Boltzmann Machine (RBM) model to detect anomalies in energy telemetry register values of malware infected application resulting from shellcode exploits. The evaluation of the proposed ML methodology on malware datasets indicate architecture-agnostic, pervasive, platform independent detectors that distinguishes malware against benign using DVFS signatures, classifies detected malware to characteristic family using EM signatures, and detect shellcode exploits on browser applications by identifying anomalies in energy telemetry register values using energy-based RBM model.

# CHAPTER 1
# INTRODUCTION

The increasing complexity of computing devices has created numerous pathways for adversaries to exploit and launch attacks that compromise the security and privacy of embedded computing devices. The abundance of malware samples necessitates intelligent data mining techniques for effective and efficient malware detection and analysis [1, 2]. An intelligent malware detection method comprises of two stages: feature extraction and classification/-clustering algorithm. Malware-specific features are captured by analyzing its behavior and monitoring application and system-level, or hardware events like permissions, APIs, system calls, network access, file access, cache access [3, 4, 5, 6]. These extracted features of different malware samples are then trained on a classification or clustering algorithm to infer previously seen or unseen signatures. In totality, data mining techniques are useful in inferring existing malware samples, identifying unseen malware samples and identifying new malware families [7].

Software-based malware detectors implemented on-device have numerous drawbacks. For instance, Anti-Virus (AV) software relying on static malware signatures is prone to evasion against zero-day malware. Also, AV engines are prone to exploits, just like conventional software. Alternatively, a dynamic malware detector overcomes the disadvantages of static analyzers and is more reliable. But, their implementation is slow and resource hungry. For instance, Rahmatian et al. have questioned the performance of host-based intrusion detectors (HIDS) in real-time by claiming an increased delay between the occurrence of an attack and the detection of the attack [8]. More recently, hardware-based approaches towards malware detection have gained relevance [6, 9, 10, 11, 12, 13, 14, 15, 16, 17]. Hardware-based malware detectors (HMD) implementations are resilient against software exploits. These are difficult to evade because malware leaves imprints as signatures in hardware, even though they remain undetectable in software. Also, HMDs implemented as hardware accelerators improve detection latency and lower power consumption [13]. Therefore, malware detection leveraging hardware-based signatures is an interesting alternative.

HMDs can be categorized into micro-architectural and non-micro-architectural based on the selection of hardware-level features. HMDs based on micro-architectural events extrapolated from hardware performance counter (HPC) in modern processors are prevalent [6, 9, 10, 11, 12, 13]. A large majority of previous research has delved into applying HPC-based HMDs in detecting android, windows, and IoT malware. Also, hardware implementation of HPC-based HMDs is shown to be a cost-effective solution compared to software approaches [13]. But, performance counter-based HMDs have come under scrutiny. The existence of non-determinism, variability in number of registers and performance events leading to lack of portability limit the applicability of performance counters in different application domains [18]. There are numerous alternatives such as leveraging non-micro-architectural events such as power domains, thermal monitors, physical side-channels, etc. These events are architecture-agnostic, pervasive and portable and not extensively explored for malware detection and analysis. This thesis demonstrates an alternative approach towards malware detection and analysis by leveraging low-level hardware signatures from on-chip power domains and external electromagnetic emissions.

The low-level hardware signatures are derived from 3 information sources explored in this thesis, voltage-frequency, electromagnetic emissions, and Running Average Power Limit (RAPL) interface. To understand their application in malware detection and analysis task, we assess their rate of information change both in time and amplitude, number of information channels and noise immunity. These metrics are summarized in Table 1.1

**Voltage-Frequency**: Power management is an integral part of the modern system-on-chips with an objective to regulate power consumption, improve energy efficiency and performance utilizing techniques such as Dynamic Voltage and Frequency Scaling (DVFS),

Table 1.1: Training and Inference configurations for Malware Detection

| **Configuration** | **Voltage-Frequency** | **EM Emissions** | **RAPL** |
|---|---|---|---|
| Update Interval (time) | 10ms | 2us | 1ms |
| Resolution (amplitude) | 30 states | 12-bit | 32-bit |
| No. of Channels | 2 per core | 1 per SoC | 1 per all core |
| Telemetry Collection | on-device | External Probes | on-device |
| Noise Immunity | Higher | Least | Lower |

clock gating and power-gating, etc. DVFS is a hardware-software co-optimization technique that dynamically configures the voltage and frequency in accordance to processor's workload. DVFS states are accessible through CPU frequency scaling interface in linux platforms. Voltage-Frequency states have lower update interval and finite information levels in comparison to power and EM emissions as shown in Table 1.1. But, DVFS states have higher noise immunity i.e. they are less sensitive to changes resulting from background processes. Prior works have explored intersections of DVFS and offensive security [19, 20, 21]. In contrast, one of the contributions of this thesis is demonstrating feasibility of power-based signatures, such as Dynamic Voltage and Frequency Scaling (DVFS) for hardware-based malware detection (HMD).

**EM Emissions**: Physical side-channels incorporates fine-grained information about the program's instruction sequences, control flows and data [22, 23]. Traditionally, EM side-channel have been exploited for offensive attacks like compromising cryptographic keys from embedded SoCs and hardware accelerators [24, 25]. However, recent works have demonstrated the capability of detecting malicious applications by remotely monitoring EM emissions from a device. The detectors leveraging EM side-channel emissions perform external monitoring and subsequent detection, thereby eliminating the need for on-device implementation. EM-based malware analysis mostly focused on anomaly-based detection that monitors deviation in EM emissions from trusted executions [26, 14, 15, 16, 27]. In contrast, the thesis demonstrates the feasibility of classifying detected malware into a characteristic family using Electromagnetic(EM) signatures. EM signatures captures fine-grained application behavior both in time and amplitude as shown in Table 1.1, which is useful for detailed fine-grained malware analysis.

**Real-time Power Estimates from On-chip Energy Telemetry Registers**: Power emissions from an SoC have been shown to reveal information about a program's behavior [22, 23]. There are different mechanisms to acquire power variations. Until recently, fine-grained power measurement required manual probing of power pins. For instance, F. Ding et al. presented "DeepPower", a Deep learning-based detector for IoT malware by measuring the IoT device power [28]. Although, fine-grained power variations are useful for detailed malware analysis as shown in [28]. But these invasive measurements require device modifications to integrate a power sampler. Alternative approach is to use a software-based power-models for different SoC components or battery power were utilized to detect malwares [29, 30, 31]. In contrast, we leverage MSR_PP0_ENERGY_STATUS register accessible using RAPL interface in x86 processors, which uses on-chip measurements of activities coupled with software-based power models to provide a real-time estimate of energy. We demonstrated detection and localizing anomalies in the MSR_PP0_ENERGY_-STATUS signatures resulting from shellcode exploits. These signatures have higher rate of information variation in amplitude but less in time as shown in Table 1.1, which is useful for detailed fine-grained malware analysis. Also, the proposed method of collecting power signatures does not require external measurement setup.

## 1.1 Research Statement

The proposed research aims to develop machine learning methodology for detecting malware applications, classifying malware family and detecting shellcode exploits from low-level power signatures and electromagnetic emissions. This includes

- Developing a signature-based detector by extracting features from DVFS states and using ML model to distinguish malware application from benign

- Developing ML model operating on spectral and wavelet features to classify detected malware to characteristic family using EM emissions
- Developing an RBM model to detect anomalies in RAPL signatures of malware infected application resulting from shellcode exploits
- Evaluating and comparing performance of malware detectors on practical malware dataset and real exploits

## 1.2 Key Contributions of this Thesis

- **Establishing unique correlation between DVFS states and an application behavior using ML model**: A correlation between DVFS states and application's runtime behavior can be established by modeling template of DVFS signatures or learning features using ML models. Template modeling is not convenient for DVFS states because it comprises background activity of concurrent threads forming uncorrelated noise. We propose ML algorithms that learns targeted application's specific DVFS features, given numerous training examples of application under different background noise conditions.
- **Demonstrating feasibility of detecting malware application using DVFS signatures**: The past studies have shown DVFS as offensive security threat [19, 20]. In contrast, we demonstrate information leakage from DVFS signatures can detect malware applications from benign. A DVFS-based detector is architecture-agnostic and pervasive applicable to broad spectrum of devices in comparison to HPC-based detection which requires selection of relevant events and varies across architecture [13].
- **Applying robust feature extraction methods to learn distinguishing characteristics of different malware family**: To extract malware specific features, we demonstrated 2D Discrete wavelet transform (DWT) on spectrogram, which improves de-

tection performance compared to 1D-STFT or spectrgram alone. This feature extraction approach is in contrast to de-noising and compression applications of wavelet transform [32, 33, 34].

- **Demonstrating feasibility of classifying detected malware into characteristic family using EM signatures**: Prior EM-based detectors emphasize deviations from trusted execution for detection, not focus on malware behavior [26, 14, 15, 27, 35]. In contract, proposed approach primarily focuses on learning characteristics of different malware family from their EM side-channel signatures.

- **Detecting and localizing anomalies resulting from shellcode exploit using RBMs**: HPC-based shellcode exploit detection is architecture-dependent and requires feature selection [9, 11]. Similarly, external power-based detection are invasive and require device modification [17, 28]. The current detection models identifies existence of anomaly in entire trace, but not localizes the anomaly. The proposed method develops an non-invasive, architecture agnostic detector that identifies and localizes anomalous executions of shellcode exploits using RBM model

## 1.3 Organization of this Thesis

**Chapter 2** presents a detailed literature survey essential to understand contributions of this thesis. It includes interactions of DVFS ans Security, power and EM emissions based detectors, feature extraction methods for hardware signals and performance evaluation metrics for the ML models.

**Chapter 3** presents supervised ML methodology to exploit the relationship between DVFS signatures and application's runtime behavior.

**Chapter 4** presents power based hardware malware detector (P-HMD) that leverages information leakage from Dynamic Voltage and Frequency Scaling (DVFS) states for classifying malware applications against benign.

**Chapter 5** presents a signal processing and machine learning methodology for EM-based malware analysis that distinguishes malware applications from benign along with identifying characteristic malware family of detected malware

**Chapter 6** presents a detection framework to identify anomalies in single register values corresponding to shellcode injections using an RBM model

**Chapter 7** summarizes key contributions of the thesis and future directions of research.

# CHAPTER 2
# BACKGROUND

## 2.1 Dynamic Voltage and Frequency Scaling (DVFS) and Security

There is a growing interest in understanding the interactions of DVFS and security. The past studies have shown DVFS as offensive security threat. For example, Tang et. al. presented a CLKSCREW methodology, which performs unconstrained undervolting/ overclocking to inject faults to perform differential fault attack (DFA) for recovering key from Trustzone on ARM processors [19]. N Chawla et. al. demonstrated the feasibility of performing application inference using DVFS states [36]. On the other hand, there have been studies exploiting DVFS for enhancing security. For example, Yang et. al. studied the use of DVFS as a countermeasure to power side channel attack on encryption engines [21]. A. Singh et. al. has demonstrated that fast DVFS enabled by on-chip regulator and adaptive clocking helps inhibit power/EM-based side-channel attack and differential fault analysis attack [20].

In contrast to the above mentioned works, the proposed research focus on establishing unique DVFS signatures of applications, and subsequently extending to detect malware against benign.

## 2.2 Leveraging Electromagnetic Emissions For Malware Detection

Physical side-channels incorporates fine-grained information about program's instruction sequences, control flows and data [37, 22]. Traditionally, power and EM side-channel have been exploited for offensive attacks like compromising cryptographic keys from embedded SoCs and hardware accelerators [24] [25].

However, recently researchers have demonstrated the capability of detecting malicious applications by remotely monitoring EM emissions from a device. The detectors leveraging EM side-channel emissions perform external monitoring and subsequent detection, thereby eliminating the need for on-device implementation. N.Sethatbaksh et.al. have shown applicability of remote EM side-channel detectors for resource constraint devices, such as Cyber-Physical Systems(CPS), IoT and embedded devices [15]. EM Side-Channel based detectors also eliminate variability arising from different architecture and software. The prior works on EM-based malware analysis mostly focused on anomaly based detection that monitors deviation in EM emissions from trusted executions. S. Clark et.al. have demonstrated "WattsupDoc" monitoring system to detect untargeted malwares on embedded medical device using AC power traces. They show 85% accuracy in detecting unknown malware on 3-NN, perceptron and random forest ML models [16]. A. Nazari have shown loop frequency in a program is amplitude modulated at operating frequency region and leaks in frequency spectrum, which is useful in identifying anomalies in code execution. They have demonstrated parametric tests such as K-S test to compare observed spectrum's with reference spectrum to detect anomalies [14]. H.A.Khan et.al. have demonstrated template based pattern matching anomaly detector for code injection attacks on Cyber-Physical Systems(CPS) like DDoS, ransomware. The template matching anomaly detection approach can detect intrusions upto 200 instructions on FPGA, code modifications with 97.5% AUC from upto 3meters [27]. N. Sehatbakhsh et.al. have demonstarted a remote EM side-channel detection framework for code-injection and code-reuse attacks on CPS. They introduced distance metric to compare peaks in the EM spectrum, under variations arising from operating clock frequency. The robustness of the proposed remote detection framework is evaluated under system level interrupts, multi-tasking, temperature variations etc [15]. Y. Han et.al. Introduced "Zues" monitoring system to maintain control flow integrity by using EM emissions from Programmable Logic Controllers (PLC) [26]. They achieved this task by learning sequence of spectrum's using stacked LSTM and identify and localize deviations from correct behavior. "Zeus" can detect malicious code executions with upto 98.9% accuracy. Y.Cheng et.al. introduced "Magattack" that violates user's privacy by identifying application launching and operation by monitoring EM side-channel emissions from laptop using magnetometer in COTS mobile devices. They demonstrate low-frequency sampling (100 Hz) of magnetometer is sufficient for workload classification and webpage identification [38].

In contrast to above mentioned works, we make the following contributions. First, the proposed approach primarily focuses on learning characteristics of different malware family from their EM side-channel signatures in comparison to prior work which profiles trusted execution behavior and find anomalous in execution paths and not focused on malware behavior [26, 14, 15, 27, 35]. Second, we applied robust feature extraction methods, like Discrete Wavelet Transform (DWT) which are useful in learning unique patterns of different malware families existing at different time and frequency regions. Third, we addressed the software updates issues through re-training of the model on new benign or malware family classes.

## 2.3 Power Based Malware Detectors

Existing works for power-based intrusion detectors target both external power side-channel as well as on-device power telemetry. Y. Liu et.al. presented a non-intrusive code execution tracking using power side-channels via hidden markov model and recover the most likely executed instruction sequence with a revised Viterbi algorithm [17]. F. Ding et.al. presented "DeepPower", Deep learning based detector for IoT malware intrusion and infection. "DeepPower" utilizes simple moving average filtering, auto-encoder model to isolate suspicious signals and infer suspicious activity [28]. The existing works for on-device power measurements acquires battery's current and models energy consumption. Jacoby et.al. have demonstrated early battery based host intrusion detection against network attacks [39]. Liu et.al. have demonstrated "Virusmeter" on Symbian devices, that exploits battery power to detect misbehavior resultant of malicious activity. The approach develops user-activity such as calling, messaging, surfing, emailing centred power model via machine learning models and develop state machine governing each activity. The generated energy profile from model is compared with actual energy profile to identify anomalies. The detector implementation has 1.5% overhead [29]. Merlo et.al. have developed energy consumption models for WiFi and CPU modules to detect battery drain attacks [30]. A trade-off is evaluated between energy measurement accuracy and precision from higher-level and lower-level measurements for energy-consumption modeling. This proposed energy consumption models are used to detect ping-flood attacks on skype calls. Caviglione et.al. have developed energy consumption of software components using high-level model of PowerTutor and values obtained from /sysfs fil system. Their utilize decision trees and neural network models to detect different covert channels on device [31].

In comparison to these above mentioned works, the proposed anomaly detection framework monitors on-chip software accessible power-models using RAPL interface. This contrasts with malware detection techniques utilizing software-defined energy models [30, 29], battery-based power [39], and external power measurements [17, 28]. In addition, we also demonstrate capability of detection framework in detecting obfuscated malware payloads.

## 2.4 Feature Extraction Techniques

*Windowed Fast Fourier Transform (W-FFT)*

We present two unique feature extraction techniques, for malware detection and family classification respectively. We applied short time fourier transform (STFT) on Electromagnetic (EM) signal as well as DVFS signatures and used amplitude of spectrum in each window to be used as features. The selection of frequency spectrum as features has been demonstrated in prior works [14, 16, 15, 26, 35]. Y. Han et al. have utilized a sequence of power spectral density (PSD) of frequency spectrum to train LSTM model [26]. S.S. Clark et al. have demonstrated energy of selective frequency components (<2.5KHz) as spectral features for malware detector [16]. A. Nazari et al. have selected spectral peaks in frequency spectrum of STFT window, where atleast $> 1\%$ signal energy is concentrated for K-S test analysis [35, 14]. In comparison to above works, the proposed detection framework utilizes amplitude of entire frequency spectrum of STFT.

*Discrete Wavelet Transform (DWT)*

We performed DWT on spectrogram derived from EM signal to extract patterns corresponding to different malware family. There are prior works that have applied DWT in side-channel analysis. J. Ai et al. have demonstrated an improved wavelet denoising method by first performing singular spectrum analysis (SSA) and detrended fluctuation analysis (DFA) on power side-channel traces, followed by wavelet transform on residual noise signal to improve success rate of Correlation Power Analysis (CPA) attack over unprocessed CPA by 30% [34]. N. Dabande et al. have demonstrated wavelet-based pre-processing for compressing power side-channel traces and pattern extraction from lower decomposition levels, to improve success of CPA attack by 50% compared to conventional time-domain CPA [32]. J. Longo et al. have demonstrated improvement in signal-to-noise (SNR) of power side-channel traces using wavelet analysis, eventually increasing t-statistic in leakage detection test [33].

In contrast to above mentioned works, feature extraction in the proposed research distinguishes in following ways. First, we demonstrate a unique data-processing pipeline for feature extraction by coupling 1D-STFT in time-domain and 2D-DWT on spectrogram. Second, we demonstrate application of DWT for feature extraction in contrast to denoising and compression applications shown in [32, 33, 34]. Third, we showed aggregation of features from multiple decomposition levels in contrast to only from last decomposition level as shown in [32].

## 2.5 Evaluation Metrics For ML Model

$F_1\ Score$

The performance of classifier is measured using $F_1\ score$ accuracy metric. The accuracy is evaluated using $F_1\ score$, which is defined as harmonic mean of micro-averaged or macro-averaged precision and recall. The precision gives ratio of True Positives (TP) and all positives (TP+FP). It shows number of samples of relevant malware family in a class and recall gives ratio of TP and (TP + FN) number of samples of relevant malware family classified correctly.

$$F_1\ score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.1}$$

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positives + False\ Positives\ (FP)} \tag{2.2}$$

$$Recall = \frac{True\ Positives\ (TP)}{True\ Positives + False\ Negatives\ (FN)} \tag{2.3}$$

*Region Operating Curves (ROC)*

Another performance metric evaluated for classifier performance is Region Operating Curves(ROC). Its shows that trend of True Positive Rate (TPR) and False Positive Rate (FPR) for different decision thresholds. Higher TPR and low false alarming rate are desired for good performance of classifier. Area under curve (AUC) is indicative of performance of the classifiers.

*Learning Curves*

We tested the generalization of model by observing the trend of training and validation score against varying number of training examples. The generalization test is performed on a 5-fold cross validation set. The ultimate objective is to study bias and variance trade-offs trends. High bias is indicated by higher training error or simpler model selection resulting in underfitting, whereas high variance is indicated by low-training and high validation error, therefore overfitting of the model. Variance is estimated using gap between training and validation curve. Lower gap is ideal for low-variance and lower training error is ideal for low-bias.

*Feature Relevance Using Mutual Information*

We used MI is for selecting features more relevant the target variable, by eliminating non-contributing features using MI score. Mutual information (MI) gives mutual dependence between two random variables. It is a measure of reduction in randomness of one random variable with knowledge about other variable. The dependence between two random variables is measured by comparing their joint probability distribution to product of marginal probability distributions. The MI for two random variable X and Y is given in equation Equation 2.4.

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p_{(X,Y)}(x,y) log \frac{p_{(X,Y)}(x,y)}{p_X(x) * p_Y(y)} \tag{2.4}$$

Here, $I(X;Y)$ denotes mutual information between two random variables, X and Y. $p_X(x)$ and $p_Y(y)$ are marginal probability distribution of random variables X and Y. $p_{(X,Y)}(x,y)$ is joint probability distribution of two random variables X and Y.

*Welsh's t-test*

Welsh's t-test is statistical measure to quantify similarity between samples drawn from two populations based on their means. T-test is a hypothesis test, where null hypothesis is EM or power signatures of two applications are selected from different malware family have similar means. The output of t-test is t-statistic and probability for rejecting or accepting the null-hypothesis. The p-value is obtained from student t-distribution function for a given degrees of freedom. The Welsh's t-test is described in equation below:

$$t - statistic = \frac{\mu_1 - \mu_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \tag{2.5}$$

Here, $\mu_1$, $\mu_2$ are sample mean, $s_1$, $s_2$ are variance and $n_1$ and $n_2$ are number of observations. The Welsh's t-test is applicable to data with unequal variances and different sizes. Based on the t-statistic and p-value, null hypothesis is accepted or rejected. If $t - statistic > 4.5$ or $t - statistic < -4.5$, null hypothesis is rejected with p-value of 0.00001 and confidence score of 99.999% [40]. The welsh's t-test has been explored in the literature for test vector leakage assessment (TVLA) of proposed countermeasures to cryptography circuits[41].

The next chapter presents characterization of DVFS signatures and assessment of information leakage from DVFS states using a ML model. The chapter builds foundation for utilizing a DVFS-based malware detection. The information leakage from power and EM emission is well established, and therefore not discussed on this thesis.

# CHAPTER 3
# DIFFERENTIATING APPLICATIONS USING DYNAMIC VOLTAGE FREQUENCY SCALING SIGNATURES

Power management is an integral part of the modern system-on-chips with an objective to regulate power consumption, improve energy efficiency and performance utilizing techniques such as Dynamic Voltage and Frequency Scaling (DVFS), clock gating and power-gating, etc. DVFS has become an integral part of modern processors to improve energy efficiency, increase battery life, and manage thermal effects. With DVFS, the supply voltage and operating frequency are scaled with respect to the varying workloads of the target system. Modern systems have multiple DVFS governors, that directs the rules for DVFS state of a core. The DVFS governors assess CPU utilization by evaluating busy time of CPU to down-scale or up-scale frequency, which in turn is workload dependent. The frequency states of all cores are updated by cpufreq driver in Linux kernel that creates a software-based information leakage path that can be utilized for detecting malware workloads.

To understand the feasibility of utilizing DVFS states for detecting malwares, a correlation between DVFS states and application's runtime behavior has to be evaluated. This can be done by modeling a template of DVFS signatures or learning features using ML models. Template modeling is not convenient for DVFS states because it comprises of information of multiple threads, executing in conjunction with profiled application. The background activity of these threads is random in nature and forms uncorrelated noise. To resolve this problem, ML techniques are an attractive alternative. ML algorithms can learn targeted application's specific DVFS features, given numerous training examples of application under different background noise conditions. In contrast to forming templates, ML models are promising in black box settings, requiring limited understanding of target implementation.

This chapter presents supervised ML methodology to exploit the relationship between

DVFS state variation with an application's behavior. We propose supervised machine learning models Support Vector Machines (SVM), Random Forest (RF) and K-Nearest Neighbors (KNN) to learn features derived from DVFS states time-series of different applications. To develop DVFS based application analysis framework, we showed various characteristics of DVFS signatures. These include uniqueness in DVFS signatures of applications, impact of DVFS governor on information leakage from DVFS signatures, and influence of background processes on DVFS signatures. The experiments are performed on the Intrinsyc Development Kit Open Q 820 SOM, hosting Android OS, with underlying linux kernel. We used applications from android benchmark dataset [42]. We used $F_1 score$ and ROC curves to evaluate ML model performance.

## 3.1 DVFS Governors in Linux Kernel

The CPUFreq module in linux kernel is a combination of three layers: cpufreq core, scaling governors and scaling drivers. The core provides framework for all platforms that support cpufreq. The scaling governors implement algorithms to decide on DVFS state by estimating CPU capacity. The scaling drivers communicate with the hardware and are responsible for actual change in the DVFS state[43]. They access the platform-specific hardware interface to change the DVFS state. There are multiple cpufreq governors accessible through linux sysfs filesystem [44].

Ondemand: The governor estimates the CPU load using its own worker routine and computes the fraction of time CPU was not idle. The ratio of non-idle to the total CPU time is estimated as load. In multi-core platforms, governor is attached to multiple CPUs. Therefore, load is estimated for all of CPU's and highest is considered for load estimate. The CPU frequency values are proportional to the estimated load, therefore maximum load corresponds to highest frequency [45].

Conservative: This governor estimates the load similar to the ondemand governor described above. It does not change frequency proportionately as ondemand governor, but

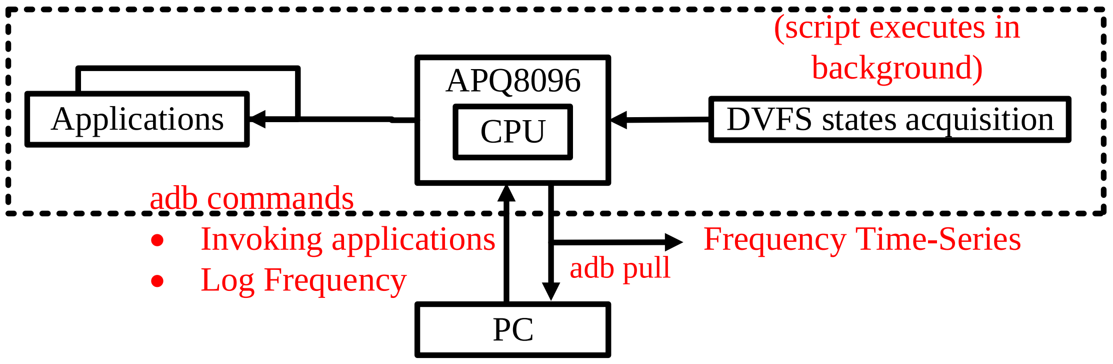

Figure 3.1: Data acquisition: (a) A methodology for DVFS states trace acquisition

instead it modulates frequency, once the load exceeds the defined higher or lower threshold of CPU load.

Interactive: This governor also estimates the load similar to ondemand and conservative governor. But, it aggressively scales the CPU frequency under intensive activities. To accomplish this, it configures a timer and if the CPU is very busy after coming out of idle, during the timer interval, it immediately ramps the frequency to the highest value. If the CPU is not sufficiently busy, then the governor evaluates the load and changes the CPU frequency.

## 3.2 Characterization of DVFS Signatures

We present following studies to characterize DVFS signatures. We show impact of cpufreq governor selection on information leakage. We demonstrate influence of background processes on DVFS signatures. We also show uniqueness in DVFS signatures of applications using Welsh's t-test.

### 3.2.1 DVFS Trace Acquisition

DVFS Traces are collected from a Intrinsyc APQ 8096 Development Kit, which has an embedded Snapdragon 820 system on module(SoM) quad-core processor. It has 2 little and 2 big cores, with highest operating frequency of 2.2 GHz. It has 32 GB of internal storage and 3GB of RAM [ Figure A.1]. The Snapdragon host android Nougat 7.0 operating system.

Evaluated Applications: The proof of concept is demonstrated by measuring DVFS states of android applications. Although evaluated android applications are benchmark applications[42], similar approach is generalized to any android application. The benchmark applications, namely whetstone, dhrystone, linpack and loops are CPU bound benchmarks. For instance, linpack benchmark performs floating point operation like addition and multiplication, busspeed and randmemi are memory bound benchmarks. Randmemi tests for data transfer speed from caches and memory. We also used some multi-threaded CPU performance benchmark in the dataset.

DVFS State Data Collection: The DVFS states of a core are derived from cpufreq module in linux /sysfs file system. We generated a compiled C binary that reads the CPU operating frequency state, logs the timestamp and save these values to a file. This file runs in the background along with profiled application as shown in Figure 3.1. The ARMv8 processor has 4 cores, two little and two big cores. The two little cores have shared power domain, and vice-versa for the other. The scheduler can assign the process to any core and the tasks can migrate between individual cores. Therefore, frequency states of both the cores are captured, when application executes on the processor. The scripts are programmed and executed simultaneously to capture the CPU frequency state of individual core.

