

Unsupervised Detection of Anomalous Behavior in Wireless Devices based on Auto-Encoders

A. Albasir, Q. Hu, M. Al-tekreeti, and K. Naik

University of Waterloo

Waterloo, Ontario

{aalbasir, q45hu, snaik}@uwaterloo.ca

N. Naik

Defence School of CIS

Ministry of Defence, UK

nitin.naik100@mod.gov.uk

A. J. Kozlowski and N. Goel

Cistech Limited

Ottawa, Ontario

{jkozlowski, ngoel}@cistech.ca

Abstract—A major problem of wireless devices is the detection of security threats in an efficient manner. Several recent incidents show that malicious applications (apps) can find their ways to online markets (e.g., Google Play Store) and be available for download and installation. Such malicious apps can collect sensitive data from millions of users and send them to a third-party servers. In this paper, we propose a methodology that leverages the power consumption of wireless devices to build a model that makes them more robust to the presence of malicious apps. The method consists of two stages: (i) Feature Extraction where stacked Restricted Boltzmann Machine (RBM) AutoEncoders (AE) and Principal Component Analysis (PCA) are used to extract features vector based on AE's reconstruction errors. (ii) Classifier where One-Class Support Vector Machine is trained to perform the classification task. The validation of the methodology is performed on a real measurements dataset. The obtained results show a good potential and prove that AEs' reconstruction error can be used as a good discriminating feature. The obtained detection accuracy surpasses previously reported techniques, where it reaches up to $\sim 98\%$ in some scenarios.

Index Terms—Malware Detection, Power Consumption Information, Denoising AutoEncoder, Wireless Devices

I. INTRODUCTION

Wireless devices and Internet of Things (IoT) devices are used everywhere in the world to make people's daily tasks easier and more convenient. In the near future, every device a person owns, and nearly any object he/she interacts with, will be connected to the Internet. Whether it is your smartphone, wearable tech or everyday household objects, the offered services will have an incredible impact on people's lives. Given the fact of the exponential growth of these devices (it is expected that we will have ~ 50 billion by 2020 [1]), and the fact of the continuous access to Internet; unfortunately, these devices represent the next frontier for hackers. Gartner, in [2], predicts that more than 25 % of identified attacks in enterprises will involve IoT/wireless devices.

To highlight the severity of the problem to be addressed, the following are recent motivational incidents: (i) The number of compromised Android devices by banking malware has increased by a factor of 3 in 2018, according to Kaspersky's reports [3] [4]. The total number of impacted users by at least one such an attack during the last year reached 1.8 million. (ii) In October 2016, there was a massive Distributed Denial of Service (DDOS) attacks that caused widespread disruption of legitimate Internet activity in the US. This denial

of service attack from unknown culprits on a US Internet service provider's domain name system caused user access to be severely restricted. The source of the attack was about 100k malicious IoT devices, according to [5]. These incidents prove that existing detection techniques very often fail to prevent such threats and hence the need for second lines of defence is paramount.

In this context, we formulate the problem of detecting malwares in wireless devices as an anomalous behavior detection problem; where the anomaly is the malware that is causing the device to behave anomalously. Earlier research efforts are implemented at different levels and can be classified based on what information is being analyzed (static program features vs. runtime behavior). In *static analysis* [6], without the need to execute an application (app), its source code and manifest file are analyzed to check if it is a malicious app or not. *Static analysis* can detect zero-day malwares; however, such an approach has limitations, namely, accessing the source code. If malware developers learn about the hypotheses in the static analysis, they can adapt their code to look benign. On the other hand, in *dynamic analysis*, an app is examined during execution time. Thus, these approaches aim to detect infected devices to minimize further damage. In *dynamic analysis* approaches, the main idea is to monitor the device to capture certain data (network based information [7], operating system based information [8], or power consumption based information [9]) to be used for analysis.

Our paper proposes a proof-of-concept, off-device monitoring and analysis approach that aims to detect anomalous behavior in wireless devices. The proposed methodology is based on: (i) monitoring the power consumption of a wireless device and treating it as a signal carrying information about the health of the device; (ii) applying data augmentation to increase the number of observations of our dataset; (iii) utilizing stacked RBM AutoEncoders to process the collected information and unsupervisedly extract/learn features based on the reconstruction errors of DEA; (iv) applying PCA to the reconstruction errors of DEA, thereby isolating noise and outliers from training of the reconstruction; and (v) training a One-Class SVM on the extracted/selected features to detect possible anomalous behavior. The solution strategy is based on the two hypotheses: (i) Every piece of software, whether malware or legitimate, would have a fingerprint in the power

consumption of the wireless device that makes it inevitable for malware to go undetected, having the right approach to process and analyze its power signals. (ii) The power consumption signals obtained from the wireless device that is “malware free” should have similar patterns (normal/expected behaviors). Such behaviors can be characterized by training AutoEncoders (AE). On the other hand, if the device starts to behave anomalously due to certain causes (malware presence in our case), its power consumption signals should reflect that as a form of deviation from the norm. Consequently, the resultant AE output - reconstruction error - should be minimal if the wireless device is still behaving normally and maximal once the device starts to behave anomalously (maliciously).

