Robust Malware Defense in Industrial IoT Applications using Machine Learning with Selective Adversarial Samples

Mahbub E Khoda, Tasadduq Imam, Joarder Kamruzzaman, Iqbal Gondal, and Ashfaqur Rahman

***Abstract*—Industrial Internet of Things (IIoT) deploys edge devices to act as intermediaries between sensors and actuators and application servers or cloud services. Machine learning models have been widely used to thwart malware attacks in such edge devices. However, these models are vulnerable to *adversarial attacks* where attackers craft *adversarial samples* by introducing small perturbations to malware samples to fool a classifier to misclassify them as benign applications. Literature on deep learning networks proposes *adversarial retraining* as a defense mechanism where adversarial samples are combined with legitimate samples to retrain the classifier. However, existing works select such adversarial samples in a random fashion which degrades the classifier's performance. This work proposes two novel approaches for selecting adversarial samples to retrain a classifier. One, based on the distance from malware cluster center, and the other, based on a probability measure derived from a kernel based learning (KBL). Our experiments show that both of our sample selection methods outperform the random selection method and the KBL selection method improves detection accuracy by 6%. Also, while existing works focus on deep neural networks with respect to adversarial retraining, we additionally assess the impact of such adversarial samples on other classifiers and our proposed selective adversarial retraining approaches show similar performance improvement for these classifiers as well. The outcomes from the study can assist in designing robust security systems for IIoT applications.**

I. Introduction

Industrial Internet of Things (IIoT), driven by Industry 4.0, refers to the use and control of smart sensors and actuators to enhance manufacturing, data analysis and decision-making process in an industrial setting [1]. Various industries including security surveillance, manufacturing, agriculture, and food processing have adopted IIoT in recent years [2]. Connected sensors, actuators, and controllers enable industries to effectively spot inefficiencies and detect operational anomalies and make intelligent business decisions by gathering a large volume of data from heterogeneous sensors and analyzing them, often in a remote server hosted in the cloud. However, these sensors have limited power and communication capacity. As a result, edge devices including small servers, desktops, laptops, routers, smartphones, and hand-held devices are used as intermediaries between the sensors and the cloud servers. These devices can collect data from sensors and temporarily store them for initial pre-processing before sending them to the local server or remote cloud.

However, such edge devices can be a point of exploitation for malware attackers. A compromised device can transmit false or spoofed information to the cloud server or lead to inaccurate assessment of data or pose threats to the security of sensitive industrial documents, business strategy, and corporate information. This may lead to major financial and reputational loss and operational inefficiencies, in addition to harming their customers. Figure 1 shows an Industrial IoT setting where sensors, actuators, controllers and other equipment communicate with edge devices that, in turn, upload data to the remote server. The server, on analyzing the data, may send instructions and control messages back to controllers and actuators through the edge devices for optimum operation of the industry. The figure also shows how malware attacks can compromise edge devices leading to critical information leak and financial loss.

Ensuring cyber security is one of the major challenges faced in an industrial IoT setting. This includes protecting edge devices from malware attacks, preventing their unauthorized access, and ensuring physical and communication privacy. Various works in the literature proposed automated techniques based on machine learning models, including deep neural networks, to ensure IIoT security regarding malware detection, fault diagnosis and anomaly detection [3]–[6]. These models, however, have also been shown to be vulnerable against adversarial attacks in the detection of image forgery, computer vision manipulation and computer malware [7]–[10]. Adversarial attacks are based on samples that are crafted by introducing small perturbation into original samples (e.g., benign computer applications, macros, scripts) so that the detection system is fooled and consequently failed to classify a crafted sample as a malicious one. [11].

After Szegedy et al. [11] demonstrated that several machine learning models, including neural networks, are vulnerable to adversarial samples, several works proposed different methods on crafting adversarial sample as well as devising defense mechanisms against it. A fast gradient sign method was introduced by Goodfellow et al. [7] that found perturbation to be added by differentiating the cost function with respect to the input. An adversarial sample crafting method based on the Jacobian matrix was proposed by Papernot et al. [9] that computes the output's sensitivity to each input feature to find the most influencing feature. Accordingly, the method iteratively modifies features so that the sample is misclassi-

M. Khoda, J. Kamruzzaman and I. Gondal are with the Internet Commerce Security Laboratory, Federation University Australia. e-mail: {m.khoda, joarder.kamruzzaman, iqbal.gondal}@federation.edu.au.

T. Imam is with School of Business and Law, CQUniversity Australia. e-mail: t.imam@cqu.edu.au.

A. Rahman is with Data61, CSIRO, Australia. e-mail: ashfaqur.rahman @data61.csiro.au.

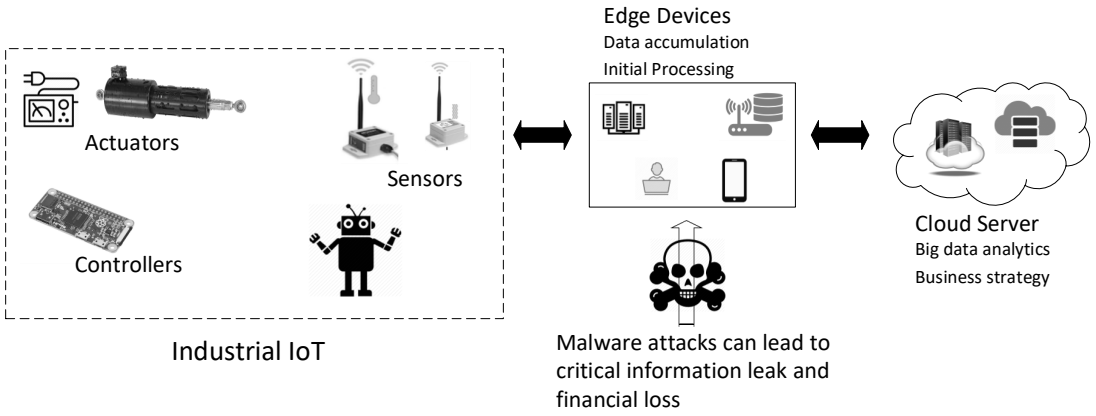


Fig. 1: Malware Attack in Industrial IoT

fied. To increases resilience of the detection system against such samples, researchers have proposed adversarial retraining method that involves first crafting the adversarial samples and then mixing these samples with the legitimate ones to retrain the classifier [11]–[13]. Even though adversarial attacks and defense against them in image processing and computer vision domain have been studied well, such works in the malware detection domain are limited and inadequate. On the other hand, it has been shown that retraining a classifier with too many adversarial samples can degrade the performance of a classifier [10], completely defeating the purpose of retraining. Hence, it is of utmost importance to carefully select the most appropriate samples to retrain a classifier since choosing the same number of samples differently can lead to a difference in classifier performance. However, in the existing works, these samples are chosen in a randomized fashion which may not necessarily provide the best performance.