Trigger, Log and Fetch: A separate script runs on desktop. It manages the invocation of DVFS state acquisition script, initialization of android application activity, pseudo-random inputs for interacting with application, and storing trace files on mobile device as shown in Figure 3.1. This is achieved through android debug (adb) interface shell commands. Each profiling android application's package name and activity name is extracted from apk dump and saved to this script running on PC. The pseudo random touch events for interaction with the application are simulated using monkey tool[46]. We set the percentage of system keys inputs to zero and gave 500ms delay between every input. The DVFS state trace collection

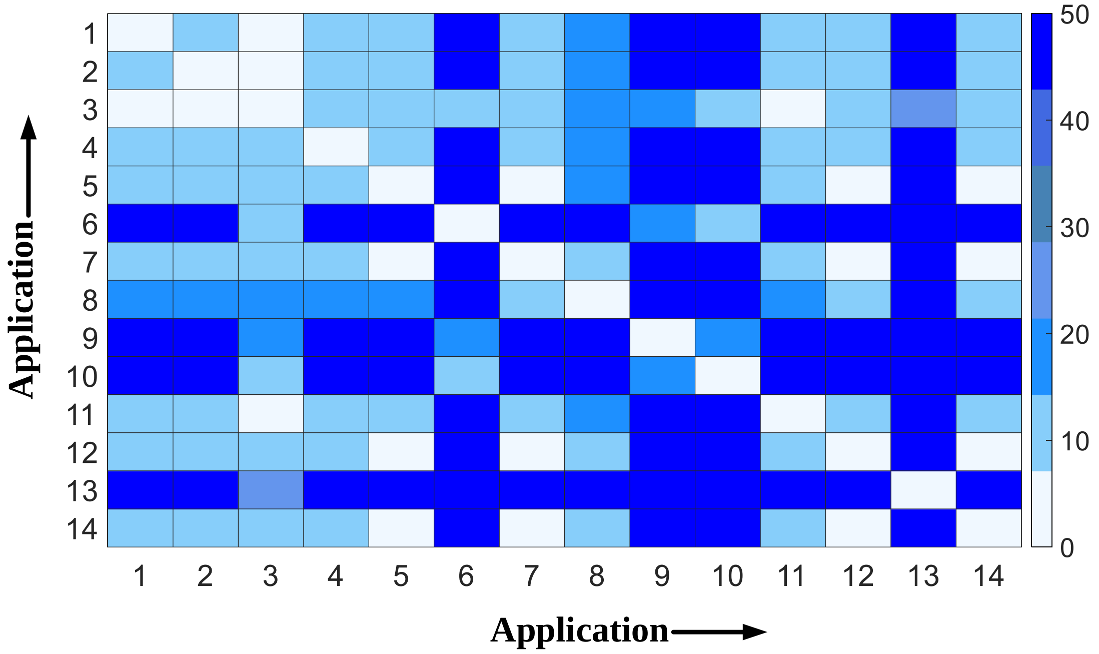

Figure 3.2: Heatmap of Maximum T-statistic between DVFS signatures for possible pairs of applications

time is set for 10 seconds for experiments. It can be tuned for different settings. Finally, the DVFS state trace is saved in the internal storage of mobile device.

### 3.2.2 Uniqueness of DVFS Signatures

The workload characteristics of applications have unique correlation with DVFS states. We applied Welsh t-test to establish uniqueness in DVFS signatures of different applications. In this particular case, the null hypothesis is ”DVFS signatures of applications selected from different classes have similar means”, therefore they are not distinguishable. We study the evidence of unique distinguishable features of different applications by obtaining Welsh t-test scores among all possible pairs of an application’s raw DVFS signatures.

First, we perform pairwise t-test on DVFS signatures of different applications to establish that signatures are unique and distinguishable. We selected ondemand governor and corresponding DVFS signatures of benchmark applications. Altogether, we perform 77 unique t-tests. We take multiple observations for DVFS traces of each application (100 traces) and perform a 2-sample t-test at each time point and obtain variation of t-statistic over time samples. We reject the null hypothesis if t-statistic at any time point crosses the threshold. The maximum t-statistic score computed over time samples for all possible combinations of t-test is represented using a heatmap as shown in Figure 3.2. The t-test results for all possible pairs of applications except (7,14) have maximum t-statistic greater than 4.5 or smaller than –4.5. This observation shows DVFS signatures of evaluated applications are distinguishable since they have distributions with different means

Second, we perform the t-test on DVFS signatures of same application collected over multiple iterations to establish they are non-distinguishable.We equally split the collected traces for an application into two groups. The traces belonging to a sub-group are selected randomly and size of each sub-group is 50. We again perform a 2-sample t-test at each time-point and observe the variation of t-statistic over time. We reject the null hypothesis if t-statistic at any time point crosses the threshold. The maximum t-statistic score computed over time samples for 14 possible t-test combinations is represented along diagonal in heatmap plot shown in Figure 3.2. We can confirm from observations in Figure 3.2, that t-statistic is below the threshold and therefore it don't disproves the null hypothesis, implying the two populations have been selected from similar distributions with equal means.

The t-test analysis establishes that DVFS signatures of different applications are distinguishable. However, the t-test itself may not provide a direct approach for detection of individual applications.

### 3.2.3 Impact of Governor Selection

We study the leakage behavior of different DVFS governors. To perform this study, we characterized an application's behavior by fixing one of the possible DVFS state and observed the likelihood of the DVFS state at every time point. We took 100 instances of DVFS state time-series of the benchmark application, "LinpackSP2" and calculated occurrence probability of that DVFS state [P(occurrence)] over 100 instances of the same trace.

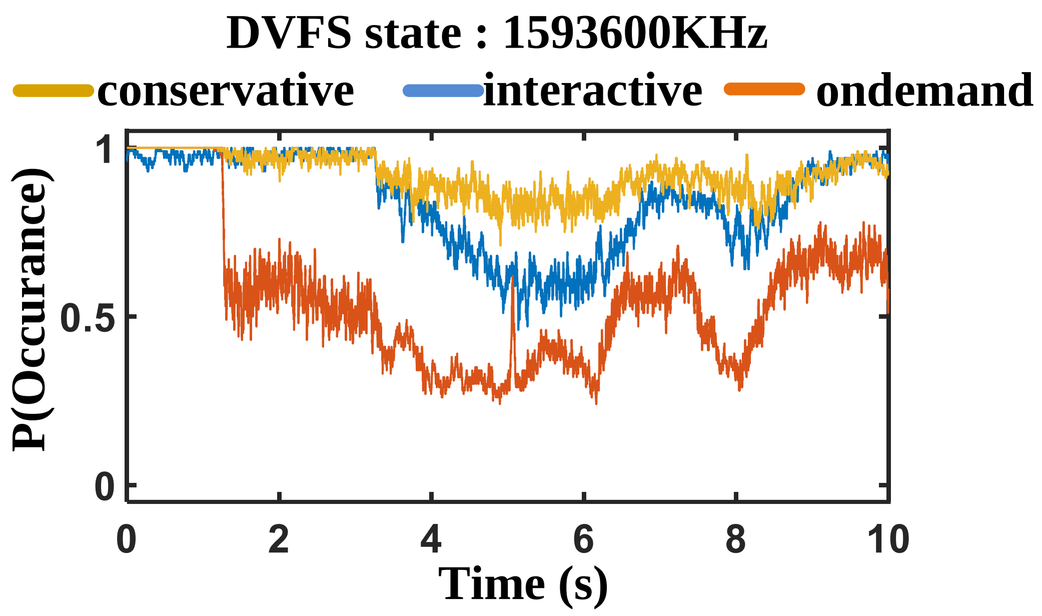

Figure 3.3: Likelihood occurrence of states 1593600KHz at every time sample for a benchmark application with all governors

Figure 3.3 shows the likelihood of occurrence of frequency state (1593 MHz) for an application with DVFS state signature generated from different CPUFreq governors. A higher likelihood of particular states across time, indicates less likelihood of its switching to different states. A value near 0.5 indicates, other states might have higher chances at those time-point. It can be concluded from Figure 3.3, governors have different leakage behavior for the same application.

### 3.2.4 Impact of Background Processes

In smartphones, foreground activity of application depends on user inputs which dynamically varies CPU utilization of the task contributing to the CPU load. In response to the CPU load, cpufreq governor scales voltage and frequency states of the CPU. In smartphones, there are numerous system level threads and user applications simultaneously contending for resources. But maximum CPU load is contributed by application running in the foreground, which the user is currently interacting with, whereas the background user level

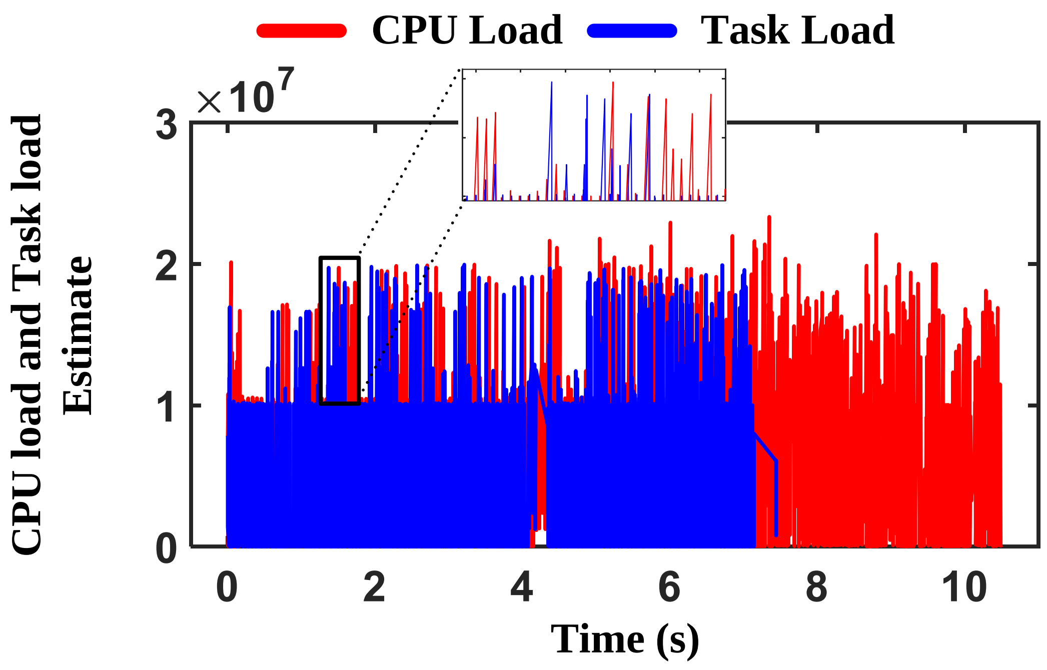

Figure 3.4: CPU load and Task Load Estimate generated using ftrace tool to demonstrate a high correlation between CPU load and foreground process

applications may not be actively executing on CPU core. Moreover, smartphones prioritize performance of applications to enhance Quality of Service(QoS). To understand the impact of background processes on DVFS state signature of targeted application, we evaluated correlation between CPU load and targeted application task load. A strong correlation between them implies reduced influence of background tasks in DVFS state signature. This is demonstrated through a motivating example detailed below.

The CPU load and task load are monitored using HMP scheduler events using ftrace framework in Linux [25]. It has multiple trace point events for kernel functions. The sched_update_task_ravg is an HMP scheduler trace event which tracks updates on currently running tasks [47]. It has various task related observable parameters which provide task specific information like PID, current CPU its running on(cpu), cpu frequency(cpu_freq), cpu utilization of task in current window (sum) and cumulative CPU demand of all the tasks in current window (curr_runnable_sum). For experimental evaluation, we initialized the tar-

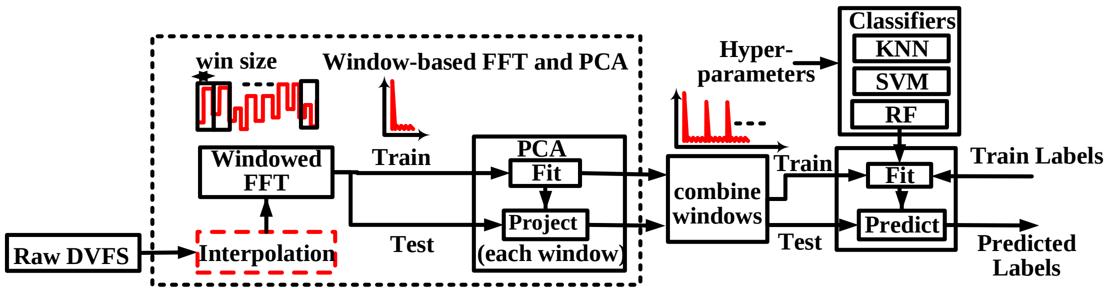

Figure 3.5: The proposed methodology for application classification. Pre-processing and feature extraction performed on short sequences of DVFS states time-series. The feature vector is generated by combining windows and subsequently passed to classifier

geted application, an android benchmark, 'LinpackSP2' [42] that executes for duration of approx. 7.5s in presence of system level threads running in the background and collected the sched_update_task_ravg trace events for duration of 10s. It comprises of all currently running processes, and we observe the task load(sum) for targeted application and cumulative CPU demand (curr_runnable_sum). Figure 3.4 shows the trend of these two variables for a particular CPU core, on which targeted application was executing for majority time. The plot depicts a high correlation between task load and cumulative CPU load, suggesting that maximum load is contributed by currently executing application.

## 3.3 Application Inference Using DVFS Signatures

### 3.3.1 Application Inference Methodology

We discuss the detailed methodology used in DVFS based application classification. Figure 3.5 shows an overview of the proposed classification approach.

Pre-processing and Feature Extraction: The input dataset comprises of current operating frequency time-series of both the CPU cores during runtime of the android application. Instead of choosing the exact frequency states, we chose frequency indexes (0 to 29) to avoid need for normalization of data as some machine learning algorithms (e.g., support vector machines) do not perform well if the input data has large values. The entire time-series length is partitioned into non-overlapping smaller sequences of finite length, and Fast Fourier Transform (FFT) is performed on each of short sequences. The amplitude of the Fourier transform is used as features. This is followed by dimensionality reduction on FFT of individual short sequence using principal component analysis (PCA). The variance in the dataset is evaluated across multiple instances of same application under background noise conditions as well as different applications. We selected the components that meet 99% variance in individual short sequence window. The number of principal eigen vectors will vary with different governors. Finally, the reduced sequences are concatenated to form feature vector. The proposed methodology is shown in Figure 3.5.

Machine Learning Algorithms: To learn program runtime characteristics from DVFS states, we chose supervised machine learning algorithms suitable for varying type of datasets. Both our training and testing datasets vary a lot in terms of dimensionality, number of observations and noise characteristics. The noise in turn depends on the selected DVFS governor, application that is being executed, the sampling speed at which the DVFS monitor is capturing the frequency states and essential system and user applications running in the background.

- Nearest Neighbors (KNN): KNN algorithm can classify datasets with linear or non-linear distributions. It selects k neighbors from the N-dimensional feature space. The neighbors are assigned weights based on distance metric (here Euclidean). K-NN performs well in low-dimensional feature space, with large number of observations which is the case with DVFS dataset.

- Support Vector Machines (SVM): Like KNN, SVM can also classify linearly or non-linearly distributed datasets with a proper kernel (linear, poly, radial basis function RBF) It constructs a hyperplane to divide the feature space. The hyperplane is chosen such that it maximizes the margin between the features. For noisy as well as high dimensional data, SVM tends to outperform other ML algorithms. But drawback is increased training time.

- Random Forest (RF): Random forest is an ensemble learning method, that uses results from multiple decision trees. Using ensemble of decision tress avoids overfitting problem. These commonly applied to multi-class classification problem. We haven't performed detailed hyper-parameter optimization but tested out classifiers performance by tuning the parameters

### 3.3.2 ML Model Building

We used several supervised machine learning algorithms to learn features derived from DVFS states time-series of different applications, and respective applications as their labels, thereby forming a classifier.

The dataset comprises of 14 android benchmark applications. During training, each android application of the dataset is executed for a duration of 10 seconds in presence of default activities executing in the system. There are no constraints imposed on background scenarios during training. The training environment is reflective of true android mobile phone scenario created by end user. As described in section 3.2, user can start multiple android applications, but at particular time, user is interacting with one application in the foreground. The DVFS signature is reflective of cumulative activity of CPU core, but its dominated by foreground application. User interface (UI) is an essential component of android applications, and in order to collect DVFS states of android application, we need to provide interactive user inputs. To clarify, "user interaction" do not imply a physical interaction between user and a device. The interaction can be virtual, aided by specific tools[37]. For instance, fuzzer tools like monkey can simulate such pseudo-random UI events. Also, user input interactions are limited to smartphones and do not extend to most of IoT devices. Moreover, the threads can be scheduled on either core and as well might migrate between cores. Therefore, it is essential to capture frequency states from both cores. In addition to this, multiple instances of DVFS state traces are acquired for every profiled application. This ensures that variations arising due to dissimilar background conditions are considered.

Table 3.1: $F_1Score$ for Application Inference with different governors

| **CPUFeq Governor** | **KNN** | **SVM** | **RF** |
|---|---|---|---|
| Interactive | 0.7 | 0.79 | 0.83 |
| Ondemand | 0.84 | 0.95 | 0.97 |
| Conservative | 0.64 | 0.64 | 0.7 |

We fixed the number of training instances per application as 75 in our experiments. The number of samples in DVFS state time-series is 20k for individual core. The feature vectors generated after pre-processing DVFS state time-series of individual core as described in subsection 3.3.1 are concatenated. There are 14 labels one for each application.

### 3.3.3 ML Model Evaluation

We collected multiple DVFS state time-series traces of all the application learned by the model under different background conditions every time. We tested each application with 25 instances. The feature vector is generated by computing FFT on windowed sequence of time-series trace and projected to corresponding eigen vectors to form the reduced representation. Reduced feature representation of windowed sequence is concatenated to form feature vector. Finally, performance of the classifier is evaluated on the testing dataset.

Accuracy Analysis: Table 3.1 shows $F_1score$ for application classification using different supervised machine learning models and DVFS states features extracted from different cpufreq governors. The $F_1score$ are highest for random forest classifiers in comparison to SVM and KNN. On comparing classification accuracies with different ML models, KNN does not perform optimal fitting. It can be concluded it is not the best algorithm when data is noisy, or features are not consistent. The trend in $F_1score$ across different cpufreq governors clearly shows the algorithmic differences of these governors. The $F_1score$ shows highest value for features extracted from ondemand governor. We also obtained ROC curves for RF inference model as shown in Figure 3.6. A higher AUC (higher TPR for lower FPR) is observed for features derived with ondemand governor compared with interactive

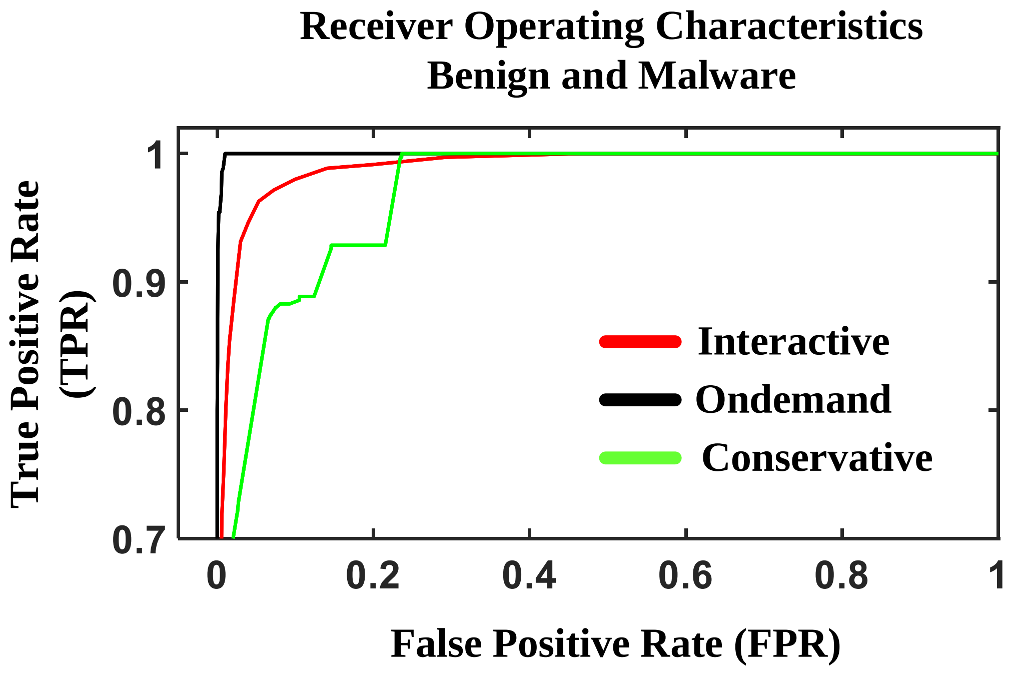

Figure 3.6: ROC curves generated for benchmark application classification with RF inference model evaluated across all governors

and conservative. These results indicates DVFS state variation of ondemand governor has more distinguishing features followed by interactive and conservative. The trends are in consensus with their algorithmic behavior as described in section 3.1.

The high detection accuracy on application classification summarized in Table 3.1 shows that ML model has extracted unique features of different applications and subsequently distinguish them with high confidence scores. The results re-confirms the evidence of unique distinguishable features of different applications as shown via Welsh t-test scores on raw DVFS signatures in subsection 3.2.2

Impact of Time-Series Length: In above application classification task, we kept the duration of DVFS signature fixed for 10s. It is essential to understand how much data is actually required to attain the $F_1 score$ as shown in Table 3.1. In general, longer duration of DVFS signature will increase data collection, training and inference time. But, it will also improve the accuracy. Therefore, it is ultimately a trade-off between model accuracy

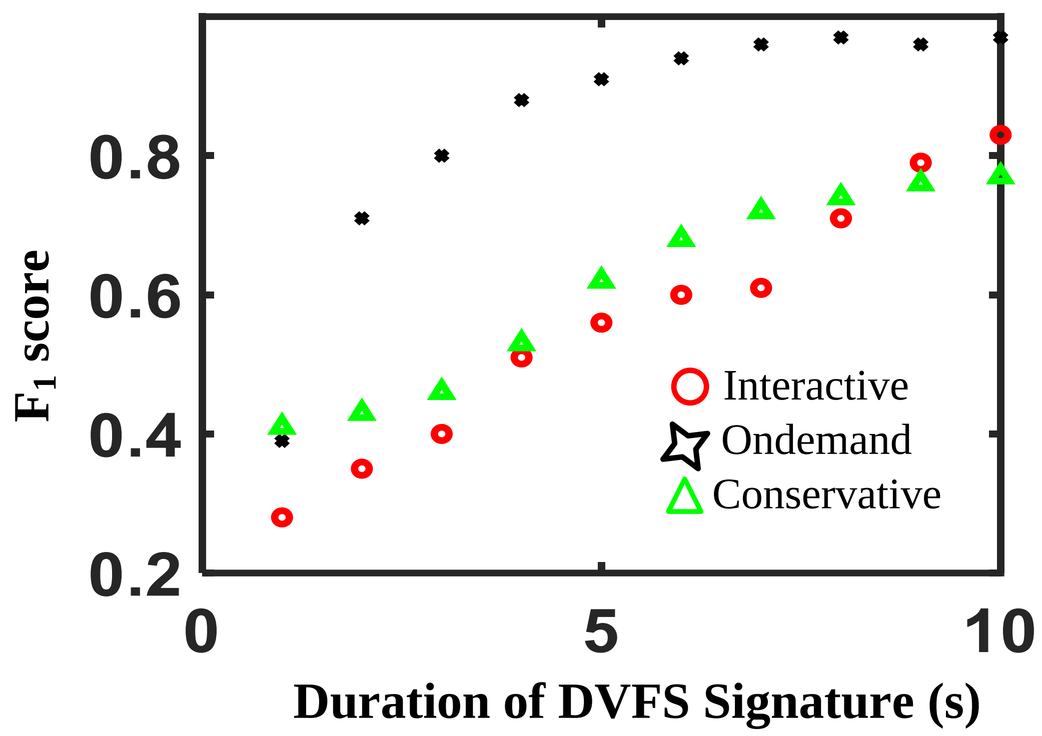

Figure 3.7: $F_1 score$ vs Duration of DVFS signature for different governors evaluated using RF model

and training time. To study the effect of time-series duration on model accuracy, we trained the model on different duration of time-series and observe the $F_1$ $score$ on testing dataset. The plot for depicting this trend is shown in Figure 3.7. As expected, we can observe that test accuracy increases with longer time-duration of the DVFS signature. Interestingly, we observe that Certainly, features derived from interactive and conservative governor require more data for improving the classification score. On the other hand, features derived from on-demand governor can classify these 14 applications with 0.97 $F_1$ $score$ with 8s time-samples. Therefore, applications can be classified with 10s of data collection with on-demand governor; however, for higher accuracy using other governors we need to collect longer duration of data.

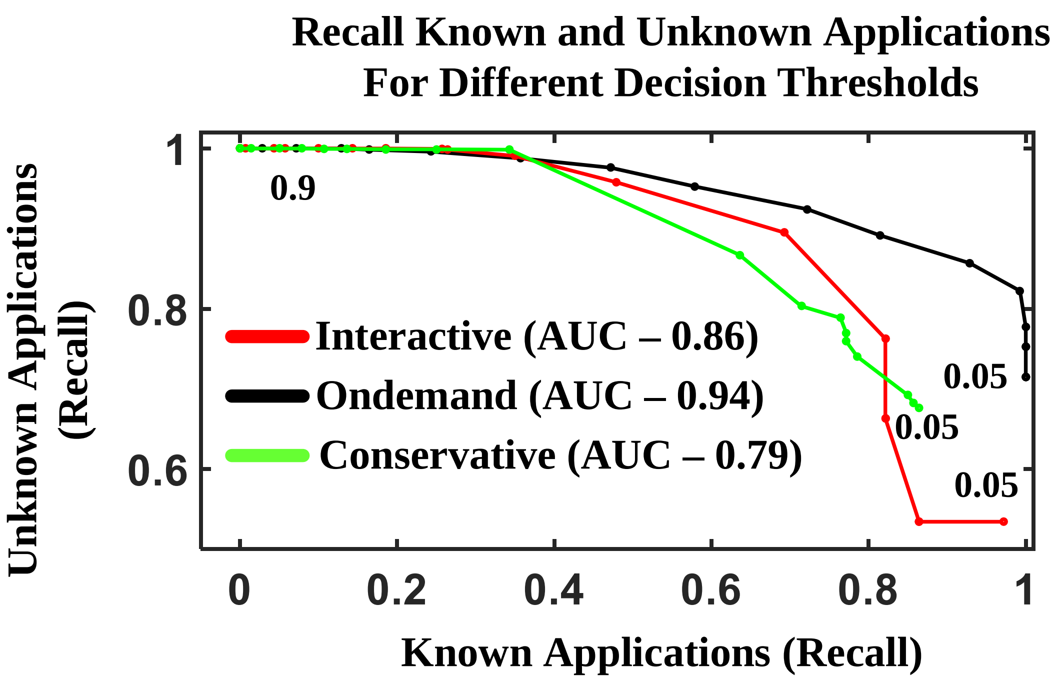

Figure 3.8: Recall for known and Unknown application generated using trained RF model evaluated across different decision thresholds

### 3.3.4 Identifying Unknown Applications

We evaluated the machine learning model on a dataset of applications whose features are not learnt by model. These are referred as unknown applications. We demonstrate that this trained model can be used to flag a potentially malicious or application not known to user. The trained model with known applications dataset is applied to a test dataset of some of the unknown applications. We used applications in MiBench suite [48] for the testing and measured 40 instances of each program. To detect the unknown program, pre-processing and feature extraction is performed, and then prediction probabilities are obtained from the machine learning model. We define a decision threshold based on probability to classify the unseen application in the unknown program class only when probability is less than decision threshold. We varied this decision threshold that subsequently introduced False negatives (FN) in identifying known applications. Therefore, there exists this trade-off between recall for known and unknown applications at different thresholds. A plot demonstrating this trend for different governors is shown in Figure 3.8. The model generates higher probability of an example learned by model in comparison to application from unknown application. Therefore, at lower threshold, higher recall is observed for known applications and vice-versa. Higher AUC for training model derived from ondemand governor indicates more distinguishability between features of known and unknown application (Figure 3.8).

## 3.4 Summary

This chapter experimentally demonstrated identification of applications by deriving features from DVFS states of a processor's core and using supervised machine learning model to establish their subsequent correlation. The Welsh t-test of analysis on DVFS signatures shows unique signatures for different applications. The $F_1 score$ of application classification is $> 0.7$ evaluated across different machine learning models and cpufreq governors. The highest $F_1\ score$ and AUC score obtained for ondemand DVFS governor indicates highest information leakage. We also showed identification of unknown applications using the learned model based on known applications. The results show $> 0.79$ AUC on recall curves for known and unknown applications evaluated with RF model and all cpufreq governors. The subsequent chapter extends the ML methodology to demonstrate feasibility of a detecting malware application from benign.

# CHAPTER 4
# SIGNATURE-BASED MALWARE DETECTION USING DYNAMIC VOLTAGE FREQUENCY SCALING SIGNATURES

The computing devices generate, process and communicate large amount of sensitive data which is susceptible to malware attacks, thereby compromising security and privacy. The software-based malware detectors, particularly AntiVirus (AV) are vulnerable to software exploits and require consistent updates. Moreover, many AV software are power hungry and hence less suitable for energy constrained devices. Alternatively, hardware-based malware detectors (HMD) have gained interest [6, 49, 12]. HMD can be implemented in hardware, remain isolated and cannot be easily tampered [12]. HMD are based on features derived from hardware events such as micro-architectural, power, electromagnetic emissions. HMD based on performance counters vary across different architectures [13]. Also, limited number of performance registers requires selection of relevant HPC. Similarly, detector based on EM emissions requires external side-channel measurements.