The main contributions of this paper are as follows:

- To the best of our knowledge, this is the first work to leverage a combination of stacked RBM AutoEncoders, data augmentation, and principal component analysis (PCA) to capture discriminative information (features) from the power consumption signals of wireless devices for malware detection;
- We utilize the extracted features to build a classifier based on the One-Class SVM that detects when a wireless device is behaving anomalously;
- We empirically validate the methodology to demonstrate its effectiveness using a real measurement dataset.

We first provide a brief overview on the related work in Section II. We then discuss the proposed methodology in Section III. Section IV and V describe the dataset used to validate our methodology, the experiments conducted to acquire the data and the results of the new proposed approach. Section VI concludes the paper.

II. RELATED WORKS

Detection of anomalies using devices’ power consumption information was firstly introduced by Kim et al. [10]. They highlighted the fact that the power consumed by devices can be used to detect anomalies which were difficult to detect by only analyzing the static characteristics of an application. The prototype that they built is for Windows phones and was used to detect number of malwares with detection accuracy reaching 80%. Similar to Kim’s work, Jacoby and Davis [11] showed how to analyze the power consumption and CPU utilization to reveal network attacks by using a battery-based Intrusion Detection System (IDS). In [12], the authors showed that the additional consumed power due to malware is not significant and is similar to the noise introduced by unpredictable user and environment interactions. Their conclusion stated that battery life information was not very discriminative to be used to detect malware on smartphones. While their claims might be correct given the fact that their conclusion is based on on-device PowerTutor application [13], our measurement approach is much different and our measurement’s sampling frequency is much higher which, in turn, facilitate higher chances of detecting events that PowerTutor fails to detect. While the PowerTutor supports 4 samples per second, the Monsoon power monitor used by us supports 5000 samples per

second of power measurement. As a common note about most of the reported techniques, the time required for the analysis is significantly long which make them impractical.

In 2014, Zefferer et al. [14] proposed a different methodology in which they extracted the power consumption of an emulated malware running on a smartphone using also the PowerTutor app. Mel Frequency Cepstral Coefficients (MFCC) was utilized to extract features from each power trace, and then they fit GMM (Gaussian Mixture Model) to classify the power traces. Following a very similar approach, Yang et al. in [15] monitored the power consumed of Samsung GALAXY S5 and LG G2 smartphones using an on-device tool. These methodologies perform well in detecting emulated malware with an accuracy of 92%, but a main drawback is that the power consumption was taken with a low sampling rate. Consequently, such a technique may fail to detect malware that are rarely active. This disadvantage has been tackled in the our work by using an external tool that has much higher measurements sampling rate and changing the way the features are extracted and the way the classification is performed.

Although there are several studies about malware detection using the power consumption data of devices, to the best of our knowledge, none of the studies in this domain have attempted to leverage a combination of stacked RBM AutoEncoders, and PCA to extract features from the power consumption signals of wireless devices in an unsupervised manner.

III. METHODOLOGY

We start this section by providing a brief background of the tools used throughout our methodology. Then, the proposed methodology and the idea behind it are described in details.

A. Restricted Boltzmann Machine (RBM)

RBMs were firstly introduced by [16]. They are intended for nonlinear dimensionality reduction and feature extraction. Later, and due to their capability to efficiently model the training data distribution, they have been found to be a suitable option for anomaly detection applications [17]–[19].

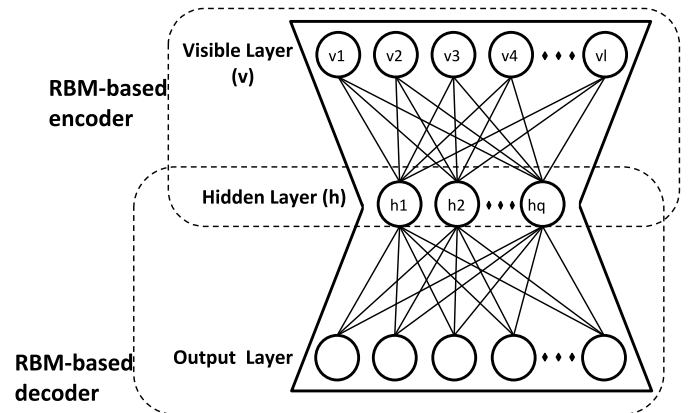


Fig. 1. Stacked RBM AutoEncoder

RBM is a stochastic graphical model which learns a probability distribution over input dataset, while restricting its

visible units and hidden units from forming a fully connected bipartite graph. As shown in Fig. 1, RBM AutoEncoder is comprised of two layers; visible layer and hidden interconnected using symmetrically weighted connections (weights). Units ($\mathbf{h}$) in the hidden layer capture higher-order correlations of the visible units ($\mathbf{v}$) in visible layer connecting it.

