Neural Computing and Applications (2020) 32:14781–14800
https://doi.org/10.1007/s00521-020-04831-9

ORIGINAL ARTICLE

On defending against label flipping attacks on malware detection systems

Rahim Taheri[1] · Reza Javidan[1] · Mohammad Shojafar[2] · Zahra Pooranian[2] · Ali Miri[3] · Mauro Conti[2]

Received: 23 July 2019 / Accepted: 4 March 2020 / Published online: 28 July 2020

Abstract
Label manipulation attacks are a subclass of data poisoning attacks in adversarial machine learning used against different applications, such as malware detection. These types of attacks represent a serious threat to detection systems in environments having high noise rate or uncertainty, such as complex networks and Internet of Thing (IoT). Recent work in the literature has suggested using the *K*-nearest neighboring algorithm to defend against such attacks. However, such an approach can suffer from low to miss-classification rate accuracy. In this paper, we design an architecture to tackle the Android malware detection problem in IoT systems. We develop an attack mechanism based on silhouette clustering method, modified for mobile Android platforms. We proposed two convolutional neural network-type deep learning algorithms against this *Silhouette Clustering-based Label Flipping Attack*. We show the effectiveness of these two defense algorithms—*label-based semi-supervised defense* and *clustering-based semi-supervised defense*—in correcting labels being attacked. We evaluate the performance of the proposed algorithms by varying the various machine learning parameters on three Android datasets: Drebin, Contagio, and Genome and three types of features: API, intent, and permission. Our evaluation shows that using random forest feature selection and varying ratios of features can result in an improvement of up to 19% accuracy when compared with the state-of-the-art method in the literature.

Keywords Adversarial machine learning (AML) · Semi-supervised defense (SSD) · Malware detection · Adversarial example · Label flipping attacks · Deep learning

1 Introduction

Machine learning (ML) algorithms have the ability to accurately predict patterns in data. However, some of the data can come from uncertain and untrustworthy sources. Attackers can exploit this vulnerability as part of what is known as *adversarial machine learning (AML)* attacks. *Poisoning attacks* or *data poisoning attacks* are a subclass of AML attacks, in which attackers inject malicious data into the training set in order to compromise the learning process, and effect the algorithm performance in a targeted manner. *Label flipping* attacks are a special type of data poisoning, in which the attacker can control labels assigned to a fraction of training points. Label flipping attacks can significantly diminish the performance of the system, even if the attacker's capabilities are otherwise limited. Recent

✉ Mohammad Shojafar
mohammad.shojafar@unipd.it; m.shojafar@ieee.org

Rahim Taheri
r.taheri@sutech.ac.ir

Reza Javidan
javidan@sutech.ac.ir

Zahra Pooranian
zahra@math.unipd.it

Ali Miri
ali.miri@ryerson.ca

Mauro Conti
conti@math.unipd.it

[1] Department of Computer Engineering and Information Technology, Shiraz University of Technology, Shiraz, Iran

[2] SPRITZ, Department of Mathematics, University of Padua, Padua, Italy

[3] Department of Computer Science, Ryerson University, Toronto, Canada

work in AML looks into effectiveness of poisoning attacks in degrading the performance of popular classification algorithms, such as support vector machines (SVMs) [38], embedded features selection methods [35, 37], neural networks [11], and deep learning systems [25]. Most attacks in the literature assume attackers can manipulate both features and labels associated with the poisoning data. However, sometimes the attacker's capabilities are limited to manipulating labels, and he is only able to flip the labels to fool the ML classifier. These types of attacks are known as *flipping attacks*. Deep neural networks (DNNs) have gained significant success in classifying well-labeled data. However, label flip-type poisoning attacks can reduce the accuracy of these algorithms [36]. Therefore, there is a need for alternative methods for training DNNs that take label flipping attacks into account. Such methods should be able to identify and correct mislabeled samples or reweight the data terms in the loss function according to the extracted label.

There are a number of works in the literature focused on identifying and dealing with poisoning attacks. For example, an algorithmic method evaluates the impact of each training sample on the performance of learning algorithms [4]. Although this method is effective in some cases, it cannot be generalized to the large dataset. Among other defensive mechanisms, the *outlier detection* is used to identify and remove suspicious samples. But, this method has a limited performance (i.e., accuracy) against label flipping attacks [26]. Another category of related works mainly focuses on learning strategies that can be applied on flip labels. Such solutions are divided into *two* categories. In the first group it can directly learn flipped labels, while in the second group, it can focus on an extra set of clean data. In the first case, the label flipping module is used to identify correctly labeled data [24, 34] and to modify the changes on the labels to reset the data terms in the loss function. Performance of this technique is significantly impacted by its label cleaning precision and its rate of flip sample estimation. In the second group of methods, an additional set of clean data is used to guide the learning agent through flipped data [28]. Despite promising results, both groups of methods have a common default. They try to fix the flipped labels, or they reweight the terms for data points. This default will inevitably cause errors for some data points.

Motivated by these considerations, in this paper, we consider the *binary classification* for sampling and analysis of Android malware. We only assume the weakest capability for the attacker. That is, we assume that the attacker has no perfect knowledge about the learning algorithm, the loss function optimized by the system, or the initial the training data and a set of features used by the learning algorithm. We show that having the system identifying and retraining the wrong label, and using our proposed semi-supervised (SS) approach to training will result in better results. To this end, we suggest a solution that covers the existing data points that are mislabeled and improves the accuracy of the classification algorithm. To do so, we present an architecture for learning flipped data. Then, we identify a small part of the mislabeled training set, whose labels are likely to be correct, and the flipped labels associated with other data are ignored. Afterward, we train a deep neural network in a SS manner based on selected data.

1.1 Contributions

In this context, several natural questions are arising, such as: How can we define attack based on label flipping algorithm which can fool the classifier? Is it possible to design an enhanced ML model to improve system security by presenting some secure algorithms against a given label flipping attack? How can we tune and test the countermeasure solutions to deal with label flipping attack? The answer to these queries is the goal of this paper. More in detail, the goal of the paper summarizes as follows: First, we rank the data points within each class and then hold the label for the points that have higher rankings. If no clean set is available, the ranking is based on the multi-way classification neural network, which is trained from the original training dataset. In fact, a binary classifier is learned that, while clean labels are available, separates data containing clean labels and flipped labels. Second, we apply a temporary ensemble for semi-supervised deep neural network training. Hence, our original contributions are as follows:

- We present an architecture for learning flipped data which reflects our main focus in the malware detection system.
- We propose a label flipping poisoning technique to attack the Android malware detection based on deep learning: where an algorithm is proposed for crafting efficient prototypes so that the attacker can deceive the classification algorithm. In this technique, we use silhouette clustering to find an appropriate sample to flip its label.
- We introduce a DL-based semi-supervised approach against label flipping attacks in the malware detection system called LSD, which uses label propagation and label spreading algorithms along with CNNs to predict the correct value of labels for the training set.
- We implement a countermeasure method based on clustering algorithms as a defense mechanism. It is a DL-based semi-supervised approach against label flipping attacks in the malware detection system that improves the detection accuracy of the compromised

classifier. In this approach, we use *four* clustering metrics and validation data to relabeled poisoned labels.
- We conduct our experiments on two scenarios on three real Android datasets using three feature types compared to the cutting-edge method and deeply analyze the trade-offs that emerge. The source code of the paper is available in GitHub [31].

To the best of our knowledge, none of the previous works in the literature has conducted a similar analysis. The closet paper to our method is KNN-based semi-supervised defense (KSSD) [26], in which the authors have entailed KNN strategy to relabel samples by considering the distance between them. However, the work in [26] is tailored to the relabeling of samples, they are unable to specify some similar samples that may be malware and benign and may mislabel the features of benign sample due to low distance of samples. Unlike the [26], in this paper we explicitly tackle the poisoning samples located far from the decision boundary and relabel them. Also the defense method presented in [26] is unable to distinguish overlapping areas of two classes and cannot correctly label the poisoning samples located there, while our defense methods impose the model to tackle such data points and relabeling them.

1.2 Organization of the paper

We organize the rest of the paper as follows: Sect. 2 overviews the related works. Section 3 details the problem definition, the presented architecture, and the related components. Section 4 presents our proposed attack model inspired by AML architecture and reports the proposed defense strategies against the raised attack. We evaluate the performance of the algorithms in Sect. 5. In Sect. 6, we detail the results of the experiment and provide some open discussion regarding our method. Section 7 presents conclusions and future work. Table 1 shows the important abbreviations used in this paper.

Table 1 Important abbreviations used in this paper

Notations	Description
AML	Adversarial machine learning
SSL	Semi-supervised learning
LSD	Label-based semi-supervised defense
CSD	Clustering-based semi-supervised defense
KSSD	KNN-based semi-supervised defense
GAN	Generative adversarial network
CNN	Convolutional neural network
LP	Label propagation
LS	Label spreading
RI	Rand index
MI	Mutual information
FMI	Fowlkes–Mallows index

2 Related work

In this section, we classify the related work in the literature into two different defense classes: (i) we will cover defense approaches that try to correct labels in Sect. 2.1, and (ii) defense strategies that ignore poisoned labels and adopt semi-supervised learning methods to protect the model against attacks are then covered in Sect. 2.2. Hence, we draw conceptual relationships and delineate the most recent defense strategies applied to tackle the label flipping attack and identify relevant major alternatives for comparison.

2.1 Defense algorithms against poisoning attacks

The problem of classification with label noise—mislabeling in class variable—is an active area of research. The paper [10] gives a comprehensive overview of both the theoretical and applied aspects of this area. Label flipping mechanism is a solution to cover label noise in the classifiers [7]. This method can model the overall label flipping probability. However, it is lack of considering individual specific characteristics in label noise. In [19], the authors create a lightweight method called *Curie* to protect SVM classifier against poisoning attacks. The preliminary idea behind this method is to distinguish the suspicious data points and remove them outside the dataset before starting the learning step of the SVM algorithm. In other words, Curie's algorithm flips labels in the training dataset to defend SVM classifiers against poisoning attacks. They cluster the data in the feature space and try to calculate the average distance of each point from the other points in the same cluster with related weight and train model and test in some datasets. They present that their defense method is able to correct 95% of samples in the training dataset. Additionally, the authors in [22] describe a poisoning algorithm to solve the bi-level optimization problem based on back-gradient optimization [21]. The proposed algorithm applies automatic differentiation technique to compute the gradient in the optimization problem. This algorithm using gradient method to resolve the optimization problem takes several computational times, and it can pose challenges in complex networks such as neural networks and deep learning. Thus, they apply a novel technique named back-gradient optimization to allow computing the gradient of interest in a more computationally efficient and stable manner to shape their ML

model. Authors in [32] explicitly investigate data poisoning attacks for the semi-online setting, unlike other works which are mostly based on the offline setting. The work in [29] argues that it is possible to perform targeted attacks on specific testing data without declining the overall performance of classifier along with any control of adversary over the labeling of training data. The methodology proposed in [3] is suitable to identify and remove poisonous data in IoT systems. This method, mainly, exploits data provenance to identify manipulated data before the training step to improve the performance of classification. Compared to our method, the defense method presented in [3] cannot correctly label the poisoning samples while our defense methods impose the model to tackle such data points and relabeling them. The work in [6] focuses on building an automatic robust multiple kernel-based logistic regression classifier against poisoning attacks without applying any cross-validation. Despite the fact that proposed classifier may improve performance and learning speed, it does suffer from the lack of any theoretical guarantees. To address this issue, they extend their method and entail new structure to resist the negative effect of random label noise as well as a wide range of non-random label noises [5].