To address this problem, in our work, we have explored adversarial learning in the malware domain in the IIoT setting, especially focusing on an intelligent selection of adversarial samples for retraining. We proposed two novel approaches for selecting adversarial samples to retrain a classifier. Firstly, we proposed a method to select adversarial samples based on their distance from the cluster center of malware and to compute this distance we experimented with different distance measures. Secondly, we selected adversarial samples based on the probability derived from a kernel based learning (KBL). Experiments with deep neural networks show that both of our selective strategies perform better than randomized selection. Though a preliminary version of the approach was proposed in [14], it was tested with only one distance measure and the characteristics of the approach was not adequately explored; while in this paper, multiple distance measures have been tested with extensive study of various characteristics of the approach.

In addition, unlike most existing works including [14] that focus mainly on deep neural networks with regard to adversarial retraining, in this work, we assessed the impact of such adversarial samples on other classifiers. Our selective adversarial training strategy showed comparable performance improvement for these classifiers as well, demonstrating the efficacy of our selection strategy across classifiers.

Our work in this paper, hence, makes the following contributions:

- We propose and evaluate two different approaches for appropriate selection of adversarial samples for retraining. Our approaches of selecting samples make the detection system far more resilient than the currently used random selection, yielding a 6% increase in detection accuracy by the KBL approach.
- We further explore the selective strategies and adversarial retraining with other well-known classifiers and show that our strategies increase the robustness of malware detection irrespective of the classifier used.
- The proposed approach will be highly useful to increase the robustness of machine learning in other industrial applications as well.

The rest of the paper is organized as follows: we first provide a summarized background of malware detection in IIoT, adversarial examples and adversarial learning in Sec. II. We then describe the methodology of our work, our adopted adversarial sample crafting technique and our proposed methods for selecting samples for adversarial retraining in Sec. III. Finally, we present the performance evaluation in Sec. IV and conclude in Sec. V.

II. Background and Related Works

In this section we briefly discuss the works in malware detection in IIoT, the background of adversarial sample crafting and adversarial learning in malware detection.

A. Malware Detection in Industrial IoT

In recent works, a lot of efforts have been made to ensure the security of IoT devices ranging from building secured IoT architectures to designing detection and prevention mechanisms of malicious IoT applications [15]–[17]. Jerald et al. [16] proposed a security architecture for integrated IoT smart services. Sun et al. [17] proposed a cloud-based anti-malware system for resource-constrained IoT devices including a scanning agent for the clients. However, in recent years handheld devices with Android and iOS systems have

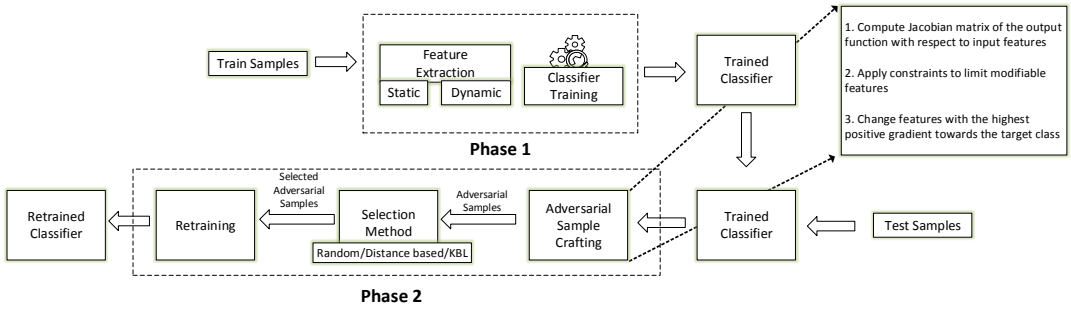


Fig. 2: Work flow of selective adversarial retraining

become an integral part of IIoT systems. These types of handheld devices including specialized mobile devices are increasingly being commissioned by industry technicians for inspection, maintenance, alarm generation, work allocations, and scheduling as well as data collection. As a result, many recent works have been focused on ensuring security on these devices. However, the recent large growth in IIoT systems and the number of applications deployed for those devices in the market have led manual inspection to be infeasible. As a result, automated malware detection techniques have been proposed in the literature based on various machine learning tools. These tools depend largely on the features extracted from applications to classify malicious and benign software. Consequently, several works studied different types of features for malware detection.

The features of such applications can be of two types: *static* and *dynamic*. Static features are extracted by analyzing the application code and related files without actually running the code. On the other hand, extracting dynamic features requires the application to run, often in a controlled environment (i.e., emulator), while its runtime behaviour is logged. Several earlier works proposed malware detection based on traditional machine learning tools leveraging simple static features (e.g., requested permissions) [18], [19]. Later works considered various other features including application programming interface (API) calls, inter-component communication (ICC) features and dynamic features to detect more sophisticated malware [20]–[23]. Several works proposed hybrid approaches where both static and dynamic features were combined to detect malicious applications [24]–[26]. Sharmin et al. [6] inspected the performance of different machine learning classification models based on static, dynamic and hybrid features of mobile apps in industrial IoT setting. However, they did not consider the vulnerability of those models against adversarial attacks.

Researchers have become increasingly interested in malware detection using deep neural networks due to promising advancement in this field in the recent past [25], [27]–[29]. However, deep neural networks have been shown to be vulnerable to adversarial samples in the computer vision domain [11]. Subsequently, the possibility of the same is explored by researchers in malware detection domain as well [10]. Hence, it is very important to increase the robustness of the detection system against these adversarial attacks.

