USING DEEP LEARNING TO CLASSIFY POWER CONSUMPTION SIGNALS OF WIRELESS DEVICES: AN APPLICATION TO CYBERSECURITY

A. Albasir, R. Soundar Raja James, K. Naik

University of Waterloo
Electrical & Computer Engg. Department
Ontario, Canada N2L 3G1

A. Nayak

University of Ottawa
School of Information Technology & Engg.
Ontario, Canada K1N 6N5

ABSTRACT

The problem of detecting malware in mobile devices is becoming increasingly important. While most of the mobile devices run on very limited resources, having anti-viruses installed on-board is not very practical, especially in IoT devices. Even if such tools exist, malware could hide or manipulate their fingerprint, making them not easy to detect. Thus, having effective countermeasures for after malware intrusion is paramount. In this work, we utilize deep learning ability to learn multiple levels of representations from raw data to classify power consumption signals obtained from smartphones. The objective is to build a framework that can intelligently tell if the smartphone has a malware or not by only monitoring its power consumption. Validation tests confirm that the proposed framework show that information contained in the measured power consumption of smartphones can in principle be used to identify malware existence and further can tell how active malware is with very high accuracy.

Index Terms— Deep Learning, Malware Detection, Signals Classification

1. INTRODUCTION

Todays' smartphones with their PC like computing capabilities are people's preferred means of performing many of their personal computing tasks. The applications range from performing online banking to health monitoring apps, to name a few. These facts make them the next big targets of malicious software and security attacks. However, mobile devices run on limited resources, which lead to the argument that detecting malware on-board is a nontrivial research challenge. To protect smartphones from these threats, detecting malicious behavior from the normal ones is a critical problem to be solved.

In literature, several studies focused on malware detection on smartphones. These approaches are either based on *static analysis* or *dynamic analysis*. In *static analysis* [1, 2, 3], the source code and manifest file of an app are analyzed without execution to check if it is malicious app or not. On the other hand, in *dynamic analysis*, an app is examined during runtime. Thus, these approaches aim to detect infected devices to minimize further damage, for example. In *dynamic analysis* approaches, the main idea is to monitor the device to capture certain information to be used for analysis. The monitoring process can be divided into *on-device* and *off-device* approaches. Also, the work done under *dynamic analysis* can be further classified based on the type of information being analyzed [4]: network based information [5], operating system (OS) based information [6], or power consumption based information. In this paper we focus on the approaches that use power consumption information to detect malware.

To this end, *on-device* monitoring based approaches have been used for detecting energy-greedy anomalies, mobile malware variants, and sensitive activities [7, 8, 9, 10, 11]. Apart from power signature analysis, machine learning methodologies have been explored in detecting suspicious activities [12, 13]. The work in reference [14] focuses on anti-malware tools adaptability for changing malware intrusion pattern. Similar research has been reported in reference [15] to show the effectiveness of a model to detect malware in different operation scenarios of mobile devices. Machine learning methods, such as Support Vector Machine (SVM), have been applied in detecting malware [16, 10, 17].

The solution strategy in this work relies on the hypothesis that every piece of software, whether malware or benign, will have a trace in the power consumption of the mobile device. This argument makes it inevitable for malware to go undetected having the right approach to process and analyze power signals. The used tool in the proposed detection methodology is Deep learning (DL) [18]. Due to its flexibility, versatility, and performance, DL has been applied to the analysis of data from diverse scientific disciplines, including time-series signals. DL is the result of training many layers of nonlinear processing units to extract features and model the data.

In this paper, we make the following contributions; (i) we propose a proof-of-concept of the idea that utilizes side channel information, represented by power consumption signals, and DL to detect malware on smartphones; (ii) we investigate the impact of the measurement's sampling frequency on the detection performance. To the best of our knowledge, we have

not come across any study that follows the same approach.

The remaining sections of the paper are organized as follows. Section 2 provides the background related to DNN. Section 3 sets the context of the addressed problem. Section 4 describes the dataset used in this work and the proposed methodology. Section 5 presents the results and some discussion. Finally, Section 6 lists some conclusions of this paper.

2. DEEP LEARNING

In this section we give an overview on Deep Neural Networks (DNNs). We discuss the architecture, the mathematical formulation, and the employed learning algorithms.