This chapter presents a power based hardware malware detector (P-HMD). The power management is an essential hardware feature which is architecture agnostic and pervasive component common to broad spectrum of devices. In particular, we focus on leveraging information leakage from power management signatures like, Dynamic Voltage and Frequency Scaling (DVFS) states for malware identification. We propose to utilize supervised machine learning models Support Vector Mchines (SVM), Random Forest (RF) and K-Nearest Neighbors (KNN) to learn features derived from DVFS states time-series of benign and malware applications. The ML methodology used to develop correlation between DVFS states and application runtime is extended to demonstrate feasibility of a malware detector (Figure 4.1). The malware detector can learn from features of DVFS signatures under different settings of governors such as ondemand, conservative, interactive. Since,

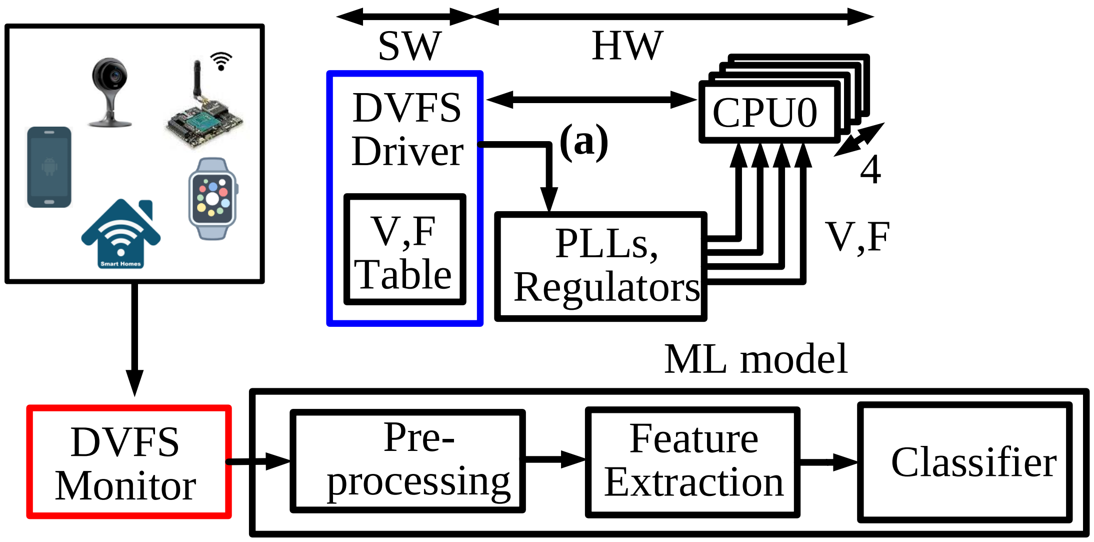

Figure 4.1: (a) Dynamic voltage frequency scaling, hardware-software co-optimization approach for power management (b) A machine learning based malware detector using DVFS states features

DVFS governors also show differences in power behavior of applications, therefore, we show existence of a power-security aware design space to meet energy efficiency as well as provide security against malware. A secure detector design is proposed for practical on-device malware detection system with cloud-based software updates features.

The experiments are performed on the Intrinsyc Development Kit Open Q 820 SOM, hosting Android OS, with underlying linux kernel.We used android benchmark dataset[42] for benign applications and malware samples were collected from Drebin dataset[50]. We have evaluated malware detection accuracy using features derived from cpufreq governors.

## 4.1 Malware Detection

We extend the machine learning (ML) methodology used for distinguishing applications for malware detection. The DVFS state time series of a processor is correlated to the application(s) scheduled on the core. Hence, DVFS state transitions can capture differences between benign and malware applications. The experimental setup used for this analysis is

Table 4.1: Training and Inference configurations for Malware Detection

| **Configuration** | **Training** | **Inference** |
|---|---|---|
| Applications | 14 benign, 16 malware | 14 benign, 144 malware |
| Traces | 2250 | 494 |
| window_size | 200 samples (100 ms) | |
| PCA | n_components = 6 | |
| RF | n_estimators = 40, gini impurity | |
| SVM | C=1, linear kernel | |

described in subsection 3.2.1

### 4.1.1 ML Model Building

The benign dataset comprises of android benchmark applications as described in section 3.2. It comprises 14 android apps. The malware samples are collected from Drebin dataset [50]. It comprises of 179 different malware families. Although benign applications are benchmarks, the methodology is generally applicable to android applications. In order to create a near balanced dataset, we selected only 16 android malware applications, each belonging to different family.

Each malware application composition has benign as well as malware proportion. Training is not performed on malicious portion separately, but instead classifier sees both benign and malware codes. Therefore, the entire DVFS state time-series is labeled as malware. Different malwares have their own activation mechanism and payloads as described in [51] and it is well understood that power behavior of these applications will be unique, but we are interested in runtime interactions of android applications with DVFS states of a processor. The dataset comprises of 14 benign benchmark applications and 16 malware applications. During training, each android application of the dataset is executed for a duration of 10 seconds in presence of default applications running in the system. As described earlier, there are no constraints imposed on background scenarios during training. In addition to this, multiple instances of DVFS state traces are acquired for every benign and malware application in the dataset. This ensures that variations arising due to dissimilar background conditions are considered. We fixed the number of training instances per application as 75. The configuration for benign and malware classification is summarized in Table 4.1.

### 4.1.2 ML Model Evaluation

The testing dataset comprises of new samples of android malware, not trained on the model. Since, it is imperative to correctly predict malwares compared to benign. Therefore, goal is to obtain lower false negatives (FN) and minimize False Positives (FP). The trained supervised machine learning model is tested on dataset comprising of 144 different malware applications which are variants of malware applications used in training. These belong to the same malware family, on which the model is trained, but it's a variant of those malware applications. It comprises of 14 android benchmarks applications measured under different background conditions, for 10s duration. We measured 25 instances each of these benign applications. The configuration during evaluation for benign and malware classification is summarized in Table 4.1.

#### *Accuracy Analysis*

Since RF classifier shows relatively higher $F_1 score$, we obtained the learning curves for RF trained model with features derived from different governors as shown in Figure 4.2. The training score attains highest value at lower number of training examples for ondemand and interactive governor compared to conservative. It indicates a low bias in the trained model. Similarly, an increase in validation scores with number of training examples and attaining close to training score indicates low-variance and a good generalizability of model for new examples in case of ondemand and interactive governor. In contrast, validation score does not improve with increasing number of training examples for conservative governor, it also shows comparitely higher variance, and poor generalization of model.

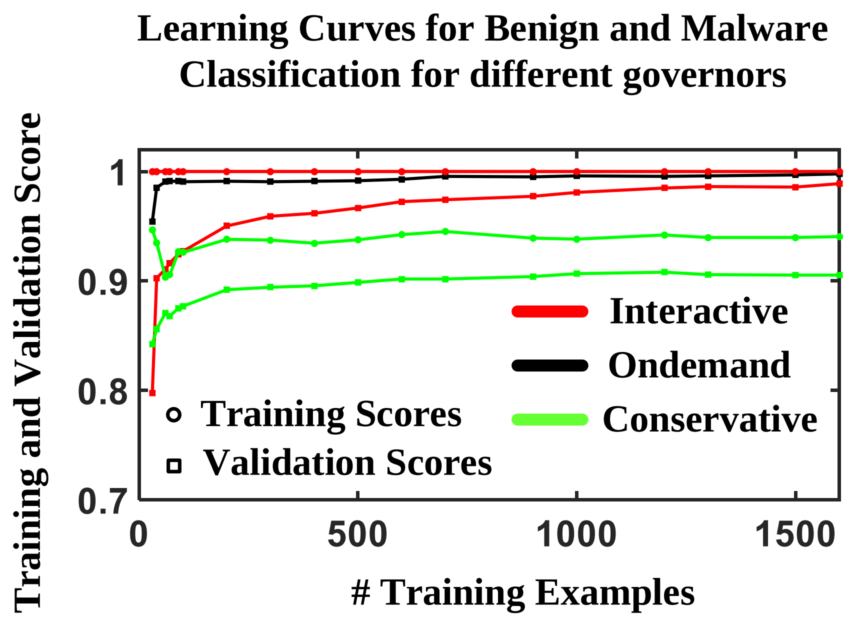

Figure 4.2: Training and Validation Score against number of training examples (Learning Curves) generated with RF model for features dervied from different governors.

Table 4.2: $F_1 Score$ for Benign and Malware Classification with different governors

| **cpufreq governor** | KNN | SVM | RF |
|---|---|---|---|
| Interactive | 0.88 | 0.78 | 0.78 |
| Ondemand | 1 | 0.72 | 0.99 |
| Conservative | 0.57 | 0.98 | 0.97 |

Table 4.2 shows the $F_1 score$ on testing dataset evaluated with different classifiers and DVFS states features generated from different governors. The $F_1 score$ are highest for RF classifiers in comparison to SVM and KNN. On comparing classification accuracies with different ML models, we can conclude, that KNN does not perform optimal fitting. These results indicate the model is generalized to variants of existing malware samples learned during training. The performance of classifier across features derived from different cpufreq governors is evaluated using # False Positives and # False Negative. Table 4.3 shows the confusion matrix values with RF classifier. FP indicates the number of benign samples being labeled as benign and vice-versa for FN. The combined FP and FN scores for

Table 4.3: Confusion Matrix Generated from Random Forest Classifier for different governors

| **cpufreq governor** | # TP | # FP | # TN | # FN |
|---|---|---|---|---|
| Interactive | 41 | 6 | 344 | 103 |
| Ondemand | 142 | 0 | 350 | 2 |
| Conservative | 144 | 15 | 335 | 0 |

ondemand is lowest followed by conservative and ondemand. These results also indicate features derived from ondemand governor have more distinguishability compared to other governors. It also indicates more information harnessed from ondemand governor.

### 4.1.3 Power and Security

The power consumption of android applications changes with different cpufreq governors and consequently malware identification accuracy as shown in Table 4.2 and Table 4.3. The tradeoffs between average power consumption of different applications and malware identification accuracy are evaluated for different governors. We measured average power consumption of android benchmark and subset of malware applications (Fig. Figure 4.3). Power is measured for application runtime duration of 10s by computing current drawn (averaged over 10 measurements) from 3.8V buck converter supplying current to the Snapdragon processor using a 5m on-board resistor. The average power consumption accumulated across all benign apps is higher compared to malware apps for all governors. Enabling conservative governor dissipates highest power for benign and malware applications. The power consumption of applications under ondemand governor is least when averaged across all applications. Moreover, features extracted from ondemand governor also shows higher application identification scores (0.97) and lower number of False Positives and False Negatives for malware identification. In summary, applications dissipate relatively less power and shows higher identification accuracy with ondemand governor compared to interactive and conservative.

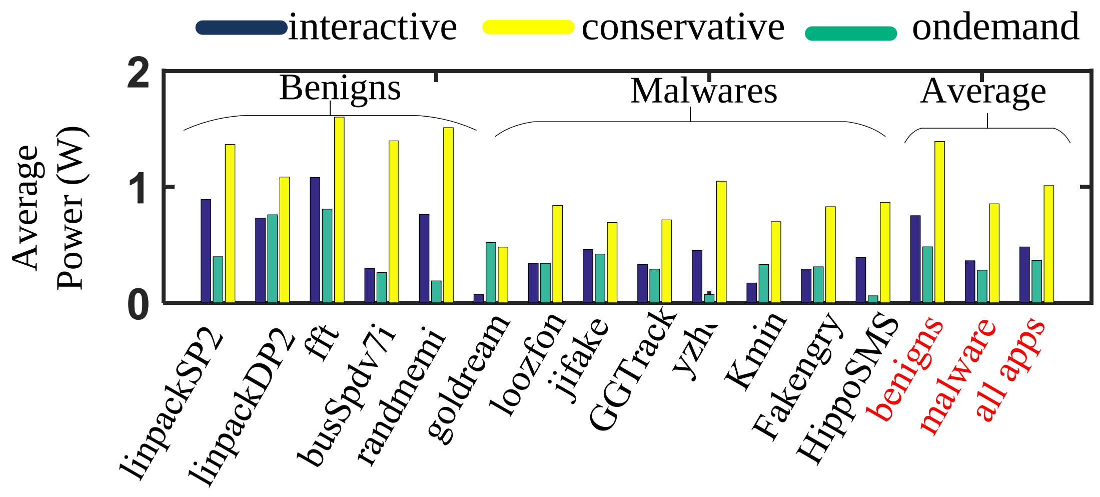

Figure 4.3: Average Power for benign and malware applications with different cpufreq governors

## 4.2 Secure Detector Design

The realization of detector encompasses multiple design choices depending on software or hardware. The hardware based implementations may include machine learning inference engines as accelerators, or dedicated core etc. In software, machine learning inference can be executed at kernel or flashed in firmware. Allocating higher privileges to detector creates more secure detection framework. First, it will not only be useful for monitoring malicious activity at application level, but they can potentially detect malware exploits at kernel level. A hardware centric approach to detect rootkits using hardware performance counters is demonstrated in [51]. Second, the detector won't fail if system level software gets compromised. More recently, more secure feature of trusted execution environments (TEE), like Trustzone are extended to IoT based platforms. Implementing such detector will ensure that underlying detector code and data is protected from tampering or adversarial attacks. The specific tasks performed by proposed detector are described below

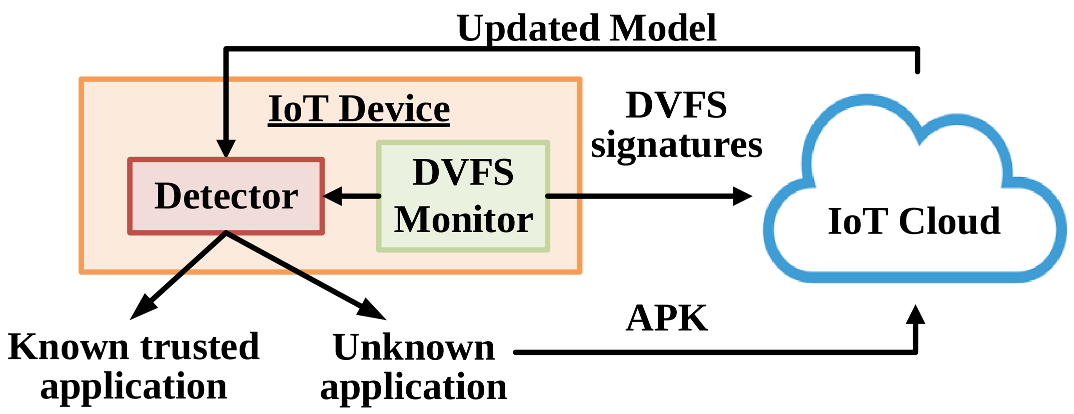

Figure 4.4: On-Device DVFS based detection for red-flagging malware and detailed malware analysis and model updates on IoT cloud servers

*Data Acquisition*

The data acquisition step involves polling values of DVFS states. In linux based platforms, DVFS states can be accessed through CPU Frequency scaling infrastructure. The current frequency state of all CPU cores are sampled periodically for a specified duration. In our proposed DVFS based detector data is collected for 10s duration.The polling of DVFS states from cpufreq module introduces noise in the profiled application's DVFS signature. We have taken this into consideration during training by collecting multiple instances of same application

*Data Analysis*

The DVFS signature of application is analyzed by detector (ML inference model) as shown in Figure 4.4. The detector determines if DVFS signature belongs to set of trusted applications (known) else it is detected as unknown. A new application(malware or benign) will be first be detected as unknown application. A detected unknown application will be classified as "malicious" by default and flagged for further analysis at the cloud servers to determine whether the application can be trusted.

*Threat Response*

Once the application is redflagged, it is deleted from the system and more detailed malware analysis is performed to scrutinize its behavior. Detailed malware analysis is both time and resource consuming task and it cannot be efficiently performed on-device. It can be analyzed on cloud servers by sending detected application. If the application is identified as threat after detailed analysis, IoT devices should be notified about updating ML model with DVFS signatures of the detected "malware" application.

*Model Updates*

ML Model updates are required under multiple scenarios. An application's power signature can change on software updates. Also, the proposed detector is trained on limited malware and trusted applications. Therefore, model would also require updates about new malware applications and new trusted applications to be included in the existing model. Considering our current approach, an update would involve retraining the machine learning model with pre-existing data and new data and finding the best fit model using hyper-parameter optimization. In our current approach, any update or a new application (trusted or un-trusted) will first be detected as an unknown application. For a software update or a new trusted application, the detected application can be mapped as "known". This involves steps like, generate a trigger to enable collection of the DVFS signatures, send the collected DVFS signature to the cloud (or gateway) for re-building of the model, and receive the new ML model from cloud once re-training is performed. (Figure 4.4).

We believe that the retraining is better performed at cloud or at the gateway depending on the available processing power.The primary overhead for the edge device will be data collection and data transfer associated with the model creation.The upload data volume for unknown application is estimated around 23MB [Samples per trace (40K) * Training Examples (75) * size(float64)]. The download data volume from cloud servers comprising of retrained model is estimated around 10MB [Apps (31) * Training Examples (75) * Feature

Vector Length (600) * size(float64)]. The model fitting and optimization will be performed at the cloud, and new ML models can essentially be deployed to all edge devices that are based on same OS.

## 4.3 Implementation Challenges

Practical implementations of the proposed detector requires addressing challenges like, scalability of the detector with increasing data, number of applications and type of platforms.

### 4.3.1 Platform Scalability

The experimental analysis is performed on an Android platform, but we have only used CPU Frequency scaling feature of the Linux OS(no Android-specific functions). Linux is a prevalent OS in many devices in the IoT ecosystem (gateway, edge, and resource constraint devices). In IoT developers survey 2019, linux is still dominant OS for gateways and edge nodes [52]. The platforms based on linux kernel have CPU frequency scaling infrastructure like cpufreq, cpuidle. Therefore, we believe the observations made in this work will be applicable to a significant portion of IoT ecosystem. Conclusively, the fundamental observations made in the paper are applicable to most of the modern processors/SoCs.

### 4.3.2 Algorithm Scalability

A new application (benign or malware) would have a unique DVFS behavior and it would eventually lead to unique features. At the best case, if features of new applications exhibit similarity with existing benign or malware class, then the current model may perform well. If the signature of the new application is different, the hyperparamaters of the current model must be re-tuned (as discussed above) to generate a new version of the current ML model. At the extreme case, as the numbers of applications increase the non-linearity in the feature space is likely to grow as well making it difficult to classify the applications using simpler

ML models. In such cases, a non-linear machine learning algorithm like RF, or ultimately, deep learning based approach, would be more scalable than linear algorithms like SVM. The future work on this topic would consider applying complex non-linear ML algorithms like deep neural networks.

#### 4.3.3 Time-Series Length

The time-duration of the data collection during training and inference will impact classification/detection accuracy. In principle, the time-series of DVFS states of individual application should be collected over long enough duration to ensure DVFS states are monitored for different phases of application execution. We expect a longer time-series of DVFS data will increase overall detection accuracy.

## 4.4 Comparison with HMD

Table 4.4: Comparison of proposed work with existing hardware based malware detectors

| Reference | Information | Platform | Performance | Features | Detection Algorithm |
|---|---|---|---|---|---|
| [6] | Performance Counters (HPC) | OMAP4460, ARM Cortex-A9, Android 4.1 | 83% AUC for 10% (FPR) | 368M samples | Decision Trees, ANN |
| [16] | Power Signal (AC) | Embedded Medical Device | 94% (known) and 85% (unknown) | 8 time-domain features (statistical), spectral | RF, 3-NN, Perceptron |
| [49] | Performance Counters (HPC) | Intel ATLASEDGE, ARM processor | 93% | 2, 4, 8 HPCs | MLP, OneR, JRip |
| Proposed | DVFS | Snapdragon 820, ARMv8 processor, Android 7.0 | $F_1 score$ = 99% (malwares variants) | DVFS states (2 CPU cores) | SVM, RF, K-NN |

There have been prior works that leverages machine learning methods using features derived from software, hardware or combined for malware detection[6][16][49][10]. But, no prior methods have focused on feasibility of DVFS based power signatures malware analysis and subsequent detection. We compare the proposed DVFS based detector against hardware feature-based machine learning techniques, like performance counters, power and EM side-channel etc. To show a fair comparison, we selected literature that explores signature-based approach for malware detection. The summary of the comparison is described in Table Table 4.4. We compare the detectors based on machine learning performance metrics, like precision, recall, AUC etc, and their feature complexity to access time to detection (inference time).

The performance of DVFS based detection is comparable to other hardware-based detection techniques as tabulated in Table 4.4. Note, datasets used for different detectors are not identical. For instance, J.Demme et.al. have evaluated the accuracy on 37 malware families [6]. S.M.P Dinakarrao et.al. have considered malwares belonging to 4 categories, namely, backdoors, trojan, virus, rootkits in [49]. S.S. Clark et.al. targeted specific malwares for medical devices [24]. The complexity of detector and time to detection can be evaluated using feature dimensions. J.Demme et.al. have demonstrated detector with 368M performance counter samples of malware and non-malware in ML model [6]. S.M.P Dinakarrao et.al. have shown reduced feature complexity of OneR classifier detection model for IoT devices using 2 HPC [49]. This would have similarities to DVFS based detector which relies on DVFS states from 2 cores. The exact comparison on feature complexity also depends on length of data collection, which indeed is different. These key insights derived from these comparisons are DVFS based malware identification is much simpler and has smaller footprint due to its low feature complexity.

## 4.5 Summary

This chapter experimentally demonstrated an ML methodology to detect malware applications from benign by deriving features from DVFS states of a CPU core. We showed an extension of the proposed machine learning methodology of distinguishing application to malware detection. The experimental results show $> 0.88$ $F_1score$ using RF model between benign and malware applications evaluated across all governors. We explored power secure awareness of selecting cpufreq governor by evaluating power consumption of different benchmark and malware applications. The applications dissipate less power on average with ondemand governor configuration and also results in higher detection accuracy. A framework for practical on-device malware detection system with cloud-based software updates features is discussed. A comparison with other HMD shows that DVFS based detection is simpler with smaller memory footprint due to its low feature complexity.

The detection of malware application is followed by analyzing its behavior such as identifying malware family. The next chapter demonstrates ML methods and spectral and wavelet-based feature extractions from Electromagnetic signal for classifying detected malware into malware family.

# CHAPTER 5
# REMOTE SIGNATURE-BASED MALWARE DETECTION AND FAMILY CLASSIFICATION USING ELECTROMAGNETIC EMISSIONS

Physical side-channels incorporates fine-grained information about program's instruction sequences, control flows and data [37, 23]. As an alternative to on-device detection, there is a growing interest in developing remote malware detectors that leverage physical side-channels such as power and EM emissions[26, 16]. Recently researchers have demonstrated the capability of detecting malicious applications by remotely monitoring EM emissions from a device[26, 16, 15, 14, 27]. The detectors leveraging EM side-channel emissions perform external monitoring and subsequent detection, thereby eliminating the need for on-device implementation.

A large proportion of malwares are variants of baseline characteristic family, which emphasizes need to evaluate malware behavior and subsequently classify them into families[53, 5]. Generally, malware writers create multiple variants of malwares by adopting obfuscation techniques to evade detection. But these variants have similar underlying functionality like registering to premium services, stealing contacts, SMS etc. Since malware samples are abundant, it is efficient to categorically separate them into groups based on their unique behavior. Moreover, malware families have similar removal techniques and their signatures guides robust anti-virus solutions [5]. Hence, along with detection, the classification of malware into families is critical for detailed analysis.

This chapter presents a signal processing and machine learning (ML) approach for EM-based malware analysis that distinguishes malware applications from benign ones along with identifying characteristic malware family of detected malware as shown in Figure 5.1. We utilize frequency and wavelet-based approaches for extracting features from EM-signals to be used for detection and classification. We show that ML methods operat-

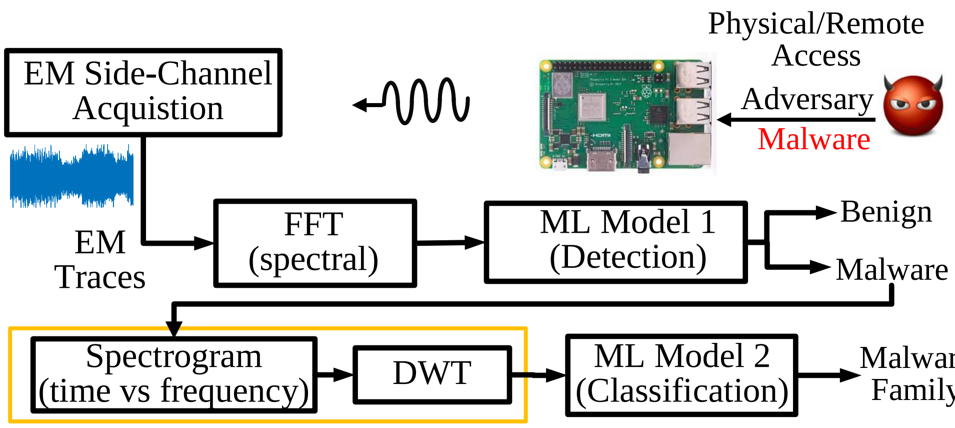

Figure 5.1: EM side-channel monitoring setup, feature extractor and machine learning model for detecting threat and classifying its characteristic behavior

ing on features computed from 1D- Short Time Fourier Transform (STFT) can be used to distinguish malwares from benign application. However, spectral features obtained from EM-traces are not sufficient to distinguish between different malware families. To obtain fine-grain temporal and spectral features from EM-traces, we propose to perform discrete wavelet transform (DWT) of a spectrogram derived from EM side-channel trace which captures power spectral density variations along time and frequency axis of the spectrogram. The developed ML methods operate on the multi-level wavelet coefficients to perform malware classifications. The ML detection and analysis model is trained on subset of benign and malware applications. Any new/updated application is identified as "unknown" by comparing prediction probability of model with pre-defined threshold. The model is re-trained with features of new/updated application.

The experimental validation is done by performing EM side-channel measurements on Intrinsyc Open-Q 820 Development Kit, Snapdragon 820 processor, Android 7.0 OS [54] using passive EM side-channel probes[55]. The ML models are trained to android benchmark applications (benign) and malware applications collected from drebin dataset [50]. We consider Support Vector Machine (SVM) and Random Forest (RF) based ML models for malware detection and classification.

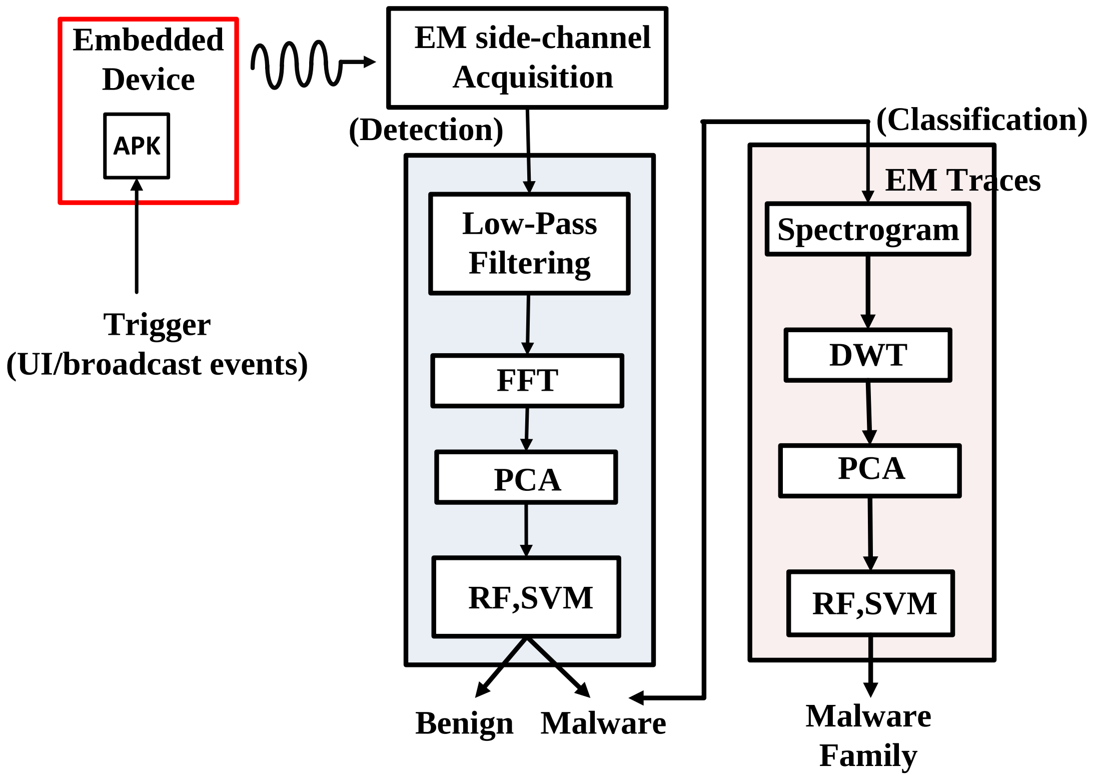

Figure 5.2: The EM side-channel detection and analysis framework comprises of EM side-channel monitoring, acquisition, post-processing, feature extraction steps and ML inference model

## 5.1 Threat Model

Threat model assumes the attacker may have physical or remote access to the embedded device to directly or indirectly install malware. The attacker can inject malware into the system by physically accessing the device. Alternatively, the attacker can package the malware within a benign application and an user interacting with device unknowingly installs the malware-infected application on the device. Once installed, the malware gets triggered using an in-build trigger mechanism.