B. Adversarial Examples

To craft adversarial samples, a small perturbation is introduced into original ones so that the classifier is misled to classify them as an incorrect class. Adversarial sample crafting can be just to mislead the classifier and simply cause misclassification in general or it can be targeted where the misclassification is biased towards a specific class. A detailed taxonomy of adversarial example crafting goals and techniques can be found in [9]. If a legitimate sample X classified by the model is represented as $M(X) = y$, an adversarial example X' is a modified version of X that is misclassified as $M(X') = y'$ where $y' \neq y$. Formally,

$$\underset{\delta_x}{\operatorname{argmin}} M(X + \delta_x) = \{y' \mid y' \neq y\} \tag{1}$$

where δ_x is the perturbation introduced to the sample X. By setting the condition to $y' = y_t$ this can be converted to targeted attack where y_t is the intended target class. However, in case of only detecting where an application is malicious or not, this effectively becomes the same since we are mostly interested in two classes (i.e., malware and benign) and therefore, crafting adversarial samples from original malware automatically means the target is benign. This means through crafting adversarial samples, attackers aim to create a minimally perturbed version of a malware so that it can be classified as benign and thereby evade detection.

However, finding a solution to this equation (Eq. (1)) is very hard due to the complex nature of the deep neural network functions. In this respect, researchers have explored two types of attack scenarios: black-box and white-box attack. In black-box attack scenario, the adversarial attacker does not have any detailed knowledge about the target; instead it can only query the model for the output. The attacker in this case is at its weakest especially when it can only get the output label from the model rather than the class probability. On the other hand, in white-box scenario the attacker has all the available information i.e., the model architecture, values of the variables and the parameters, and the training data. Consequently, making a model robust to the white-box scenario means the model can defend against the strongest attacks. Most of the approaches in the literature are white-box in nature and the attacker has the full knowledge of the network architecture and its parameters [7], [9]–[11]. Hence, in our work we adopted a white-box attack scenario.

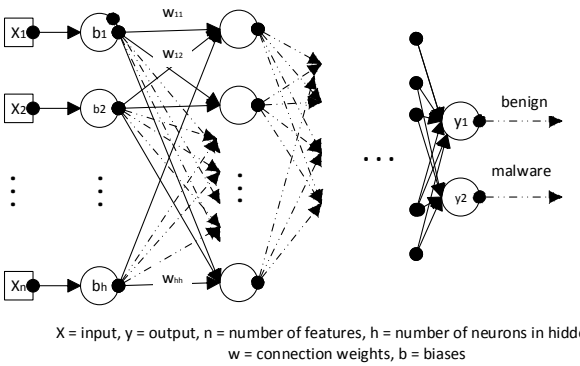


Fig. 3: Deep neural network

The details of the particular methods we adopted are described in Sec. III-B.

C. Adversarial Malware Crafting and Defense

There are only limited works in literature that have reported adversarial example crafting techniques and their impact on the detection accuracy of malware, while even less attention has been focused on adversarial malware attack in industrial IoT setting. Grosse et al. [10] studied adversarial example crafting and adversarial retraining as a defense mechanism for android malware detection. The work showed improved performance with adversarial retraining while considering several networks trained with varying ratios of malware and benign samples. The work also reported that retraining the classifier with too many adversarial samples can degrade classification performance due to overfitting. This observation is also verified by our experiments. A reinforcement learning based approach was adopted by Anderson et al. [30] for crafting adversarial examples so that detection of malware in Microsoft Windows-based portable systems can be evaded. Their method tried to evade a detection model by modifying a malware file iteratively until it was misclassified by the target model. Interestingly, in the black box attack, the model achieved a better evasion rate than in a more informed white box attack scenario. Dang et al. [31] also explored crafting adversarial examples for malware in a black-box scenario. They proposed a morphing technique to craft adversarial malware based on a hill-climbing approach where the number of morphing steps was leveraged as a scoring mechanism. The work reported a 100% evasion rate of the adversarial malware crafted by their technique. A malware recomposition variation approach was proposed by Yang et al. [32] for crafting adversarial malware samples. Through semantic-feature mutation and phylogenetic analysis on different malware families, the work performed program transplantation to automatically mutate malware bytecode to generate new malware variants. Unlike adversarial retraining, Papernot et al. [33] proposed a different defense mechanism based on distillation. In this method, a classification model M is first trained using the original data and the probability of the samples to belong to certain classes are recorded. A second classification model M' is then trained using these probabilities as the label. However, in [10] the authors showed that this mechanism is not as effective in malware detection as it was in computer vision. Other defenses against adversarial attacks include classifying adversarial samples as a separate class and differentiating them based on their statistical property [34], [35].

Unlike image processing and computer vision domain, the effectiveness of adversarial malware attack and its defense in industrial IoT settings is a relatively unexplored area. Furthermore, intelligent selection of samples has shown improved performance in different areas of machine learning applications such as dealing with imbalanced data [36], [37]. However, such selective sampling is yet to be applied for adversarial retraining in industrial IoT malware detection. The research presented in this work fills this gap.

III. Methodology

In this section, we describe the methodology of our work. Figure 2 shows the workflow of adversarial retraining using our selective samples strategy. The process of adversarial sample crafting and retraining is divided into two main phases. In the first phase, we train a classifier with clean data, ie., with the original dataset without adversarial samples included. This phase includes extracting features from the samples, representing them as feature vectors and training the classifier. In the second phase, test samples are used to craft adversarial samples from the previously trained classifier. Afterwards, a subset of adversarial samples is selected and combined with the original samples to retrain the same classifier. For this purpose, we propose two mechanisms to select adversarial samples. In the subsequent sections we describe the feature extraction process, the classifier we used (deep neural network), adversarial sample crafting process and adversarial sample selection methods in detail.

A. Feature Extraction and Representation

As detailed in Sec. II-A, features of an application can be static or dynamic. Several works showed that combining both static and dynamic features (known as the hybrid approach) leads to the best performance regarding malware detection [24], [27]. Hence, in our work, we extracted both static and dynamic features for malware detection. After the features are extracted, each application is represented by a binary vector,

$X \in \{0,1\}^n$, where n is the number of features and $X_i = 1$ indicates that feature i is present in the application and $X_i = 0$ indicates that the feature is absent.