The training procedure minimizes the overall energy so that the data distribution can be well captured. The used energy function is defined by:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \mathbf{v}} bv_i - \sum_{j \in \mathbf{h}} ch_j - \sum_{i,j} v_i h_j w_{ij} \quad (1)$$

where $\mathbf{v}$ is the input signal vector that forms the visible units and $\mathbf{h}$ is a vector that forms the hidden units (features); $\theta = \{w, b, c\}$ are model parameters. Starting with random weights, the state of hidden units $\mathbf{h}$ is set based on the joint probability defined using:

$$p(\mathbf{v}, \mathbf{h}; \theta) = \frac{1}{Z(\theta)} \exp(-E(\mathbf{v}, \mathbf{h}; \theta)) \quad (2)$$

where $Z(\theta)$ is a normalizing factor called partition function. The marginal distribution over the visible layer $\mathbf{v}$ is:

$$p(\mathbf{v}; \theta) = \frac{1}{Z(\theta)} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta)) \quad (3)$$

We can build a more robust model that can sufficiently extract high level abstract features by stacking layers of RBMs to form stacked RBM AutoEncoders [20]. The stacked AE can be trained via a greedy layer-wise procedure. Each layer is trained as an RBM using contrastive divergence (CD) strategy (Gibbs sampling for example [21], [22]); after each RBM layer of the stacked AE has been trained, weights are clamped and a new layer is added. For more details on the structure of RBM and stacked RBM AEs and the way they are trained, the reader is directed to [20], [22], [23].

B. Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique that reduces the number of features in the dataset, and at the same time, it tries to retain as much information as possible by preserving the maximum variance in the original dataset [24]. Therefore, by linearly transforming the original data points to another space, the inherent structure of the data makes it easier to recognize/classify. PCA is performed by using singular value decomposition (SVD). Given $\mathbf{X}$ which is a matrix of signals of size $M \times n$. Each row is signal/observation x that forms a vector of a size $1 \times n$. The first step of PCA is to mean center the matrix of signals, which is accomplished by subtracting the mean of signals from each signal x_i as shown in Eq. III-B

$$\hat{x}_i = x_i - \mu_i, \quad \mu_i = \frac{1}{M} \sum_i^M x_i$$

The resultant mean centred matrix $\hat{\mathbf{X}}$ (it is of the same size as $\mathbf{X}$) is then decomposed using SVD,

$$\hat{\mathbf{X}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

Here, $\mathbf{U}$ is a $n \times M$ unitary matrix, $\mathbf{V}$ is a $n \times n$ unitary matrix, and $\mathbf{\Sigma}$ is a diagonal matrix comprising the singular values of $\hat{\mathbf{X}}$ in decreasing order [25]. A reduced dimensional representation of $\hat{\mathbf{X}}$ can be obtained by discarding columns of $\mathbf{U}$ and $\mathbf{V}$,

$$\hat{\mathbf{X}} \approx \mathbf{U}_{0:j} \mathbf{\Sigma}_{0:j,0:j} \mathbf{V}_{0:j}^T$$

Here, j denotes the number of columns (principle components) retained.

C. Classification - One Class SVM

One-Class Support Vector Machines (OC-SVM) is an anomaly detection technique. It was originally developed by scholkopf [26] to identify novelty unsupervisedly. In his formulation [26], given a training set $\mathbf{X} = \{x_i\}_{i=1}^m$, the algorithm learns a function (soft boundary) that returns +1 in a region capturing “majority” of the training observations if x_i falls within “normal region”, and -1 elsewhere. In the case where the data is not linearly separable, the so called kernel trick is applied, where each point is implicitly projected to a higher dimensional feature space (through linear/nonlinear kernel) to separate the data set from the origin. Then one needs to solve the following quadratic optimization problem:

$$\text{Minimize} \quad \frac{1}{2} \|\mathbf{W}\|^2 + \frac{1}{vm} \sum_{i=1}^m \xi_i - \rho \quad (4)$$

subject to

$$\langle \mathbf{W}, \Phi(x_i) \rangle \geq \rho - \xi_i, \quad \xi_i \geq 0 \quad (5)$$