## 5.2 Malware Detection and Analysis Framework

The remote malware analysis framework comprises of an EM side-channel acquisition unit that monitors and collects EM side-channel emissions from the computing device. The malware detection block collects the sampled EM side-channel traces and subsequently learns patterns in EM signatures through ML models. The objective of malware detection block is to red-flag EM signature of malware infected applications during their runtime. The detection of malware application is generally followed by analyzing a malware application to identify patterns representative of known malware behaviors. The malware analysis unit derives fine-grained features from the collected EM side-channel signatures and classifies the detected EM side-channel signature among known malware families. The malware detection and analysis framework assume no prior knowledge about the application (malware or benign) under execution. The framework can recognize the application start by abrupt modulations in EM side-channel activity.

### 5.2.1 EM Side-Channel Acquisition

The objective of the EM side-channel acquisition block is to obtain a EM side-channel trace. A trace comprises of uniformly sampled time-varying Electromagnetic energy. The side-channel acquisition block comprises of far-field antenna, a low-noise amplifier (LNA) and oscilloscope to sample the acquired signal. The details on selection of EM probes (near and far-field), amplifiers and signal acquisition setup have been explored in context on EM side-channel acquisition. The remote side-channel detection would require high-gain, high bandwidth antennas. The prior works on leveraging EM side-channel for mounting attacks and more recently detecting malwares have demonstrated use of loop antenna's [24], horn antenna, panel antenna [27] and even microphones to collect EM side-channel signatures [26]. Similarly there are numerous choices in signal acquisitions, like oscilloscopes, spectrum analyzers, and software defined radio's (SDR).

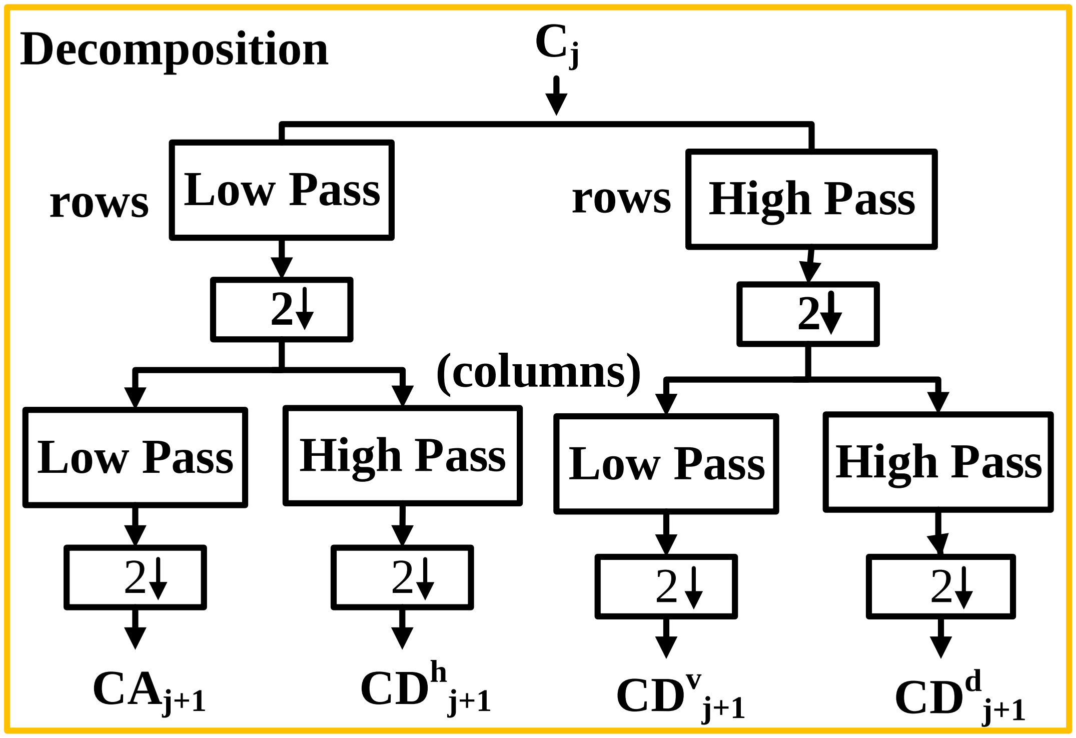

Figure 5.3: 2D DWT as filter bank decomposes input spectrogram into approximate, horizontal, vertical and detail coefficients.

### 5.2.2 Feature Extraction

The collected EM side-channel traces are input to the malware detection unit. The feature extraction block derives application specific features from the EM side-channel traces. Generally, both time and frequency domain-based information are used for feature extraction. Fourier techniques are one of the unique descriptors, which gives spectral content of a signal at multiple frequency points. We applied windowed FFT on EM side-channel traces to derive features for detection.

The FFT based feature extraction method ignores the temporal variations in EM side-channel trace. The malware analysis unit classifies the detected malware into respective families, which requires deriving fine-grained patterns pertinent to malware family from EM side-channel trace. These fine-grained patterns can lie at different time points in dif-

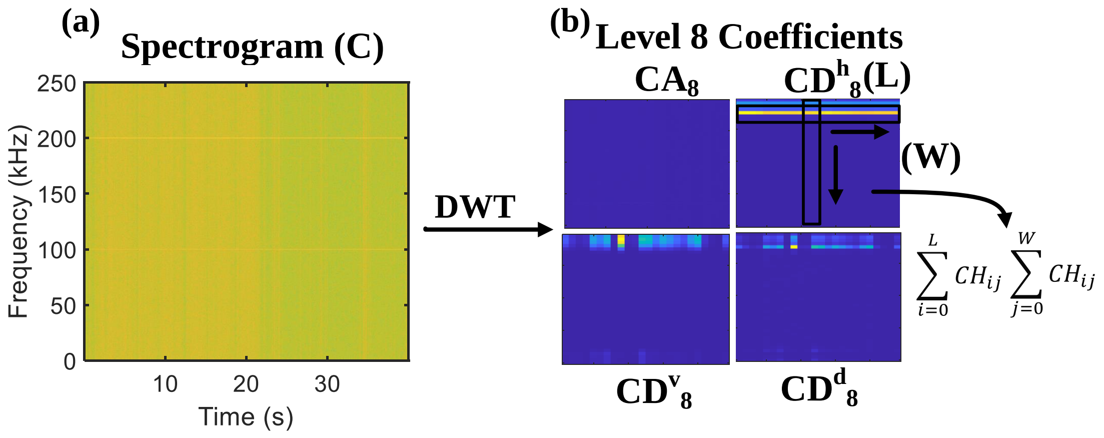

Figure 5.4: (a) Spectrogram of EM side-channel trace gives power spectral density variations (psd) against time. (b) DWT applied on spectrogram resolved at level 8

ferent frequency regions. Therefore, first we segment EM side-channel trace into multiple short windows, compute short time fourier transform (STFT) and finally aggregate windows to obtain a spectrogram. It depicts power spectral density (PSD) variations of the EM side-channel trace over time. A spectrogram generated from EM side-channel trace is demonstrated in Figure 5.4(a). The subsequent step is to derive features from the spectrogram. Since, unique patterns can lie in multiple frequency regions, the spectrogram is resolved at multiple frequency bands with a wavelet basis by performing a 2D – DWT. A 2D – DWT is computed by applying scaling and wavelet basis functions to 2D input like spectrogram. It comprises of one scaling function and 3 wavelet basis function. The wavelet transform on 2D input is demonstrated in the equation below

$$w_\phi(j_0, m, n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\phi_{(j_0,m,n)}(x, y) \tag{5.1}$$

$$w_\psi(j, m, n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\psi^i_{(j,m,n)}(x, y) \tag{5.2}$$

where, $w_\phi(j_0, m, n)$ are approximation coefficients, $j_0$ is the starting scale factor and $w^i_\psi(j, m, n)$ represents the horizontal, vertical and diagonal coefficients. M and N represents size of 2D input $f(x, y)$. The wavelet decomposition steps at each level is described in Figure 5.3. At the first level, rows of the 2D input are lowpass and highpass filtered followed by downsampling. Next, columns of filtered rows are high-pass and lowpass filtered and downsampled. The wavelet transforms on a 2D input, like an image measures the intensity variations across horizontal, vertical and diagonal directions. Also, scaling factor of wavelet basis can be modulated to resolve the 2D input at multiple levels. Lower resolution levels give information about larger structures in 2D input and higher resolution give details. A plot demonstrating the 2D – DWT on spectrogram at lower resolutions (level 8) with symlet basis function is demonstrated in the Figure 5.4(b).

The multi-level wavelet decomposition on the spectrogram is followed by aggregating coefficients to be used as features. There can be multiple statistical metrics that can be used to derive features from the wavelet coefficients. We used average of the coefficients along both axes as features. The feature vector comprises of aggregating the mean values of all wavelet coefficients from all the decomposition levels. DWT also depends on selection of mother wavelet or wavelet basis function. We explored multiple wavelet basis including haar, symlet, daubechies and evaluated the performance for classifying malware family using these evaluated wavelet bases. The feature vector ($F.V.$) obtained from the wavelet decomposition is given by:

$$\begin{aligned} F.V. = \sum_{i=0}^{L} CA^k \sum_{i=0}^{W} CA^k \sum_{i=0}^{L} CH^k \sum_{i=0}^{W} CH^k \\ \sum_{i=0}^{L} CV^k \sum_{i=0}^{W} CV^k \sum_{i=0}^{L} CD^k \sum_{i=0}^{W} CD^k ... \end{aligned} \tag{5.3}$$

where, $CA^k$, $CH^k$, $CV^k$, $CD^k$ represents different wavelet coefficients at decomposition level k. L and W denotes the length the length and width of wavelet coefficients.

### 5.2.3 Dimensionality Reduction

The feature extraction block is followed by dimensionality reduction. The feature vector comprises of FFT coefficients from high-dimensional segmented EM side-channel trace. Similarly, aggregating wavelet coefficients from multiple levels of decomposition results in a high-dimensional feature vector. We applied principal component analysis (PCA) for reducing the feature dimensions. PCA is a dimensionality reduction technique that projects the multi-dimensional feature vectors along directions that maximizes variance. The directions or number of principal components is a hyper-parameter that is selected such that 90% variance is retained in reduced feature vector. The principal component has highest variance in their increasing order.

### 5.2.4 Machine Learning Algorithms

The reduced dimensional feature vector is fitted using ML models. The task of malware detection and malware analysis requires distinguishing malwares against benign and different malware families respectively, therefore a classification-based ML model is used for fitting the feature vector. The selection of ML model is based on complexity of feature space. We evaluated the linear separability in the feature space by selecting linear kernel of SVM model and set the regularization parameter to very high value. This forces the optimizer to make 0 error in classification. We fit the training data and evaluated the accuracy on the same dataset. A 100% accuracy on the training dataset implies 0 training error, and optimal fitting with linear kernel. We examined SVM(linear kernel) can linearly separate features of benign and malware applications, but not individual malware families.

Random Forest(RF): To learn non-linear complexities of the feature space, we also evaluated random forest classifier for malware detection and family classification. Random Forest is an ensemble learning technique, which fits the training data to number of decision tress specified by hyper-parameter (n_estimators) and uses averaging to improve the prediction accuracy and avoid overfitting. We selected random forest algorithm because decision trees can fit non-linear training data and they are computationally less intensive compared to neural network. We explored different hyper-parameter for RF model like quality of split criteria, maximum depth of tree, maximum features for split.

Support Vector Machines(SVM): SVM algorithm finds a maximum separable hyper-plane to divide the feature space. The SVM is a kernel based classifier with linear, rbf and polynomial kernel functions to fit complex feature space. We explored hyper-parameters like regularization parameter, non-linear kernels for the explored dataset.The SVM classifier for multi-class classification is implemented in one-vs-one scheme

We applied randomized search to find optimal hyper-parameter configurations for high validation accuracy and subsequently used the model with optimal configuration for inference. We used the scikit-learn python machine learning libraries for RF and SVM classifier implementations.

## 5.3 EM Side-Channel Trace Collection

The proposed approach is demonstrated on Intrinsyc Open-Q 820 Development Kit [54]. The EM side-channel acquisition setup in described in Figure A.1. The EM side-channel probes are placed in proximity to device for experimental feasibility purposes, but practical detector would involve high gain antenna placed at distance from the device under analysis.

There are three computing platforms used for side-channel measurements as shown in Figure 5.5. A Laptop is used to interface with experimental platform, a desktop is used to record and save the EM side-channel traces by reading memory buffer of the calibrated oscilloscope and experimental platform (Snapdragon 820) is used for triggering workloads. A script is used to launch applications, generate trigger inputs for application as well as oscilloscope to start and stop measurement. This script executes on laptop. The trigger inputs to application are simulated using monkey tool and are sent using android debug interface (adb) shell commands. We used broadcast events as well as user touches, swipes, system level events as trigger inputs. Similarly, trigger inputs are sent to oscilloscope using

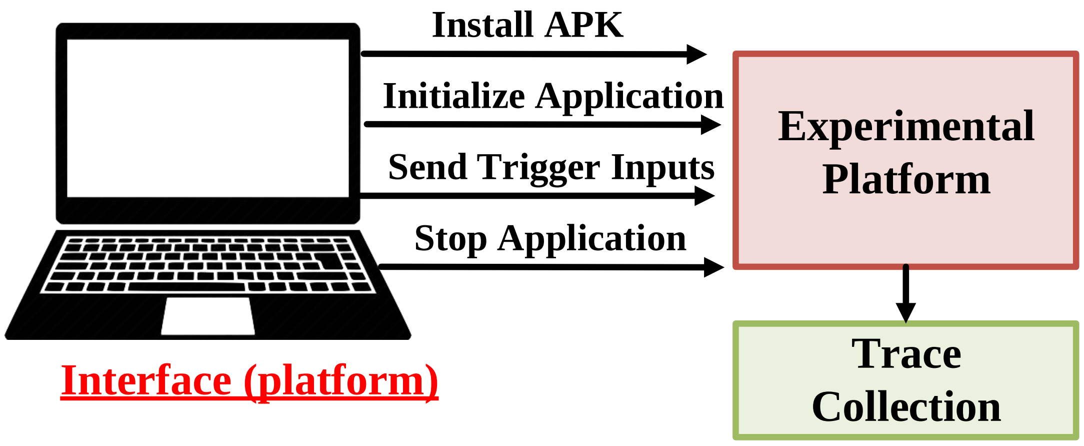

Figure 5.5: EM side-channel Trace collection setup and laptop interfaced with the experimental platform to trigger workloads and synchronize data collection

Arduino Uno. A MATLAB script executes on desktop to read collected EM side-channel samples and store in a file. The sequence of steps followed to capture a single trace is detailed below:

- Initialize the default activity of targeted application using adb shell command
- Send a trigger (0-1 transition) to oscilloscope through Arduino Uno to start recording EM side-channel signature
- Send trigger inputs to android application using adb shell commands for profiled process duration
- Stop the targeted application using adb shell command
- Send a trigger (1-0 transition) to stop the oscilloscope

These steps are repeated to collect multiple EM side-channel traces of every single evaluated application. The sequence of trigger inputs to application is kept same for all captures. In case of malware applications, each trace capture is followed by rebooting the platform to re-initialize to original state for subsequent capture. The two scripts running on laptop and desktop are synchronized by specifying delay equivalent to time for saving a trace file. Once, the signatures are collected, training and testing steps are subsequently performed offline.

## 5.4 Distinguishing Malware Against Benign Applications

### 5.4.1 Dataset Generation

Benign Dataset: The benign dataset evaluated for this study comprises of 14 android benchmarks. The benchmark applications comprises of whetstone, dhrystone, linpack and loops, randmemi. Among these some are CPU bound benchmarks. For instance, linpack benchmark performs floating point operation like addition and multiplication while others like busspeed and randmemi are memory bound benchmarks. Randmemi tests data transfer speed from caches and memory.

Malware Dataset: For detection, we selected 1 application (1 sample) each from 14 randomly chosen malware families during training. In addition to this, we selected 146 other malware applications, selected randomly from same 14 malware families. But these, 146 malware applications are evaluated during inference and these are variants of 14 malware applications selected during training.

The performance of malware detection during inference depends on uniqueness of the derived features of malware and benign applications. The uniqueness in feature vectors of these applications is contingent to similarities in EM side-channel traces. The EM side-channel emission's unique correlations with application during its runtime has been established experimentally in prior works [36]. It can be concluded from those observations that malware and benign applications have distinctive EM side-channel signatures. The training and inference methodology steps are detailed in subsequent subsections.

## 5.4.2 ML Model Building

The malware detection methodology is evaluated on dataset comprising of benign and malware applications. We selected 14 android benchmark applications and 14 malware applications. Multiple EM side-channel traces are collected for each application in benign and malware dataset. Increasing number of traces per applications improves learning under uncorrelated noisy environment. The number of traces per application is fixed at 75. The EM side-channel traces comprises of noise resulting from multiple factors, including measurement noise, OS scheduling policies, context switching and concurrent thread execution. We haven't specifically introduced any constraints like core pinning, core isolation etc. to minimize system level noise during EM side-channel data collection. In addition to this, we have disabled external communication peripherals like Bluetooth, WiFi and Modem, so that malware cannot send sensitive information to external servers. The EM side-channel signatures are captured for duration of 10s and each signal is sampled at 2MHz frequency. EM side-channel traces are first segmented into 100ms window and each segmented trace is filtered using low-pass butterworth filter at 2KHz cut-off frequency. The filtered sub-trace is transformed into frequency domain by performing FFT on each window. This windowed FFT is performed on all sub-sequences of EM side-channel traces. After performing FFT, PCA is applied separately on individual sub-traces to obtain a reduced dimensional representation of feature. The number of principal components in each window is experimentally evaluated for retaining high-variance and set as 40. The feature vector is formed by concatenating reduced representation of each sub-trace sequentially and fitted using SVM and RF model. We explored linear kernel SVM and non-linear RF model as detector. Table 5.1 shows different hyper-parameter configuration chosen with SVM and RF model.

Table 5.1: Training and Inference configurations for Malware Detection

| **Configuration** | **Training** | **Inference** |
|---|---|---|
| Applications | 14 benign, 14 malware | 14 benign, 146 malware |
| Traces | 2100 | 496 |
| Filtering | butterworth, 2KHz cut-off frequency | |
| window_size | 200K Samples (0.1 sec) | |
| PCA | n_components = 40 | |
| RF | n_estimators = 40, gini impurity | |
| SVM | C=1, linear kernel | |

### 5.4.3 Model Evaluation

The test dataset comprises of 25 traces of 14 benign applications collected in different background state and 146 new malware applications selected from 14 malware families evaluated in the dataset. The new malware applications are variants of malware families selected in training. EM side-channel traces are first segmented and FFT is applied on each sub-trace. The windowed FFT of each sub-trace is projected to principal component axis to obtain reduced representation of each window. Finally, feature vector is generated by concatenating reduced feature dimensions from all windows. The feature vector is tested on the trained ML model and performance is evaluated using $F_1\ score$. The validation curve is obtained for malware detection using SVM and RF model as shown in Figure 5.6. The training error is low with increasing number of training examples. This indicates low bias and higher validation scores with increasing number of training examples indicates lower varaince (no overfitting). The selected ML models have optimally fitted on EM side-channel features. The cumulative test dataset accuracy ($F_1\ score$) with SVM model is 99$\%$ and RF is 97$\%$. The $F_1\ scores$ of benign and malware classes with SVM and RF model are described in Table 5.2. The training and inference configurations for malware detection is shown in Table 5.1

Table 5.2: F1-scores for benign and malware applications with different detection models

| **Class/Model** | SVM | RF |
|---|---|---|
| Benign | 0.99 | 0.97 |
| Malware | 0.98 | 0.94 |

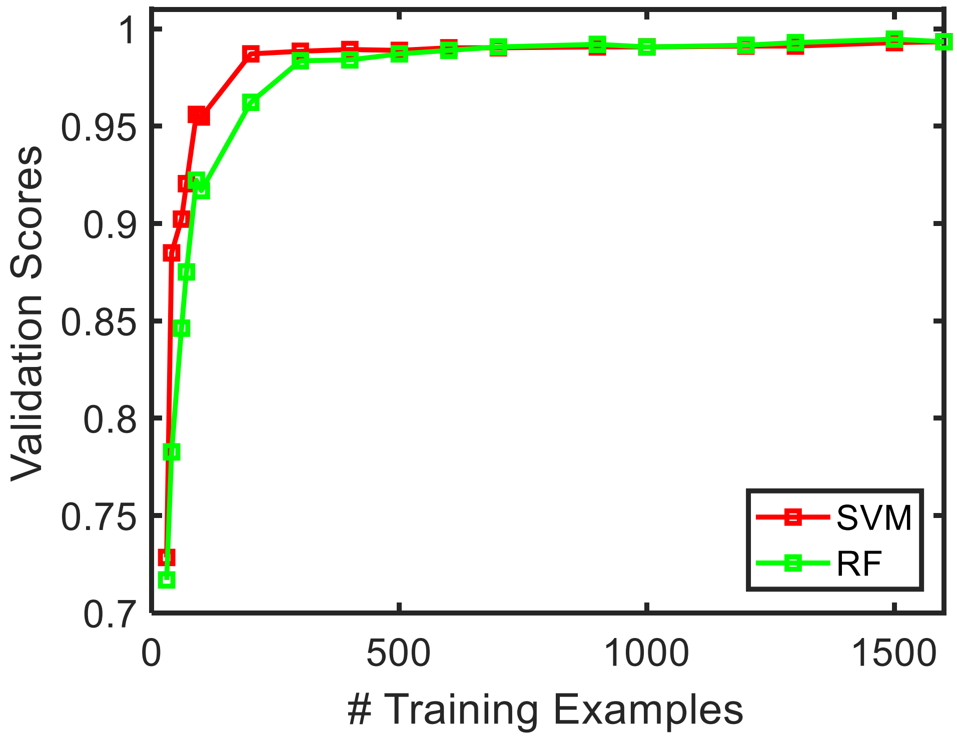

Figure 5.6: Validation Score against number of training examples (Learning Curves) using SVM and RF model for malware detection

## 5.5 Distinguishing Malware Family

### 5.5.1 Dataset Generation

For classification, 137 malware applications selected from 8 malware classes. We selected these malware families, “Adrd” , “BaseBridge”, “DroidKungFu”, “Geinimi”, “GinMaster”, “Goldream”, “Kmin” and “Yzhc”. We selected these 8 malware families out of 154 different families in drebin dataset for following reasons. First, selected malware families are

Table 5.3: Number of Samples and Selected Malware Families from Drebin Dataset

| Malware Family | Samples |
|---|---|
| Adrd | 11 |
| BaseBridge | 25 |
| DroidKungFu | 19 |
| Geinimi | 3 |
| GinMaster | 19 |
| Glodream | 19 |
| Kmin | 18 |
| Yzhc | 23 |

among the top 20 common malware families in the drebin dataset [50], which have higher number of samples in comparison to other 134 malware families, having fewer than 10 malware samples. Second, datasets aside from drebin, for instance MalwareGenome [51], Contagio-Dump [56] and VirusShare [57] also includes these 8 selected malware families, which are among the common malware families. Third, selected malware families have been explored in various android malware detection/family classification research, even most recently [58, 59]. Finally, selected malware families share one of the common mechanisms for activation, i.e., registering to android system events, "BOOT_COMPLETED", which triggers malware activity on system reboot [60]. The number of samples of each malware family is shown in Table 5.3. Although, malware classification framework is analyzed on 8 malware families, But the proposed framework can be extended to multiple malware families.

### 5.5.2 Distinguishing Malware Family Using Welsh's t-test

An application's unique correlation with EM signatures is not indicative of similarity between features of multiple applications belonging to same malware family. There are multiple approaches both statistical and visualization based utilized to analyze distinguishably between two signatures like KL divergence, dynamic time warping, t-test, T-SNE etc.

We present Welsh's t-test to evaluate ***"if the EM side-channel signatures of two mal-***

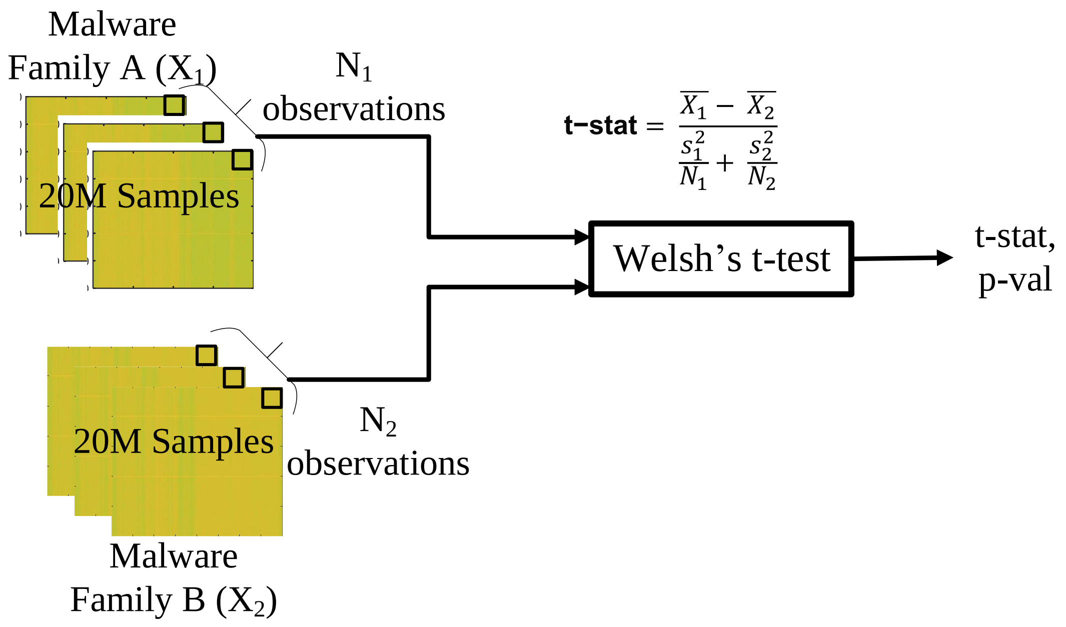

Figure 5.7: Welsh t-test analysis on spectrograms of applications selected from different malware families

***ware families are distinguishable".***. Eventually, the uniqueness in feature vectors of a malware family is contingent to similarities in EM side-channel traces. Therefore, we perform welsh's t-test on input space rather than reduced feature space. We performed welsh's t-test analysis on spectrogram input. The welsh's t-test analysis is compared using dissimilarity score (D.S.), which gives proportion of trace points where $t - statistic > |4.5|$. A higher dissimilarity score indicates more distinguishable input signatures of malware families. Dissimilarity score is given by equation Equation 5.4 and its value ranges between 0 and 1

$$D.\ S. = \frac{Trace\ Points,\ \ni\ (t - statistic\ > |4.5|)}{Trace\ Length} \tag{5.4}$$

First, a pair-wise t-test is performed on applications selected from different malware families to establish distinguishability between malware family. The t-test is applied at all time/frequency points of the represented input. Welsh's t-test analysis on spectrogram input of applications is shown in Figure 5.7. The number of observations ($n_1$ or $n_2$) includes

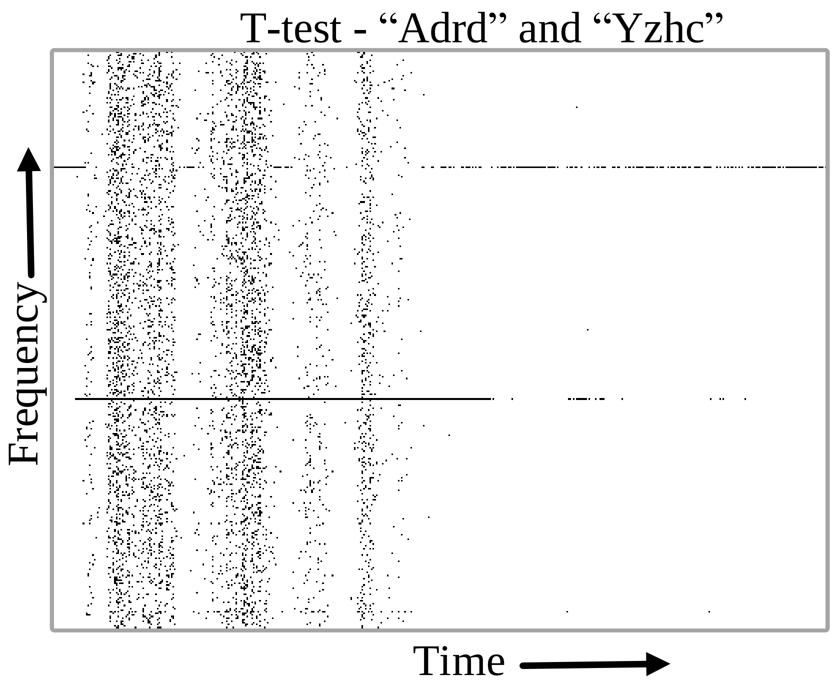

Figure 5.8: T-statistic scores represented as 2D plot, where black regions represent $t - statistic > 4.5$ or $t - statistic < -4.5$

multiple instances of all applications in the selected malware family. The number of observations vary between malware families. The null hypothesis is rejected for every time, frequency point where t-statistic crosses the threshold. The t-statistic plot for a spectrogram input can be represented in a 2D plane as shown in Figure 5.8. The plot shows t-score plane corresponding to t-test between “Adrd” and “Yzhc” malware family. The black portions of the trace correspond to rejection in null hypothesis, which eventually indicates regions of statistical variations or distinguishability between two malware families and vice-versa for white regions.