2.2 Semi-supervised learning defense algorithms

Another active area of research is the one dealing with learning from unlabeled data. The semi-supervised learning approach, along with applying unlabeled data to learn better models, is particularly relevant to our work. The semi-supervised approaches include multi-view learning like [9], co-training [28, 33], graph-based methods like [15], and semi-supervised ML solutions like SVM [20], and our proposed work (DL-based semi-supervised solution). These approaches try to tackle that many successful learning algorithms need access to a large set of *labeled data*. To address this issue, i.e., lack of availability of labeled data, a combination of tri-training with a deep model is used in [9] to build *Tri-Net*, which can use massive set of unlabeled data to help to learn with limited labeled data. The semi-supervised deep learning model generates *three* modules to exploit unlabeled data by considering model *initialization*, *diversity augmentation*, and *pseudo-label editing*. Graph-based transduction approach that works through the propagation of few labels, called *label propagation*, was used in [15] to improve the classification performances and obtain estimated labels. This method consists of two steps. In the first step, the classifier trains through labeled and predicts pseudo-labeled. In the second step, the nearest neighbor graph constructs based on the previous trained classifier. A limitation to this approach is that practically graph models are often mis-specified. However, this could potentially be overcome by employing highly expressive model families like neural networks [17]. Hence, in S3VM method [20], the authors adopt the SVM solution to find the flipped label examples in a dataset and improve the safeness of the semi-supervised support vector machines (S3VM). They indicate that the performance of their method is not statistically significantly worse than the solution shaped with labeled data alone. The major limitation of this method is that it is not easy to use such method for large amounts of noisy samples and outliers, and it exponentially reduces ML performance.

3 System model and proposed architecture

In this section, we first provide a formal definition our problem (Sect. 3.1). Then, in Sect. 3.2, we introduce the proposed Android malware detection architecture used in the paper. In particular, Fig. 1 will describe the components of the proposed architecture.

3.1 Problem definition

Consider the datasets as follows.

$$D = \{(x_i, y_i) \in (X, Y)\}, \quad i = 1, \ldots, n \tag{1}$$

where n is the number of malware samples. If x_i has the j feature, we have $x_{ij} = 1$. Otherwise $x_{ij} = 0$, and $X \subseteq \{0,1\}^k$—a k-dimensional space. The variable y represents the label of the samples with $y_i \in \{0,1\}$ and the D set has an unknown distribution on $X \times Y$. We assume the training set is defined as follows.

$$S = \{(x_k, y_k)\}, \quad k = 1, \ldots, m \tag{2}$$

where S is the label set. The flipping attack label aims to find a collection such as P containing samples in S so that when their labels are flipped, it minimizes the desired target for the attacker. For simplicity, we assume that the attacker's goal is to maximize the loss function which we define it as $L(w, (x_j, y_j))$.

3.2 Proposed architecture

In this section, we present our architecture to tackle the Android malware detection problem in IoT systems (Fig. 1).

In Fig. 1, we present a general scheme of our proposed architecture and the proposed attack and defense algorithms which use for Android applications. In this architecture, we assume a complex set of IoT devices (i.e., IoT systems), which are communicating with each other,

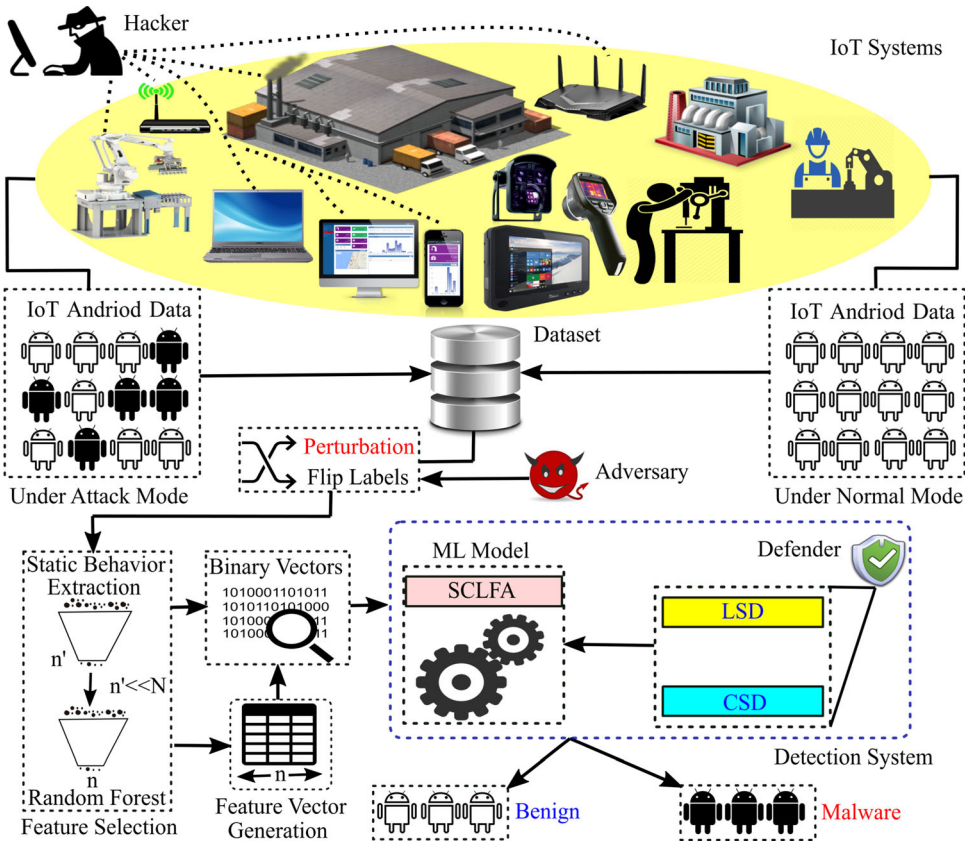


Fig. 1 Architecture overview of proposed method. ML = machine learning; SLFA is our attack method and LSD and CSD are our defense methods

represented by the yellow oval in the figure. We assume that some of the IoT devices are using Android OS platforms. We also assume that an attacker can get access to some of the IoT devices. Hence, he can manipulate the data they transferring to each other. As a result, the data traffic of each Android data can include those from malware apps, represented by the black Android app symbol in our figure. Each Android app, whether malware and benign, presents as a vector of different features with various labels. ML algorithms exposed to adversary attacks can add a variety of perturbations to data to fool ML algorithms. Hence, in this architecture, an adversary can get access to the dataset and flip the labels by adding some perturbation of existing labels. Our feature selection component gives the ability to select the choice of features. We then generate a binary vector of each Android app and input the result to the ML model. A final component of our architecture is the detection system composed of the ML model and our proposed defense algorithms. Our architecture can increase the robustness of our detection system against label flipping attacks and increase the accuracy of malware/benign classifications. In the following section, we explore our attack and defense algorithms.

4 Proposed attack and defensive solutions

In this section, the proposed classification algorithm used in the paper is described first in Sect. 4.1. We then describe our attack strategy, inspired by silhouette clustering method in Sect. 4.2. Section 4.3 presents our two defense solutions against the attack proposed in the previous section. Finally, we report the computational complexity of our strategies in Sect. 4.4.

4.1 Classification algorithm

In this paper, we incorporate a deep CNN to classify the binary samples. We adopt the overfitting method to find out how good our dataset size is. *Shift invariant* or CNN is a multilayer perceptron strategy to tackle the fully connected neurons in each layer and help to prone the overfitting data and can include more complex patterns. To do so, we try to classify our data using a training set and then repeat the classification using cross-validation. If we increase the data size, it gives better results in CNN classification processing.

Figure 2 presents the proposed CNN architecture for the classification algorithm. In this figure, we can see that we apply three sequential layers of one-dimensional convolution (Conv-1D) that has 16, 32, and 64 filters. In each of

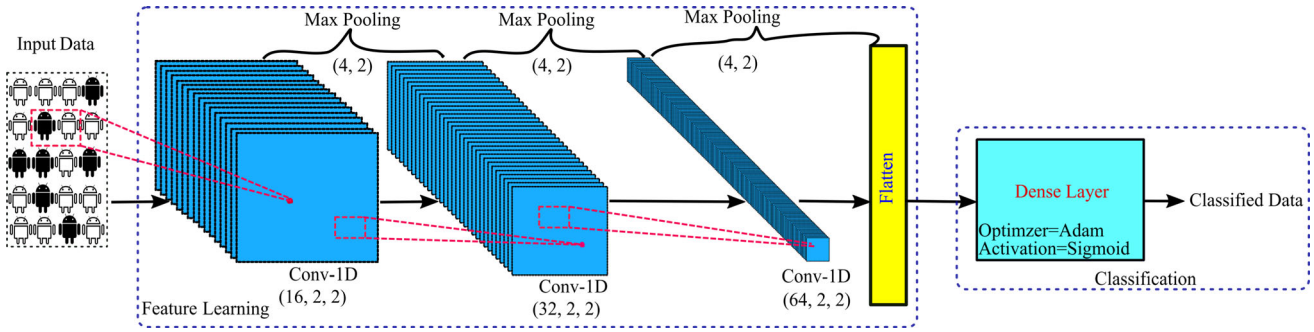


Fig. 2 Proposed classification algorithm architecture. Conv = convolution; (A, B, C) = (filters, kernel size, stride); (C, D) = (pool size, stride)

these layers, we have kernel size with value 2 and stride with value 2. We apply maxpooling between the convolution layers to prevent overfitting by reducing the computational load, memory, and number of parameters. Each maxpooling layer creates four pool sizes with two strides. After applying three convolutional layers, we adopt a flattened layer and a dense layer. In the dense layer, we use *Adam* optimizer and *Sigmoid* activation function to shape the classification algorithm and the outside of the dense layer is the classified data.