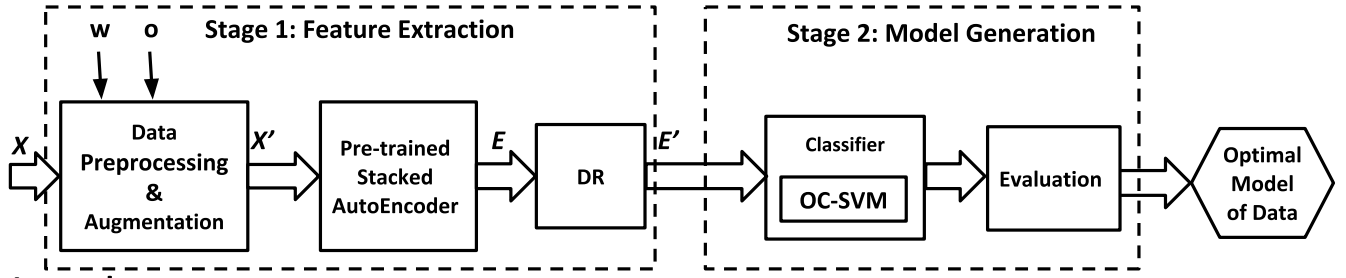
The parameter $v \in (0, 1]$ sets an upper bound on the fraction of outliers and a lower bound on the number of training examples used as support vectors.

D. Model Pipeline

The basic concept of our methodology can be explained using two main procedures. Procedure 1, shown in Fig. 2 compromises of two main stages, namely *Feature Extraction* stage and *Model Generation* stage. Procedure 1 is used to train and test the model on the wireless devices collected dataset. Procedure 2, shown in Fig. 4, which makes use of the learned models from Procedure 1, describes the process of labeling a new unlabeled measurement obtained from a wireless device as whether it came from a compromised device or not.

E. Procedure 1: Features Extraction and Model Generation

The input data to Procedure 1 is $\mathbf{X}$, which represents the raw dataset used for training and testing. The formal definition of $\mathbf{X}$ is as follows: $\mathbf{X} = \{x_1^{(t)}, x_2^{(t)}, \dots, x_i^{(t)}, \dots, x_m^{(t)}\}$, which is a matrix of size $m \times n$. $x_i^{(t)}$ represents one reference signal (time-series) and is taken by keeping the same time duration and conditions in terms of configuration of a wireless device. Each of the rows which forms the matrix $\mathbf{X}$ is represented by $x_i^{(t)}$, which is described as follows: $x_i^{(t)} = [x_1, x_2, \dots, x_i, \dots, x_n]$. Hence, $x_i^{(t)}$ is a vector of size $1 \times n$. In other words, the elements in $x_i^{(t)}$ represent the values of the power consumed by a device sampled at frequency F_s for a time $t = T$.



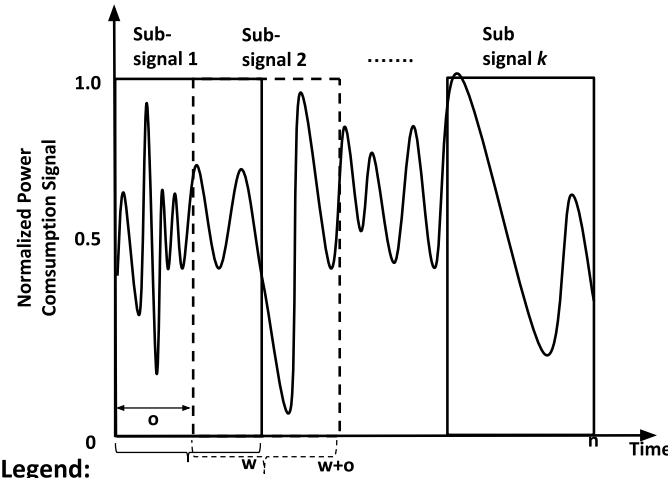
Legend:

X - a matrix of m # of reference signals by n # of observed values in each signal, o - offset of the window
 W - window size (# of sampels in each sub signal), X' - a transformed version of X of size $K*m$ by W where $K = \text{floor}((n-w)/o)$, E - a matrix which represents the reconstruction of errors of the same size as X'
 E' - a matrix of size m by W' , where $W' \ll W$, **OC-SVM** - 1-Class Support Vector Machine, **DR** - Dimensionality Reduction

Fig. 2. Overview of the methodology: Signals Transformation and Model Training & Testing

1) *Stage 1: Feature Extraction (FE)*: The first step in this stage is data pre-processing where we start with *Min-Max Normalization*. Since power consumption signals from different devices have different scales, we normalize (using the formula described in Eq. 6) all signals to be in the range [0,1]. This is also necessary for a faster convergence when training the AE.