2.1. DNN Structure

In its generic form, DNN consists of many non-linear processing units, called neurons or perceptrons, arranged in multiple layers. It has three main layers: input layer; hidden layers; and the output layer, as shown in Fig. 1. The basic building block of a network is a neuron. In general, except for the input layer ($l = 0$), neurons are arranged into a network of neurons. Each column of neurons is called a layer, and one network can have multiple layers. The architecture of the neurons in the network is often called the network topology. A deep learning model can be constructed with different DNN architectures (e.g., Deep Belief Networks (DBN) [19], and Convolutional Neural Networks (CNN)) [20]). The notational representation of DNN that maps inputs to outputs is introduced in the following.

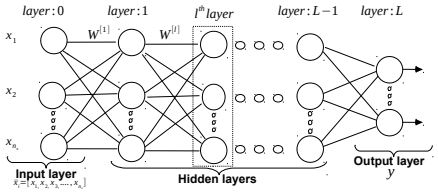

Fig. 1. Generic structure of fully connected DNN

Let X represent the input matrix of the dataset. $X = [\bar{x_1}, \bar{x_2}, \bar{x_3}, .., \bar{x_i}, .., \bar{x_m}]$, where $\bar{x_i}$ is a n_x dimensional column vector that represents the i^{th} training example, the size of X is $m \times n_x$. L is the number of layers in the network. y is the label (ground truth) vector of the dataset of size $m \times 1$, and y_i is the label of the i^th example. $\hat{y}$ represents the output vector of the network (the prediction). W^l is the weight matrix of the l^{th} layer, the size of W^l is (# of units in next layer * # of units in the previous layer). b^l is the bias vector in the l^{th} layer. For simplicity, we are dropping the bias term b in Fig. 1. Lastly, n_h^l is the number of hidden units in the l^{th} layer, and n_y is the number of hidden units in the output layer.

Each neuron has a set of inputs, each of which is given a specific weight W^l. The neuron, after it computes the sum of the weighted inputs ($z_l = \bar{a_l} * W^l$), applies activation function that introduces the non-linearity. To illustrate, in input layer ($layer\ l = 0$), $\bar{a_0} = \bar{x_i}$, and in the l^{th} layer, $\bar{a_l} = g(z_{l-1})$. Possible activation functions are: (i) Sigmoid $g(z_l) = \frac{1}{1+e^{-z_l}}$; (ii) Rectified Linear Unit (Relu) $g(z_l) = max(0, z_l)$; and (iii) Tanh function $g(z_l) = \frac{e^{z_l} - e^{-z_l}}{e^{z_l} + e^{-z_l}}$. The choice of the activation function in the output layer is strongly constrained by the type of problem that is modeled.

2.2. Operation of DNN

The intuition behind neural network idea can be explained as follows [21]: (i) by using multiple nonlinear processing units stacked in multiple layers and having enough data, enough computational powers, and enough training time, a network is able to learn on its own almost any complex function; (ii) once a DNN learn to represent a function with parameters (weights), through applying sophisticated optimization techniques (e.g. stochastic gradient decent (SGD)) on a loss function (Cross Entropy (X-E) and Mean Square Error (MSE) are widely used loss functions), optimal weights can be learned. In other words, DNN learns a parameterized function that represents the input data with multiple levels of abstraction. The learning procedure results in a complex function that can optimally represent the structure of the dataset.

To put the pieces together, Algorithm 1 shows the steps of Feed-forward and Back-propagation in pseudocode format for a better visualization. It starts with performing Feed-forward (line # 4). The mathematical equations used to compute the forward pass are as follows: (i) in layer #l: $z_l = W^l * \bar{a}_{l-1}$ and $\bar{a_l} = g(z_l)$. Both z_l and $\bar{a_l}$ are n_h^l dimensional column vectors. After iterating over all of the training examples, the output is the prediction vector $\hat{y}$ of size $m \times 1$.

Algorithm 1: Learning algorithm

Input : Dataset input & label matrices X & Y: $(\bar{x_i}, y_i),\ i \in \{1, ..., m\}$
Output: Optimal parameters {W,b}
1 Parameter Initialization {W,b};
2 **while** $epoch < N_{epochs}$ **do**
3 **for** *each input* $\bar{x_i}$ *in* X **do**
   `/* Feed-forward */`
4 Feed-forward $\bar{x_i}$ through all layers
5 $z_l = W^l \cdot \bar{x_i} + \bar{b} \rightarrow \bar{a_l} = g(z_l)$
6 **if** *l = L (Output layer)* **then**
7 $\hat{y_i} = \bar{a_l}$
8 **end**
9 **end**
   `/* Back-propagation */`
10 Propagate error back to adjust the weights
11 Evaluate $C_{MSE}(W, b) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y_i})^2$
12 Minimize $C_{MSE}(W, b)$ & Update Parameters
13 **end**

In line # 11, the error is propagated back through the chain

of rule of derivatives. The starting point is the output layer $l = L$, where the error is computed by evaluating the loss function. The error derivative with respect to the output of a unit is computed by differentiating the loss function (also known as cost function). As explained in Algorithm 1, if the cost function is MSE, then this gives $(y - \hat{y})$. Once the $\frac{\partial C_{MSE}}{\partial W^L}$ is known, the weight W^{L-1} can be updated. Using the chain of rule of derivatives, the wight matrix of the layers before get updated accordingly till $layer\ l = 1$ is reached.