4.2 Attack strategy: Silhouette Clustering-based Label Flipping Attack (SCLFA)

In this subsection, we apply silhouette clustering method to flip the labels. We name this attack *Silhouette Clustering-based Label Flipping Attack (SCLFA)*. Silhouette clustering is a type of clustering technique in which we can interpret and validate the consistency of data clusters. Silhouette provides a concise visual presentation object classifications. This technique defines a measurement called *silhouette value (SV)* that expresses the self-cluster similarity or cohesion of per object comported to other clusters or separation, which is between $[-1, 1]$. If the silhouette value is one, it presents well matching of the object to its own cluster and is less likeness to other neighboring clusters. If the majority of the objects in a cluster have high SVs, it indicates that the cluster objects and the clustering are appropriately configured. We utilize a Euclidean distance method to calculate the SV in this paper. We define the *label flipping attack (LFA)* as follows:

Definition 1 *LFA in SCLFA*: LFA is a type of attack that the attacker tries to use some algorithms to modify the label of features and changes the interval of each sample in a cluster. In this paper, we use the silhouette clustering algorithm to implement LFA. To put it simply, in SCLFA, we assign an interval $[-1, 1]$ for each sample, which indicates whether the sample is in the correct cluster. If the silhouette value (SV) is negative, it means that the selected sample is a good candidate for flipping the label, and according to the silhouette algorithm, it is definitely belonging to another cluster. Hence, we change the label of such sample. Let L_i be the label of the i-th sample out of n samples in the dataset. Thus, we can write it as Eq. (3):

$$L_i = \begin{cases} (x_i, y_i), & \text{SV} > 0 \\ (x_i, |1 - y_i|), & \text{otherwise} \end{cases} \tag{3}$$

Algorithm 1 presents the label flipping poisoning attack.

Description of Algorithm 1. In this algorithm, we present the proposed method, SCLFA, for the flipping label of the training sample. This method is based on the K-means clustering algorithm. In this way, we first create a model based on the K-means algorithm that divides the X_train samples into *two* clusters and predicts the label for each sample (lines 1–2). Then, in line 3, we calculate the silhouette values for samples and predicted labels for the samples. As previously stated, values close to 1 indicate that the sample is fitted in the appropriate cluster, and as the values of silhouette are less than 1 and close to -1, it means that the sample is clustered incorrectly. In the proposed method, we flipped the label of samples that have a silhouette value less than zero. In this way, we probably have chosen the examples that have the potential to be in the other cluster (lines 4–8).

Algorithm 1 Silhouette-based Label Flipping Attack (SCLFA)

Input: X_train , Y_train
Output: Poisoned_Y_train

1: $M_K \leftarrow$ **Make Model** with two Clusters Using KMeans Algorithm
2: $Labels \leftarrow$ **Predict** labels of X_train using M_K
3: $S \leftarrow$ **Compute** Silhouette values using X_train using $Labels$
4: **for** each $row \in X\_train$ **do**
5: **if** $(S[row] \leq 0)$ **then**
6: $Poisoned\_Y\_train[row] = abs(1 - Y\_train[row])$
7: **end if**
8: **end for**
9: **return** $Poisoned\_Y\_train$

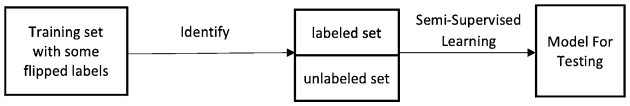


Fig. 3 Overview of SSL model

4.3 Defensive strategies

In this subsection, we discuss these countermeasures against the label flipping attack. In detail, we describe label-based semi-supervised defense (LSD) and clustering-based semi-supervised defense (CSD), which are presented in Sects. 4.3.1 and 4.3.2, respectively. In this paper, we assume our data are only partially labeled. Our defense strategies begin by investigating which validation data in training samples may have been flipped. It would then predict new labels for these data and replace their labels. Figure 3 shows the overview of the semi-supervised learning (SSL) model for both defense strategies.

4.3.1 LSD defense

In this section, we design LSD algorithm to give a priority between semi-monitoring learning (SML) methods. In other words, we adopt validation data as inputs of SML algorithms to predict the label for each sample and then rank the predicted labels. The goal of the LSD algorithm is to find the samples for which the labels in the flipped training set are likely to have the correct values. Then, we need to give the selected data and its labels to the SSL algorithm. We need to create a validation set to monitor the training process and select the suitable parameters. That is, in the LSD method, we first rank the data points within each class and then hold the label for the points that have the highest rankings. If no clean set is available, ranking is applied which is designed based on the multi-way classification neural network. Hence, ranking is trained from the original training dataset. In fact, in this defense mechanism, we try to learn a binary classifier, while clean labels are available. Then, we separate data containing clean labels and flipped labels. Formally speaking, in this defense strategy, in the first stage, we apply the *label propagation (LP)* algorithm to assign the labels to unlabeled data points. Then, in the next stage, we use *label spreading (LS)* to minimize the noises happen in labeling the samples. In the LSD method, we plan to design a method which works like an ensemble learning such that it uses propagation models to predict labels for flipping. In this way, we provide a *two-stage framework* for learning flipped labels. In the following, we describe LP and LS.

- *Label propagation (LP)* LP is a type of semi-supervised ML algorithm that can give a label to the unlabeled sample data. First, LP gives labels a small dataset of samples and makes classifications. In other work, LP aims to propose the labels to the unlabeled data points. That is, LP helps to find the community structure in real complex networks [2]. LP compared to the other practical methods in the literature has much lower processing time and could support a priori information needed about the network structure, and it does not require any knowledge of data point and samples before propagation. However, LP could produce several solutions for each set of data points.
- *Label spreading (LS)* LS algorithm is a type of propagation method that can apply the normalized graph Laplacian and soft clamping in an affinity matrix to influence on the labels. It also can diminish the regularization properties of a loss function and make it robust against the noise [18]. LS algorithm repeats on the modified version of a graph of data points and can normalize the edge weights by computing the normalized graph Laplacian matrix.

LP and LS algorithms create on a kernel of the system in which positively effect on the performance of the algorithm and enhance the chance of scalability of the problem. To be precise, and as an example, the RBF kernel can generate a fully connected graph that can demonstrate a dense matrix. Such big size matrix, in each iteration, could join with the cost performance of full matrix multiplication calculation and results in increasing the time complexity, which causes a problem for scalable case studies. In this paper, we fix the problem by utilizing LP and LS algorithms on a KNN kernel system which provides much more memory-friendly sparse matrix and can exponentially save on execution latency.

In the *first stage* of LSD algorithm, we use validation set to train the LP and LS algorithms. Then, we use these algorithms to predict labels of training set. At the same time, we train the CNN classifier with the validation data and predict new labels for training set samples. In the second stage, we use voting between all available labels, i.e., LP output, label spreading, CNN predicted labels, and poisoned labels.

In the *second stage* of LSD algorithm, we apply a temporary ensembling for semi-supervised deep neural network training. Then, we present a semi-supervised two-stage algorithm for training flipped labels, which include two main components. We discover and select some samples from the labeled training set, for which there are strong indications that their labels are correct. Afterward, we aim to learn a semi-supervised deep neural network that only uses the selected labels from the first previous stage. Finally, the ML model network can easily classify previously unseen test data. We summarize the proposed LSD countermeasure algorithm in *Algorithm 2*.

Description of Algorithm 2 It presents the semi-supervised defense, which is based on Label estimation. As seen in this algorithm, in lines 3–5, the label spreading algorithm is applied, which is used to find labels of training data. The label spreading algorithm is trained using validation data and then created a model used to predict labels of training data. Similarly, lines 6–8 use the label propagation algorithm to predict training data labels. This algorithm, like the label spreading algorithm, is a semi-supervised algorithm. In lines 9 and 10 of this algorithm, convolutional neural network as the third part of the ensemble learning approach is used, which is trained with validation data and is used to predict the training data label. The final part of the LSD method is the voting between the results of the three methods described and the poisoned label, which is the result of voting as the label for training samples.

Algorithm 2 Label-based Semi-supervised Defense (LSD)

Input: $X\_validation, Y\_validation, X\_train, Poisoned\_Y\_train$
Output: $Y_{Corrected}$

1: $X \leftarrow X\_train$
2: $Y \leftarrow Y\_validation$
3: $M_s \leftarrow$**Make Model** using LS algorithm
4: **Fit** the M_s Model on X and Y
5: $L_s \leftarrow$ **Predict** labels of X_train using M_s
6: $M_p \leftarrow$ **Make Model** using LP algorithm
7: **Fit** the M_p Model on X and Y
8: $L_p \leftarrow$ **Predict** labels of X_train using M_p
9: $M_{cnn} \leftarrow$ **Make Model** using proposed CNN algorithm
10: $L_{cnn} \leftarrow$ **Predict** labels of X_train using M_{cnn}
11: $Y_{Corrected} =$ **Voting**$(Y_{Corrected}, L_s, L_p, L_{cnn})$
12: **return** $Y_{Corrected}$

4.3.2 CSD defense

The main idea behind this approach is to use clustering techniques to correct flipped labels. As each of the clustering methods has its specific measure, in this method it is suggested to use the voting between the labels determined by different clustering methods for determining the label of the flipped samples. Hence, we use four indices to analyze the accuracy of our generated clusters and the predicted one and identify the most likely adversarial examples and flip their labels.

Description of Algorithm 3 In this algorithm, we explain the CSD method. In lines 1–3 of this algorithm, we use the proposed CNN model and validation data and predict the labels of the training data. In lines 4–7, the algorithm describes four cluster metrics, namely RI, MI, HM, and FMI, and computes their values. Each of these metrics is a measure for the accuracy of clustering. The main idea behind this approach is that the training samples are labeled in such a way that the mentioned measure does not differ significantly from the values calculated from the validation data. Therefore, in lines 8–16, we add one sample of the training data to the validation dataset, calculate the values of the clustering with four metrics, and compare them with the base values. If the difference is less than 0.1 (i.e., we consider as a threshold), then we consider the sample to be properly labeled. As a result, the output of this algorithm is the labeled sample, which can be used as a validation data and selected sample for training the ML model.

Algorithm 3 Clustering-based Semi-supervised Defense (CSD)

Input: $X\_validation, Y\_validation, X\_train, Poisoned\_Y\_train$
Output: $Y_{Corrected}$

1: $X \leftarrow X\_validation$
2: $M_{cnn} \leftarrow$ Make Model Using proposed convolutional neural network
3: $Y_{Corrected} \leftarrow$ predict labels of X_train using M_{cnn}
4: $R \leftarrow$Compute **Rand Index** using X and $Y_{Corrected}$
5: $M \leftarrow$Compute **Mutual Information** using X and $Y_{Corrected}$
6: $H \leftarrow$Compute **Homogeneity Metric** using X and $Y_{Corrected}$
7: $F \leftarrow$Compute **Fowlkes-Mallows Index** using X and $Y_{Corrected}$
8: **for** each $row \in X\_train$ **do**
9: $X\_temp \leftarrow X\_validation + row$
10: Compute R_{temp} ,M_{temp} ,H_{temp} ,F_{temp}
11: $S \leftarrow |((R_{temp} - R) + (M_{temp} - M) + (H_{temp} - H) + (F_{temp} - F))|$
12: **if** $(S \leq 0.1)$ **then**
13: $X \leftarrow X + row$
14: $Y_{Corrected} \leftarrow Y_{Corrected}$+Label related to row
15: **end if**
16: **end for**
17: **return** $Y_{Corrected}$

The indices are defined as below.