$$x^{(t)'} = \frac{x^{(t)} - \min(x^{(t)})}{\max(x^{(t)}) - \min(x^{(t)})}; \text{ where } x^{(t)} \in \mathbf{X} \quad (6)$$



Legend:

n - # of samples in each signal, w - window size, o - an offset,
 $k = \text{floor}\{(n-w)/o\}$ - # of sub-signals

Fig. 3. Data Augmentation with Sliding Window

Signals Partitioning with Sliding Window: Then, for each normalized power signal we apply a sliding window to partition the signals into sub-signals as shown in Fig. 3. Based on a window size W and an offset o , we partition each signal accordingly (we call this process “Data Augmentation”. The output of this process is a transformed version of our input dataset X' . X' is also a matrix of size $m * K * W$, where

$K = \lfloor \frac{n-W}{o} \rfloor$. The reason behind this step is that in the case where the dataset is limited (small dataset), implementing a way to augment the data is very important to deploy a robust model and avoid the over-fitting problem. Next, we use the subsignals in X' to train the RBM AE and compute the reconstruction error.

Pre-trained Stacked RBM AutoEncoders: This is the main block in our FE stage. After data pre-processing, we use X' as an input to train the RBM AE. Once the AE is trained, we use the pre-trained AE as our feature extraction unit. The structure of our Pre-trained RBM AE includes 2 RBM layers each of a size (number of hidden units) 3000 and 500, respectively. We follow Yoshua et. al (in [27], [28]) way of training our RBM. We use power consumption signals obtained from a “malware free” wireless device to perform greedy layer-wise unsupervised training to the two RBM layers. Then we train the stacked AutoEncoder using the training algorithm described in Algorithm 1. The objective is to minimize the mean squared error and iteratively updates the parameters of the stacked AE. After initializing the network parameters, we compute the mean squared error and then update the parameters accordingly until they qualitatively converge using the stochastic gradient descent method.

Algorithm 1: Stacked AE training algorithm.

Input : Dataset X' :

x'_i, i^{th} training example, $i \in \{1, \dots, k\}$

Output: Optimal parameters $\Theta^* = \{\Theta_E^*, \Theta_D^*\}$

1 Initialization of Network Parameters Θ_E, Θ_D ;

2 **while** $epoch < N_{epochs}$ **do**

3 Compute mean square error (MSE)

4 $L_{MSE}(\Theta; x'_i) = \frac{1}{k} \sum_{i=1}^k \|x'_i - \mathcal{D}(\mathcal{E}(x | \Theta_E) | \Theta_D)\|^2$

5 Update Parameters Θ_E, Θ_D using SGD

6 **end**

In detection procedure (Procedure 2), the input is the pre-processed/manipulated observations X' , and the output is the resultant reconstruction residual error E .

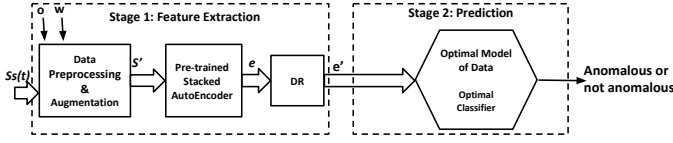


Fig. 4. Detection of Anomalous Behavior

Dimensionality Reduction (DR): Given that we are using a large window size and offset to augment the data, as pre-processing step, we can generate huge number of sub-signals. In order for us to avoid the over-fitting problem when training the OC-SVM classifier, and also to extract main components (which contain most of the discriminative information) from the AE’s output, we apply dimensionality reduction techniques, namely Principal Component Analysis (explained earlier), to the AE’s reconstruction residual error vectors before training the classifier. The input is the resultant reconstruction error E , and the output is E' , which is a reduced version of E of a size $m \times W'$, where $W' \lll W$.

F. Classification: OC-SVM

The task of detection in our methodology is performed using a semi-supervised learning technique. As shown in Fig. 2, we employ One-Class Support Vector Machines (explained earlier) as an anomaly detection technique. Since only data from one-class is available (power consumption signals obtained from normal behaving wireless devices), the objective of using OC-SVM is to learn a function that returns $+1$ for the normal behavior region (**not anomalous**) and -1 elsewhere - malicious behavior region (**anomalous**). In order to obtain more versatile decision boundaries, we use two kernel functions (namely Sigmoid and RBF kernel) to investigate their effectiveness on the detection performance. We train the OC-SVM using the feature matrix E' with M rows (number of observations) and W' columns (number of features).

G. Procedure 2: Detection Procedure

The detection step shown in Fig. 4 also includes 2 stages, reconstruction error concatenation and classification. The reconstruction error concatenation stage is concatenating the reconstruction errors of every window split from each signal. Then, using PCA to do feature extraction and finally perform the classification using the SVM algorithm.