3. PROBLEM DESCRIPTION

This article focuses on the problem of detecting malware running on smartphones through investigating only their power consumption measurements without using any other sort of information from the smartphone OS. This approach is challenging due to the fact that the measured power consumption is the total power consumed by the smartphone, and thus some of the smartphone hardware components exhibit nonlinear energy consumption characteristics. This means if the malware code is utilizing specific piece of hardware and at the same time a benign application is also using it, the power consumption may not be linearly increasing or decreasing, at least if the measurement sampling rate is not high enough to capture such an event. The nonlinearity behavior is also characterized by tail energy concept. This means that a specific piece of hardware stays powered on for a short period of time after it does its job. For instance, the 3G wireless network interface stays on for some time after the smartphone has received/transmitted all the data [22]. Therefore, the main question that this paper aims to answer is that, by having a high power consumption sampling rate, can we utilize smartphone power consumption to detect interesting events?

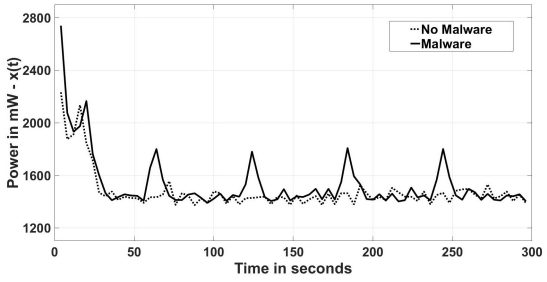

Fig. 2. Two power traces of a smarphone

4. DATASET AND SOLUTION STRATEGY

In this section, we discuses the proposed methodology and dataset used to evaluate it.

4.1. Dataset

The dataset consists of raw power consumption signals of Android smartphone that were collected using Monsoon Power Monitor [23] with maximum sampling rate $f_s = 5000$ Hz (for more details on test bench, refer to [17]). The main experiment was, using WiFi connection, to stream for $T = 300$ seconds of Youtube video. Based on the sampling frequency, this comes to 1,500,000 samples per trace. This experiment was conducted under a very controlled execution environment and repeated under 7 different scenarios. In the first scenario, it was ensured that no other apps are running in the background other than Youtube. This scenario is called *No Malware* class. In the other six scenarios, an *emulated non-adaptive malware code* was running in the background on the smartphone while streaming the same Youtube video (the power consumption signal of an extreme case where the malware is highly active (Dut = 12%) is shown in Fig. 2 - solid line). The emulated malware has a tunable activity cycle, called "malware activity duty cycle" (Dut), which represents the percentage of time the malware is active. When the malware is active in the background, the task it performs is downloading some content from a remote server. These six scenarios are: *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. In each of the 7 scenarios, we repeated the experiment 15 times. Therefore, in total the dataset has 105 power traces.

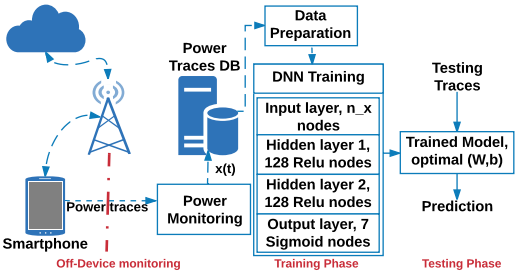

Fig. 3. Deep learning model for malware detection

4.2. Method

Motivated by the fact that DNN has the capabilities to work on raw data, we came to the idea of applying DL directly to our dataset, allowing DL to learn the embedded information features by itself. In a previous work [17], we hand-crafted features from raw data and applied SVM to classify signals. The performance was not very good and suffered from high false negatives. The reason behind the bad detection rate is that by using the mean and standard deviation as extracted features, we tend to lose much discriminative information.

In this work, however, we treat these power consumption traces as signals that carry information about the operational state of the smartphone. Our methodology has been illustrated in Fig. 3, where we start with samples preparation. Then we train DNN with 3 layers to classify power signals.