B. Crafting Adversarial Malware Samples

Even though the technique of crafting adversarial samples described in Sec. III-B can be applied to any differentiable classification function, the most common application of this is with deep neural networks (DNN) [7], [10], [11]. In our work also we adopted DNN to craft the adversarial samples. Note that, once adversarial samples are crafted using DNN, the extended dataset containing these samples and the original ones are used to train and test multiple classifiers to investigate the effectiveness of our selective strategies in adversarial learning. For this reason, below we present a brief discussion about the deep neural networks.

A neural network is a machine learning architecture that is structured with layers of neurons which are the primary computing units of the network. Multiple hidden layers between the output and input layers make a neural network deep which allows for more complex relationships among the features to be modeled by the network.

Figure 3 exemplifies a simple architecture of a deep neural network. The input layer takes the feature vectors as input. Each subsequent layer of the network takes input from the previous layer and transforms it into some abstract representation which is produced as the output for the next layer. The network hierarchically extracts the complex relationship among the features through the learning process. Formally a deep neural network model M can be represented as the composition of multi-dimensional and parametric functions m_i (corresponding to each layer i) that maps each input to a particular output (e.g., class label):

$$M : \boldsymbol{x} \longmapsto m_n(...m_2(m_1(\boldsymbol{x}, \boldsymbol{\theta_1}), \boldsymbol{\theta_2})..., \boldsymbol{\theta_n}) \tag{2}$$

where $\boldsymbol{x}$ is the input to the model, $\boldsymbol{\theta_i}$'s are the parameters for each layer i, i.e., the weights connecting layer i and $i-1$ (w's) and the biases of each neuron in layer i (b's). These weights and biases are learned during the training process.

The goal of crafting adversarial samples in malware detection is to craft samples from original malware so that the classifier outputs them as benign applications. To achieve this we start with a *malware* sample $X \in \{0,1\}^n$ and note the prediction outcomes using the deep neural network. Using two neurons to represent two classes, the final layer of the trained DNN model M then outputs two values $M(X) = [M_0(X), M_1(X)]$, indicating the probability of X being a benign application or a malware respectively. As the prediction, we choose the class that has the highest probability. For crafting adversarial example, we now want to find a small perturbation δ_x so that the output $M(X+\delta_x)$ is different from the original prediction and matches the attacker's goal. This can be done by solving Eq. (1) described in Sec. II-B.

However, the complex functions learned by neural networks are generally non-linear and non-convex. As a result, finding a solution to this problem is very hard. Hence, researchers have proposed heuristic solutions based on forward gradient [7] and using saliency map based on Jacobian matrix [9] to find perturbations.

In our work, we adopt adversarial sample crafting based on the Jacobian matrix since it is more suitable for binary features [9]. The goal here is to craft adversarial malware samples so that they are classified as benign applications. The Jacobian matrix is defined as

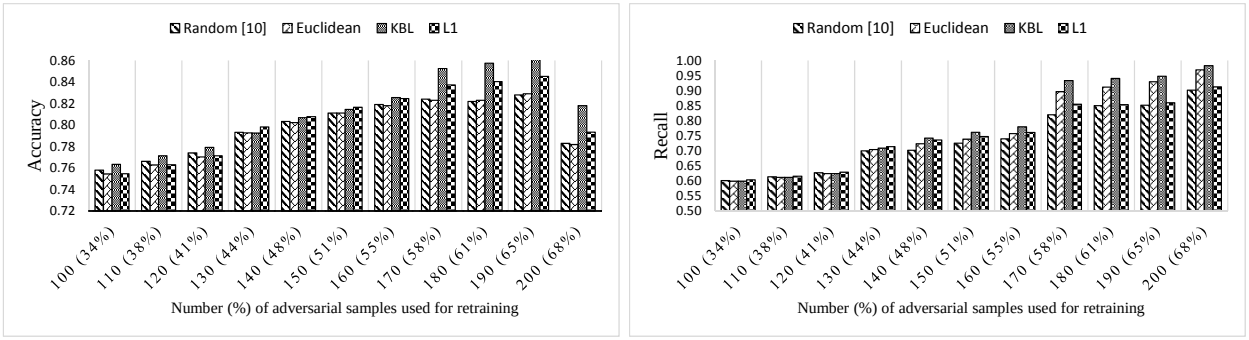
$$J_M = \frac{\nabla M(X)}{\nabla X} = \begin{bmatrix} \frac{\nabla M_0(X)}{\nabla X_0} & \cdots & \frac{\nabla M_0(X)}{\nabla X_n} \\ \frac{\nabla M_1(X)}{\nabla X_0} & \cdots & \frac{\nabla M_1(X)}{\nabla X_n} \end{bmatrix} \tag{3}$$

here ∇ denotes the gradient of a function. After computing the Jacobian matrix, i.e., the derivatives of the cost function with respect to the input, we modify the feature that is most influential towards our target class. Since our target class is 0 (i.e., to fool the classifier to misclassify the adversarial sample as begin) we find the feature that has the highest value in $\nabla M_0(X)$ for modification. This process is iterated until the malicious application is misclassified as benign.

However, modifying the value of such features is not as straightforward. For example, the method was originally proposed for image processing where several constraints were imposed so that the modification of pixel values does not visually distort the image. An example of such a constraint can be allowing only a small change in pixel value i.e., keeping the change under a certain threshold. However, unlike image processing, where the values of the image pixels are continuous, we only have 0 and 1 as feature values in malware detection. As a result, thresholding for feature values cannot be applied here. Furthermore, an attacker will make sure that the malicious functionality of a malicious application is not hampered while adversarial samples are crafted. For this, we imposed a different set of constraints for crafting adversarial malware samples. These are described below.

Constraints for Adversarial Malware Crafting: As crafted malware need to preserve their malicious functionality, the features that could potentially hamper the maliciousness of an application cannot be modified. Taking the malicious behaviour of a broad range of malware into account, we impose the following specific constraints on adversarial malware crafting to preserve its malicious functionality.

- Features are only allowed to be added but not to be removed since removing a feature could potentially break down the corresponding code execution. This means we only allow modification to the features that have value 0.
- Most applications have a resource file associated with it that lists information such as application metadata, other used resources, and used packages (e.g. manifest.xml file in Android apps). We allow the addition of features in such files since adding features in this way does not necessitate modification of the original code, hence original functionality of the code can be preserved.
- Modification of dynamic (i.e., system call) features is not allowed. This is because adding a dynamic feature (e.g., system call) requires modifying the code for executing the corresponding function which in turn may hamper the original functionality of the malware.