Second, t-test is performed among applications belonging to same malware family to demonstrate more similarity and less distinguishability. Here, input traces of applications selected from malware family are split into two groups. Then t-test is performed at all time / frequency points of the representative input. The number of observations ($n_1$ and

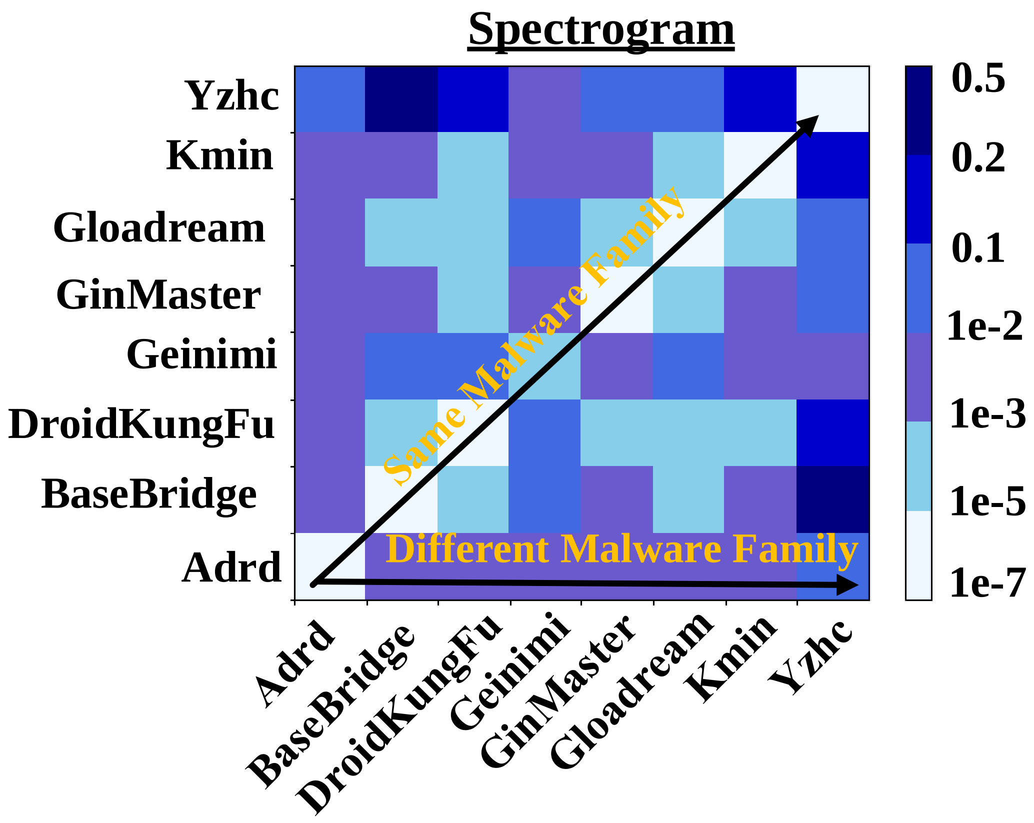

Figure 5.9: Heatmap plot of dissimilarity score (time or frequency points), where $t - statistic > |4.5|$ evaluated for spectrogram of raw trace

$n_2$) for t-test is comparatively lower than testing for distinguishability between malware families. The null hypothesis is rejected for every time, frequency bin when t-statistic crosses the threshold. We obtained t-score plane corresponding to applications belonging to "Adrd" malware family. We observed that the number of black regions on the t-score plane got significantly lowered, with very few rejections of the null hypothesis, indicating more similarity and lesser distinguish-ability between applications of "Adrd" malware family.

Welsh's t-test results: The input traces of applications within same malware family should retain more similarities and less differences when compared across malware families. We tested this hypothesis by measuring dissimilarity score for all possible pairs of malware families and obtained a heatmap plot of dissimilarity score for spectrogram input as shown in Figure 5.9. The plots are symmetric along the diagonal which corresponds

Table 5.4: Training and Inference Configurations For Malware Family Classification

| Configuration | Training | Inference |
|---|---|---|
| Applications | 137 | 137 |
| Traces | 548 | 137 |
| PCA | n_components=20 | |
| Spectrogram | window = hamming, window_size = 10000<br>overlap = 50%, mode = psd | |
| Wavelet | haar, symlets (order=5), daubechies (order=5) | |
| RF | n_estimators = 1 to 500, information gain = gini,entropy<br>max_depth = 2 to 20, max_features = 2 to 8 | |
| SVM | C,gamma = 1,10,100,1000, kernel=linear,rbf,poly<br>degree = (2 to 6) | |

to applications of same malware family. The lower intensity regions on heatmap indicates more similarity and less distinguishability between selected pairs and vice-versa for high-intensity regions on heatmap plot. The low-intensity regions across diagonal indicates more similarity among applications of same malware family, and increasing intensity regions, indicates more distinguishability among applications of different malware family.

### 5.5.3 ML Model Building

The dataset comprises of 137 malware applications selected from 8 malware families in drebin dataset [50]. We collected 5 traces for each selected application. The training dataset is formed by collecting EM side-channel signatures of 80% of applications selected from each of malware family. These applications are selected at random by keeping fixed seed. The number of applications per malware family is shown in Table 5.3. The EM side-channel trace includes noise arising from numerous sources like measurement noise, OS scheduling policies, kernel level threads and context switching. We did not specifically impose any constraints during side-channel acquisition of profiled application.

First step in training procedure is to generate a spectrogram of each EM side-channel trace. We tested different configurations of window size for generating spectrograms. We experimentally varied length of segment for computing the windowed FFT and obtained the power spectral density (PSD) variations with time. The length of segment is selected appropriately for finer resolutions in both time and frequency. The output comprises of frequency bins, time samples and PSD at each frequency and time-point. In second step, the spectrogram is resolved at multiple levels and each level generated set of wavelet coefficients for performing 2D-DWT. We experimented with different wavelet basis function namely haar, symlets and daubechies. We followed similar steps of feature extraction as described in subsection 5.2.2 to obtain feature vector. Lastly, we reduced the feature dimensions by applying PCA and finally fit the reduced dimensional feature vector using ML model. We explored multiple hyper-parameter configurations described in Table 5.4 and used randomized search algorithm to derive at best hyper-parameter configurations for highest cross validation accuracy. In our experiments, different hyper-parameter configurations ranges were explored, and 100 iterations of random sub-sampling was used to select hyper-parameter configurations. Different configurations during training are detailed in the Table 5.4.

### 5.5.4 Model Evaluation

The trained ML classification model is tested on dataset comprising of 20% of the remaining applications of each malware family. The testing dataset has 137 examples belonging to 8 malware families. The EM side-channel traces are converted to a spectrogram and DWT is applied to resolve the spectrogram at multiple levels, with each level generating set of approximation and wavelet coefficients. The feature vector is averaged of coefficients at all decomposition levels. The feature vector dimensions are projected to PCA to form reduced dimensional vector and output is evaluated on trained machine learning model with optimal hyper-parameter configurations.

Accuracy Results: The inference accuracy for different ML models and feature extraction techniques is summarized in Table 5.5. The test dataset inference accuracy ($F_1\ score$)

Table 5.5: $F_1$ $scores$ for Malware family classification with different ML models

| Features/Model | SVM | RF |
|---|---|---|
| 2D-DWT (haar) | 0.87 | 0.81 |
| 2D-DWT (sym5) | 0.88 | 0.81 |
| 2D-DWT (db5) | 0.74 | 0.77 |

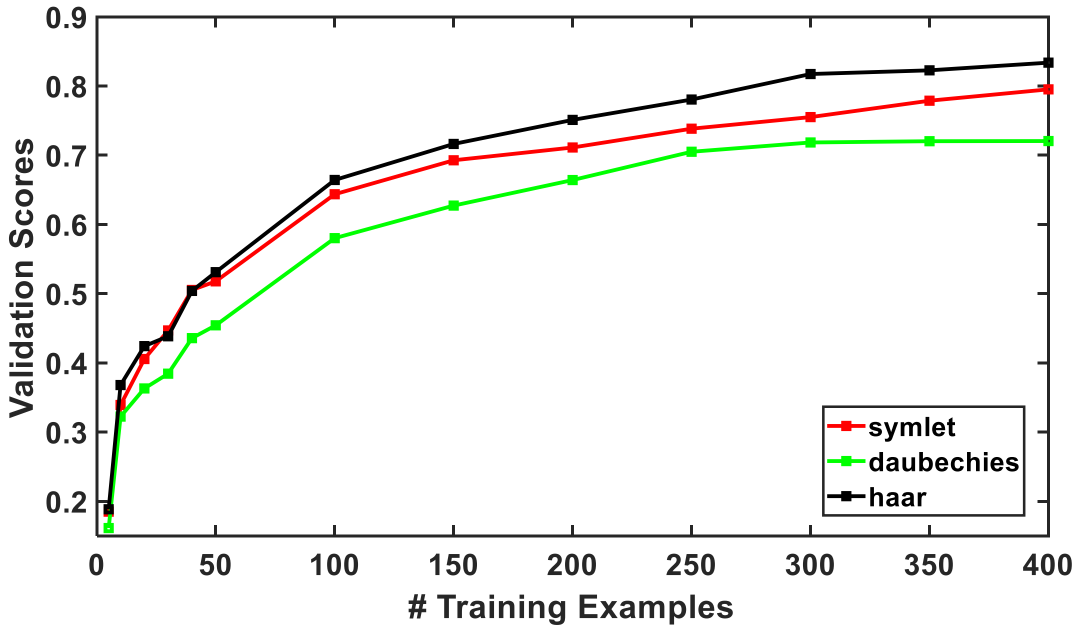

Figure 5.10: Validation Score against number of training examples for different wavelet basis using SVM model for malware family classification

is also evaluated for different wavelet basis as described in Table 5.5. The $F_1$ $scores$ are calculated for optimal configurations of both SVM, RF model. The optimal hyper-parameter configurations for SVM are ($C = 1$, $gamma = 10$, $kernel = rbf$) and RF are ($n\_estimators = 350$, $max\_features = 5$, $gini\ impurity$). The optimal configurations for both ML models depict a higher non-linearity in features of different malware family. We observe that selection of wavelet basis function also influences the classification accuracy. We obtained higher inference accuracies with "haar" and "symlet" basis compared to "Daubechies".

Learning Curves: We obtained the validation curves corresponding to best hyper-parameters

Table 5.6: Impact of PCA on $F_1$ *scores* For malware family classification

| **Features/Model** | PCA | No-PCA |
|---|---|---|
| 2D-DWT (haar) | 0.87 | 0.86 |
| 2D-DWT (sym5) | 0.88 | 0.89 |
| 2D-DWT (db5) | 0.81 | 0.82 |

for different basis function as shown in Fig. Figure 5.10. Low bias obtained during training with SVM model is result of higher training score with increasing number of training examples. Furthermore, comparatively higher variance is obtained during training for all evaluated basis function. Although, validation score is increasing with number of training examples, it implies given more training data, better fitting can be achieved. We also infer trend in validation scores with increasing training examples are similar for different wavelet basis, but selection of haar wavelet offers more generalized model with comparatively lower-variance.

Impact of PCA: To verify the significance of applying PCA, we evaluate classification accuracy in two scenarios. In first case, feature vector is directly fitted to classification model and in second case, reduced feature vector after applying PCA is fitted. We selected SVM model and fixed the optimal hyper-parameters configurations that were obtained for malware family classification. $F_1$ *scores* are summarized in Table Table 5.6 for malware family classification using SVM model. We observe the $F_1$ *scores* are very close or similar, both with or without PCA as processing step. Therefore, we conclude (PCA) step is not applied to improve separability of feature space, but instead to remove redundant information and obtain less-complex feature space for optimal fitting. The feature vectors obtained prior to applying PCA are already distinct for each malware family.

Welsh's t-test vs $F_1$ *scores* Comparison: We compare relations between t-test derived dissimilarity score and $F_1$ *scores*. This comparison answers: "**Does more distinguishable malware families have higher classification accuracy ?**". To draw this comparison, we obtained confusion matrix for malware family classification with highest inference accu-

|  | Adrd | BaseBridge | DroidKungFu | Geinimi | GinMaster | Gloadream | Kmin | Yzhc |
|---|---|---|---|---|---|---|---|---|
| **Adrd** | **10** |  |  |  |  |  | 1 |  |
| **BaseBridge** |  | **24** | 1 |  |  |  |  |  |
| **DroidKungFu** | 1 | 2 | **10** |  | 3 | 1 | 2 |  |
| **Geinimi** |  |  |  | **3** |  |  |  |  |
| **GinMaster** |  |  | 1 |  | **17** | 1 |  |  |
| **Gloadream** |  | 1 | 2 |  |  | **16** |  |  |
| **Kmin** |  |  | 1 |  |  |  | **17** |  |
| **Yzhc** |  |  |  |  |  |  |  | **23** |

Figure 5.11: Confusion matrix for malware family classification using proposed feature extraction techniques and SVM model

racy in Table 5.5 and compared with heatmap plot of dissimilarity score evaluated for DWT output at level 2 as shown in Figure 5.9.

We make following key deviations: First, we observe, “Yzhc” malware family has a greater number of high-intensity regions against other malware families. This is also reflected in higher $F_1 scores$ and no False Positives (FP) and False Negatives (FN) in confusion matrix for this malware class as shown in Figure 5.11. Second, we observe higher intensity regions for “BaseBridge” malware family against 4 others, where dissimilarity score $> 0.1$ as shown in Figure 5.9. Also, “BaseBridge” malware family is classified with higher $F_1$ $scores$ and fewer False Negatives (FN). Third, “DroidKungFu” and “Adrd” have greater number of low-intensity regions. A lower dissimilarity score for “DroidKungFu” is also reflected with higher FP and FN in confusion matrix for this malware class. But, “Adrd” has contrasting behavior. Although comparison between t-test and classifier output provides interesting insights. However, t-test results may not necessarily dictate a classifiers output. Even though t-test results may show close similarity between two malware families, an ML model may still separate their feature vector using complex non-linear kernels.

### 5.5.5 Analysis on Feature Selection

To understand the significance of applying a second discrete wavelet transform (DWT) on spectrogram, we evaluate relevance of features using mutual information metric and inference accuracy (F1 scores). The ML model is evaluated for different feature extraction techniques to quantify the significance of applying DWT on spectrogram. We evaluate the inference accuracy for three different feature extraction techniques.

1D-DWT on Raw Trace: A 1D-DWT decomposes raw time domain EM signal using a scaling and a wavelet function to generate two sets of coefficients: approximate and detail. The EM signal is also resolved at multiple decomposition levels. We derived 5 time-domain features by computing mean, variance, skewness, kurtosis, interquartile range on approximation and detail coefficients. These time-domain features have been explored in the earlier works on malware detection [16]. The feature vector comprises of aggregating these 5 time-domain features derived from wavelet coefficients at all decomposition levels.

Spectrogram: A spectrogram resolves EM side-channel trace into time, frequency bins and gives frequency response over time. We obtained power spectral density (PSD) in each frequency bin of spectrogram and used them as features.

2D-DWT on Spectrogram: Features are derived from DWT coefficients by applying 2D-DWT on spectrogram. A DWT on spectrogram reveals PSD variations over 2D space of time and frequency. We obtained mean of wavelet coefficient (approximation, horizontal, vertical, and diagonal) along both time and frequency axis and used them as features. The average value of approximation coefficient accumulates PSD variations. The average value of horizontal, vertical, and diagonal coefficient aggregate PSD variations along respective directions. The feature vector comprises of aggregating mean of all wavelet coefficients from all the decomposition levels.

Table 5.7: $F_1$ $scores$ for different features - Malware family classification

| **Features/Model** | **SVM** | **RF** |
|---|---|---|
| Windowed FFT | 0.74 | 0.47 |
| 1D-DWT (haar) | 0.18 | 0.37 |
| Spectrogram | 0.69 | 0.67 |
| 2D-DWT (haar) | 0.87 | 0.81 |
| 2D-DWT (sym5) | 0.88 | 0.81 |
| 2D-DWT (db5) | 0.74 | 0.77 |

The inference accuracy for different ML models and feature extraction techniques is summarized in Table 5.7. We obtained lowest inference scores on 1D-DWT with time-domain EM signal. We obtained highest $F_1$ $score$ for features selected from DWT on spectrogram instead of spectrogram alone or DWT applied on raw trace. We obtained 2.1x and 1.2x improvement in $F_1$ $scores$ with DWT on spectrogram in comparison to 1D-DWT on raw trace and spectrogram alone. Additionally, results show that spectral features have higher relevance to malware family classification task, as depicted with higher $F_1$ $scores$.

**Mutual Information Results**: We evaluate MI between features and label (y) for evaluating malware family classification. The higher relevance of a feature is reflected in higher MI score. Figure 5.12 shows variation of mutual information scores for first 625 selected features derived for two cases: spectrogram alone and DWT on spectrogram. We only selected features from approximation coefficients of DWT because they had highest MI scores compared to horizintal, vertical and diagonal coefficients. Therefore, Figure 5.12 compares approximation coefficients at level 2 and level 3 decomposition. We can conclude that information gain is enhanced by performing a second transformation (DWT) on spectrogram as observed with higher MI scores for most features. Therefore, mutual dependence of features derived from DWT on spectrogram is higher in comparison to spectrogram alone. Similarly, we also observe mutual dependence of features at level 3 decomposition is slightly higher compared to level 2. The MI analysis and $F_1$ $scores$ shows that

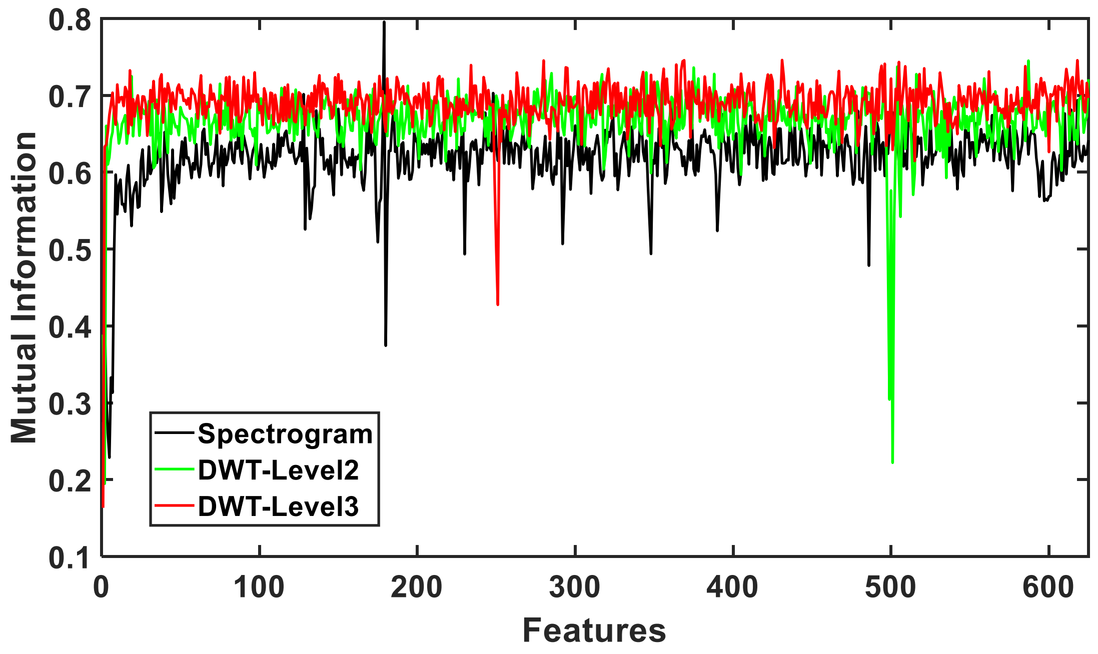

Figure 5.12: Mutual Information (MI) scores for features derived from spectrogram and DWT with haar basis

a second transform i.e., DWT on spectrogram improves the classification performance.

## 5.6 Unknown Application Detection and Machine Learning Model Updates

The proposed malware detection and family classification framework is trained on subset of benign and malware applications. The model would require updates under following scenarios. The software updates to the applications in training dataset or addition of new benign or malware applications to existing dataset. The ML model updates can be managed in two steps.First, the trained ML model should detect new/updated benign or malware application as “unknown”, features of which are not seen before. Once the application is detected as unknown, further forensic analysis scrutinizes to find existence of anomalies in its behavior. Second step is to re-train the model with features of new/updated applications.

Since, EM side-channel based detection is external, and not implemented on monitored device, feature extraction and re-training of new/updated EM side-channel signatures can be performed remotely. Subsequently, we discuss the methodology for detection of

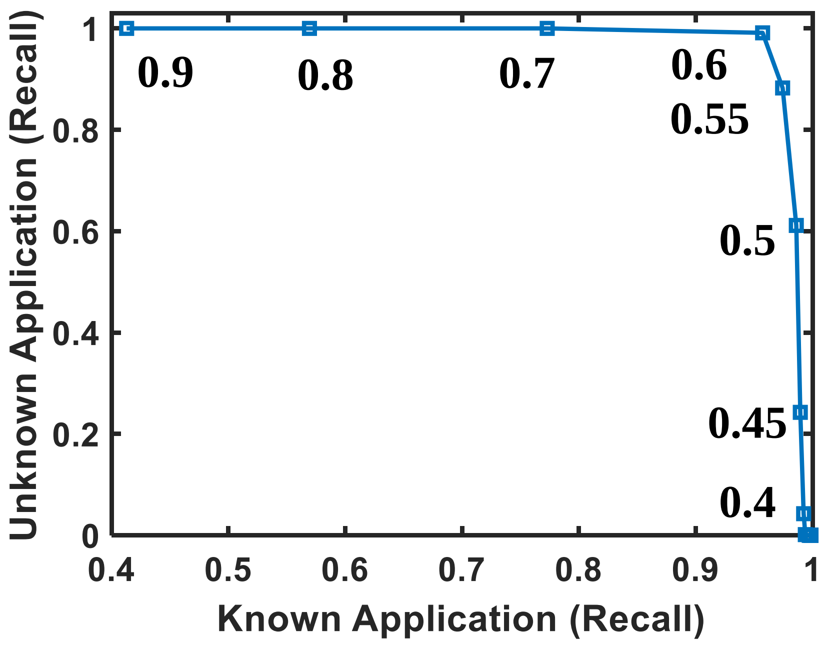

Figure 5.13: Recall Known vs Unknown Malware Family evaluated at different decision thresholds using RF model

unknown applications.

### 5.6.1 Detecting Unknown Using Prediction Probability

We detect an unknown application by comparing predicted probabilities derived from classifier model with a pre-defined threshold. If prediction probability of test sample is less than decision threshold, ML model has lower confidence is classifying the test sample as "known" sample and therefore, it is detected as unknown. The key observation of utilizing prediction probability derived from classification model for unknown application detection is due to disparity between prediction probabilities of an already seen "known" and "unknown" example. Generally, higher probability is expected from an already seen "known" example in comparison to out-of-distribution unknown example. The probability estimates from classifier's output is obtained using Platt scaling method.

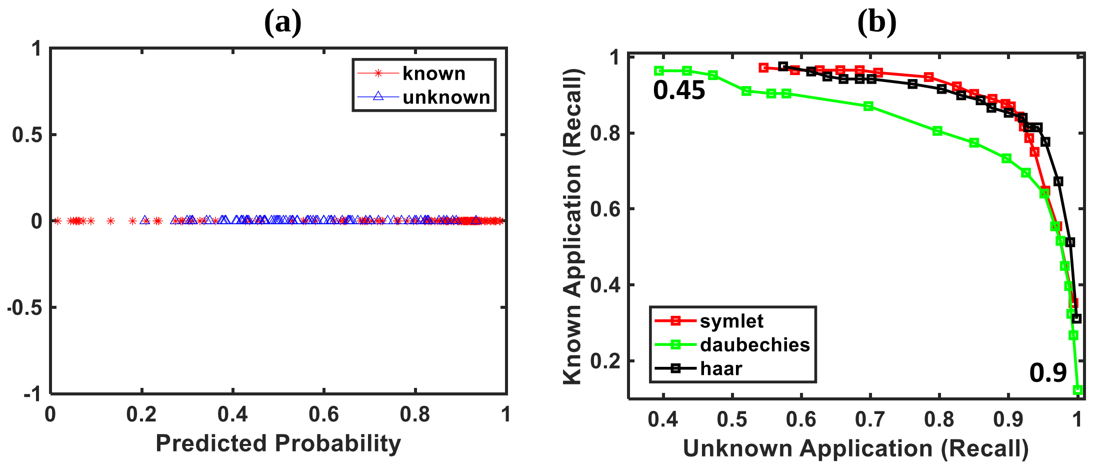

Figure 5.14: (a) Distribution of predicted probability estimates for unknown malware family "DroidDream" and known malware family. (b) Recall known vs unknown malware family evaluated at different decision thresholds with SVM model

## 5.6.2 Unknown Application Detection

We selected benchmarks from MiBench suite as "unknown" dataset for evaluation purposes [39]. We selected 10 benchmarks from MiBench suite at random and collected 40 EM side-channel traces for each benchmark for 10s duration. Therefore, unknown dataset comprises of 400 examples. The ML model is already trained on a set of benign and malware applications described in Table 5.1. The number of "known" test examples are 496. We applied similar post-processing, feature-extraction steps as described in subsection 5.4.3 for the "unknown" dataset and finally obtained predicted probabilities. The selection of decision threshold to detect unknown example would require re-evaluation of $F_1$ *score* for known and unknown examples. We obtained variation in recall scores of both known and unknown malware examples for different selected thresholds as shown in Figure 5.13. The optimal selection of detection threshold should give a higher recall for both known and unknown applications. A plot depicting the recall for known and unknown applications is shown in Figure 5.13 with RF model. We observe a decision threshold of (0.6) can detect unknown applications with 99% accuracy and known applications with 95%.

### 5.6.3 Unknown Malware Family Detection

We selected “DroidDream” malware family from drebin dataset for evaluating unknown malware family detection using trained malware classification model. We selected 16 applications from “DroidDream” malware family and 5 traces are collected for each application. EM side-channel traces are collected for 40s duration. In total, we have 137 examples of known malware family and 80 examples of unknown family. We applied similar DWT feature-extraction steps described in subsection 5.2.2 and finally obtained the prediction probabilities.

Figure 5.14(a) shows distribution of predicted probability of an unknown malware family, “DroidDream” and known malware family during inference. We observe probability estimates of known examples is concentrated more towards probability greater than 0.8. But probability of unknown examples is distributed between 0.2 and 0.8. We selected a decision threshold and re-evaluated recall scores of both known and unknown examples. Finally, we obtained a variation in recall scores of both known and unknown malware examples for different selected thresholds as shown in Figure 5.14(b). We expect higher recall for correctly classifying known examples at lower threshold (<0.5). This is because probability estimates for known examples is higher in comparison to unknown which gives fewer False Negatives (FN) at lower threshold. We observe exactly opposite trend at higher thresholds (>0.9). The three curves in Figure 5.14(b) depict the trend for wavelet basis selection. We expect higher AUC for recall curves for “symlet” and “haar” basis due to higher baseline classification accuracy obtained with these basis function as described in Table 5.5.