Data Preparation: A key aspect of our method is data preparation. In our dataset, each power trace $x(t)$, is a vector of size $T * f_s$ samples, as we explained in Section 4.1. We introduce the concept of the "labeling window", where the size of this window represents the number of selected samples n_x. For example, if $n_x = 5000$, this means the window

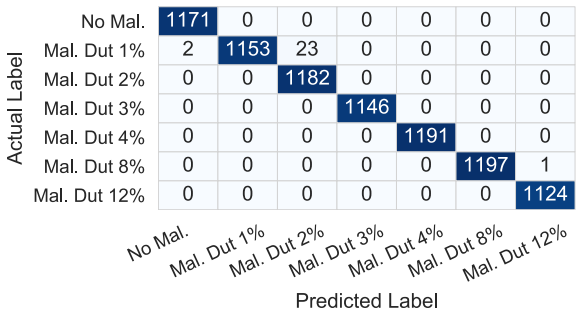

Fig. 4. Confusion matrix for $f_s = 5000$ and $n_x = 5000$

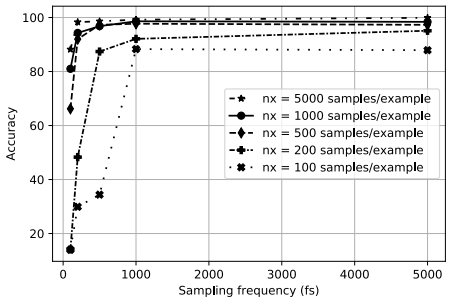

Fig. 5. Detection accuracy vs sampling rate

has 5000 samples which is equivalent to 1 second of the experiment time. In this case, the task involves labeling each 1 sec of each power trace with same label of the original signal. For example a power trace $x(t)$ with the label *No Malware* ($y = 0$), generates $x(t)/n_x = 1,500,000/5000 = 300$ training example $\bar{x}$ labeled with the same original label ($y = 0$). In total, in this example we get from 105 power traces in our dataset $105 * 300 = 31,500$ training examples.

Model: Our DNN architecture model contains three layers (L = 3). The input layer of size n_x followed by two hidden layers, each has n_h neurons. The output of the these two layers is fed to a classier layer, which is simply a fully connected (affine) softmax output layer of a size n_y (the number of classes), as illustrated in Fig. 3. The softmax layer produces a distribution over the 7 classes in our problem.

5. EXPERIMENTS AND RESULTS

The problem is formulated as a multi-class classification problem, where we define $\bar{x} \in X$ as input training example vector, and $y \in \{0, 1, 2, 3, 4, 8, 12\}$ represents a possible label. $y = 0$ means class *No malware*, $y = 1$ means class *malware with Dut = 1%*, and so on. In the context of our application, the above notation translates into the following: The input sample $\bar{x_i}$ is a n_x time series samples of power consumption measurement, and y_i is the corresponding label. Given $\bar{x_i}$, the model predicts $\hat{y_i}$ which tells whether the smartphone has a malware or not. If it does have malware, it can further identify the malware activity.

Experimental Setup: All of the experiments were performed using Tensorflow and Keras [24] Python libraries. The two hidden layers are identical with 128 neurons and Relu non-linear activation function in each. The output layer, that is the classifier, has 7 neurons with Sigmoid non-linear activation function to produce likelihood over the 7 classes of labels. The used loss function is Cross-Entropy and the used optimizer in the learning/training phase is Adam [25], which is a method for stochastic optimization. We evaluate our model empirically on our dataset described in Section 4.1. For a better model generalization and to maintain similar classes' distribution in the training and testing dataset, we use stratified sampling algorithm to randomly split the dataset into 70% for training and the remaining 30% for testing. As we explained in Section 4.2, the parameter that we will be varying is the size of "labeling window", n_x, i.e. we investigate how the size of the "labeling window" impacts the overall detection performance. We also investigate the impact of the sampling rate (f_s) on the detection performance.

Classification Results: We use accuracy as the performance metric. The detection performance for the case of "labeling window" $n_x = 1000$ & $f_s = 5000$ is shown by the confusion matrix in Fig. 4. These results shows that the model was not only able to detect malware with high accuracy ($\tilde{9}9.98\%$), but also was able to differentiate the degree of activity of each malware, with a very low False negatives (FN). A low FN rate is a useful measure for malware detection techniques since we have more cost when anomalous behavior is present but does not get detected. This shows a good potential of using smartphone power consumption signals to detect more insights about the operational state of the devices.