(a) Accuracy (b) Recall

Fig. 4: a) Accuracy and b) recall values of deep neural network after adversarial retraining when adversarial samples are selected randomly or selectively based on the Euclidean distance, L1 norm or KBL

To implement the above constraints we first define a vector $\nu \in \{0,1\}^n$, where $\nu_i = 1$ if feature i is allowed to be modified and $\nu_i = 0$ otherwise. After computing the derivatives from Eq. (3) we obtain the final score using the following equation

$$S(X) = \nabla M_0(X) \times neg(X) \times \nu \tag{4}$$

where $neg(X)$ is a negation function on vector X i.e., for each index it changes the value from 0 to 1 and vice versa. After this we choose the index i for modification using the following equation.

$$i = \underset{j}{\operatorname{argmax}}\, S(X_j) \tag{5}$$

C. Proposed Selection Methods for Adversarial Retraining

In the following, we propose two methods for selecting adversarial samples, one based on the distance from malware cluster center and another based on the probability derived from a kernel based learning.

1) Method 1: Based on distance from cluster center: In this method adversarial samples are selected by computing the distance of these sample from the malware cluster center. For this, first a distance score, $S_d(X)$, is calculated for each adversarial samples X using the following equation:

$$S_d(X) = \min_{\forall k \in \mathbb{K}} Dist_k(X) \tag{6}$$

where $\mathbb{K}$ is the set of malware clusters, and $Dist_k(X)$ is a distance measure between sample X and cluster center k. Note that malware samples may form multiple clusters depending on the types of malware in the dataset. Any clustering algorithm, e.g., K-means clustering, can be used for this purpose. To investigate the impact of distance measure types on the proposed selection strategy, we experimented with various distance measures, namely, the Euclidean distance, Hamming distance, and L1 norm. Thereafter, the samples were sorted based on the distance score and the first n number of samples with the least score were selected for retraining. Intuitively, these are the samples that should have been correctly classified since they are closer to the cluster center, yet they were able to fool the classifier.

2) Method 2: Based on probability derived from kernel based learning (KBL): In this method, adversarial samples are selected based on the probability derived from a kernel based learning. We implement this by first training a support vector machine (SVM) kernel on the dataset that is free of any adversarial sample. The decision function $f(x)$ is computed such that $sign(f(x))$ is used to predict the label of a sample. Afterwards, in order to compute the class probability $Pr(y = 1|x)$, i.e., the probability that a sample is a malware, we consider the following approximation based on Platt [38]

$$Pr(y = 1|x) \approx P_{A,B}(f) \equiv \frac{1}{1 + exp(Af + B)} \tag{7}$$

where $f = f(x)$. Assuming f_i is an estimate of $f(x_i)$ the best value of A, B is determined by minimizing the negative log likelihood of the training data:

$$\min_{A,B} \quad -\left[\sum_i t_i log(p_i) + (1 - t_i) log(1 - p_i)\right] \tag{8}$$

where $p_i = P_{A,B}(f_i)$ (Eq. (7)) and t_i's are the target label for the optimization problem defined as:

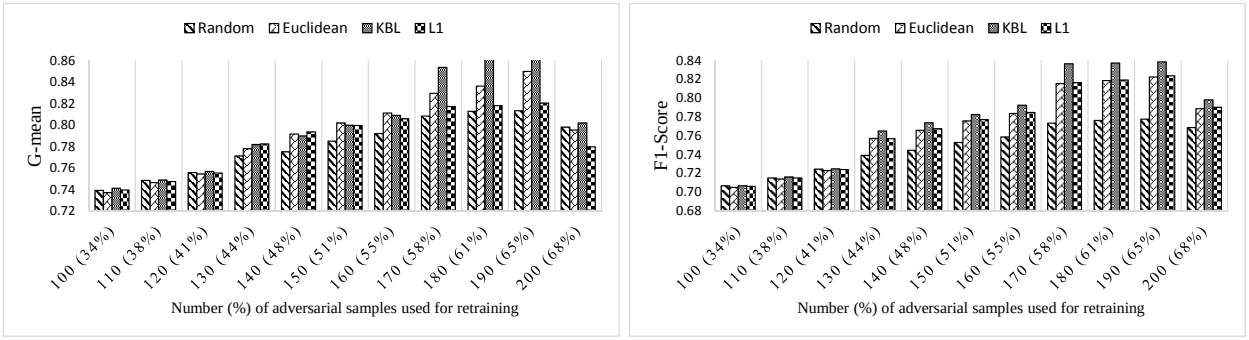
$$t_i = \begin{cases} \frac{N_+ + 1}{N_+ + 2}, & \text{if } y_i = +1. \\ \frac{1}{N_- + 2}, & \text{if } y_i = -1. \end{cases} \tag{9}$$

where N_+ and N_- are the number of positive and negative samples respectively. Using Eq. (7) - (9), the probability of adversarial samples to be a malware is computed. We chose the first n adversarial samples that have the least probability of being a malware. Having lower probability of being a malware means these samples are more likely to fool a classifier.

IV. Experiments and Results

A. Dataset and Simulation Tools

We used publicly available malware dataset [20] for experiments. A total of 3000 malware and 5000 benign applications were selected from this dataset for experiments. Permissions, API calls, and ICC features (activities, android intents, content providers and broadcast receivers) were extracted as static



(a) G-mean (b) F1-score

Fig. 5: a) G-mean and b) F1-score of DNN after adversarial retraining with random and selective samples

features using Androguard tool [39]. For dynamic feature extraction, each application was run in genymotion emulator [40] and the 'monkey' tool [41] was used to generate random events while the application is being executed. System calls of the applications during the runtime were logged as dynamic features. The deep neural network and adversarial sample crafting were implemented using tensorflow and Python library [42], [43]. All the other classifiers were implemented using the Python scikit-learn library [44].

B. Performance Metrics

The following metrics were used for performance evaluation:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \times 100\%$$

$$Precision = \frac{TP}{TP+FP} \times 100\%$$

$$Recall = \frac{TP}{TP+FN} \times 100\%$$

$$Specificity = \frac{TN}{TN+FP} \times 100\%$$

$$F1\text{-}score = 2.\frac{Precision \times Recall}{Precision+Recall}$$

$$G\text{-}mean = \sqrt{Specificity \times Sensitivity}$$

where TP is the number of true positives, i.e., malware accurately predicted; FP is the number of false positives, i.e., benign apps that were predicted as malware; TN is the number of true negatives, i.e., benign apps accurately predicted and FN is the number of false negatives, i.e., malware that were predicted as benign apps.