## 5.7 Comparison with EM Side-Channel based Detector

Refer to Table 5.8

Table 5.8: Comparison with existing EM side-channel based malware detection

| Reference | Platform | Detection | Algorithm | Features | Performance |
|---|---|---|---|---|---|
| [16] | Embedded Medical Device | Anomaly | 3NN, Perceptron, RF | Spectral, statistical (time) | 85% Known, 94% Unknown |
| [14] | A13-OLinuXino Board | Anomaly | STFT, KS test | Spectral peaks in STFT | Detect injections ($>$315K instructions) |
| [26] | PLC(Allen Bradley) | Control Flow Integrity | Stacked LSTM Network | EM Spectrum | 98.9% accuracy |
| [27] | FPGA ,TS-7250, A13-OLinuXino board | Anomaly (DDoS, ransomware, code-modification) | Template based Pattern Matching | EM Signal (time-domain) | Detect Intrusions (instructions $>$ 200K) |
| [15] | Altera FPGA Nios II, TS-7250, OlimexA13 | Anomaly | Hierarchical DBSCAN | Short Spectrum | FP ($<$ 0.1%) |
| [28] | D-Link (D-934L) ESCAM (E-G02) Xiaofang (X-1S) | Anomaly + Signature | Autoencoders, Seq2Seq, CNN | Mel-frequency spectrogram | TPR = 92.7%, FPR = 2.9% |
| [61] | Samsung Galaxy SIII, Samsung Galaxy S Duos, Asus Padfone Infinity | Signature | KNN, KNN+DTW, SVM, RF, NN | - | Recall = 95.65%, Precision= 89.19% |
| [35] | Arduino Uno, IntelAltera DE0-CV, TS-7250 Single-Board-Computer, A13-OLinuXino | Anomaly | K-S Test | Spectral Spikes in STFT | TP = 100%, FP = 0% |
| **Proposed** | **Intinsyc Open-Q 820 System on Module** | **Signature** | **SVM, RF** | **STFT, 2D-DWT on spectrogram** | $F_1 score$ **= 99% (detection),** $F_1 score$ **= 88% (classification)** |

## 5.8 Summary

This chapter demonstrated ML methodology to detection malware applications and classify detected malwares into characteristic malware families. Unique data-processing methods are demonstrated by coupling 1D-STFT and 2D-DWT on spectrogram of EM side-channel traces to extract fine-grained patterns of different malware families. The uniqueness in spectrograms of applications selected from different malware family is evaluated using Welsh's t-test. Welsh t-test analysis showed more similarity and less distinguishable DWT coefficients for applications selected from same malware family and vice-versa. The experimental analysis on Snapdragon 820 shows 0.99 $F_1\ score$ in detecting malwares and 0.88 $F_1\ score$ in classifying detected malware into 8 respective families using SVM, RF models. Updates to malware detection and analysis framework are demonstrated by first identifying "unknown" application or malware family and subsequently re-training model with new EM signatures. The results showed 0.99 recall for detecting "unknown" benchmark applications and 0.87 recall in detecting "unknown" malware family.

The next chapter performs analysis on the detected malware to identify actions such as shellcode injections by learning from RAPL traces of non-infected application and detecting anomalies in free energy variation of an RBM model.

# CHAPTER 6
# ANOMALY-BASED SHELLCODE EXPLOIT DETECTION USING RESTRICTED BOLTZMANN MACHINE

The persistent threat and evolving nature of malware require robust and early-to-detect engines to identify malware or potential intrusions in the early stages before infecting the end device. This requires detection of malware actions in the early stages by identifying deviations in the correct operation of the targeted executable. A typical attacker launches malware payload in a multi-stage process by first sending crafted shellcode to exploit a target's vulnerability. Second, shellcode residing in the victim's memory of the vulnerable program either reverts a shell session offering attackers remote access or download and execute task-specific malicious payload to achieve the attacker's objective.

A malware detection aimed at exploit-centered detection identifies malware infection as early as shellcode completes execution. Detecting the malware in exploit stage would substantially minimize the severe repercussions from the spread of malware infection. But, the challenge of detecting shellcodes is their short-lived execution. A. Tang et al. have shown stage-1 shellcode detection on Internet Explorer (IE) exploits by identifying anomalies in performance counters using oc-SVM [9]. Since, performance counters-based anomaly detectors require the selection of micro-architectural events, it is more intuitive to measure a single event comprehensive of HPC that simplifies the feature selection process. Moreover, anomaly patterns can be localized in this single event over time, as demonstrated in [28].

We utilized Running Average Power Limit (RAPL) interface accessible on Intel platforms to detect anomalous patterns resulting from shellcode executions. RAPL is an interface comprising of non-microarchitectural model specific registers (MSR) [62]. There are multiple regsiters with RAPL interface. We used MSR_PP0_ENERGY_STATUS register that reports actual energy consumption of a core power plane [62]. This specific register

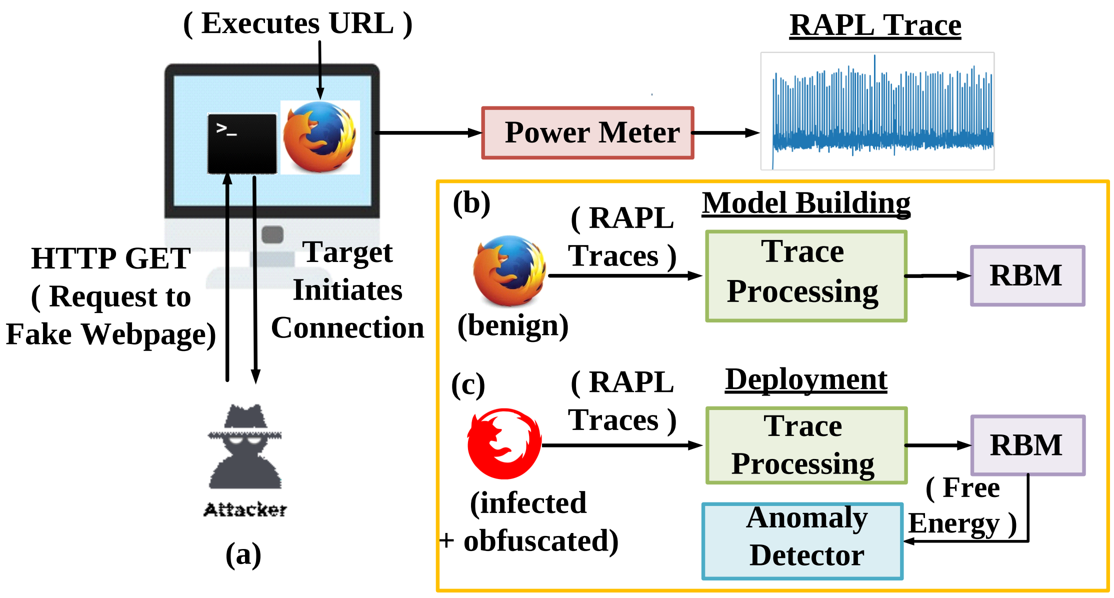

Figure 6.1: (a) A Remote Shellcode Injection Exploit Stage in malware infection process. (b) RBM learns patterns in RAPL traces under normal operation of victim application. (c) RBM detects anomalies in RAPL sequence by comparing free energy of infected browser against benign

represents the total amount of energy consumed [62]. The register uses on-chip measurements of activities coupled with software based power models to provide a real-time estimate of energy. We utilize MSR_PP0_ENERGY_STATUS register within RAPL interface because of easy accessibility in software.

This chapter presents a detection framework to identify anomalous patterns in MSR_PP0_ENERGY_STATUS register values that corresponds to shellcode injections using a Restricted Boltzmann machine (RBM) model as shown in Figure 6.1. An RBM is a generative stochastic artificial neural network described by an energy-based model. An RBM model learns the variations in MSR_PP0_ENERGY_STATUS register value corresponding to a targeted vulnerable application by minimizing the free energy between input and reconstructed output as shown in model building phase in Figure 6.1(b). We observed that variations in MSR_PP0_ENERGY_STATUS register values are modulated during vulnerability exploit phase. These abrupt variations in MSR_PP0_ENERGY_STATUS register are reflected as increase in free energy of RBM model, which forms the basis for detection.

The experimental validation is performed on Lenovo Ideapad 3 comprising of Intel i5, $10^{th}$ Generation IceLake processor, Quad-Core hosting Ubuntu 18.04 OS. The exploits are generated using Metasploit penetration testing tool. To evaluate the framework, we simulated real exploits targeting *CVE-2014-8636* and *CVE-2015-0802* vulnerabilities of the Firefox browser and selected reverse TCP shell as malicious payload. The Firefox binary is obfuscated using *shakata ga nai* encoder to generate a polymorphic malware payload [63].

## 6.1 Characterization of MSR_PP0_ENERGY_STATUS Register Trace

### 6.1.1 RAPL Interface

The RAPL interface is a feature introduced in the Sandy Bridge Architecture on Intel Processors. The RAPL interface has multiple power domains, package, core, uncore and DRAM power planes. For each of these components, RAPL interface allows setting power limits of core, uncore, DRAM components and also provides energy consumption information. The RAPL energy counter MSR_PP0_ENERGY_STATUS for core power plane (PP0) can be accessed through model-specific registers (MSRs). These counters are 32-bit registers that indicates the total amount of energy consumed since the last time this register was cleared. The counters are updated approximately once every 1ms. The MSR's are accessible on linux platforms using *msr* driver in $/dev/cpu/ < coreid > /msr/$ directory.

### 6.1.2 MSR_PP0_ENERGY_STATUS Trace Acquisition

The experimental validation is performed on Intel platform. There are numerous control registers on x86 platform, known as Model specific registers (MSR) which are useful for program debugging, and performance monitoring. The energy consumption information of a core package are updated in model-specific register (MSR), MSR_PP0_ENERGY_STATUS on the selected platform [62]. This register comprises of 32-bit value, representing the total energy consumed by all core devices and MSR is updated every 1ms. To profile a workload, the MSR is polled by a profiler concurrently while profiled workload executes on the core. The MSR read interval is tuned relative of register update interval, which is 1ms. The register values represents a count and can be converted to actual power values using simple transformation. But, we didn't use actual power values for the analysis. The profiler and workload are executed on seperate core to minimize effect of noise resulting from MSR read on the profiled workload. Finally, the timestamp and MSR count are logged during workload's execution and collected as trace files.

### 6.1.3 Code Injections in Benchmarks

Before describing the anomaly detection framework, it is important to examine if there are reasonable deviations in MSR_PP0_ENERGY_STATUS values resulting from a code injection within a profiled program. To evaluate this task, we injected two arbitrary functions within a CPU benchmark as described in algorithm below. In the first case, we injected a square root function with an execution time of 100ms between two loops, each of which counts floating-point and integer operations. In the second case, we injected a 20ms delay function within the loop that counts number of floating-point operations.

**Algorithm 1** Code Injection Between Loops

```
1: for $i \leftarrow 1$ to $5$ do
2:     FLOPSBenchmark()
3: end for
4: sqrt()
5: for $i \leftarrow 1$ to $5$ do
6:     IOPSBenchmark()
7: end for
```

In both cases, we first profiled the benchmark without code injection by executing the benchmark on a single core and polling MSR_PP0_ENERGY_STATUS values by executing profiler on another core for a 1sec duration. Then, we collected MSR_PP0_ENERGY_STATUS values for the benchmark with code injections. The acquired MSR_PP0_ENERGY_STATUS traces are processed to remove systemic noise and traces are aligned. The plot

**Algorithm 2** Code Injection Within Loop

```
   for $i \leftarrow 1$ to $5$ do
2:     FLOPSBenchmark()
       delay()
4: end for
   for $i \leftarrow 1$ to $5$ do
6:     IOPSBenchmark()
   end for
```

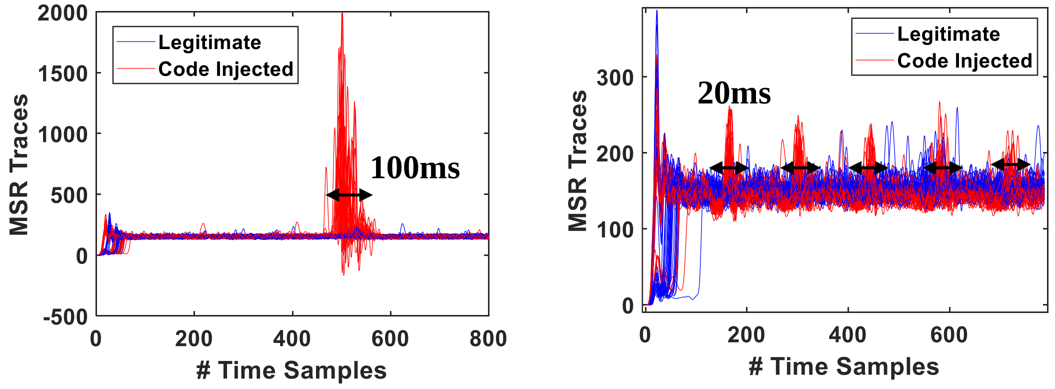

Figure 6.2: MSR_PP0_ENERGY_STATUS Traces of Code Injections in Benchmark application (a) Between Loops (b) Within Loop

of the MSR_PP0_ENERGY_STATUS traces for both the cases is shown in Figure 6.2. We observe the amplitude of MSR_PP0_ENERGY_STATUS samples increases in the region of code injection for both cases. The duration of code-injection is also reflected in MSR_-PP0_ENERGY_STATUS traces. The execution of 100ms square root function increases the MSR_PP0_ENERGY_STATUS samples as shown in Figure 6.2 (a). Similary, the 20ms delay function reduces the MSR_PP0_ENERGY_STATUS samples as shown in Figure 6.2 (b). Also, the peak corresponding to delay function repeats showing the loop frequency.

## 6.2 Anomaly Detection Framework

There has multiple components of functionality to attain anomaly detection task, namely trace acquisition, post-processing traces, RBM inference and anomaly detection. The data acquisition block monitors and periodically samples MSR_PP0_ENERGY_STATUS reg-

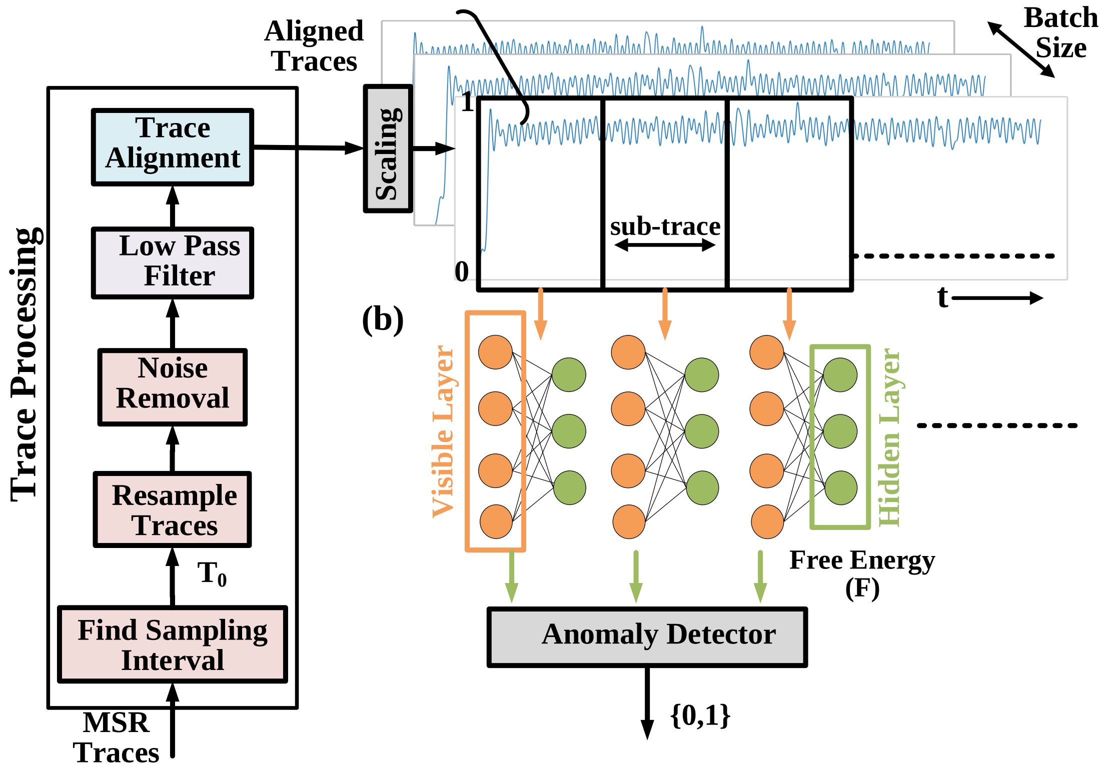

Figure 6.3: (a) Processing acquired MSR_PP0_ENERGY_STATUS traces to remove systemic noise, outlier removal, filtering, and alignment. (b) Building RBM model by training each segmented sub-trace individually after scaling

ister. The acquired traces are post-processed to remove systemic noise, period noise by filtering and alignment methods. The post-processed trace is evaluated on trained RBM model to obtain free energy of the trace. The sequence of MSR_PP0_ENERGY_STATUS trace that exceeds free energy beyond a specified threshold is detected as anomalous sequence. The sub-section below elaborates specifics of individual component.

### 6.2.1 MSR_PP0_ENERGY_STATUS Trace Collection

The acquisition of MSR_PP0_ENERGY_STATUS trace is described in subsection 6.1.2. To train the DL model, we collected multiple iterations MSR_PP0_ENERGY_STATUS trace for the profiled workload.

### 6.2.2 Post-Processing Traces

A MSR_PP0_ENERGY_STATUS trace represents variation of counter representing energy consumption information against time. The raw MSR_PP0_ENERGY_STATUS trace has multiple noise sources. The multiple noise sources are systemic noise due to uncertain background core activity, periodic noise, non-uniform sampling interval, missing or repeated counter values, etc. Therefore, it is post-processed to remove or minimize these noise sources before analyzing on an RBM network. The post-processing steps are described below.

#### *Resampling*

The acquired MSR_PP0_ENERGY_STATUS traces have non-uniform intervals between consecutive counter sample. A uniform sampling interval is required across all MSR_PP0_-ENERGY_STATUS traces because analysis is performed in time-domain, and to estimate anomaly detection time. A uniform sampling interval is obtained by finding highest occurrence of non-uniform interval first across entire trace, and then across all power traces in the dataset. After obtaining uniform sampling interval, we generate uniform timing axis based on number of samples. Finally, the samples of power traces are interpolated on uniform time axis.

#### *Noise Removal*

We tacked to remove numerous noise sources using different schemes. The systemic noise is introduced by abrupt changes in total workload of a core, which overshoots MSR_PP0_-ENERGY_STATUS register counter. These abrupt fluctuations are noticeable randomly across a MSR_PP0_ENERGY_STATUS trace. We remove the traces affected by systemic noise as these are outliers in the training dataset. The outlier trace is identified if the amplitude of the counter at any time point exceeds the average trace by 3 times standard deviation. The average trace is mean of all collected traces of a profiled program. Since, MSR resets after an overflow, numerous missing counter values are encountered in MSR_PP0_-ENERGY_STATUS trace. These 0 values are linearly interpolated from nearby samples and updated. On the other hand, the repeated counter values are avoided by selecting the MSR read interval, i.e. 1ms.

*Low Pass Filtering*

The MSR_PP0_ENERGY_STATUS trace also comprises of periodic noise introduced by power supply. The period of the periodic noise can be derived by computing auto-correlation of a MSR_PP0_ENERGY_STATUS trace for different lags. An auto-correlation plot shows higher correlation for lag value that matches closely with noise period and lower elsewhere. We derived the noise period by identifying average time interval between peaks in the auto-correlation plot. But, we observed that removal of periodic frequency does not completely eliminate periodic noise because of existence of harmonics of base frequency. Therefore, we performed low-pass filtering with cut-off frequency at 50Hz to eliminate high-frequency variations in MSR_PP0_ENERGY_STATUS trace.

*Cross-Correlation Based Trace Alignment*

The system-level threads context-switches with profiled program of a core introduces steady mis-alignments across multiple MSR_PP0_ENERGY_STATUS traces of a profiled program. In addition to this, activity of remaining core is also varying across multiple runs of the same program, which also affects MSR_PP0_ENERGY_STATUS trace alignment. The MSR_PP0_ENERGY_STATUS traces should be aligned because analysis is performed in time-domain. We proposed to diminish the effect of misalignment by cross-correlating MSR_PP0_ENERGY_STATUS trace with a template trace. The template trace is randomly selected from among the noise-free filtered traces. The steps followed by cross-correlation alignment is described here. First, we segment the template MSR_PP0_ENERGY_STATUS trace to select a region to be aligned with. Second, we obtained the segmented trace in the same time-region that is to be aligned. Third, a pearson correlation is evaluated between segmented trace to be aligned and template trace for different offsets. Finally, the offset corresponding to maximum correlation is applied to actual MSR_PP0_ENERGY_STATUS trace (non-segmented) to attain an aligned trace.

The sequence or post-processing steps on acquired MSR_PP0_ENERGY_STATUS traces are shown in Figure 6.3(a)

### 6.2.3 Model Building

RBM is a generative stochastic artificial neural network described by an energy-based model demonstrated below. RBM is a two layered neural network with no intra-layer connections. For a two layered RBM, energy function as shown in equation (1). Here, v is visible layer state, h is hidden layer state, b is visible layer bias, c is hidden layer bias and W is the weight matrix.

$$E(v,h) = -c^T v - b^T h - v^T W h \tag{6.1}$$

The joint probability over visible and hidden units can be expressed using energy function as shown in equation (2). Here, z is a normalization constant known as partition function, which is intractable. Z is given by summing over all possible pairs of visible and hidden vectors.

$$P(v,h) = \frac{e^{-}E(v,h)}{Z} \tag{6.2}$$

Since there are no within layer connections, probability of hidden variable is independent given state of visible layer variables and vice-versa. The conditional probability of hidden variable given visible variable is interpreted as stochastic neuron with sigmoid function as shown in equation (3). Similarly, conditional probability of visible layer given hidden state is described in equation (4). Free Energy of RBM is denoted by -ln(Z). In reduced form, free energy is described in equation (5).

$$P(h = 1|v) = \sigma(W^T v + b) \tag{6.3}$$

$$P(v = 1|h) = \sigma(W^T h + c) \tag{6.4}$$

$$F(v) = -c^T v - \sum log(1 + e^{Wv+b}) \tag{6.5}$$

The loss function or log likelihood over marginal distribution of v can be expressed as difference between free energy of model and training data point v (shown in equation (6)). The RBM is trained to maximize the log likelihood to estimate the parameters.

$$L = -Ln(P(v)) = F^c(v) - F \tag{6.6}$$

Here, $F^c(v)$ can be easily computed as h can be summed out analytically due to conditional independence between layers. But, computing is intractable for F. The approximations to log likelihood is achieved using contrastive divergence (CD) learning. In a single step of CD for binary RBM's, first visible layer is initialized with training examples followed by forward pass to attain activations for hidden layer. The hidden state is obtained by sampling activation value from Bernoulli distribution. In backward pass, visible layer activations are generated based on hidden state and visible state is obtained after sampling from distribution.

To localize the anomaly in the MSR_PP0_ENERGY_STATUS trace, we first segment the entire trace into multiple sub-traces and train each sub-trace on an RBM separately as shown in Fig Figure 6.3 (b). The MSR_PP0_ENERGY_STATUS samples are real-valued, which has to be scaled between [0,1]. Also, hidden and visible states are changed to expectations of activation values instead of binary sampling used in Bernoulli RBM. The rest of the steps in training procedures remains similar to binary RBMs.

### 6.2.4 Rule-based Anomaly Detection

The outcome of the RBM model is free energy of each sub-trace in the MSR_PP0_ENERGY_STATUS trace as shown in Figure 6.3 (b). The minor variations across multiple MSR_PP0_ENERGY_STATUS traces of the same profiled program also shows minor variation in free energy values. We obtained the mean and variance (spread in values of free energy) of free energy across multiple MSR_PP0_ENERGY_STATUS traces of profiled program. During deployment phase, free energy of MSR_PP0_ENERGY_STATUS trace are compared against free energy (validation). Any significant deviation in MSR_PP0_ENERGY_STATUS register count value of profiled program during deployment is reflected as increase in free energy for that sequence. The free energy of sub-trace that deviates from the mean value by standard deviation is detected as an anomaly as shown in equation (6.7).

$$Anomaly = |\mu_{validation} - F_{TestSample}| > \sigma_{validation} \tag{6.7}$$

## 6.3 Detecting Shellcode Injections in Browser Exploits

We experimentally demonstrate the proposed methodology in detecting stage-1 shellcode attacks on vulnerable browser applications. From among the vast number of vulnerabilities at OS and application level, we selected browser applications as the targets because according to Common Vulnerability and Exposures (CVE) databases, browser applications such as Chrome, Firefox and Internet Explorer report numerous distinct vulnerabilities. We explored reverse tcp shell as stage-1 payloads, where the target machine initiates a connection to a network host (attacker). An attacker can launch advanced attackers upon access to victim's machine in stage 1.

### 6.3.1 Firefox Browser Vulnerabilities

Although there exist multiple vulnerabilities to hijack the Firefox browser during execution, we target *CVE-2014-8636* and *CVE-2015-0802* listed in the CVE database to demonstrate proof of concept. *CVE-2014-8636* is a javascript injection in privileged URI scheme chrome://windows in Firefox. The javascript code is configured to call Firefox XPCOM API to spawn reverse shell. This vulnerability affects Firefox 31-34 version. *CVE-2015-0802* is javascript injection exploiting a privilege escalation bug in resources:// URI, which affects Firefox 35-36 version.

*Firefox Proxy Prototype RCE - CVE-2014-8636*

The details of the firefox exploit using proxy objects as means to execute privileged code is dissected by Rapid 7 [64]. The exploit is implemented by leveraging improper privileges exercised using proxy objects and `messageManager` interface in chrome:// window to spawn reverse shell. Proxy objects create a proxy for another object, which can intercept and redefine fundamental operations for that object. The proxy objects allow the execution of chrome:// URI, a privileged zone permitted to run from unprivileged code. Firefox's chrome:// URI has full privileges of the browser. Any javascript executing from chrome:// URI can give the attacker a fully-working remote shell on user's machine. To inject remote shell, `messageManager`, a privileged Firefox API is accessible inside inside chrome:// window is utilized to send message between processes. The steps outlined in the exploit is accessible in firefox_proxy_prototype exploit module on Metasploit [65]. The various steps in executing this exploit is summarized below:

- Creating a Proxy Object

```
var pro = Object.getPrototypeOf(document)
Object.setPrototypeOf(document,Proxy.create(props))
```

- Click on HTML page

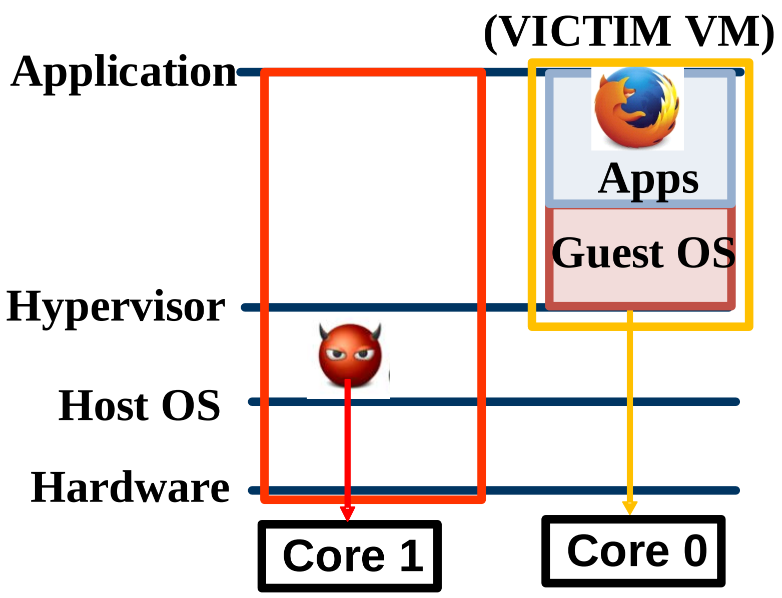

Figure 6.4: Threat model describes ring-0 privileges for attacker on host machine. Attacker initiates a reverse tcp shell listener on Core 1. The victim executes vulnerable Firefox browser application inside virtual machine dedicated Core0

```
if (!window.top.x && n=='nodeType')
```

- Open Privileged Chrome Window

```
window.top.x=window.open("chrome://browser/content/
browser.xul", "x");
```

- Run reverse TCP shell payload

```
opts = { key => run_payload }
x.messageManager.loadFrameScript('data:,'+key, false)
```

Joe Vennix describes "`run_payload` method will return a configured piece of Javascript code that will call Firefox's XPCOM APIs to spawn a reverse shell" [64].

### 6.3.2 Threat Model

We assume that malware binary comprising of reverse tcp shell payload for remote access enters the victim's system through certain social engineering attack. The reverse tcp shell can bypass the firewall because of connections initiated by victim. Meanwhile, the attacker creates a remote listener session corresponding to the payload. As the victim downloads and executes this payload, an outgoing tcp session is initiated, reverting the remote shell to the attacker. We emulated this attack scenario by isolating the vulnerable application in a virtualized environment using VM with dedicated resources and a secure boundary as shown in Figure 6.4. The host (attacker) sets up listener reverse TCP session via Metasploit on Core 1 as shown in Figure 6.4. The victim has a vulnerable firefox browser application running inside VM on a dedicated core. Both attacker and victim have similar privileges.

### 6.3.3 System Configurations

The detector is tested on a personal computing platform Lenovo Ideapad 3, comprising of Intel i5, $10^{th}$ Generation IceLake processor. The data acquisition phase requires polling MSR_PP0_ENERGY_STATUS MSR at 1ms update interval. We minimized the effect of external noise factors by tuning system configurations such as disabling hyperthreading, disabling Intel P-state, setting constant OPP. The profiler is scheduled on Core 1 and Core 0 is dedicated to virtual machine (VM) processing. We tested the framework on Windows 7 Ultimate Edition and Linux 16.04. The Firefox exploits are executed on target VM using Metasploit Framework ver. 6.0.4 running on host OS. The user interactions (UI) with browser are automated using selenium. The host machine and VM have connected to same network.