Figure 5 summaries our results and shows the accuracy for different sampling frequency rates. The accuracy is higher than 90% for all of the cases where the sampling rate is higher than 1000 samples per second. The takeaway from these results is that, the higher the sampling rate, the more is the discriminative information captured by the model. We also have tested our model on traces that was not included in the dataset, traces that the model has never been exposed to. The obtained accuracy is 90% and the FN rate is 5%. Results shows that our model generalizes well on unseen test examples as well.

6. CONCLUSION

This paper introduces a new proof-of-concept smartphone malware detection technique that is based on deep learning and the power consumption signals of the smartphone. The measured power readings were treated as signals and used as raw data to train a deep learning model. The results were promising and show a potential of applying this technique to detect more insights about the smartphone operational state (e.g. hardware failure). As a next step, we will validate our approach using real malwares and including other apps to build more realistic scenarios.

7. REFERENCES

[1] William Enck, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX*, 2010, OSDI'10.

[2] I. Bulut and A. G. Yavuz, "Mobile malware detection using deep neural network," in *2017 25th Signal Processing and Communications Applications Conference (SIU)*, May 2017, pp. 1–4.

[3] Karim O Elish, Danfeng Yao, and Barbara G Ryder, "User-centric dependence analysis for identifying malicious mobile apps," in *Workshop on Mobile Security Technologies*, 2012.

[4] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 6:1–6:42, Mar. 2008.

[5] Shun Tobiyama, Yukiko Yamaguchi, Hajime Shimada, and et. al., "Malware detection with deep neural network using process behavior," in *the 40th Annual IEEE Conference (COMPSAC)*. IEEE, 2016.

[6] Ahmet İlhan Ayşan and Sevil Şen, "Api call and permission based mobile malware detection," in *Signal Processing and Communications Applications Conference, 2015 23th*. IEEE, 2015, pp. 2400–2403.

[7] Muhamed Halilovic and Abdulhamit Subasi, "Intrusion detection on smartphones," *CoRR*, 2012.

[8] Grant A Jacoby, Randy Marchany, and NathanielJ Davis, "Battery-based intrusion detection a first line of defense," in *Information Assurance Workshop. Proceedings from the Fifth Annual IEEE SMC*. IEEE, 2004.

[9] Hahnsang Kim, Joshua Smith, and Kang G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proceedings of the 6th International Conference MobiSys '08*, 2008.

[10] Mordechai Guri, Boris Zadov, Eran Atias, and Yuval Elovici, "Led-it-go: Leaking (a lot of) data from air-gapped computers via the (small) hard drive LED," *CoRR*, 2017.

[11] T. K. Buennemeyer, "Mobile device profiling and intrusion detection using smart batteries," in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, Jan 2008, pp. 296–296.

[12] S. Papadopoulos, A. Drosou, and D. Tzovaras, "A novel graph-based descriptor for the detection of billing-related anomalies in cellular mobile networks," *IEEE Transactions on Mobile Computing*, 2016.

[13] S. Kumar, A. Viinikainen, and T. Hamalainen, "Machine learning classification model for network based intrusion detection system," in *11th Int. Conference for Internet Technology and Secured Trans. (ICITST)*, 2016.

[14] Y. Xue, G. Meng, Y. Liu, T. H. Tan, H. Chen, J. Sun, and J. Zhang, "Auditing anti-malware tools by evolving android malware and dynamic loading technique," *IEEE Transactions on Info. Forensics and Security*, 2017.

[15] H. Kim, K. G. Shin, and P. Pillai, "Modelz: Monitoring, detection, and analysis of energy-greedy anomalies in mobile handsets," *IEEE Trans. on Mobile Computing*, vol. 10, no. 7, pp. 968–981, 2011.

[16] Hyo-Sik Ham, Hwan-Hee Kim, Myung-Sup Kim, and Mi-Jung Choi, "Linear svm-based android malware detection for reliable iot services," *Journal of Applied Mathematics*, vol. 2014, 2014.

[17] R. Soundar Raja James, A. Albasir, K. Naik, and et. al., "Detection of anomalous behavior of smartphones using signal processing and machine learning techniques," in *2017 IEEE 28th Annual International (PIMRC)*, 2017.

[18] John Cristian Borges Gamboa, "Deep learning for time-series analysis," *arXiv:1701.01887*, 2017.

[19] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, 2006.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, 2015.

[22] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference IMC '09*. 2009, ACM.

[23] "Monsoon power monitor," = https://www.msoon.com/LabEquipment/PowerMonitor/.

[24] François Chollet et al., "Keras," 2015.

[25] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.