C. Resilience by Retraining with Deep Neural Network

In this section, we present the performance evaluation of adversarial retraining with deep neural networks when the samples are chosen using our proposed selection strategy and compared with the current work in the literature where those are selected randomly. We first trained a deep neural network with 3 hidden layers consisting of 2000, 1000, and 500 neurons respectively. We used 4500 benign and 2700 malware for training and 500 benign and 300 malware for testing. The network achieved 98.3% accuracy, which is comparable to the performance attained in other key works in this area [10], [20], [45]. This concludes Phase-1 in Fig. 2. By applying the method and satisfying the constraints outlined in Sec. III-B, 293 adversarial samples were generated by modifying the malware samples used in the test dataset. The classification accuracy dropped to 71% when tested on the adversarial data. Thereafter, a certain number (n) of adversarial samples were selected for retraining the classifier either randomly or our proposed strategies. For our first proposed method, i.e., based on the distance from the malware cluster center, we considered the malware samples as a single cluster for implementation simplicity. However, it can easily be extended for multiple clusters using Eq. (6) and any clustering algorithm. The rest of the adversarial samples are combined with the clean test data for evaluation. Once the adversarial samples are selected, the classifier is retrained using a new training set consisting of original training samples and the selected adversarial samples as additional malicious samples. This completes Phase-2 in Fig. 2. We incremented the value of n for retraining from 100 with an interval of 10. Each of these experiments was run for 10 trials varying random initialization of DNN weights and biases and the average of these trials is reported in the results presented below.

Figure 4 shows the comparison of accuracy and recall values between random selection [10] and selection based on the Euclidean distance, L1 norm or KBL after a DNN is retrained with adversarial samples. Results show that our proposed selection methods outperform the random selection method. It is also noteworthy that here KBL performs better than other selective strategies.

We postulate that, the probability function f in Eq. (7) used in the KBL method essentially reflects a distance from the hyperplane, separating the samples in the transformed kernel space. Arguably, the better separation between samples, as learned by the kernel for the transformed feature space, also captures better information of the samples leading to better performance.

A comparison of random selection with our other distance-based selection methods shows that in only a few cases random selection performs slightly better. This happens mostly in the cases where the number of samples for retraining is small (e.g.,

TABLE I: Accuracy and G-mean of adversarial retraining with different deep neural network architectures using random, the Euclidean distance based, L1 norm based and KBL selection. 65% adversarial samples were used for retraining.

# of hidden layers (# of neurons)	Accuracy (%)				G-mean			
	Random	Euclidean	L1	KBL	Random	Euclidean	L1	KBL
2([500, 500])	80.27	83.83	82.45	82.55	0.781	0.8218	0.8056	0.809
2([1000, 500])	80.27	83.83	83.73	85.25	0.781	0.8229	0.8171	0.8535
3([1000, 1000, 500])	79.85	83.61	84.04	85.74	0.7777	0.8211	0.8179	0.8616
3([2000, 1000, 500])	**80.69**	**84.39**	**84.52**	**86.08**	**0.7843**	**0.8262**	**0.8202**	**0.8655**
4([2000, 1000, 500, 500])	80.17	83.24	79.33	81.8	0.7802	0.8148	0.7796	0.8018
4([2000, 1000, 750, 500])	78.40	79.65	79.65	80.58	0.7763	0.7898	0.7898	0.7997

100, 110 samples) where the probability of randomly selecting better samples is higher. But the performance difference is not significant in these cases and this effect diminishes as more samples are chosen for retraining. The figure shows the results of KBL using the *RBF* kernel. In addition, we also explored the possibility of using a *linear* kernel, however, it did not show any performance gain. It has already been shown in the literature that the linear kernel is a degenerate case of RBF kernel and hence, a properly tuned RBF kernel gives similar or better performance in most cases [46].

Figure 4a further shows that a selection of 58% to 65% adversarial samples for retraining achieves better performance for all methods. Especially, when compared to random selection, KBL achieves a 6% accuracy improvement when trained with 65% of adversarial samples. Figure 4b also shows that KBL attains better recall value which is more prominent when more than 50% of adversarial samples are selected for retraining. Higher recall value indicates the misclassification of fewer malicious samples which is a desired property since the cost of misclassifying a malicious application is significantly higher than that of misclassifying a benign application.

We further recorded G-mean and F1-score that represent a balance between the false positives and false negatives which are shown in Fig. 5. The figure shows that in most of the cases the selective sampling, especially KBL, performs better than the random selection strategy and the highest value for G-mean (Fig. 5a) and F1-score (Fig. 5b) were obtained when the DNN model was retrained with 65% of the adversarial samples.

However, when more than 65% adversarial samples are selected for retraining, the performance degrades with the increasing number of samples. If too many adversarial samples are used for retraining the classifier starts to over-fit towards malware and starts misclassifying benign samples at a higher rate. This can further be observed in Fig. 4b that shows that the recall value keeps increasing as more adversarial samples are used for retraining indicating that more malicious applications are being correctly classified i.e., the number of false negative is decreasing. This occurs at the expense of increased misclassification of benign samples when adversarial samples are too many such as above 65%. As a result, a gradual decrease of precision values was also observed as more adversarial samples are used for retraining.

A Wilcoxon Signed-Rank Test [47] comparing random selection and KBL based selective methods yielded a p-value of 4.4×10^{-4} indicating that the performance improvement of KBL, as compared to random selection, is statistically significant.