- *Rand index (RI)* Rand measure/index is a statistical index to calculate the similarity between two data clusterings [27]. It is a value between zero and one such that zero indicates that two sets of clustered data do not have any pair point and one indicates that the data clustering is the same. Also, RI can be used to adjust a group for elements that we called them *adjusted Rand index*. In other words, RI is a metric of the accuracy of two sets of data points, which represents the frequency of occurrence of total pairs. Formally speaking, RI presents the probability of how can we randomly select two pairs X_1 and X_2 in two partitions of the same big set.
- *Mutual information (MI)* MI, or *information gain*, is a measure to realize the amount of information and dependency between two separate variables by observing them [23]. It is a type of entropy of a random variable that can understand the joint distribution of a pair data point which calculates by the product of the marginal distribution of those pair samples. Since the

data we are dealing with are fallen in the group of discrete data with discrete distribution, we can calculate the $\mathcal{I}$ MI of two jointly discrete random variables X_1 and X_2 as follows:

$$\mathcal{I}(X_1, X2) = \sum_{x_1 \in X_1} \sum_{x_2 \in X_2} p_{(X_1,X_2)}(x_1, x_2) \log_2 \left(\frac{p_{(X_1,X_2)}(x_1, x_2)}{p_{X_1}(x_1) p_{X_2}(x_2)} \right) \quad (4)$$

where $p_{(X_1,X_2)}$ is a joint probability mass function for the two samples of X_1 and X_2 , and p_{X_1} is a marginal probability of sample X_1 and p_{X_2} is a marginal probability of sample X_2.

- *Homogeneity metric (HM)* This metric uses for validating the data points which are members of a single class. HM is independent of being changed the score value of data point when a permutation of the class or labels is applied [14]. We can define HM values as $\mathcal{HM}$ as follows:

$$\mathcal{HM} = 1 - \frac{H(Y_T|Y_{PR})}{H(Y_T)} \quad (5)$$

where $\mathcal{HM}$ can be between 0 and 1. Note that low values of $\mathcal{HM}$ explain a low homogeneity or vice versa. If we have a sample data Y, we define Y_{PR}, Y_T are the predicted and the corrected values for that sample; hence, $H(Y_T)$ is the HM value for that sample when it is correctly placed and predicted to be placed in one single class, respectively. Besides, the $\frac{H(Y_T|Y_{PR})}{H(Y_T)}$ indicates that the predicted sample is not placed correctly in a single class. We aim to approach this fraction smaller and reach it to zero ($\mathcal{HM} \rightarrow 1$). We can achieve this goal when we reduce the knowledge of Y_{PR} and diminish the uncertainty of Y_T that results in the fraction above become smaller, and we have HM around 1.
- *Fowlkes–Mallows index (FMI)* Fowlkes–Mallows Index (FMI) metric is a popular metric to understand the similarity between two generated clusters, whether hierarchical or benchmark classification clusters [12]. The higher similarity between two clusters (created cluster and the benchmark one) indicates higher FMI values. FMI is an accurate metric used to evaluate the unrelated data and also is reliable even with added noises to the data results.

4.4 Computational complexity

In following section, we evaluate computational complexity analysis on the presented attack and defensive methods. Assume that the number of samples in *X_train* and *X_validation* is n and m, respectively. We list the computational complexity of the methods. So, we have

- *Time complexity of SCLFA attack* Focusing on SCLFA, the computation of all possible configurations in lines 1–2 of Algorithm 1 creates a model based on the *K*-means method and predicts the correct n training samples, resulting in $\mathcal{O}(n^2 \times k)$. Since, in this method, $k = 2$, the time complexity is in the order of $\mathcal{O}(n^2)$. In line 3 of this algorithm, silhouette values are computed for n training data samples, which has a complexity of $\mathcal{O}(n^2)$. Lines 4–8 of the algorithm include a *for* loop that performs the correction of the m validating labels and has a complexity of $\mathcal{O}(m)$. Overall, the computational complexity of Algorithm 1 is in the order of $\mathcal{O}(n^2) + \mathcal{O}(n^2) + \mathcal{O}(m) = \mathcal{O}(n^2), \forall\ n \gg m$.
- *Time complexity of LSD defense* Focusing on LSD, the computation of Algorithm 2 directly relates to the LS method, which has a complexity of $\mathcal{O}(n)$. Similarly, in lines 6–8, the model is based on the LP algorithm, which has a complexity of $\mathcal{O}(n)$. Then, lines 9 and 10 present CNN model creating, according to [13], which has a computational complexity of all convolutional layers. CNN computational complexity is $\mathcal{O}(\Sigma^d{}_{i=1} n_{(l-1)} \times s_l^2 \times m_l^2)$, where l is the index of a convolutional layer; d is the depth (number of convolutional layers); n_l is the width or the number of filters in the l_{th} layer–$n_{(l1)}$ is the number of input channels of the l-th layer; s_l is the spatial size (length) of the filter; and m_l is the spatial size of the output feature of CNN which has a time complexity in the order of $\mathcal{O}(n^3)$. Then, we perform voting between results that has a complexity of $\mathcal{O}(1)$ (line 11). Overall, the computational complexity of LSD defense algorithm is $\mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n^3) + \mathcal{O}(1) = \mathcal{O}(n^3)$.
- *Time complexity of CSD defense* Focusing on CSD, the computation of Algorithm 3 relies on CNN model construction based on validation data (lines 1–2). Then, we predict the label for training data samples based on this generated ML model. Therefore, the computational complexity of this part is in the order of $\mathcal{O}(n^3)$. Focusing on the RI, MI, HM and FMI clustering metric calculations, they have a complexity of $\mathcal{O}(n)$ (lines 4–7). Then, we calculate the values of these parameters for m samples. Hence, the complexity of this loop of the CSD algorithm is in the order of $\mathcal{O}(n \times m)$ (lines 8–16). As a result, the overall computational complexity of CSD defense method is $\mathcal{O}(n \times m) + \mathcal{O}(n^3) + \mathcal{O}(n) = \mathcal{O}(n^3), \forall\ n \gg m$.

5 Experimental evaluation

In this section, we report the results of our proposed attack and defense algorithms in different scenarios: with feature selection consideration (WFS) and without feature

selection consideration (WoFS). Given the two scenarios, we conduct our experiments on our attack (SCLFA) and defense algorithms (LSD and CSD) against KNN-based semi-supervised defense (KSSD) [26]. The source code of the paper is available in GitHub [31].

5.1 Simulation setup

We describe the test metrics, datasets, features, classification parameter, and comparison defense algorithm below.

5.1.1 Test metrics

To provide a comprehensive evaluation of our attack and defense algorithms, we use the following indices: accuracy, precision, recall, false-positive rate (FPR), true-negative rate (TNR), miss rate (FNR), F1-score, and area under cover (AUC):

- *Accuracy* Accuracy metric is defined in:

$$\mathrm{Acc} = \frac{\Omega + \chi}{\Omega + \chi + \Lambda + v} \tag{6}$$

  where Ω is true positive; χ is true negative; Λ is false positive; and v is false negative metrics.
- *Precision* Precision is the fraction of relevant samples between the retrieved samples which is shown in

$$\mathrm{Precision} = \frac{\Omega}{\Omega + \Lambda} \tag{7}$$

- *Recall* The recall is expressed in

$$\mathrm{Recall} = \frac{\Omega}{\Omega + v} \tag{8}$$

- *F1-score* This metric defines as a harmonic mean of precision and recall which is defined as

$$\begin{aligned} \text{F1-Score} &= \frac{1}{\frac{1}{\mathrm{Recall}} + \frac{1}{\mathrm{Precision}}} \\ &= 2 * \frac{\mathrm{Precision} * \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \end{aligned} \tag{9}$$

- *False-positive rate (FPR)* This metric represents a ratio between the number of negative events incorrectly classified as positive (false positives) and the total number of actual negative events. This metric is described in Eq. (10):

$$\mathrm{FPR} = \frac{\Lambda}{\Lambda + \chi} \tag{10}$$

- *Area under curve (AUC)* AUC measures the trade-off between misclassification rate and FPR. This metric can be calculated as (11):

$$\mathrm{AUC} = \frac{1}{2}\left(\frac{\Omega}{\Omega + \Lambda} + \frac{\chi}{\chi + \Lambda}\right) \tag{11}$$

- *False negative rate (FNR)* This metric is a method for determining the case that the condition does not hold, while in fact it does. In this work, we also called it *miss rate*. This metrics can be calculated as (12):

$$\mathrm{FNR} = \frac{v}{v + \Omega} \tag{12}$$

5.1.2 Datasets

Our experiments utilized the following *three* datasets:

- *Drebin dataset* This dataset is an Android example collection that we can apply directly. The Drebin dataset includes 118,505 applications/samples from various Android sources [1].
- *Contagio dataset* It consists of 11,960 mobile malware samples and 16,800 benign samples [8].
- *Genome dataset* This dataset is an Android example which is supported by the National Science Foundation (NSF) project of the United States. From August 2010 to October 2011, the authors collected about 1200 samples of Android malware from different categories as a *genome* dataset [16].

5.1.3 Features

In this paper, we consider various malicious sample features like permissions, APIs and intents. We summarize them as follows:

- *Permission* Permission is a essential profile of an Android application (apk) file that includes information about the application. The Android operating system processes these permission files before installation.
- *API* API feature monitors various calls to APIs on an Android OS, such as sending SMS or accessing a user's location.
- *Intent* Intent feature applies to represent the communication between different components which is known as a *medium*.

5.1.4 Parameter setting

We rank the features to better manage the huge amount of features using the *RandomForestRegressor* algorithm. Then, we repeat our experiments for 300 manifest features with higher ranks to determine the optimal number of features for modification in each method. In each test, we randomly consider 60% of the dataset as training samples,

20% as validation samples, and 20% as testing samples. We run our experiments on an 8-core Intel Core i7 with speed 4 GHz with 16 GB RAM on an OS Win10 64-bit.

5.1.5 Comparison of defense algorithms

We compare our proposed algorithms to defend against label flipping attacks with KNN-based semi-supervised defense (KSSD) [26] and GAN-based defense [30]. The comparison results show that our proposed methods are more robust in detecting label flipping attacks. In the KSSD method, authors adopt *K*-nearest neighbor (KNN) method to mitigate the effect of label flipping attacks. A relabeling mechanism for suspected malicious malware is suggested. The KNN algorithm uses the training set to assign a label to each sample. The aim is to ensure the homogeneity of the label between the close examples, especially in areas that are far from the decision boundary. In the training set, authors first select *K*-nearest neighbors using the Euclidean distance. Then, if the fraction of the data points that are among the most commonly enclosed labels in *K* are equal to or greater than the threshold of t with $0.5 \leq t \leq 1$ they select them. The training sample available in the *K*-nearest neighbor is relabeled with the most common label. Given that we only have two types of labels in detecting malware, they assign the dominant label in *K* to the nearest neighbor to the sample. *Algorithm 4* presents the KSSD defense.