### 6.3.4 Collecting Clean Browser State

We collected MSR_PP0_ENERGY_STATUS traces during the normal operation of the Firefox browser. We selected ten commonly visited web pages as mentioned on Alexa [1].

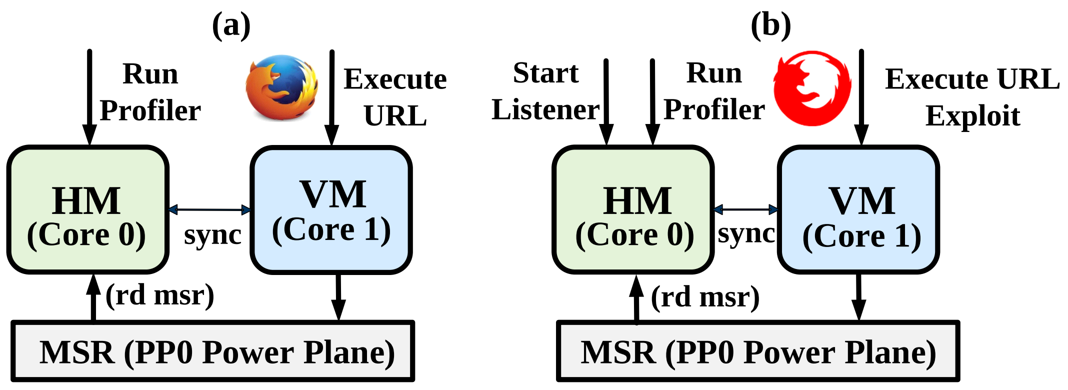

Figure 6.5: (a) Collecting MSR_PP0_ENERGY_STATUS Traces of Clean Browser State executing legitimate URLs. (b) Collecting MSR_PP0_ENERGY_STATUS Traces of Infected Browser State execution

Table 6.1: RBM Training and Inference Configurations For Detecting Firefox Exploits

| **Configuration** | **Training** | **Testing** |
|---|---|---|
| Total traces | 800 | 400 |
| RBM | Visible Layer Size (v) = 50<br>Hidden Layer Size (h) = 20<br>Epoch = 100<br>Sub-Trace Length = 50<br>Learning Rate = 0.01 | |
| | Batch Size = 50 | Batch Size = 1 |
| Filtering | butterworth, 50Hz cut-off frequency | |

These include social networking, entertainment, and content-based websites. Using the selenium browser automation tool, we simulated user interactions such as clicks, scroll, and text on the browser. We profiled only a few out of all possible states of browser operation because it is infeasible to model the entirety. The model would require updates as new phases of browser states are available. Therefore, RBM's training in the proposed framework is semi-supervised.

The first step in collecting MSR_PP0_ENERGY_STATUS traces of a clean browser state is setting the configurations. The profiler runs on a dedicated core. The Firefox browser initiates a session and executes a URL in Linux 16.04 VM dedicated to an isolated core. The profiler reads the MSR while simultaneously browser runs inside VM. The file-sharing mechanism synchronizes HM and VM. Figure 6.5 (a) summarizes the configuration settings for collecting MSR_PP0_ENERGY_STATUS traces of clean browser state.

To train the RBM model, we collected 800 MSR_PP0_ENERGY_STATUS traces each for a duration of 5s. These MSR_PP0_ENERGY_STATUS traces correspond to 10 selected webpages navigated on opening a browser session. We introduced variability in trace collection by randomly selecting from these 10 websites and executing URL. We obtained the MSR_PP0_ENERGY_STATUS traces of different webpages and observed a unique pattern, where phase of the browser during initial loading is similar but it diverges on loading a specific webpage. The trace collection encounters numerous noise sources resulting from the indeterminate background state of CPU core, context-switches, and variable network latency. Therefore, these traces of clean browser state follows the detection framework's post-processing block. We followed the steps outlined in the post-processing block of resampling, noise removal, filtering and alignment to obtain noise-free aligned MSR_PP0_-ENERGY_STATUS traces. Since RBM model trains on real-valued data between 0 and 1, the amplitude of aligned power traces is scaled. To train the RBM model, we first segment the entire trace into multiple sub-traces. There are multiple RBMs to train each batch of sub-traces. The number of RBMs equals the total number of sub-traces. The configuration parameters of the RBM model are tuned to minimize the loss function. We experimented with various configuration parameters and selected the ones which minimized the training loss. Table 6.1 summarizes the configurations of multiple parameters during training.

### 6.3.5 Detecting Infected Browser State

To evaluate the proposed framework in detecting malicious browser states, we have to generate exploits targeting vulnerabilities in Firefox browser. We used the Metasploit tool to create exploits. Metasploit has multiple in-build exploit and payload modules used for easy execution of exploit code. We setup the required configuration parameters corresponding

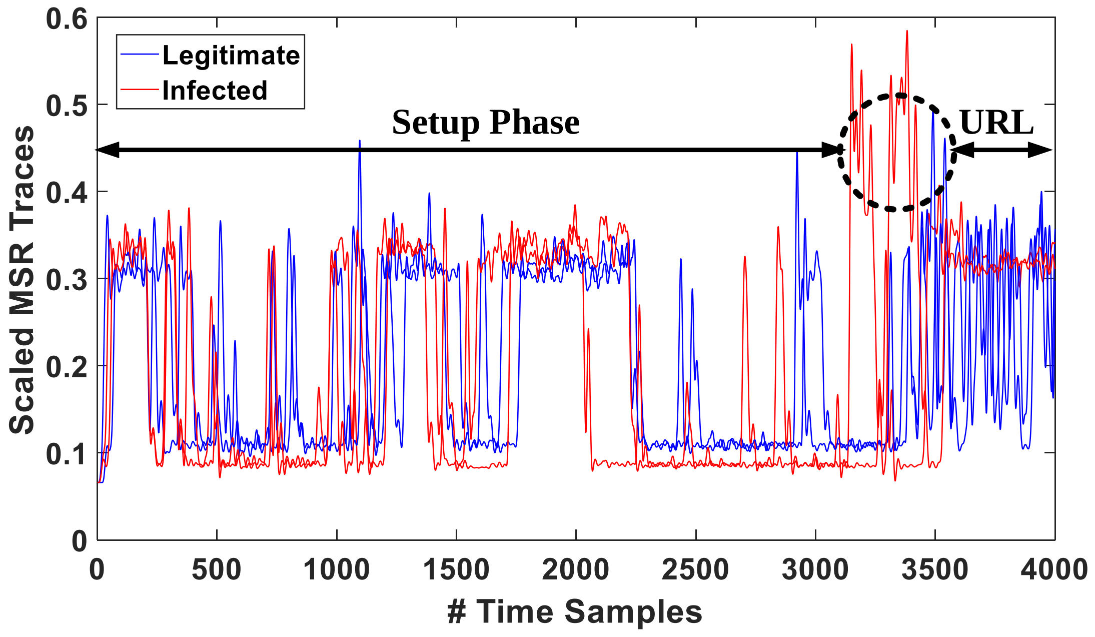

Figure 6.6: Scaled MSR_PP0_ENERGY_STATUS Traces of Infected Browser State changes during shellcode exploit execution

to the exploit in Metaploit `msfconsole` tool such as IP address of target machine to execute a remote shell payload. Once, exploit executes, a listener event starts on the attacker's machine (HM). Figure 6.5 (b) summarizes the configuration required to collect MSR_PP0_-ENERGY_STATUS traces of the infected browser state. Here, the profiler and Metaploit module executes on a dedicated core. The Firefox browser initiates a session and runs a malicious URL inside Linux 16.04 VM dedicated to an isolated core. Upon navigating the malicious URL, exploit code runs and reverse shell session reverts to HM. Once the TCP connection is established, file system of VM is accessible inside HM.

The objective is to detect the generation of this reverse TCP session. Therefore, we simultaneously collect the MSR_PP0_ENERGY_STATUS traces while the exploit runs inside VM. We run the exploit code multiple times in different background state of the core and collected 200 MSR_PP0_ENERGY_STATUS traces each for 5s. In addition, we collected 100 traces of clean browser state for evaluating the trained model. We followed similar post-processing steps outlined in the detection framework block to obtain the aligned trace.

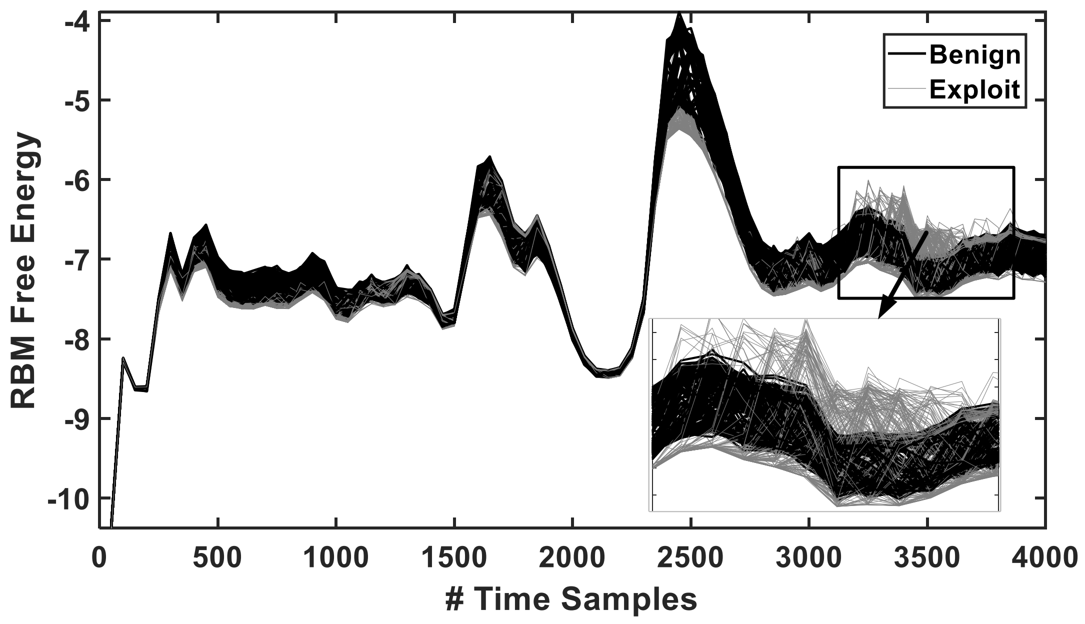

Figure 6.7: RBM's Free Energy Variation over time during normal browser operation is lower compared to shellcode exploit execution in region of exploit

Although we applied trace alignment step, it's not required during testing. Next, we segment the MSR_PP0_ENERGY_STATUS traces using same sub-trace length used in training to obtain multiple sub-traces. Further, all the sub-traces are evaluated on multiple trained RBMs to get free energy scores. We collected free energies from individual RBMs and fed to the anomaly detection block. The free energy of the sub-traces that satisfies the rule for detecting anomaly is labeled 1, while others are 0. We compare the predicted labels against the true labels for sub-traces in the exploit region to derive $F_1 score$ and FPR scores.

*Experimental Results*

First, we plotted the scaled power traces both for the infected and clean browser state as shown in Figure 6.6. We observe that the amplitude of scaled MSR_PP0_ENERGY_STATUS traces for infected browser state increases in the exploit region and deviates from clean browser state. We also concur that MSR_PP0_ENERGY_STATUS trace during loading of browse application is similar initially and only change when webpage or malicious URL runs. Second, we observe the trend in free energy over time both for MSR_PP0_ENERGY_-STATUS traces of clean browser state and infected browser state as shown in Figure 6.7. This trend is obtained by concatenating free energy corresponding to each sub-trace. We observe that free energy in exploit region for infected browser state increases relative to clean browser state as shown in Figure 6.7 (zoomed). Although, we expect free energies in non-exploit region to be similar because of identical browser behavior. But, we observe numerous non-exploit regions with non-overlapping free energy. This is attributed to irregularities in MSR_PP0_ENERGY_STATUS traces and inefficient trace alignment. Third, we observe the outcome of the anomaly detection block for a selected infected browser state as shown in Figure 6.8. The plot shows the absolute difference between mean free energy of clean browse state and infected browser state denoted as DFM. It also depicts variance between free energy of clean browser state. The region where DFM exceeds variance is detected as anomaly (labeled 1). We observe anomaly detected in the exploit region. But, anomalies were flagged in the non-exploit region as well. These irregularities corresponding to MSR_PP0_ENERGY_STATUS variation resulting from variable network latency and inefficient trace alignment.

We evaluate the efficacy of the proposed anomaly detector using $F_1 score$ and FPR. The $F_1 score$ and FPR for correctly detecting shellcode exploits in the exploit region is 0.875 and 0.06.

## 6.4 Obfuscated Payload Detection

In practical malware attacks, the original shellcode or multi-stager payloads are concealed to evade detection by intrusion detection system (IDS) on victim's machine. Among various defense bypassing mechanisms, we discuss obfuscation techniques for instance, string manipulation, padding, polymorphism, encryption, compression, and metamorphism.

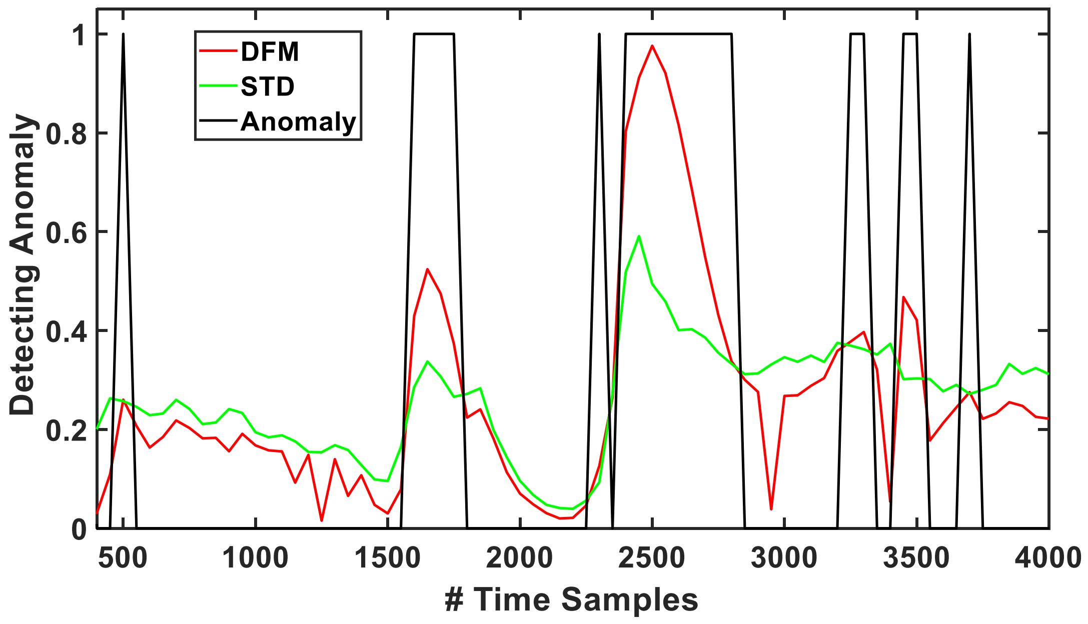

Figure 6.8: Anomaly detected where absolute difference between free energy of infected and clean browser state is greater than standard deviation

### 6.4.1 Obfuscation Techniques

String manipulation and interleaving shellcode with NOP instructions is shown successful in bypassing rudimentary static analysis. A more refined approach is to compress shellcode along with executable binary. The inherent encryption increases effectiveness of evading detection. In most cases, a static analyzer with most-updated signature repository can detect simple obfuscations. Alternatively, polymorphism is a technique of morphing static shellcode to evade signature analyzers. A polymorphic engine comprises of transfer function which decrypts the morphed binary to executable code. The executable is loaded and run to achieve malicious actions. Finally, a new key is derived, and reverse transfer function is applied on original executable to get back new morphed code. Polymorphic malware can bypass static analysis because of changing nature of static binary. A Metamorphic malware obfuscates binary by inserting benign instructions, substituting opcodes, swapping registers, and changing control flow. A metamorphic malware involves five steps. Disas-

Table 6.2: RBM Training and Inference Configurations For Detecting Obfuscated Payloads

| **Configuration** | **Training** | **Testing** |
|---|---|---|
| Total traces | 1600 | 900 |
| RBM | Visible Layer Size (v) = 20<br>Hidden Layer Size (h) = 10<br>Epoch = 100<br>Sub-Trace Length = 50<br>Learning Rate = 0.01 | |
| | Batch Size = 50 | Batch Size = 1 |
| Filtering | butterworth, 50Hz cut-off frequency | |

sembling opcodes, compressing disassembled code, permutation, expanding instructions, and changing control flow jumps. A metamorphic malware can evade signature analyzers, but the increased size and CPU workload makes them prone to side-channel based detection.

### 6.4.2 Generating Obfuscated Payloads

We obfuscated malware payload using a polymorphic XOR additive feedback encoder, shakata ga nai (SGN) included in Metasploit framework. The multiple steps in SGN algorithm is summarized here [63]. First, a key is initialized. Then, location relative to EIP is obtained. This step points to the location of starting payload. SGN uses `fnstenv` instruction to get the location. In the next step, a loop is iterated to decode the instructions of the payload. A register is zeroed to be used as a counter to iterate the loop. SGN decodes instruction addresses by adding location relative to EIP and XORing with key obtained in first step. Then relative location to EIP is modified and key is also modified. Finally, decoded instructions are move towards execution. Generally, these steps are repeated for multiple iterations. We assembled reverse TCP shell payload using SGN encoder for 3 iterations and packaged binary in Firefox executable as separate thread. Generally, packaged malware has higher chance of evading during anti-virus scanning compared to vanilla binary payload. We uploaded the packaged malware on VirusTotal website to check existence of signatures in current anti-malware engines. We evaluated the detection of SGN obfuscated malware executable.

To train the RBM model, we collected 1600 MSR_PP0_ENERGY_STATUS traces of Firefox browser during normal operation. We collected MSR_PP0_ENERGY_STATUS traces for only 1 sec duration for initializing browser. We collected MSR_PP0_ENERGY_-STATUS traces for only 1 sec because malware operation of spawning a reverse TCP shell is confined to within this region. First, the configurations of traces acquisition is set as shown in Figure 6.5. In this case, Firefox executes on Windows 7 Ultimate Edition VM dedicated to Core 0 and simultaneously profiler executes on host machine. The number of energy samples to be collected are changed for 1s duration. Further, we followed the steps outlined in the post-processing block to obtain aligned traces. the amplitude of aligned MSR_PP0_ENERGY_STATUS traces is scaled between 0 and 1. To train the RBM model, we first segment the entire MSR_PP0_ENERGY_STATUS trace into multiple sub-traces. Each batch of the sub-trace is fed to individual RBM. We experimented with different sub-trace lengths and observed best detection results when sub-trace length equals exploit runtime (20ms here). The other RBM training configuration parameters such as learning rate and epochs are tuned to minimize and stabilize the loss function. Table 6.2 summarizes the configurations of multiple parameters during training.

### 6.4.3 Detecting Obfuscated Payload

The objective is to detect the obfuscated reverse TCP shell payload inside the Firefox browser binary. We simultaneously collected the MSR_PP0_ENERGY_STATUS traces while the obfuscated Firefox binary runs inside VM. The obfuscated Firefox binary runs multiple times under different background state of the core and we collected 500 MSR_-PP0_ENERGY_STATUS traces each for 1s. We also collected 200 traces of Firefox browser under normal operation for validating the trained model. We followed similar post-processing

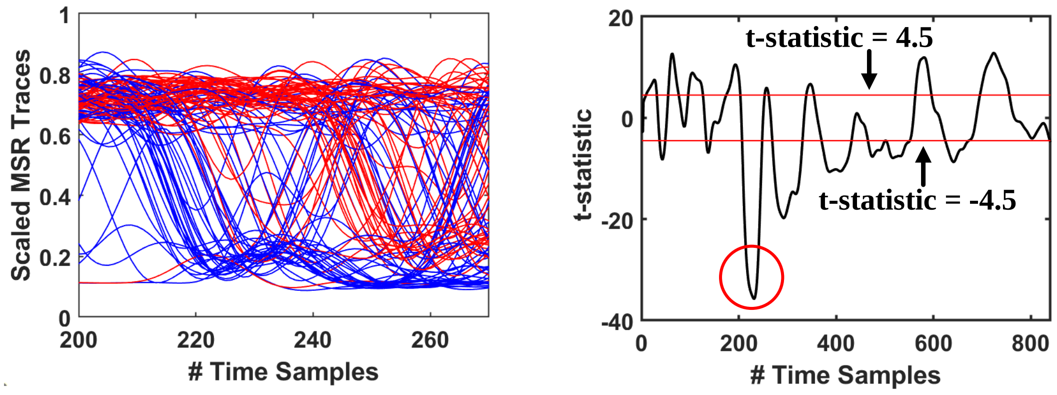

Figure 6.9: (a) Scaled MSR Traces of infected and clean Firefox browser (b) t-statistic variation vs time samples identifies region of differences in MSR traces of two Firefox version

steps outlined in the detection framework block to obtain the aligned trace. The scaled aligned traces are evaluated on the trained model to obtain free energy scores. The free energy scores are fed on anomaly detector. The free energy of the sub-traces that satisfies the rule for detecting anomaly is labeled 1, while others are 0. Finally, we evaluate detection performance using $F_1 score$ and FPR.

*Experimental Results*

First, we plotted the scaled MSR_PP0_ENERGY_STATUS traces both for obfuscated Firefox binary and clean Firefox binary as shown in Figure 6.9 (a). Since we couldn't highlight the differences visually, we plotted the region of code obfuscation execution. We deduce that MSR_PP0_ENERGY_STATUS profile may not necessarily increase amplitude as shown earlier in Figure 6.6. Here, MSR_PP0_ENERGY_STATUS profile of obfuscated Firefox is not modified, instead it extends for 50ms further to execute hidden payload as shown in Figure 6.9(a). Second, we applied welsh's t t-test between MSR_PP0_ENERGY_-STATUS traces of clean and obfuscated Firefox to validate that MSR_PP0_ENERGY_STA-TUS profile is altered by payload execution.. We observed highest leakage is visible in the t-test plot in the region of payload execution as shown in Figure 6.9 (b). The region of in-

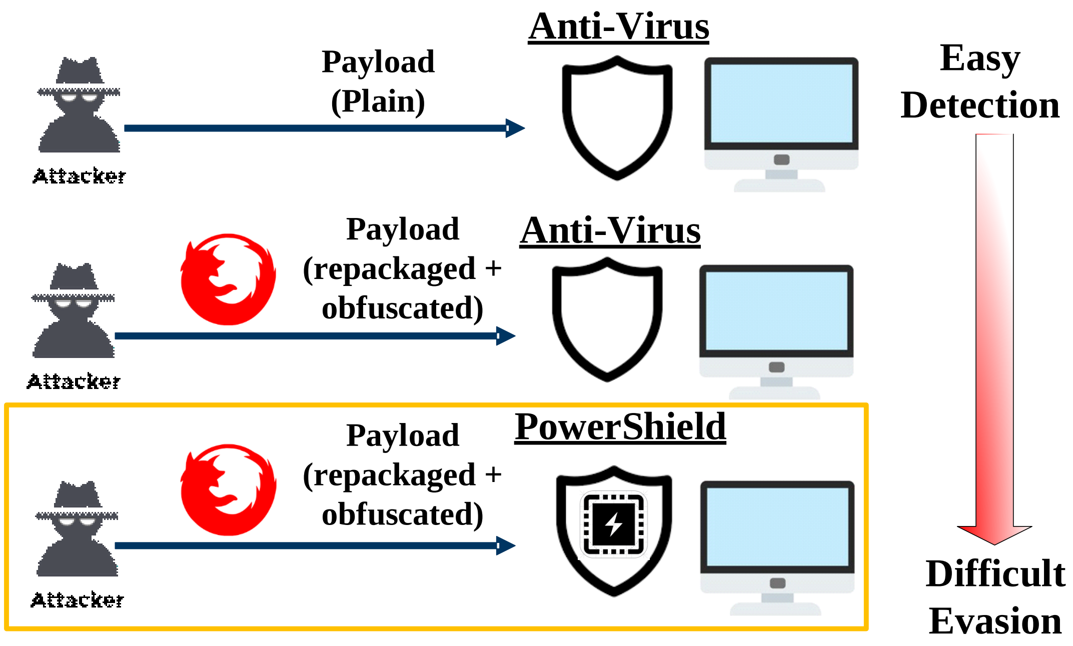

Figure 6.10: Comparison of proposed anomaly detector against existing anti-virus engines for different payloads

creased leakage indicates differences in MSR_PP0_ENERGY_STATUS profile of two Firefox versions. Third, we show the outcome of the anomaly detection block corresponding to a MSR_PP0_ENERGY_STATUS trace of obfuscated binary in Figure 6.9(c). We observed anomalies detected in the region of payload execution.

We evaluate the efficacy of detecting obfuscated payloads using $F_1score$ and FPR. We obtained 0.768 $F_1score$ in detecting 700 MSR_PP0_ENERGY_STATUS traces of clean and obfuscated Firefox browser. We obtained 0.52 as a false positive rate. A High FPR indicates many false positives in detecting obfuscated payloads. A high FPR is observed due to similar free energy scores, which increases difficulty to distinguish. We demonstrate effectiveness of deploying anomaly detector for malware detection as shown in Figure 6.10. An attacker sending a plain malicious payload cannot bypass conventional anti-virus engine. But, an obfuscated and packaged payload increases chance of evading anti-virus detection. We noticed 50% detection rate among 60 anti-virus engines on VirusTotal for obfuscated generated payload. Since `shakata ga nai` is an existing popular encoder, few anti-virus software are updated to detect the presence of this encoder. Moreover, it is even more challenging to evade a low-level hardware-based detector. Although, we only evaluated polymorphic techniques, but we envision similar behavior against metamorphic malware, with increased complexity.

*Effect of Payload size*

The shellcode injections have smaller memory footprint. Moreover, it has shorter executions time (in ms interval). These injections may alter MSR_PP0_ENERGY_STATUS profile in amplitude as shown in Figure 6.6 or in time as shown in Figure 6.9 (a). Therefore, larger payload sizes are easier to detect. We performed an experiment to test effect of payload size on detection accuracy. We introduced delay function as code injection in a CPU benchmark. In first case, delay function runs 100ms between two loops. In second case, delay function runs 20ms within a loop. We first trained RBMs on original benchmark, and then evaluated on code-injected benchmark. We observe 95% accuracy in 100ms delay injection and 83% accuracy in 20ms delay injection.

## 6.5 Comparison with HMD

Table 6.3 compares hardware-based anomaly detectors against various parameters. A. Tang et.al. and F. Ding et.al. showed the detection of real world malware attacks similar to the proposed work [9, 28]. We observe lower accuracy for detection of stage-1 shellcode exploits using RAPL interface and performance counters as shown in [9].Also, False positive rate (FPR) is higher in comparison to [9, 11]. Also, there is no requirement for additional feature selection in comparison to [9]. The action-based anomaly detection primarily focuses on detecting malware's distinct atomic actions such as stealing information, DDOS, ransomware as shown in [11, 15]. The proposed detector focuses on the initial stage-1 shellcode injection stage compared to atomic malware action as shown in [9, 28]. The confidence of detection would improve by monitoring atomic actions. The RAPL interface is accessible on-device compared to external power or electromagnetic emissions shown in [28, 15]. RBM and RDA detection algorithms localizes and detects the regions of anomalous executions in comparison to generalized oc-SVM [9, 11]. Finally, we showed for the first time the detection of polymorphic malware payload using hardware-based signatures.

Table 6.3: Comparison with existing Hardware-Based Anomaly Detectors

| Reference | Platform | Application | Algorithm | Detection | Information | Performance |
|---|---|---|---|---|---|---|
| [20] | Intel i7 Ivy-Bridge | Internet Explorer, Adobe Flash Player | oc-SVM | Exploit-based | 4 HPC | AUC = 0.995 |
| [21] | Android Dev. Board | Synthetic Android Malware | oc-SVM, markov model | Action-based | 4 HPC | FPR = 20% |
| [48] | DCS-834L IP Camera | IoT Malware (Mirai Botnet) | RDA + LSTM | Exploit + Action-based | External Power | TPR = 92.7% FPR = 2.9% |
| [5] | Altera FPGA Nios II, TS-7250, OlimexA13 | Synthetic Benchmarks + DDOS, ransomware | Hierarchical DBSCAN | Action-based | EM Signal | TPR $>$ 0.99, FPR $<$ 0.1% |
| **Proposed** | **Intel i5 IceLake** | **Firefox Browser** | **RBM** | **Exploit-based** | **RAPL Interface** | **FPR = 0.06;** $F_1 score$ = **0.87** |

## 6.6 Summary

This chapter experimentally demonstrated detection of shellcode exploits and obfuscated malware payloads by identifying anomalies in MSR_PP0_ENERGY_STATUS register values using free energy of an RBM model. The MSR_PP0_ENERGY_STATUS trace processing methods remove systemic noise removal, filter periodic noise, and align traces. The framework is evaluated by detecting shellcode injections in Firefox browser application exploiting existing vulnerabilities. The experimental demonstration on Intel i5, 10th

Generation Core shows 87% $F_1\ score$ in detecting shellcode injections. We further demonstrated the detection of a reverse TCP shell payload obfuscated through shakata ga nai polymorphic encoder. The detection framework shows 76% $F_1\ score$ but with 50% FPR in detecting obfuscated payloads in comparison to a 50% detection rate across 60 anti-virus engines on VirusTotal.