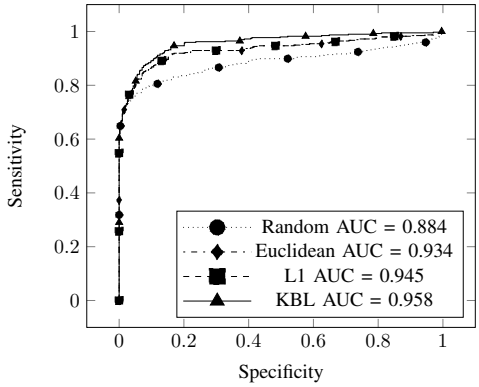


Fig. 6: ROC curve and AUC for adversarial retraining using random, the Euclidean distance and KBL selection

We further evaluated the impact of network architecture on the performance of adversarial retraining. Table I shows the accuracy and G-mean values of different adversarial retraining methods with varying network architecture. The table shows a slight variation in terms of performance depending on the number of layers and neurons in each layer of the network. Since networks with less than two hidden layers are considered shallow, we varied the number of hidden layers from 2 to 4 with a different number of neurons in each layer. Results show that our proposed selection mechanisms yield better performance than blind random selection, such as 80.69% and 0.7843 (random) vs 86.08% and 0.8655 (KBL) in terms of accuracy and G-mean respectively.

A plot of Receiver Operating Characteristic (ROC) curve and the corresponding Area Under the Curve (AUC) using the selection methods is shown in Fig. 6. As the figure shows our selection methods achieve better AUC than random selection and the best AUC of 0.958 is achieved by the KBL method vs 0.884 in random selection.

D. Resilience by retraining with Other Classifiers

We further evaluated the selective strategies of adversarial retraining with three other widely used classifiers, namely, Random Forest, Support Vector Machine (SVM) and Bayesian classifier. We first experimented with how effectively the adversarial samples, that are crafted using deep neural networks, can mislead those classifiers. For this, we first trained a classifier with original samples and tested it with clean

test samples to make sure its performance was good enough. Afterwards, we mixed adversarial samples with the clean samples and evaluated the classifier's performance. Table II shows the accuracy of different classifiers when tested with clean samples and samples mixed with adversarial ones. The table shows a significant drop in classifiers' performance when tested with adversarial samples. This suggests that adversarial samples crafted using deep neural networks are similarly effective in misleading other classifiers as well.

TABLE II: Accuracy of different classifiers when tested on clean samples and samples mixed with adversarial samples.

Classifier	Accuracy (%)	
	Clean Samples	Mixed with adveresarial samples
SVM	98.5	73.19
Random forest	97.75	73.83
Bayesian	97.25	72.82

Thereafter, we retrained these classifiers with the adversarial samples crafted using DNN and compared the efficacy of our selective methods with that of random selection. Figure 7 shows the comparison of malware detection accuracy after adversarial retraining with the classifiers a) SVM, b) Random forest, and c) Bayesian. It is observed that KBL- and the Euclidean distance-based selection method clearly outperforms the random selection method for adversarial retraining in all the classifiers while L1 norm-based selection performs better than random selection for SVM and Random forest classifier. Calculating G-mean and F1-score also showed improved performance by our KBL method. For example, using the SVM classifier at 65% samples, G-mean and F1-scores are 0.906 and 0.902 respectively for KBL selection, and 0.871 and 0.862 for random selection.

For these classifiers, a Wilcoxon Signed-Rank Test comparing KBL with random selection method yielded a p-value $< 3.8 \times 10^{-4}$, validating the performance improvement being statistically significant.

V. Conclusion

In this work, we studied the vulnerability of machine learning based malware detection systems against adversarial attacks in Industrial IoT setting. Rather than selecting random adversarial samples for retraining a classifier, we proposed two techniques for selecting adversarial samples. Experimental results reveal that, compared to the random selection strategy proposed in current malware literature, our selective sample selection strategy leads to better performance. In fact, our KBL selection method achieved 6% performance improvement over randomized selection. While existing works are mostly based on deep neural networks, we show that other well-known classifiers are also vulnerable against adversarial samples in malware detection. Our selection methods yielded similar performance improvement for those classifiers as well. For the IIoT system, the strategies proposed in this study will aid in designing robust and secured systems which, in turn, will lead to operational efficiency and financial benefits.

References

[1] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (iiot): An analysis framework," *Computers in Industry*, vol. 101, pp. 1–12, 2018.

[2] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[3] M. He and D. He, "Deep learning based approach for bearing fault diagnosis," *IEEE Transactions on Industry Applications*, vol. 53, no. 3, pp. 3057–3065, 2017.

[4] F. Cheng, J. Wang, L. Qu, and W. Qiao, "Rotor-current-based fault diagnosis for dfig wind turbine drivetrain gearboxes using frequency analysis and a deep classifier," *IEEE Transactions on Industry Applications*, vol. 54, no. 2, pp. 1062–1071, 2017.

[5] A. Ahmed, V. Krishnan, S. Foroutan, M. Touhiduzzaman, A. Srivastava, Y. Wu, A. Hahn, and S. Sindhu, "Cyber physical security analytics for anomalies in transmission protection systems," in *2018 IEEE Industry Applications Society Annual Meeting (IAS)*. IEEE, 2018, pp. 1–8.

[6] S. Sharmeen, S. Huda, J. H. Abawajy, W. N. Ismail, and M. M. Hassan, "Malware threats and detection for industrial mobile-iot networks," *IEEE access*, vol. 6, pp. 15 941–15 957, 2018.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, vol. abs/1412.6572, 2015.

[8] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," *arXiv preprint arXiv:1702.02284*, 2017.

[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.

[10] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 62–79.

[11] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2014.

[12] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," *arXiv preprint arXiv:1412.5068*, 2014.

[13] T. Miyato, S.-i. Maeda, S. Ishii, and M. Koyama, "Virtual adversarial training: a regularization method for supervised and semi-supervised learning," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[14] M. E. Khoda, T. Imam, J. Kamruzzaman, I. Gondal, and A. Rahman, "Selective adversarial learning for mobile malware," in *18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom19)*. IEEE, 2019.

[15] S. Chakrabarty and D. W. Engels, "A secure iot architecture for smart cities," in *2016 13th IEEE annual consumer communications & networking conference (CCNC)*. IEEE, 2016, pp. 812–813.

[16] A. V. Jerald, S. A. Rabara, and D. P. Bai, "Secure iot architecture for integrated smart services environment," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2016, pp. 800–805.

[17] H. Sun, X. Wang, R. Buyya, and J. Su, "Cloudeyes: Cloud-based malware detection with reversible sketch for resource-constrained internet of things (iot) devices," *Software: Practice and Experience*, vol. 47, no. 3, pp. 421–441, 2017.