Algorithm 4 KNN-based Semi-Supervised Defense (KSSD) [26]

Input: training set S, Threshold t
Output: S

for $i = 1 \leq m$ **do**
  $K_{NN} \longleftarrow$ **Find** K nearest neighbor for sample i
  $F \longleftarrow$ **Find** the fraction of samples in KNN with most frequency
  if $F > t$ **then**
    $y_i =$ label with most frequency in K_{NN}
  end if
end for
return S

We indicate that poisoning sample points that are far from the decision boundary are likely to be relabeled and reduce the negative performance consequences on the classification algorithm. Although the algorithm gains validation of genuine points at the same time, i.e., in areas where the two classes overlap (especially for values of t close to 0.5), we can have a similar amount of the correct points that are labeled in two classes, and it confirms that the KSSD label correction solutions presented in Algorithm 4 must be the same for the two classes. Therefore, this type of labeling shall not considerably influence the classification algorithm.

Another comparison made in this paper is the GAN-based defense presented in [30]. Algorithm 5 illustrates the proposed method in this study. This algorithm works by generating new samples to train the machine learning model again. Specifically, in this paper, we use the GAN as a synthetic data generator set. GAN has two functions called *Generator* and *Discriminator*. The former one can modify the less likely malware samples. To do so, in the training phase, it selects one random feature from the highest ranked features with zero value. Then, it changes the selected feature value to one to generate new sample. In the latter function, the GAN uses this function as a classifier to predict the class variable. It modifies the features until the discriminator function is cheated and labels such a sample among the benign samples. Besides, we gather the wrongly estimated malware samples into a synthetic data generator set. Besides, we use 80% of the synthetic data generator set with the training dataset to update the AML model. We use the remaining synthetic data generator samples (i.e., 20% of the data samples) with the test dataset to test/analyze the classification. It is found that the proposed methods even outperform the GAN-based method, since the proposed GAN is only flipping-focused research with respect to the important features of decision making, while the proposed methods in this paper are based on the value of labels.

Algorithm 5 GAN: pseudo-code of the GAN defense [30]

Input: training set $S(X, Y)$, X^*, Y^*, λ
Output: $Model_{new}$

$Model_{poison} \leftarrow$ Fit a model on X_{trn} using LR
$Lesslikely \leftarrow$ Fit 10% model of X_m with KNN to $Model_{poison}$
for each $x \in Lesslikely$ **do**
  $x_{new} \leftarrow x$
  while $x_{new} \in Model_{poison}$ Classify as M **do**
    $x_{new} \leftarrow$ Add ranked(λ) features from $X_b \cup x_{new}$
  end while
  $synthetic_{data} \leftarrow synthetic_{data} \cup x_{new}$
end for
$Model_{new} \leftarrow$ Fit Model on $X_{trn} \cup synthetic_{data}$
$Y_{corrected} \leftarrow$ Use $Model_{new}$ to predict label of X
$S \leftarrow$X and related label($Y_{corrected}$)
return S

5.2 Experimental results

In this section, we test our presented attack algorithm (SCLFA) on our originally trained classifiers and validate our defense algorithms (LSD and CSD) against adversarial label flipped examples (KSSD) and GAN-based synthetic data generator on the above three Android malware datasets.

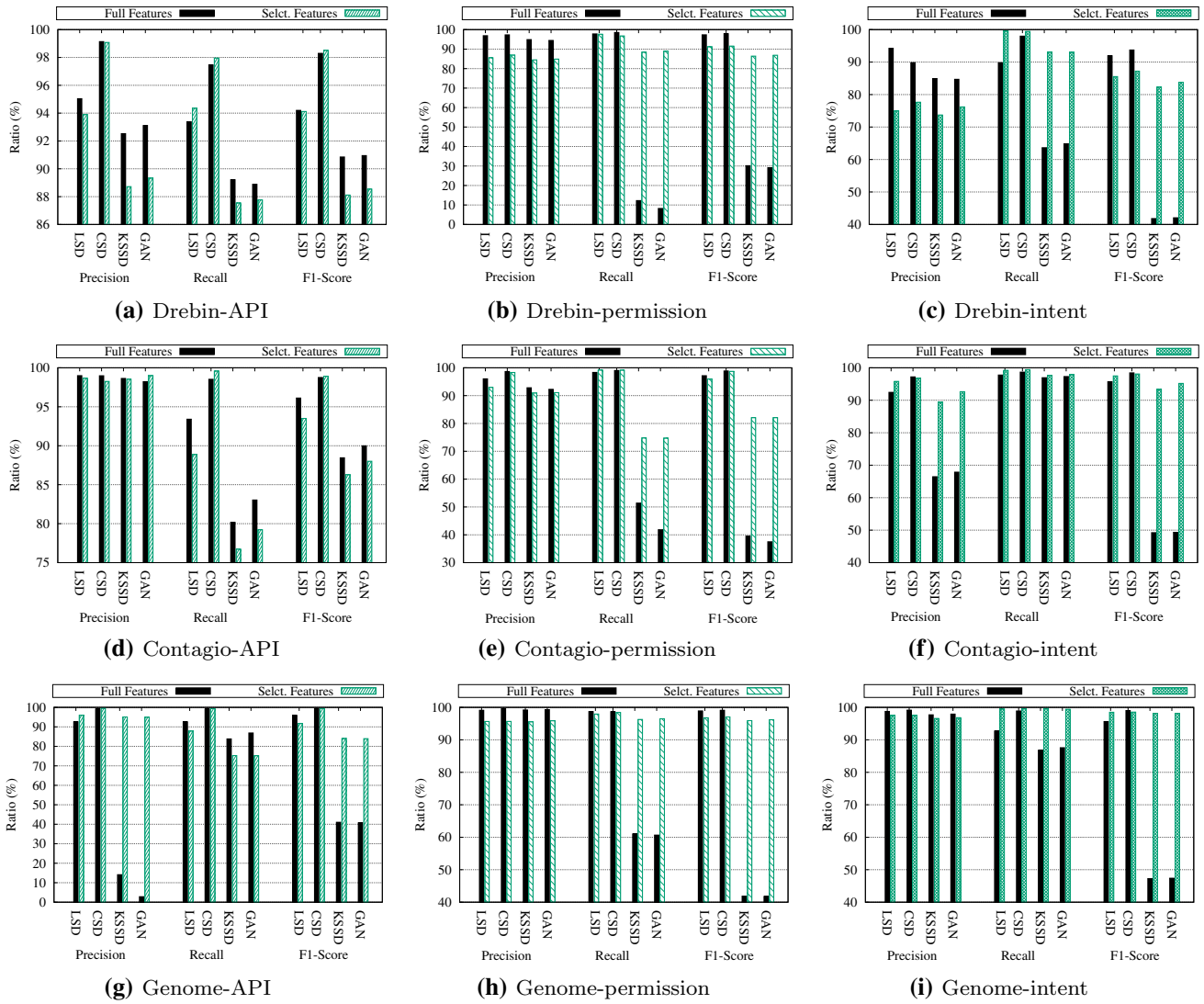


Fig. 4 Comparison between DEFENSE algorithms with reference to precision, recall, and F1-Score for API, intent and permission features in various datasets

5.2.1 Comparing methods based on precision, recall, F1-score

In this test scenario, we aim to compare the defense algorithms (Fig. 4) and attack method compares with the data without triggering the data, i.e., no-attack (Fig. 5). Specifically, in Fig. 4, we provide precision, recall, and F1-score values for different defense algorithms. Both recall (sensitivity) and precision (specificity) metrics indicate generated errors. The recall is a measure that could show the rate of total detected malware. That is, the proportion of those correctly identified is the sum of all malware (i.e., those that are correctly identified by the malware plus those that are incorrectly detected by benign). Our goal in this section is to design a model with high recall that is more appropriate to identify malware. To give more insight, Fig. 4a–i reports the permission, API and intent data for the Drebin, Contagio, and Genome datasets, respectively. *Three* considerations hold in this figure: (i) the value of precision/recall and even F1-score for KSSD algorithm and GAN-based algorithm is clearly lower than our LSD and CSD methods (as expected), and it confirms that our proposed defense algorithm is able to identify the more benign samples compared to other methods correctly. (ii) CSD algorithm has higher precision and recall values compared to the LSD algorithm. (iii) In this feature group, our proposed algorithms have a higher precision/recall/F1-score value for intent-type features in all datasets compared to two other feature sets in which our defense algorithm can detect more benign samples correctly in different data samples.