IV. DATASET

The dataset consists of 105 raw power consumption signals of an Android smartphone that were collected using an external tool, namely, Monsoon Power Monitor. The used test bench is explained in [29]. The Monsoon Power Monitor is a reliable device which measures different electrical parameters, namely, voltage and current in real time of a device while provides constant direct current voltage to it. The measurements are taken at sampling rate (F_s) of 5000 samples/s; thus, we can assert that it has high sensitivity to detect any short activity running on a smartphone. Furthermore, the monitoring device

is connected to a computer to store the measurements that will be used for the analysis.

The raw power consumption signals in our dataset represent two main classes: (i) *No Malware* class - Reference signals, where the power measurements are obtained from a smartphone running YouTube, and no other applications are running in the background; and (ii) *Malware* class - Malicious signals, where the power measurements are obtained from a smartphone running YouTube and an *emulated malware* running in the background. The emulated malware has a tunable activity cycle, called “malware activity duty cycle” (Dut). Dut represents the percentage of the time when the malware is actively performing the tasks in the background. To illustrate, if *Dut* is 1%, it means that the emulated malware is active for 0.6 seconds in every 60 seconds. The *emulated malware* is a piece of code that mimics the activities of a real malware and runs in the background. Just like real malwares, the emulated malware performs the following tasks: (i) fetches some information; (ii) downloads the collected information into the phone; and (iii) goes to the sleep mode till the next activity cycle comes. Therefore, the Malicious signals class has several subclasses depending on the “Malware activity duty cycle” (Dut), as we show below.

The dataset has 15 power traces from each of these classes. Each power signal has 1,500,000 samples that characterize a smartphone in either of the above mentioned classes. 15 of the 105 signals represent the *No Malware* class and are basically a YouTube video streaming with no other applications running in the background. The remaining 90 of the 105 signals represent the *Malware* class and are basically the power consumption of the device while streaming a YouTube video plus the *emulated malware code* running in the background. The 90 signals represent 6 classes of different malware intensity (15 signals in each class): *malware with Dut = 1%* class, *malware with Dut = 2%* class, *malware with Dut = 3%* class, *malware with Dut = 4%* class, *malware with Dut = 8%* class, and *malware with Dut = 12%* class.

V. EXPERIMENTS AND RESULTS

Experiments: The problem of detecting malicious behavior in wireless devices is a semi-supervised classification problem. Consequently, given the dataset that we have, we perform six binary classifications to validate our method. The healthy class (*No Malware*) is tested against the six scenarios of of malware intensity ($Dut = i\%$, $i \in \{1, 2, 3, 4, 8, 12\}$). The results reported in our experiments are based on 3-fold cross-validation. Note that the training sets only consisted of data from normally behaving devices (*No Malware*), but the validation sets were mixed with both *No Malware* and *Malware with Dut = i%*. The effectiveness of the chosen AE Error based features in our methodology was verified by building an alternative baseline model. A brief description of the baseline model is as follows: We extract two features (namely the mean and standard deviation) out of each power trace. Then we train traditional Support Vector Machine (SVM [30]) on the extracted features. The results are discussed in