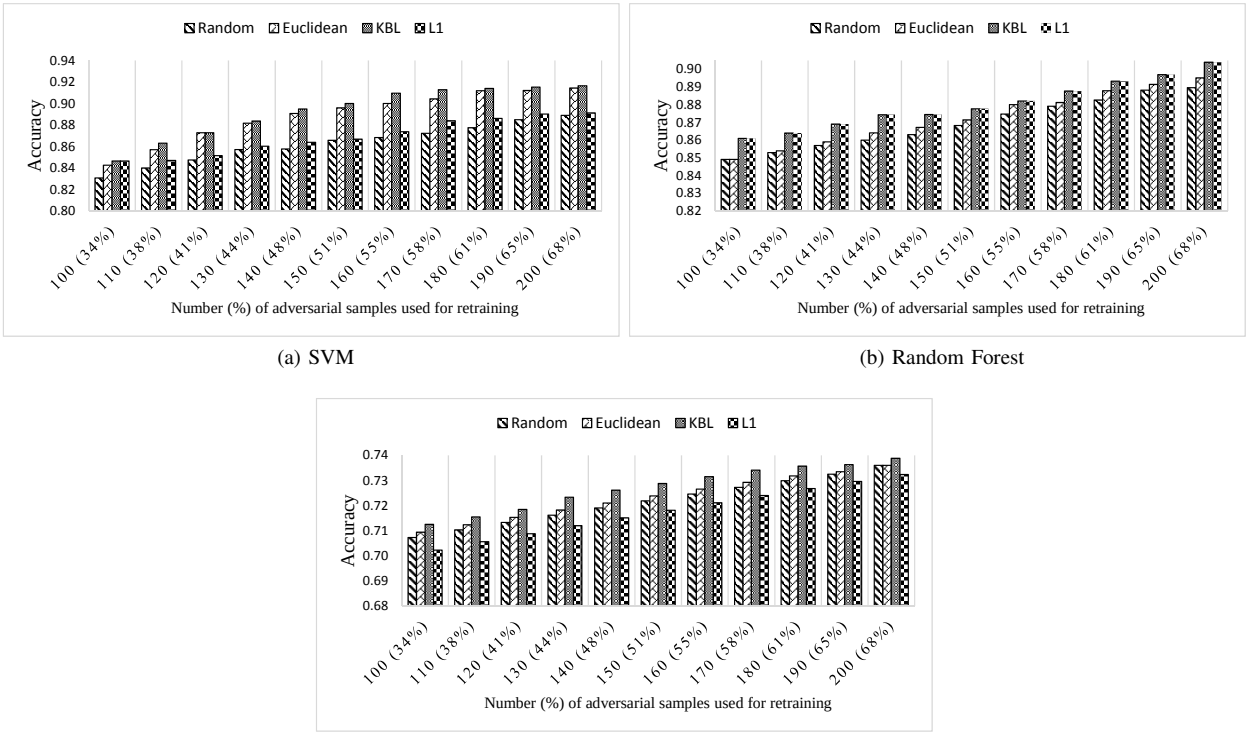
[18] A. A. A. Samra, K. Yim, and O. A. Ghanem, "Analysis of clustering technique in android malware detection," in *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2013, pp. 729–733.

[19] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of bayesian classification-based approaches for android malware detection," *IET Information Security*, vol. 8, no. 1, pp. 25–36, 2014.

[20] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[21] K. Xu, Y. Li, and R. H. Deng, "Iccdetector: Icc-based malware detection on android," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1252–1264, 2016.

[22] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, "Identifying android malware using dynamically obtained features," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, 2015.



(a) SVM

(b) Random Forest

(c) Bayesian

Fig. 7: Malware detection accuracy of adversarial training for a) SVM b) Random Forest and c) Bayesian classifier.

[23] 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*. ACM, 2016, pp. 1–8.

[24] A. Kapratwar, "Static and dynamic analysis for android malware detection," 2016.

[25] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: deep learning in android malware detection," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 371–372.

[26] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in android," *Journal of Parallel and Distributed Computing*, vol. 103, pp. 22–31, 2017.

[27] X. Su, D. Zhang, W. Li, and K. Zhao, "A deep learning approach to android malware feature learning and detection," in *Trustcom/BigDataSE/ISPA, 2016 IEEE*. IEEE, 2016, pp. 244–251.

[28] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[29] Z. Wang, J. Cai, S. Cheng, and W. Li, "Droiddeeplearner: Identifying android malware using deep learning," in *Sarnoff Symposium, 2016 IEEE 37th*. IEEE, 2016, pp. 160–165.

[30] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading machine learning malware detection," *Black Hat*, 2017.

[31] H. Dang, Y. Huang, and E.-C. Chang, "Evading classifiers by morphing in the dark," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 119–133.

[32] W. Yang, D. Kong, T. Xie, and C. A. Gunter, "Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps," in *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 2017, pp. 288–302.

[33] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 582–597.

[34] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," *arXiv preprint arXiv:1702.06280*, 2017.

[35] Z. Gong, W. Wang, and W.-S. Ku, "Adversarial and clean data are not twins," *arXiv preprint arXiv:1704.04960*, 2017.

[36] S. Ertekin, J. Huang, L. Bottou, and L. Giles, "Learning on the border: active learning in imbalanced data classification," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM, 2007, pp. 127–136.

[37] J. Hernandez, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, "An empirical study of oversampling and undersampling for instance selection methods on imbalance datasets," in *Iberoamerican Congress on Pattern Recognition*. Springer, 2013, pp. 262–269.

[38] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *ADVANCES IN LARGE MARGIN CLASSIFIERS*. MIT Press, 1999, pp. 61–74.

[39] Androguard, "androguard/androguard," Nov 2017. [Online]. Available: https://github.com/androguard/androguard

[40] "Genymotion." [Online]. Available: https://www.genymotion.com/

[41] "Monkey tool," https://developer.android.com/studio/test/monkey.html, accessed: February 17, 2019.

[42] M. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[43] "Craft image adversarial samples with tensorflow." [Online]. Available: https://github.com/gongzhitaao/tensorflow-adversarial

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] Z. Zhu and T. Dumitraș, "Featuresmith: Automatically engineering features for malware detection by mining the security literature," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 767–778.

[46] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with gaussian kernel," *Neural computation*, vol. 15, no. 7, pp. 1667–1689, 2003.

[47] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.