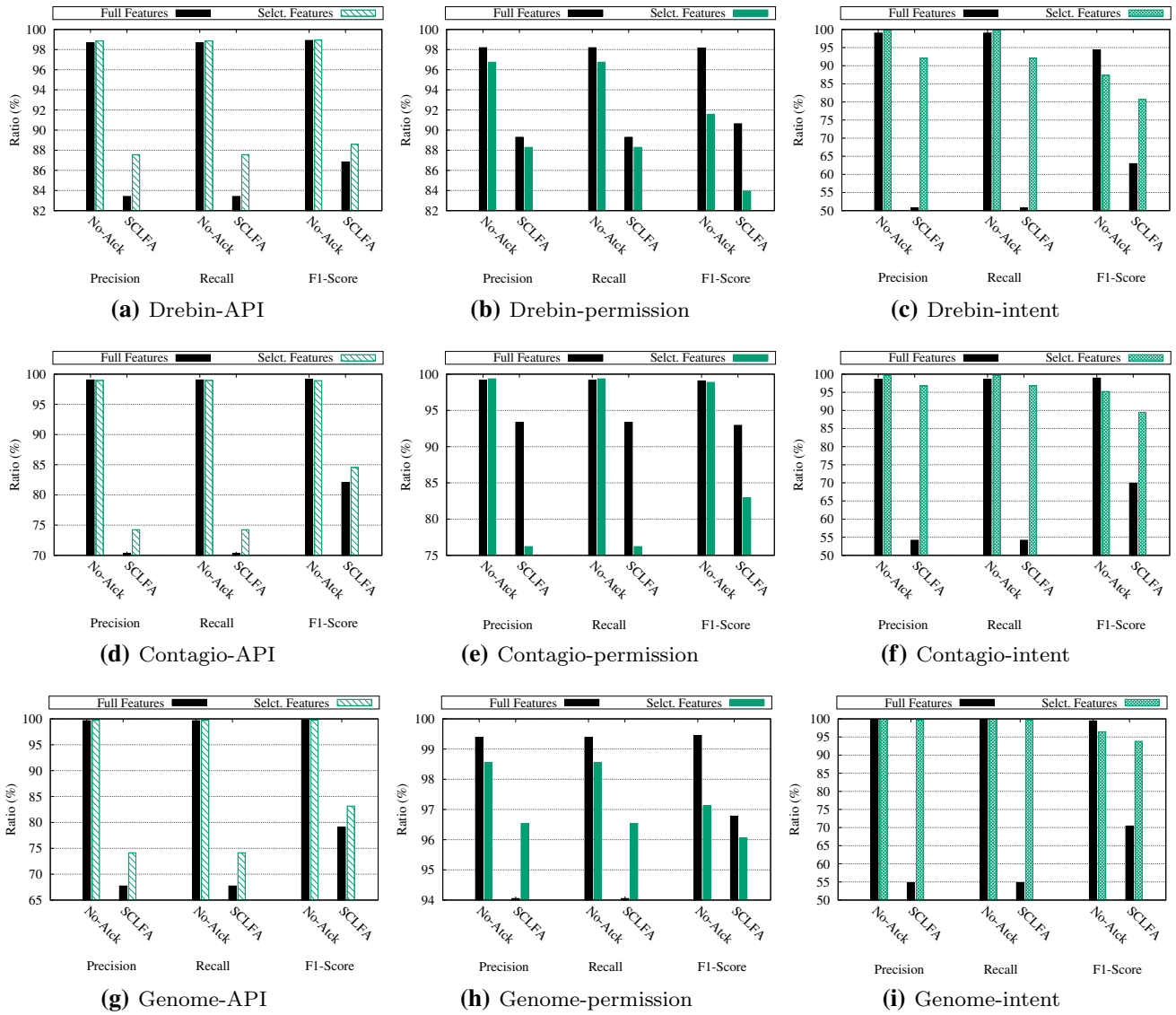
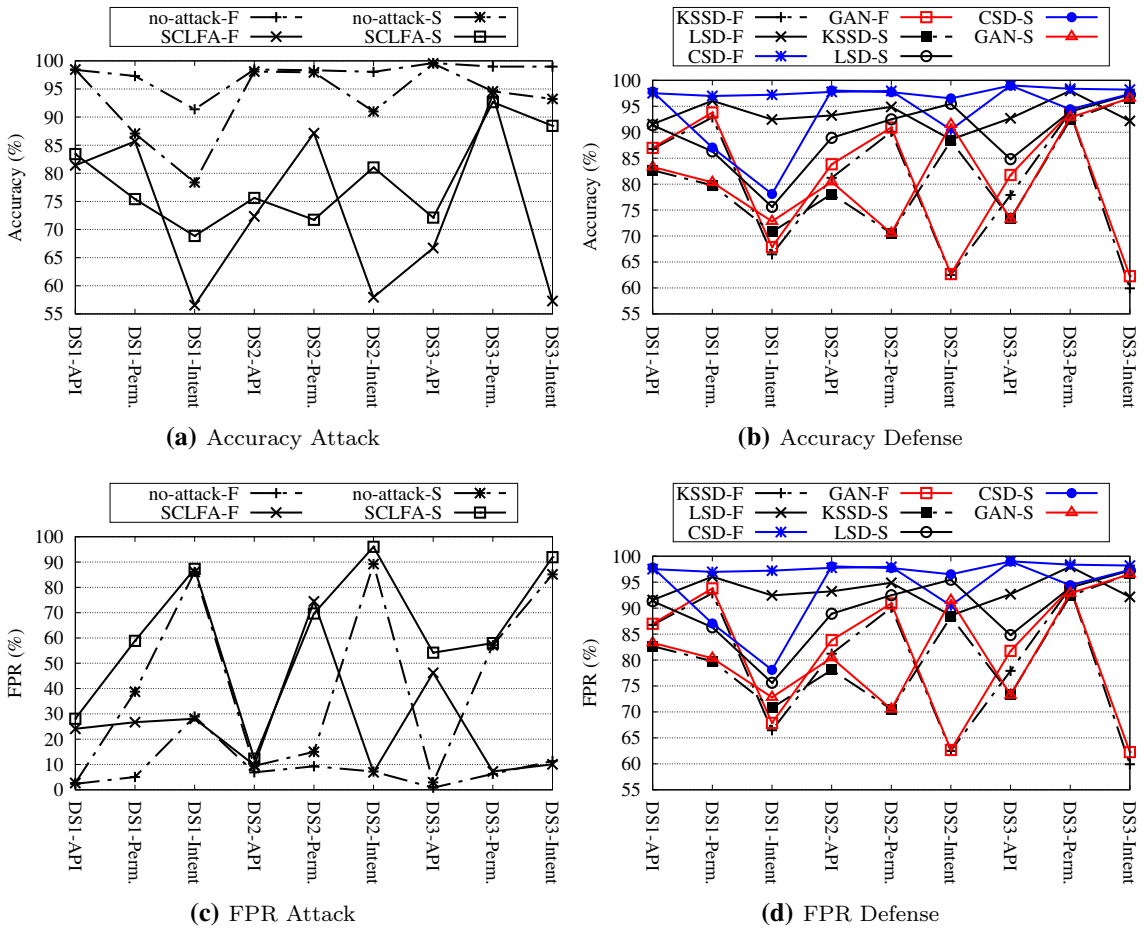


Fig. 5 Comparison between ATTACK algorithms with reference to precision, recall, and F1-Score for API, intent and permission features in various datasets

Focusing on the attack consequences, Fig. 5 presents the precision, recall, and F1-score values for our attack algorithm, SCLFA, and the similar data when there is no any attack triggered for the ranked selected features and all three datasets with full features. In this figure, we understand that our attack strategy can completely fool the ML model and impose to falsify classification and exponentially decrease the precision and recall values. It can be seen that the diminishing rate is about 45% for intent features in all three datasets and its ratio is higher for Drebin dataset (the precision/recall pair bars in Fig. 5c). It is considering that the feature selection has a positive influence on the ML performance. From the attacker point of view, it is essential to impact negatively the ML model classification ability. Thus, our attack strategy gives the lead for such cases, and its adverse effects are more apparent in Contagio API features, where SCLFA influences on both selected features and full feature scenarios and can misclassify about 23% of the samples (the pair bars in Fig. 5d).

5.2.2 Comparing methods based on FPR and accuracy

In this part, we present the FPR and accuracy values for the attack and defense algorithms for nine states, which consists of three features and three datasets, and show them in Fig. 6. The attack algorithm aims to increase the FPR rate, and defense algorithm seeks to improve the accuracy and decrease the FPR rate accordingly. Considering these points, we evaluate the algorithms on two mentioned



(a) Accuracy Attack **(b)** Accuracy Defense

(c) FPR Attack **(d)** FPR Defense

Fig. 6 Comparison between attack and defense algorithms with reference to accuracy and FPR for API, intent and permission features in various datasets. (DS1 = Drebin; DS2 = Contagio; DS3 = Genome) and $-F$ = full feature; $-S$ = selected features

scenarios: considering full features (labeled "$-F$") and feature selection scenario (labeled as "−Selc").

Focusing on Fig. 6a, we compare our attack algorithm (SCLFA) with the no-attack mode for two mentioned scenarios. As we can understand from this figure, our SCLFA label flipping method compared to the no-attack mode poses problem for all nine states (i.e., listed in the x-axis of the figure) and results in lower accuracy to all feature types and their major drastic are higher in intent features for all datasets (see the 'X' shape marks in the lower part of Fig. 6a). In this case, the accuracy has dropped more than 20% compared to the absence of an attack. It confirms that the proposed attack method is more successful in attack to the intent features. In all datasets, the algorithms behave roughly the same, and the reduction in the accuracy of the API is more intense. As we see, the accuracy of the proposed attack method in the case of using all data is lower than that of the 300 features.

Focusing on Fig. 6b, we compare the accuracy of defense algorithms for full-featured and selected features scenarios. In this figure, almost in all cases, the CSD method can provide higher accuracy than other defense algorithms (the blue mark points in Fig. 6b) and is fallen in a range of (62%, 98%). Therefore, it can detect more benign samples. The CSD method is more accurate than LSD in all 9 states, and its average accuracy is about 95%, 97.6% and 98.5% for full feature consideration scenario in Drebin, Contagio and Gnome datasets, respectively, while this value for KSSD defense algorithm is about 80%, 79% and 77%, respectively.

Focusing on Fig. 6c, the FPR value of our attack algorithm (SCLFA) is compared to the time we have no attack in datasets. Concerning the intent features in all datasets, the SCLFA algorithm, it has an FPR value and can fool more malware samples compared to other features in all datasets. In other words, increasing the FPR values means increasing the number of false positives, which is the goal of an attacker, and as can be seen from the comparison of Fig. 6c with a, by increasing FPR, the accuracy value decreases. As a result, the results of these two figures confirm each other.

Table 2 AUC and FNR comparisons in percent (%) for presented algorithms in various features, datasets in two test scenarios: WFS and WoFS

Other ML metrics		Datasets											
Ratio (%)		Drebin				Contagio				Genome			
Algorithms	File type	WoFS		WFS		WoFS		WFS		WoFS		WFS	
		FNR	AUC	FNR	AUC	FNR	AUC	FNR	FNR	FNR	AUC	FNR	AUC
Attack													
SCLFA	Permission	10.71	82.64	11.71	60.56	6.67	59.06	23.79	60.76	5.95	96.21	3.46	68.80
	API	16.59	83.22	12.43	80.78	29.70	94.35	25.75	92.91	32.29	74.40	25.86	70.24
	Intents	49.21	62.97	7.94	42.28	45.83	95.87	3.21	43.55	45.22	94.26	0.31	48.32
Defenses													
LSD	Permission	2.17	93.90	2.33	70.77	1.65	80.32	0.81	61.64	1.29	93.89	2.05	68.65
	API	6.62	90.77	5.65	88.44	6.59	95.34	11.13	94.03	7.40	96.59	12.15	55.23
	Intents	10.20	91.97	0.43	43.70	6.28	89.42	0.75	50.07	7.27	91.82	0.53	50.41
CSD	Permission	1.44	95.08	3.22	73.98	0.94	93.18	0.84	91.36	1.33	97.06	1.58	69.07
	API	2.53	98.40	2.04	98.28	1.47	95.05	0.42	91.39	0.84	98.42	0.65	96.83
	Intents	2	93.74	0.54	45.60	1.75	93.18	0.55	50.31	1.10	94.12	0.40	51.43
GANX [30]	Permission	48.95	47.00	11.06	70.93	47.91	0.54	25.28	61.19	33.62	55.49	3.51	69.14
	API	11.11	87.37	12.46	78.92	16.96	93.80	20.78	95.58	27.26	59.77	24.78	70.76
	Intents	87.35	8.96	6.99	43.50	33.46	57.75	2.10	71.60	26.36	60.84	0.57	83.29
KSSD [26]	Permission	4.95	91.17	11.58	70.59	3.53	61.88	25.19	90.94	5.60	90.91	3.72	68.57
	API	10.78	86.27	12.46	78.92	19.81	94.45	23.27	94.25	19.91	72.48	24.73	71
	Intents	39.10	70.93	6.91	42.39	35.11	89.02	2.35	44.58	41.79	90.04	0.30	50.51

WFS With feature selection, *WoFS* without feature selection

Focusing on Fig. 6d, we compare the FPR values of defense algorithms. From this figure, we can understand that the CSD algorithm tries to decrease the FPR values more than two other defense algorithms in most of the states. It is important to note that having high accuracy does not mean that the defense algorithm can successfully protect the dataset against the poisoning data, and it is essential to decrease the FPR in that state. Hence, we need to point-to-point check each state of this figure with a similar state in Fig. 6b. From these comparisons, we conclude that the CSD algorithm performs better than LSD and KSSD.