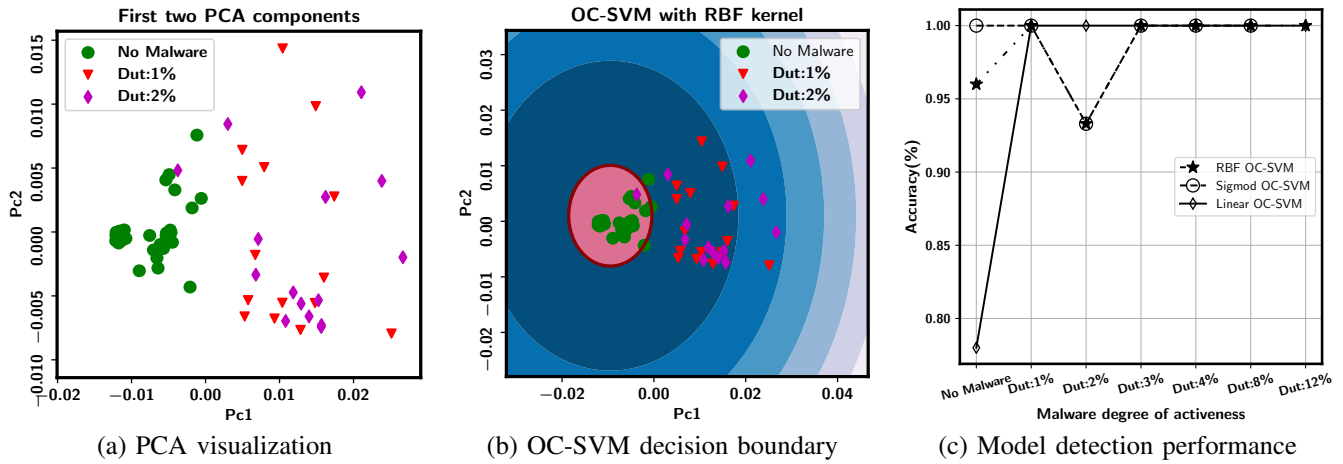


Fig. 5. Performance results of our methodology

the following subsection. Furthermore, the impact of PCA on the classification performance was verified. The way we investigate that is to train our OC-SVM model once on the PCA data and the other time on raw AE error data. Also, we investigate how the size of the “window” W impacts the overall detection performance. Finally, all of the training and testing experiments were performed on Google’s powerful analysis platform (Colab) [31]. The deep learning tasks were carried out in Python using Google Tensorflow framework and Keras [32] Python libraries.

Classification Results:

Before we start reporting our results, we visualize whether the proposed AE based feature extraction discussed in Sec. III-E1 is effective for wireless devices’ anomalous behavior detection. Figure 5-(a) shows how the data points from normally and maliciously devices are distributed using PCA’s first two components. As can be noticed in Fig. 5-(a), although dimensionality reduction loses some information and visualization results of low-dimensional space cannot fully reveal what are in the high-dimensional space, it is still insightful that the extracted features carry good information that can be used to construct an effective detection model.

Figure 5-(b) illustrates the decision boundary found by the kernelized OC-SVM. The used kernel function is Radial Basis Function (RBF), which is a kind of Gaussian kernels. The decision boundary shows that the model is able to separate most of the positive (normal malware-free observations) from the negative points correctly. We show only two extreme cases of the emulated malware in Fig. 5-(b), namely $Dut = 1\%$ - the least active case, and $Dut = 12\%$ - the most active case; however, the results show good detection performance regardless how active the emulated malware is (as we show also in Fig. 5-(c) and in Table I), thanks to the good features learned by the RBM AE. Figure 5-(c) highlights the impact of the OC-SVM kernel functions on the accuracy of the model as the malware intensity varies ($Dut = i\%$). In our training phase of the OC-SVM we perform a grid search to tune the hyper-parameters and explore the capabilities of the OC-SVM.

TABLE I
COMPARISON RESULTS

Model	Detection Accuracy %
Our Model: {RBM AE + OC-SVM}	98 ($\pm 2\%$)
Baseline: {mean&STD} + SVM	65 ($\pm 6\%$)
[14]: {FCC + GMM}	68 ($\pm 5\%$)

Finally, to demonstrate the effectiveness of our AE based features extraction technique, we compare the average detection accuracy of our methodology against the baseline model, as well as against previously reported techniques. Table I shows that AE based features outperform the used baseline model with at least 30% performance gain.

VI. CONCLUSIONS

This paper introduces a new malware detection technique that is based on devices’ power consumption signals and stacked RBM AutoEncoders. We treated the measured power readings as signals carrying discriminative information that can be learned by AutoEncoders. The results were promising and show a potential of applying this technique to detect more insights about the wireless devices’ operational state (e.g. hardware failure).

As a next step, we will be including more realistic anomalous behaviors of wireless devices. This will cover a wide range of emulated malware as well as real malware. Moreover, the experimental setup in the current paper is based on smartphone, in the future work, we will be building an IoT application so that we can validate the methodology on IoT devices as well. We will also simulate some well-known cyber-security attacks such as DDOS to investigate how well the proposed methodology performs. This will ensure the robustness of the methodology in detecting real anomalous behaviors for a wider range of devices.