# CHAPTER 7
# CONCLUSION AND FUTURE WORK

Modern processors have numerous hardware events such as power domains, voltage, frequency, accessible through software interfaces for performance monitoring and debugging. These hardware events are architecture-agnostic, platform independent and does not require event selection in comparison to HPC. These events have not been explored for defenses against malware threats. This thesis demonstrates an alternative approach towards malware detection and analysis by leveraging low-level hardware signatures from on-chip power signatures and electromagnetic emissions. The proposed research developed machine learning methodology for detecting malware applications, classifying malware family and detecting shellcode exploits from low-level power-based signatures and electromagnetic emissions. The contributions of the thesis are summarized for each chapter in Section 7.1 and directions for future research in Section 7.2.

## 7.1 Dissertation Summary

**Chapter 3** concludes the applications executing on processor can be identified by deriving features from DVFS states of a processor's core using supervised machine learning model. The t-test analysis shows that DVFS signatures of different applications are distinguishable. The $F_1\ score$ of application classification is $>$ 0.7 evaluated across different machine learning models and cpufreq governors. An unknown application can also be identified using the learned model based on known applications. The results show $>$ 0.79 AUC on recall curves for known and unknown applications evaluated with RF model and all cpufreq governors.

**Chapter 4** concludes ML models can distinguish malware applications against benign by deriving features from DVFS states of a CPU core. $F_1\ score$ is $>$ 0.88 using RF based classification between benign and malware applications evaluated across all governors. Power security awareness of selecting cpufreq governor shows less power is dissipated on average with ondemand governor configuration and and higher detection accuracy is obtained. A discussion on a practical framework for on-device malware detection on edge devices with cloud-based software updates is presented.

**Chapter 5** concludes that ML models can detect malware applications and classifies detected malwares into characteristic malware families using EM emissions. The experimental analysis on Snapdragon 820 and DREBIN dataset shows 0.99 $F_1$ $score$ in detecting malwares and 0.88 $F_1$ $score$ in classifying detected malware into 8 respective families using SVM and RF models. The proposed feature extraction utilizing DWT on spectrogram improves detection accuracy by 1.2x compared to spectrogram alone. The experimental results shows 0.99 recall in detecting “unknown” benchmark applications and 0.87 recall in detecting “unknown” malware family.

**Chapter 6** concludes that shellcode exploits and obfuscated malware payloads can be detected by identifying anomalies in MSR_PP0_ENERGY_STATUS register values using free energy of an RBM model. The experimental evaluation of the framework on the Firefox browser application shows 0.87 $F_1$ $score$ and 0.06 FPR in detecting shellcode injections exploiting existing vulnerabilities of the browser application. A reverse TCP shell malware payload obfuscated through shakata ga nai polymorphic encoder show 0.76 $F_1$ $score$ and 50% FPR.

## 7.2 Future Research Directions

### 7.2.1 Multiple Channel Fusion of Hardware Events For Malware Detection and Analysis

This thesis analyses hardware events individually on the proposed AI models. The future work can extend to improve malware detection performance by fusing multiple channels voltage, frequency, thermal monitors, power (core and memory). The frequency scaling framework on modern systems extends beyond CPU to non-CPU devices. For instance,

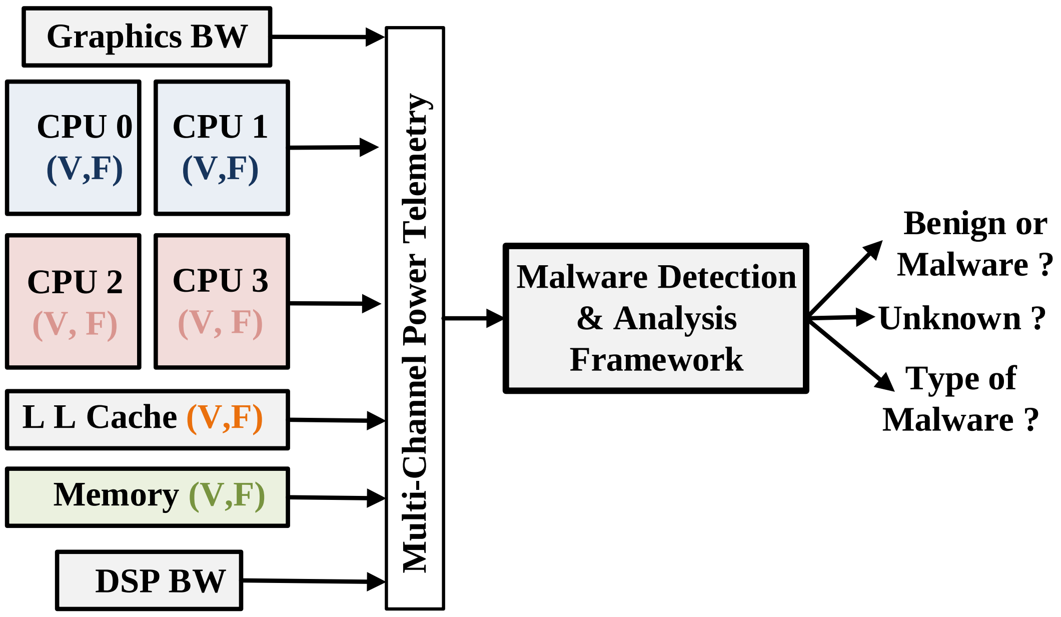

Figure 7.1: Multi-Channel Power Telemetry Fusion For Malware Detection and Analysis

devfreq modules in /sysfs subsystem of linux kernel comprises of frequency scaling governors for CPU bandwidth, memory latency, last level cache bandwidth, dsp and graphics. Similarly, power domains extents beyond core to on-core memory devices. The malware detection based on CPU DVFS can be extended to multi-channel fusion of Hardware events. In addition, a new malware analysis unit can be introduced to classify peculiar behaviors/activities. deep neural network (DNN) models for multi-channel fusion could further improve performance. The detection and analysis framework should also recognize out-of-distribution test examples, not seen by the model as unknown applications. The key challenge in multi channel input space is determining correlation and causation between events to reduce the high dimensional space.

### 7.2.2 Improving Prediction Likelihood of Detecting Out-of-Distribution Examples using Model Uncertainty

The unknown applications are detected by comparing predicted probabilities derived from classifier model with a pre-defined threshold. Suppose prediction probability of test sample is less than decision threshold. In that case, ML model has lower confidence is classifying the test sample as "known" sample and therefore, it is detected as unknown. The probability estimates from the classifier's output is obtained using Platt scaling method. Platt scaling algorithm produces probability estimates by fitting classifier's output, such as scores of SVM model to a parametric model i.e., a sigmoid function. But, these probability estimates are frequentist and not bayesian. The prediction probability of detecting unknown can be improved using model uncertainty. Bayesian approaches for uncertainty estimation to detect "unknown" applications with higher confidence. In addition to detecting unknown, estimating the likelihood of unknown belonging to benign or malware would further improve the detection model.

### 7.2.3 Extending Low-Level Hardware Based Detection to Multi-Workload Environment

The data collection experiments conducted in this thesis profiled an application under analysis individually without considering the impact of noise on power / EM profile resulting from concurrent background process execution. The proposed malware detection and analysis model applies to single workload behavior and its performance might drop in a multi workload environment. The current detection models should integrate features from both core and package level granularity. Voltage and frequency are updated are each core, which forms local signatures and power consumption is updated at the package level, which forms the global signature of an application. Including both local and global power-based feature would improve the model performance in a multi-workload environment.

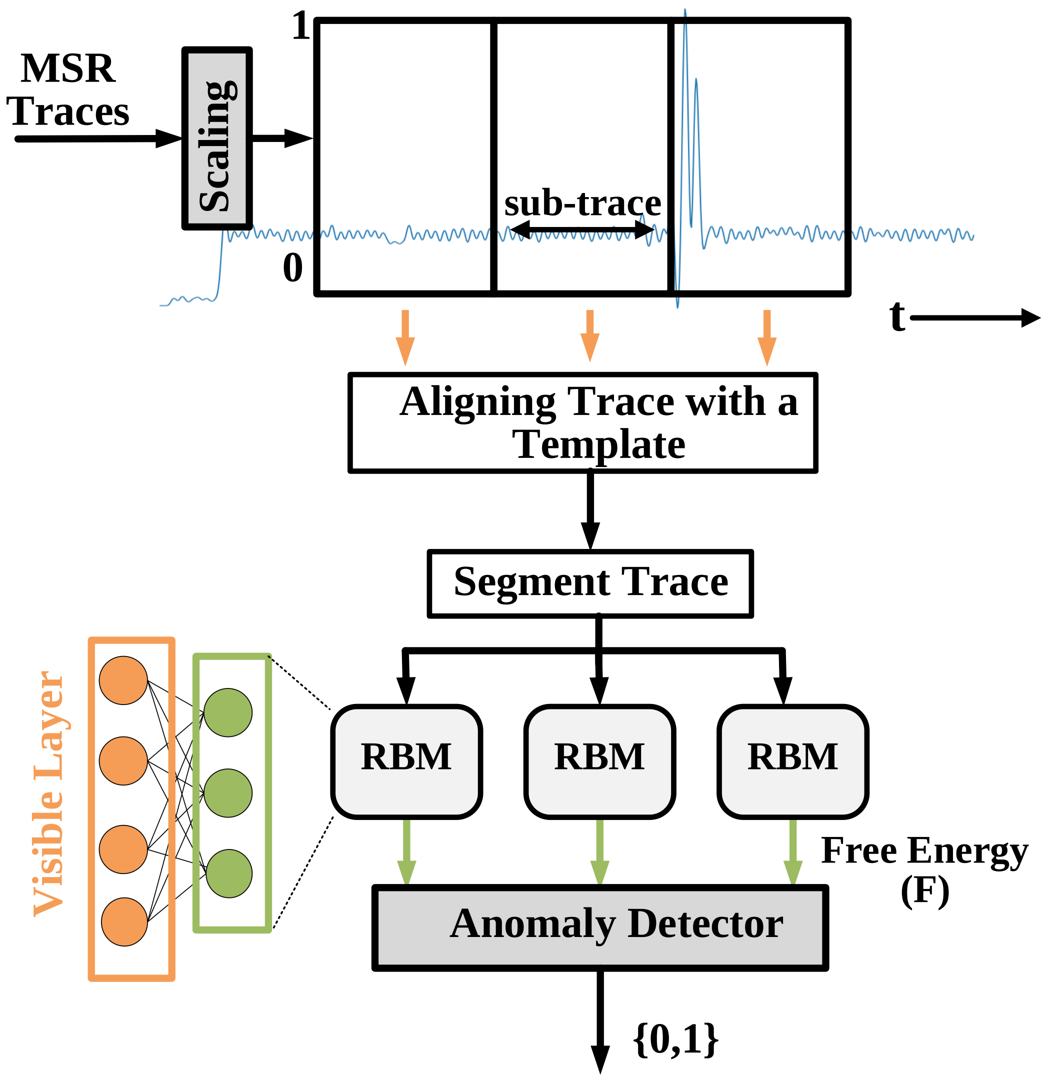

Figure 7.2: Real-time anomaly detection using RBMs

### 7.2.4 Real-time Anomaly Detection using RBMs

The proposed RBM model in Chapter 6 identifies deviation in MSR_PP0_ENERGY_STATUS register values to detect anomaly resulting from shellcode exploits on Firefox browser. The early detection of shellcode exploits prior to infecting the device requires a real-time detection of an anomaly in MSR_PP0_ENERGY_STATUS register values. There are multiple challenges in extending the current RBM model to a real-time detector.

- **Synchronization of Training and Real-time Traces** : In the current approach, the RBM is trained on MSR_PP0_ENERGY_STATUS traces corresponding to clean states of browser operation. The real-time detection would require analyzing MSR_-PP0_ENERGY_STATUS register values in a moving-window fashion. The traces collected in real time during inference are not perfectly synced with start of browser activity. The evaluation of non-synchronized traces on trained RBM would lead to misleading results.

- **Differentiating between background vs Target workload**: The activity of a core continuously varies in real-time. Multiple real-time processes such as interrupts, kernel worker threads are triggered at random times. Also, users can open and interact with multiple applications in real-time, which requires a detection model to identify the region of Firefox browser operation before detecting an anomaly in Firefox browser states.

The background activity of core under no operation can be detected by simple correlation between a MSR_PP0_ENERGY_STATUS template trace and real-time trace. Similarly, one possible approach to synchronize between template and real-time traces is by using alignment as shown in Figure 7.2. The future work would explore appropriate methods and perform cost analysis to address real-time detection using hardware signatures.

# Appendices

# APPENDIX A
# EM SIDE-CHANNEL DATA ACQUISITION SETUP

## A.1 Platform

The EM Side-Channel Data Acquisition comprises of Intrinsyc Open-Q 820 Development Kit [54]. The Open-Q 820 system on module (SOM) comprises of Snapdragon 820 quad-core processor. The CPU is based on ARM big.LITTLE architecture. The development kit has 32GB internal storage, 3GB RAM and LCD user interface. The processor hosts Android 7.0 OS.

## A.2 EM Probes

We selected 100 series EMC probes from beehive electronics. The EM side-channel signatures are captured using passive probes. The probe has loop diameter 0.85, tip diameter 1.0, and 50Mz 3dB bandwidth [55].

## A.3 Acquisition

The side-channel signatures are recorded on Tektronix DPO5204 oscilloscope (2GHz bandwidth). Arduino Uno is used to generate trigger for oscilloscope for starting capture.

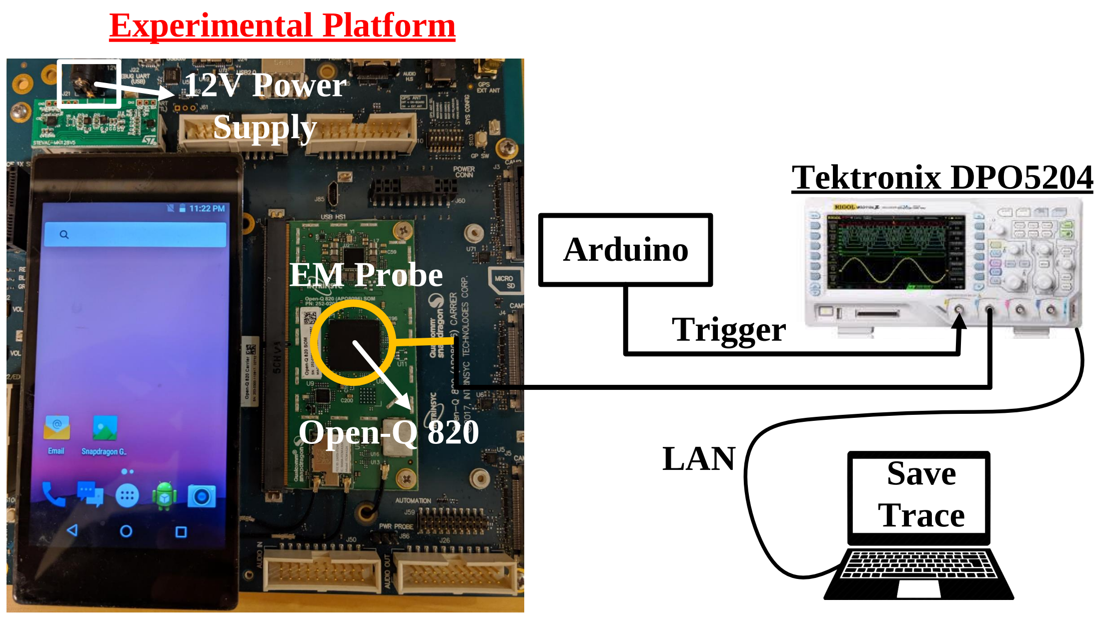

Figure A.1: EM side-channel data collection setup comprising Open-Q Snapdragon 820 Development Kit, EM probes, oscilloscope for acquisition and Arduino For Trigger

# APPENDIX B
# MALWARE CHARACTERISTICS

Y Zhou et al. have classified malware families based on installation, activation and payload mechanisms[51]. The selected malwares are repackaged versions of benign counterpart. Android specific malware installation methods include repackaging and updates as described in [51]. Repackaging benign application to include malware payloads is among the most common way malware gets installed by user. Among other methods include malware payload installation during an application update.

## B.1 Triggering Malware

Once installed, these malwares may stay dormant or actively start executing in the background. A major proportion of malware applications observe system level broadcast events for activating their payload [51]. The malware explored in this dataset observes broadcast event "BOOT_COMPLETED" for activation. These applications get triggered as soon as system completes the boot process. In addition to this, there are multiple other broadcast events either generated by android system or by other applications that malware registers in its manifest to trigger its action. These list of events are described in [51]. Therefore, we generated necessary trigger events as inputs to malware application. The broadcast receiver mechanism is one of the common techniques for triggering malware APKs [60]. In the experiments, we use android debug interface (adb) commands to send different broadcast receiver events to targeted malware application for activating malware payload.

We tested the activation of malware by monitoring frequency of system calls, while triggering the applications with user touches and broadcast events. Since, the malware payload is packaged in benign application, we tested for malware action by comparing the system call behavior of malware infected benign application and benign application itself.

## B.2 Malware Payload

The malware executes its payload after activation. Y Zhou et al. have categorized malware payloads into three broad categories; stealing personal information, broadcasting to remote servers, subscribing users to premium services and escalating privileges by exploiting android system vulnerabilities to remotely control device [51].

# REFERENCES

[1] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, Jun. 2017.

[2] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," vol. 8, pp. 1–22, Dec. 2018.

[3] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Evaluation of android malware detection based on system calls," in *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, ser. IWSPA '16, New Orleans, Louisiana, USA: Association for Computing Machinery, 2016, pp. 1–8, ISBN: 9781450340779.

[4] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2018.

[5] T. Chakraborty, F. Pierazzi, and V. S. Subrahmanian, "Ec2: Ensemble clustering and classification for predicting android malware families," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 2, pp. 262–277, 2020.

[6] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, Tel-Aviv, Israel: ACM, 2013, pp. 559–570, ISBN: 978-1-4503-2079-5.

[7] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a side channel based disassembler," in *Transactions on Computational Science X: Special Issue on Security in Computing, Part I*, M. L. Gavrilova, C. J. K. Tan, and E. D. Moreno, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 78–99, ISBN: 978-3-642-17499-5.

[8] M. Rahmatian, H. Kooti, I. G. Harris, and E. Bozorgzadeh, "Hardware-assisted detection of malicious software in embedded systems," *IEEE Embedded Systems Letters*, vol. 4, no. 4, pp. 94–97, 2012.

[9] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Research in Attacks, Intrusions and Defenses*, A. Stavrou, H. Bos, and G. Portokalidis, Eds., Cham: Springer International Publishing, 2014, pp. 109–129, ISBN: 978-3-319-11379-1.

[10] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *Pro-*

*ceedings of the 18th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 9404*, ser. RAID 2015, Kyoto, Japan: Springer-Verlag, 2015, pp. 3–25, ISBN: 9783319263618.

[11] M. Kazdagli, V. J. Reddi, and M. Tiwari, "Quantifying and improving the efficiency of hardware-based mobile malware detectors," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-49, Taipei, Taiwan: IEEE Press, 2016.

[12] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 651–661.

[13] N. Patel, A. Sasan, and H. Homayoun, "Analyzing hardware based malware detectors," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17, Austin, TX, USA: Association for Computing Machinery, 2017, ISBN: 9781450349277.

[14] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, "Eddie: Em-based detection of deviations in program execution," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17, Toronto, ON, Canada, 2017, pp. 333–346, ISBN: 978-1-4503-4892-8.

[15] N. Sehatbakhsh *et al.*, "Remote: Robust external malware detection framework by using electromagnetic signals," *IEEE Transactions on Computers*, vol. 69, no. 3, pp. 312–326, 2020.

[16] S. S. Clark *et al.*, "Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in *Proceedings of the 2013 USENIX Conference on Safety, Security, Privacy and Interoperability of Health Information Technologies*, ser. HealthTech'13, Washington, DC: USENIX Association, 2013, pp. 9–9.

[17] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, "On code execution tracking via power side-channel," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, Vienna, Austria: Association for Computing Machinery, 2016, pp. 1019–1031, ISBN: 9781450341394.

[18] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "Sok: The challenges, pitfalls, and perils of using hardware performance counters for security," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 20–38.

[19] A. Tang, S. Sethumadhavan, and S. Stolfo, "Clkscrew: Exposing the perils of security-oblivious energy management," in *Proceedings of the 26th USENIX Conference on*

*Security Symposium*, ser. SEC'17, Vancouver, BC, Canada, 2017, pp. 1057–1074, ISBN: 978-1-931971-40-9.

[20] A. Singh, M. Kar, S. K. Mathew, A. Rajan, V. De, and S. Mukhopadhyay, "Improved power/em side-channel attack resistance of 128-bit aes engines with random fast voltage dithering," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 2, pp. 569–583, 2019.

[21] S. Yang, W. Wolf, N. Vijaykrishnan, D. N. Serpanos, and Y. Xie, "Power attack resistant cryptosystem design: A dynamic voltage and frequency switching approach," in *Design, Automation and Test in Europe*, Mar. 2005, 64–69 Vol. 3.

[22] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "Scandalee: A side-channel-based disassembler using local electromagnetic emanations," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 139–144.

[23] J. Park, X. Xu, Y. Jin, D. Forte, and M. Tehranipoor, "Power-based side-channel instruction-level disassembler," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[24] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer, "Stealing keys from pcs using a radio: Cheap electromagnetic attacks on windowed exponentiation," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 207–228, ISBN: 978-3-662-48324-4.

[25] D. Genkin, I. Pipman, and E. Tromer, "Get your hands off my laptop: Physical side-channel key-extraction attacks on pcs," in *Cryptographic Hardware and Embedded Systems – CHES 2014*, L. Batina and M. Robshaw, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 242–260, ISBN: 978-3-662-44709-3.

[26] Y. Han, S. Etigowni, H. Liu, S. Zonouz, and A. Petropulu, "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*, ser. CCS '17, Dallas, Texas, USA: ACM, 2017, pp. 1095–1108, ISBN: 978-1-4503-4946-8.

[27] H. A. Khan *et al.*, "Idea: Intrusion detection through electromagnetic-signal analysis for critical embedded and cyber-physical systems," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2019.

[28] F. Ding *et al.*, "Deeppower: Non-intrusive and deep learning-based detection of iot malware using power side channels," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20, Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 33–46, ISBN: 9781450367509.

[29] L. Liu, G. Yan, X. Zhang, and S. Chen, "Virusmeter: Preventing your cellphone from spies," in *Recent Advances in Intrusion Detection*, E. Kirda, S. Jha, and D. Balzarotti, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 244–264, ISBN: 978-3-642-04342-0.

[30] A. Merlo, M. Migliardi, and P. Fontanelli, "Measuring and estimating power consumption in android to support energy-based intrusion detection," *J. Comput. Secur.*, vol. 23, pp. 611–637, 2015.

[31] L. Caviglione, M. Gaggero, J. Lalande, W. Mazurczyk, and M. Urbański, "Seeing the unseen: Revealing mobile malware hidden communications via energy consumption and artificial intelligence," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 4, pp. 799–810, 2016.

[32] N. Debande, Y. Souissi, M. A. E. Aabid, S. Guilley, and J.-L. Danger, "Wavelet transform based pre-processing for side channel analysis," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops*, ser. MICROW '12, USA: IEEE Computer Society, 2012, pp. 32–38, ISBN: 9780769549200.

[33] J. Longo, E. De Mulder, D. Page, and M. Tunstall, "Soc it to em: Electromagnetic side-channel attacks on a complex system-on-chip," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 620–640, ISBN: 978-3-662-48324-4.

[34] J. Ai, Z. Wang, X. Zhou, and C. Ou, "Improved wavelet transform for noise reduction in power analysis attacks," in *2016 IEEE International Conference on Signal and Image Processing (ICSIP)*, 2016, pp. 602–606.

[35] N. Sehatbakhsh, M. Alam, A. Nazari, A. Zajic, and M. Prvulovic, "Syndrome: Spectral analysis for anomaly detection on medical iot and embedded devices," in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2018, pp. 1–8.

[36] N. Chawla, A. Singh, M. Kar, and S. Mukhopadhyay, "Application inference using machine learning based side channel analysis," in *2019 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2019, pp. 1–8.

[37] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "Scandalee: A side-channel-based disassembler using local electromagnetic emanations," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 139–144.

[38] Y. Cheng *et al.*, "Magattack: Guessing application launching and operation via smartphone," in *Proceedings of the 2019 ACM Asia Conference on Computer and Com-*

*munications Security*, ser. Asia CCS '19, Auckland, New Zealand: Association for Computing Machinery, 2019, pp. 283–294, ISBN: 9781450367523.

[39] G. A. Jacoby, R. Marchany, and N. J. Davis, "Using battery constraints within mobile hosts to improve network security," *IEEE Security Privacy*, vol. 4, no. 5, pp. 40–49, 2006.

[40] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Advances in Cryptology - EUROCRYPT 2009*, A. Joux, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461, ISBN: 978-3-642-01001-9.

[41] T. Schneider and A. Moradi, "Leakage assessment methodology," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 495–513, ISBN: 978-3-662-48324-4.

[42] R. Longbottom, *Updated android benchmarks for 32 bit and 64 bit cpus from arm and intel contents*, Mar. 2018.

[43] *MS Windows NT kernel description*, https://www.kernel.org/doc/html/v4.15/admin-guide/pm/cpufreq.html, Accessed: 2010-09-30.

[44] *Linux cpufreq governors*, https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt, Accessed: 2019-11-21.

[45] V. Pallipadi and A. Starikovskiy, "The ondemand governor: Past, present and future," in *Proceedings of Linux Symposium, vol. 2, pp. 223-238*, 2006.

[46] *Ui/application exerciser monkey*, https://developer.android.com/studio/test/monkey, Accessed: 2019-11-21.

[47] *Scheduler extensions*, https://android.googlesource.com/kernel/msm/+/android-msm-marlin-3.18-nougat-dr1/Documentation/scheduler/sched-zone.txt.

[48] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, Dec. 2001, pp. 3–14.

[49] S. M. Pudukotai Dinakarrao, H. Sayadi, H. M. Makrani, C. Nowzari, S. Rafatirad, and H. Homayoun, "Lightweight node-level malware detection and network-level malware confinement in iot networks," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2019, pp. 776–781.

[50] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *NDSS*, 2014.

[51] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 95–109.

[52] *Iot developer survey 2019*, https://outreach.eclipse.foundation/download-the-eclipse-iot-developer-survey-results.

[53] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, "Monet: A user-oriented behavior-based malware variants detection system for android," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1103–1112, 2017.

[54] *Open-q 820 development kit*, https://www.intrinsyc.com/snapdragon-embedded-development-kits/snapdragon-820-development-kitsnapdragon-820-apq8096/, (Accessed February 9, 2021).

[55] *Emc probe set*, https://beehive-electronics.com/probes.html, Accessed: 2020-09-03.

[56] *Contagio malware dump*, http://contagiodump.blogspot.com/, (Accessed February 9, 2021).

[57] *Virusshare.com - because sharing is caring*, https://virusshare.com/, (Accessed February 9, 2021).

[58] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, 2017, pp. 31–38.

[59] S. K. Dash *et al.*, "Droidscribe: Classifying android malware based on runtime behavior," in *2016 IEEE Security and Privacy Workshops (SPW)*, 2016, pp. 252–261.

[60] J. Yu, Q. Huang, and C. Yian, "Droidscreening: A practical framework for real-world android malware analysis," *Sec. and Commun. Netw.*, vol. 9, no. 11, pp. 1435–1449, Jul. 2016.

[61] A. Azmoodeh, A. Dehghantanha, M. Conti, and K.-K. R. Choo, "Detecting crypto-ransomware in iot networks based on energy consumption footprint," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 4, pp. 1141–1152, 2017.

[62] *Intel 64 and ia-32 architectures software developer's manual combined*, https://software.intel.com/, (Accessed October 4, 2021).

[63] *Shikata ga nai encoder still going strong*, https://www.mandiant.com/resources/shikata-ga-nai-encoder-still-going-strong, (Accessed February 9, 2021).

[64] *R7-2015-04 disclosure: Mozilla firefox proxy prototype rce*, https://www.rapid7.com/blog/post/2015/03/23/r7-2015-04-disclosure-mozilla-firefox-proxy-prototype-rce-cve-2014-8636/, (Accessed November 15, 2021).

[65] *Firefox proxy prototype exploit module*, https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/multi/browser/firefox_proxy_prototype.rb, (Accessed November 15, 2021).