5.2.3 Comparing methods based on FNR and AUC

In this part of our work, we compare the AUC and FNR values for attack and defense algorithms over three datasets using three different feature sets. In Table 2, we present these results for two scenarios: with feature selection (WFS) and full features or without feature selection (WoFS). Concerning the FNR concept, it shows the misclassification rate of data in a dataset. In the flipping attack, the FNR rate increases, and it decreases with defense algorithms. Also, the AUC and FNR values indicate that by performing a flipping attack on the labels, FNR values increase and AUC values decrease. The AUC and FNR values are based on Eqs. (11) and (12) which relates to true negative (Λ) and true positive (Ω). FNR and AUC values with increasing FPR rates during an attack increase the Ω value and are pleasant for the attacker. Defense strategies try to decrease the FNR or miss rate of malware sample corrections and help to increase the AUC. In Table 2, we understand that both of our defense algorithms have higher AUC and lower FNR rate for all datasets, and they confirm our recently mentioned points.

5.2.4 Computational complexity comparisons

In this part of the paper, we compare the computational complexity of our attack and the defense algorithms against (KSSD) [26] and GAN-based Defense [30]. Table 3 compares the time required for the testing phase of various datasets and different features among the different proposed algorithms for ranked features using RF and without feature selection methods. In this table, focusing on the defense algorithms, the implementation of KSSD defense is the fastest compared to LSD, CSD, and GAN-based algorithms. The reason behind it is that it randomly selects and modifies the label of the features. However, its accuracy is much lower than LSD and CSD algorithms (the

Table 3 Computational complexity comparisons in seconds (s) for presented algorithms in various features, datasets in two test scenarios: WFS and WoFS

Computational complexity		Datasets					
Time (s)		Drebin		Contagio		Genome	
Algorithms	File type	WoFS	WFS	WoFS	WFS	WoFS	WFS
Attack							
SCLFA	Permission	140.09	4.04	87.66	3.56	130.10	3.11
	API	7.14	4.71	4.84	3.88	4.21	3.74
	Intents	150.99	3.83	209.89	2.87	106.07	2.92
Defenses							
LSD	Permission	385.79	101.16	417.62	107.81	348.62	106.02
	API	123.91	114.64	117.35	112.75	109.87	105.38
	Intents	963.97	105.17	747.98	96.81	501.85	108.10
CSD	Permission	148.15	11.50	118.77	9.51	123.45	9.16
	API	21.76	15.77	17.22	13.27	14.56	12.77
	Intents	281.83	11.26	235.24	9.21	198.63	11.42
KSSD [26]	Permission	95.90	5.20	83.91	4.15	90.83	5.16
	API	9.95	7.59	8.53	6.46	8.41	6.42
	Intents	210.99	5.17	206.77	4.12	146.55	5.12
GAN [30]	Permission	425.13	211.64	471.33	194.55	394.65	176.38
	API	94.23	67.45	86.56	64.75	75.34	57.14
	Intents	515.41	276.54	495.32	209.21	436.97	196.45

WFS With feature selection, *WoFS* without feature selection

curves in Fig. 6b). However, both LSD and CSD algorithms require processing of the DL algorithm on the malicious dataset, with the LSD and GAN being the slowest method among the *four* methods in correcting the labels of poisoning samples.

Focusing on the computational complexity of ranked features scenario, we can understand that the LSD and CSD methods, running faster in intent and permission features and these results are even quicker when we compared them with API feature cases. We can conclude that the distribution of API features may require more computations in calculating LSD and CSD, and this is a normal behavior of the algorithms. Because in the feature selection, we select 300 API features that have the highest rank based on the RF feature selection algorithm, and this covers about 95% of the API features from every data sets, but the selection of 300 features from intents and permission intends selecting at most 20% of the main features. Therefore, the computational complexity of the proposed algorithm over the API features when the 300 features are chosen is close to the state in which all the features are used.

Focusing on the computational complexity of full feature comparisons scenario, we can see that since the number of API features is much less than the intents and permissions, the computational time of proposed algorithms is less on these features. Similarly, we can conceive the same results for the computational complexity of proposed algorithms on the permission features than the results of intent features (the permission row values for WFoS cases in all datasets in Table 3). Also, from Table 3 we can realize that the proposed methods are slower than the KSSD method. However, as we understand from the comparisons of ML metrics, the KSSD method is a weak method for label flipping attack (LFA) compared to our proposed defense algorithms.

Additionally, the computational time of CSD with taking into account the high accuracy of this method and its take less running time compared to the LSD method. So, this behavior converts the CSD method into an attractive way to defend against the LFA. Another point to be added about the time of the LSD method is to consider the structure of the method in which a CNN network is used, whose time complexity is at least $\mathcal{O}(n^3)$ and this can be the major drawback when it compares to the CSD algorithm.

6 Discussions

In the following, we explain the achievements and some constraints on our attack and defense algorithms. From the results, we can conclude that the proposed methods based on semi-supervised learning can modify the flipped labels to increase the accuracy of classification methods, including CNN. Despite the promising results achieved by our attack and defense algorithms, it is clear that our approaches have some intrinsic limitations. Firstly, the critical

point in the proposed methods is the need for more calculations. Notably, the use of a CNN in the LSD method increases the computational complexity of this method. Secondly, the proposed malware detection algorithms implement static features, and the features are binary and are in the sparse matrix. Hence, it is easier to calculate clustering measures. However, for other applications, it may not be possible to perform calculations of the clustering measures efficiently.

Another limitation of our defense methods is the classification algorithm used in this paper. Formally speaking, in this work, we design a three-layer CNN, which has high accuracy, and use CNN to investigate the results of the proposed algorithms. The classification accuracy with other classification algorithms is another issue that needs to be addressed. The accuracy and FPR in the comparison figures indicate that when the feature selection applied (second scenario), the proposed methods still have acceptable values for these two measurements, but LSD and CSD algorithms running faster than the case that testing of the full features are employed (as expected and visible).

7 Conclusions and future work

In this paper, we design an attack and two defense algorithms which target Android malware detection system, namely a Silhouette-based Label Flipping Attack (SCLFA), a label-based semi-supervised defense algorithm (LSD), and a clustering-based semi-supervised defense (CSD) algorithm. We compare our defense algorithms against the KNN-based label flipping attack on Android mobile dataset using *three* public datasets, i.e., the Drebin, Genome, and Contagio datasets, using different API, intent and permission features. We test our models on a CNN classification algorithm. The comparison of proposed CSD and LSD methods against the KSSD method reveals that the proposed methods have higher accuracy than the KSSD, while the KSSD algorithm is faster. To be precise, the CSD algorithm is slightly slower than the KSSD algorithm, but since in many cases, it has approximately 19% higher accuracy than the KSSD and has about 15% lower FPR compared to the KSSD. For future work, we suggest using semi-supervised methods based on deep learning techniques, such as autoencoder and various types of GAN networks. They can be used along with clustering techniques. Using these methods as an ensemble learning can provide excellent results against label flipping attacks.

Acknowledgements Mauro Conti and Mohammad Shojafar are supported by a Marie Curie Fellowship funded by the European Commission (agreement PCIG11-GA-2012-321980 and agreement MSCA-IF-GF-2019-839255, respectively).

Compliance with ethical standards

Conflict of interest There is no conflict of interest for the paper.

Appendix

In this section, we explain why we select CNN classifier as the main classification algorithm applied in the paper. To this end, we tested our attack and defense algorithms against the KSSD [26] and GANX [30] defense algorithms on present various classifications algorithms, namely RF, SVM, DR, NN and CNN, and compute accuracy and FPR metrics for different features to study on different datasets. Table 4 presents the results. As the results presented in this table, with the change of the classification algorithm, there is no significant difference in the superiority of one method to another. However, from this table, we conclude that the accuracy of CNN classification method for all the attack and defense algorithms compared to other classification methods in all datasets for all features is higher. Similarly, the FPR rate of CNN method is lower compared with other classification methods. As a result, *we select CNN method to design our ML model for attack and defense algorithms*.

Table 4 Testing the proposed methods (attacks and defenses) using training classification algorithms tested on API, permission and intent features in various datasets