REFERENCES

- [1] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," CISCO white paper, Tech. Rep., 2011.
- [2] S. B. Alaybeyi, E. Perkins, and R. Contu, "Market guide for iot security," October 2016.
- [3] K. Lab. (2019) Financial cyberthreats in 2018. . [Online]. Available: <https://securelist.com/financial-cyberthreats-in-2018/89788/>
- [4] T. I. T. Avast. (2019) Android apps on google play store come with nasty surprise. . [Online]. Available: <https://blog.avast.com/adware-plagues-google-play>
- [5] S. Hilton, "Dyn analysis summary of friday october 21 attack. 2016," <http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/>, October 2016.
- [6] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, Siemens, and C. E. R. T., "Drebin: Effective and explainable detection of android malware in your pocket," in *NDSS*, vol. 14, 2014, pp. 23–26.
- [7] S. Tobiyama, Y. Yamaguchi, H. Shimada, and et. al., "Malware detection with deep neural network using process behavior," in *the 40th Annual IEEE Conference (COMPSAC)*. IEEE, 2016.
- [8] A. İ. Ayşan and S. Şen, "Api call and permission based mobile malware detection," in *Signal Processing and Communications Applications Conference, 2015 23th*. IEEE, 2015, pp. 2400–2403.
- [9] A. Albasir, R. S. R. James, K. Naik, and A. Nayak, "Using deep learning to classify power consumption signals of wireless devices: An application to cybersecurity," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 2032–2036.
- [10] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proceedings of the 6th international conference on Mobile systems, applications, and services*. ACM, 2008, pp. 239–252.
- [11] G. A. Jacoby, R. Marchany, and N. Davis, "Battery-based intrusion detection a first line of defense," in *Information Assurance Workshop. Proceedings from the Fifth Annual IEEE SMC*. IEEE, 2004.
- [12] J. Hoffmann, S. Neumann, and T. Holz, "Mobile malware detection based on energy fingerprints a dead end?" in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2013, pp. 348–368.
- [13] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 105–114.
- [14] T. Zefferer, P. Teufl, D. Derler, and et. all, "Towards secure mobile computing: Employing power-consumption information to detect malware on mobile devices," *Int. journal on advances in software*, vol. 7, 2014.
- [15] H. Yang and R. Tang, "Power consumption based android malware detection," *Journal of Electrical and Computer Engineering*, vol. 2016, 2016.
- [16] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: <https://science.sciencemag.org/content/313/5786/504>
- [17] T. Tagawa, Y. Tadokoro, and T. Yairi, "Structured denoising autoencoder for fault detection and analysis," in *Asian Conference on Machine Learning*, 2015, pp. 96–111.
- [18] E. Marchi, F. Vesperini, F. Weninger, F. Eyben, S. Squartini, and B. Schuller, "Non-linear prediction with lstm recurrent neural networks for acoustic novelty detection," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–7.
- [19] Y. Kawaguchi and T. Endo, "How can we detect anomalies from subsampled audio signals?" in *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2017, pp. 1–6.
- [20] G. E. Hinton, "Deep belief networks," *Scholarpedia*, vol. 4, no. 5, p. 5947, 2009, revision #91189.
- [21] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, pp. 721–741, 1984.
- [22] G. E. Hinton, "A practical guide to training restricted boltzmann machines," in *Neural Networks: Tricks of the Trade (2nd ed.)*, ser. Lecture Notes in Computer Science, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Springer, 2012, vol. 7700, pp. 599–619. [Online]. Available: <http://dblp.uni-trier.de/db/series/lncs/lncs7700.html#Hinton12>
- [23] M. Daoud and M. Mayo, "Using swarm optimization to enhance autoencoders images," *CoRR*, vol. abs/1807.03346, 2018. [Online]. Available: <http://arxiv.org/abs/1807.03346>
- [24] L. I. Smith, "A tutorial on principal components analysis," Tech. Rep., 2002.
- [25] R. M. A. Dawson, Z. Shen, D. A. Furst, S. Connor, J. Hsu, M. G. Kane, R. G. Stewart, A. Ipri, C. N. King, P. J. Green, R. T. Flegel, S. Pearson, W. A. Barrow, E. Dickey, K. Ping, C. W. Tang, S. V. Slyke, F. Chen, J. Shi, J. C. Sturm, and M. H. Lu, "Design of an improved pixel for a polysilicon active-matrix organic LED display," in *SID Tech. Dig.*, 1998, vol. 29, pp. 11–14.
- [26] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection," *NIPS*, vol. 12, pp. 582–588, 1999, cited By :41. [Online]. Available: www.scopus.com
- [27] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, 2007, pp. 153–160.
- [28] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.
- [29] R. James, A. Albasir, K. Naik, M. Dabbagh, P. Dash, M. Zaman, and N. Goel, "Detection of unknown applications in smartphones: A signal processing perspective," in *Canadian Conference on Electrical and Computer Engineering*, 2017, pp. 1–6.
- [30] C. Cortes and V. Vapnik, "Support-vector network-. machine learning 20: 273–297," *Portfolio Selection, Journal of Global Optimization*, vol. 43, no. 2-3, 1995.
- [31] T. Carneiro, R. V. Medeiros Da Nóbrega, T. Nepomuceno, G. Bian, V. H. C. De Albuquerque, and P. P. R. Filho, "Performance analysis of google colaboratory as a tool for accelerating deep learning applications," *IEEE Access*, vol. 6, pp. 61 677–61 685, 2018.
- [32] F. Chollet et al., "Keras," <https://keras.io>, 2015.