Algs.	RF		SVM		DT		NN		CNN	
	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR
Drebin										
API										
No-A	98.00	3.81	98.40	2.08	97.85	2.25	97.78	4.58	98.45	2.70
SCLFA	83.03	28.81	83.37	27.58	82.24	27.63	82.75	29.92	83.42	28.08
LSD	90.92	17.91	91.29	16.46	90.74	16.56	90.67	18.87	91.34	17.01
CSD	97.39	3.63	97.78	1.90	97.23	2.07	97.16	4.40	97.83	2.52
KSSD [26]	82.27	31.56	82.61	30.38	82.06	30.42	81.99	32.70	82.66	30.87
GANX [30]	82.82	28.31	82.90	28.38	82.54	27.72	84.67	25.79	83.30	29.07
Permission										
No-A	86.61	39.86	87.02	38.31	87.71	39.02	87.23	38.36	87.07	38.76
SCLFA	74.97	59.97	75.38	58.52	76.23	58.44	75.59	58.88	75.42	58.88
LSD	85.85	45.10	86.26	43.59	85.90	43.04	86.47	43.71	86.30	44.01
CSD	86.59	40.03	87.00	38.49	87.52	39.46	87.21	38.54	87.04	38.93
KSSD [26]	79.31	44.41	79.72	42.88	80.70	42.78	79.93	43.00	79.77	43.31
GANX [30]	79.86	45.69	80.00	40.89	83.23	43.22	82.46	39.68	80.34	43.03
Intents										
No-A	78.26	85.81	78.53	85.37	77.76	86.39	77.93	85.62	78.38	86.10
SCLFA	68.75	87.04	69.01	86.68	68.25	87.54	68.41	86.87	68.86	87.30
LSD	75.50	87.32	75.76	86.93	74.99	87.85	75.16	87.13	75.61	87.59
CSD	77.98	86.18	78.24	85.74	77.48	86.76	77.64	85.98	78.10	86.47
KSSD [26]	70.68	88.66	70.94	88.28	70.17	89.20	70.34	98.94	70.79	88.94
GANX [30]	70.53	92.07	71.22	86.13	72.56	88.16	67.96	98.90	72.82	89.21
Contagio										
API										
No-A	98.45	2.70	98.45	6.72	98.27	6.38	97.11	10.36	97.95	12.90
SCLFA	75.97	9.52	75.79	9.28	74.98	11.17	75.48	15.84	75.62	12.32
LSD	89.28	7.84	89.10	7.54	88.60	8.02	88.78	14.08	88.92	10.60
CSD	98.39	12.61	98.21	12.46	97.42	14.08	97.89	19.06	98.04	15.47
KSSD [26]	78.44	7.28	78.25	6.96	77.67	8.31	77.94	99.72	78.08	10.03
GANX [30]	79.11	9.27	78.60	0.86	80.49	9.46	80.72	99.67	80.43	7.84
Permission										
No-A	98.30	12.02	98.30	8.61	98.87	10.73	97.60	18.10	97.95	15.02
SCLFA	72.09	65.40	72.09	62.61	72.15	68.14	71.39	73.93	71.74	69.67
LSD	92.83	65.40	92.83	62.61	93.29	68.14	92.13	73.93	92.48	69.67
CSD	98.07	12.61	98.07	9.20	98.63	11.36	97.37	18.71	97.72	15.62
KSSD [26]	70.89	64.81	82.30	8.22	72.43	42.89	70.19	73.31	70.54	69.07
GANX [30]	70.86	74.68	82.52	7.38	75.11	43.89	73.93	56.29	70.54	68.69
Intents										
No-A	90.43	92.81	90.43	92.81	91.75	75.07	90.37	85.87	96.84	29.13
SCLFA	80.49	98.11	80.49	98.11	81.80	87.50	80.51	93.75	86.89	61.55
LSD	89.06	93.24	89.06	93.24	90.43	77.00	89.01	86.90	95.47	35.77
CSD	90.11	94.03	90.11	94.03	91.43	76.27	90.05	87.02	96.52	30.54
KSSD [26]	81.92	96.98	81.92	96.98	83.24	86.06	81.93	92.53	88.33	59.18
GANX [30]	82.09	90.09	82.27	92.58	81.51	89.36	84.16	75.34	91.49	49.46

Table 4 (continued)

Algs.	RF		SVM		DT		NN		CNN	
	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR
Gnome										
API										
No-A	99.37	4.08	99.16	4.96	98.59	5.04	98.89	8.02	99.52	2.94
SCLFA	71.97	53.88	71.76	55.37	71.55	50.84	71.49	54.58	72.12	54.20
LSD	84.66	83.45	84.45	86.62	86.46	48.55	84.18	81.48	84.81	85.51
CSD	98.83	6.94	98.62	7.85	97.03	8.02	92.76	46.67	98.98	5.88
KSSD [26]	73.18	52.77	72.97	54.31	76.16	55.70	72.70	53.57	73.33	53.07
GANX [30]	73.40	59.39	73.33	44.49	79.41	60.96	75.81	34.86	73.22	53.42
Permission										
No-A	94.72	54.69	94.93	51.63	93.91	59.24	94.54	62.66	94.57	57.14
SCLFA	92.80	55.51	93.01	52.44	91.99	60.08	92.62	63.52	92.65	57.98
LSD	94.12	55.83	94.34	52.70	93.31	60.52	93.94	64.04	93.97	58.37
CSD	94.57	55.10	94.78	52.03	93.76	59.66	94.39	63.09	94.42	57.56
KSSD [26]	92.53	55.92	92.74	52.85	91.72	60.50	92.35	63.95	92.38	58.40
GANX [30]	93.22	62.34	93.10	43.60	91.98	59.36	92.71	47.73	92.85	57.59
Intents										
No-A	93.19	84.03	92.86	80.45	93.28	83.59	93.04	82.58	99.22	8.78
SCLFA	88.42	91.22	88.09	88.86	88.15	94.07	88.27	90.27	94.45	43.03
LSD	91.21	89.84	90.88	86.69	91.54	90.75	91.06	88.56	97.24	25.00
CSD	91.33	87.82	91.00	84.76	91.53	88.20	91.18	86.58	97.36	24.44
KSSD [26]	90.55	88.86	90.22	86.05	90.13	90.60	90.40	96.07	96.58	30.88
GANX [30]	88.96	94.51	90.58	79.02	90.25	87.26	90.64	95.89	96.55	30.25

No-A No-attack algorithm, *RF* random forest, *NN* neural network, *SVM* support vector machine, *DT* decision tree

References

1. Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K, Siemens C (2014) Drebin: effective and explainable detection of android malware in your pocket. Ndss 14:23–26
2. Aviles-Rivero AI, Papadakis N, Li R, Alsaleh SM, Tan RT, Schonlieb CB (2019) Beyond supervised classification: extreme minimal supervision with the graph 1-Laplacian. arXiv:1906.08635
3. Baracaldo N, Chen B, Ludwig H, Safavi A, Zhang R (2018) Detecting poisoning attacks on machine learning in IoT environments. In: 2018 IEEE international congress on internet of things (ICIOT). IEEE, pp 57–64
4. Bhagoji AN, Cullina D, Mittal P (2017) Dimensionality reduction as a defense against evasion attacks on machine learning classifiers. arXiv:1704.02654
5. Bootkrajang J (2016) A generalised label noise model for classification in the presence of annotation errors. Neurocomputing 192:61–71
6. Bootkrajang J, Kabán A (2014) Learning kernel logistic regression in the presence of class label noise. Pattern Recognit 47(11):3641–3655
7. Bootkrajang J, Kabán A (2012) Label-noise robust logistic regression and its applications. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 143–158
8. Contagio Dataset (2020) http://contagiominidump.blogspot.com/. Accessed 22 Feb 2020
9. Dong-DongChen W, WeiGao ZH (2018) Tri-net for semi-supervised deep learning. In: Proceedings of twenty-seventh international joint conference on artificial intelligence, pp 2014–2020
10. Frénay B, Kabán A et al (2014) A comprehensive introduction to label noise. In: ESANN, pp 667–676
11. Ganin Y, Ustinova E, Ajakan H, Germain P, Larochelle H, Laviolette F, Marchand M, Lempitsky V (2016) Domain-adversarial training of neural networks. J Mach Learn Res 17(1):2030–2096
12. Guo B, Tian L, Zhang J, Zhang Y, Yu L, Zhang J, Liu Z (2019) A clustering algorithm based on joint kernel density for millimeter wave radio channels. In: 2019 13th European conference on antennas and propagation (EuCAP). IEEE, pp 1–5
13. He K, Sun J (2015) Convolutional neural networks at constrained time cost. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5353–5360
14. Hirakawa K, Parks TW (2005) Adaptive homogeneity-directed demosaicing algorithm. IEEE Trans Image Process 14(3):360–369
15. Iscen A, Tolias G, Avrithis Y, Chum O (2019) Label propagation for deep semi-supervised learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5070–5079
16. Jiang X, Zhou Y (2012) Dissecting android malware: characterization and evolution. In: Proceedings of IEEE S&P, pp 95–109

17. Kaiser Ł, Sutskever I (2015) Neural gpus learn algorithms. arXiv:1511.08228
18. Label Propagation (2020) https://scikit-learn.org/stable/modules/label_propagation.html. Accessed 22 Feb 2020
19. Laishram R, Phoha VV (2016) Curie: a method for protecting SVM classifier from poisoning attack. arXiv:1606.01584
20. Li YF, Zhou ZH (2014) Towards making unlabeled data never hurt. IEEE Trans Pattern Anal Mach Intell 37(1):175–188
21. Maclaurin D, Duvenaud D, Adams R (2015) Gradient-based hyperparameter optimization through reversible learning. In: International conference on machine learning, pp 2113–2122
22. Muñoz-González L, Biggio B, Demontis A, Paudice A, Wongrassamee V, Lupu EC, Roli F (2017) Towards poisoning of deep learning algorithms with back-gradient optimization. In: Proceedings of the 10th ACM workshop on artificial intelligence and security. ACM, pp 27–38
23. Mutual Information (2020) https://nlp.stanford.edu/IR-book/html/htmledition/mutual-information-1.html. Accessed 22 Feb 2020
24. Natarajan N, Dhillon IS, Ravikumar PK, Tewari A (2013) Learning with noisy labels. In: Advances in neural information processing systems, pp 1196–1204
25. Papernot N et al (2016) Distillation as a defense to adversarial perturbations against deep neural networks. In: Proceedings of IEEE S&P, pp 582–597
26. Paudice A, Muñoz-González L, Lupu EC (2018) Label sanitization against label flipping poisoning attacks. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 5–15
27. Rand WM (1971) Objective criteria for the evaluation of clustering methods. J Am Stat Assoc 66(336):846–850
28. Ren M, Zeng W, Yang B, Urtasun R (2018) Learning to reweight examples for robust deep learning. arXiv:1803.09050
29. Shafahi A, Huang WR, Najibi M, Suciu O, Studer C, Dumitras T, Goldstein T (2018) Poison frogs! targeted clean-label poisoning attacks on neural networks. In: Advances in neural information processing systems, pp 6103–6113
30. Taheri R, Javidan R, Shojafar M, Conti M et al (2020) Can machine learning model with static features be fooled: an adversarial machine learning approach. Cluster Comput. https://doi.org/10.1007/s10586-020-03083-5
31. Taheri R, Shojafar M (2020) Source code of label flipping attack/defenses on android data. https://github.com/mshojafar/sourcecodes/blob/master/Taheri%20et%20al-NCAA2020.zip. Accessed 22 Feb 2020
32. Wang Y, Chaudhuri K (2018) Data poisoning attacks against online learning. arXiv:1808.08994
33. Xia Y, Liu F, Yang D, Cai J, Yu L, Zhu Z, Xu D, Yuille A, Roth H (2018) 3D semi-supervised learning with uncertainty-aware multi-view co-training. arXiv:1811.12506
34. Xiao H, Biggio B, Nelson B, Xiao H, Eckert C, Roli F (2015) Support vector machines under adversarial label contamination. Neurocomputing 160:53–62
35. Xiao H, Biggio B, Brown G, Fumera G, Eckert C, Roli F (2015) Is feature selection secure against training data poisoning? In: International conference on machine learning, pp 1689–1698
36. Yang C, Wu Q, Li H, Chen Y (2017) Generative poisoning attack method against neural networks. arXiv:1703.01340
37. Zhang F, Chan PP, Biggio B, Yeung DS, Roli F (2016) Adversarial feature selection against evasion attacks. IEEE Trans Cybernet 46(3):766–777
38. Zhou Y, Kantarcioglu M, Thuraisingham B, Xi B (2012) Adversarial support vector machine learning. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 1059–1067