sensors

MDPI

Article

Adversarial Samples on Android Malware Detection Systems for IoT Systems

Xiaolei Liu [1], Xiaojiang Du [2], Xiaosong Zhang [1], Qingxin Zhu [1], Hao Wang [3,*] and Mohsen Guizani [4]

[1] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China; liuxiaolei@uestc.edu.cn (X.L.); johnsonzxs@uestc.edu.cn (X.Z.); qxzhu@uestc.edu.cn (Q.Z.)
[2] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA; dux@temple.edu
[3] Department of Computer Science, Norwegian University of Science and Technology, 7491 Trondheim, Norway
[4] Department of Computer Science and Engineering, Qatar University, Doha 2713, Qatar; mguizani@ieee.org
* Correspondence: hawa@ntnu.no

Received: 22 December 2018; Accepted: 20 February 2019; Published: 25 February 2019

Abstract: Many IoT (Internet of Things) systems run Android systems or Android-like systems. With the continuous development of machine learning algorithms, the learning-based Android malware detection system for IoT devices has gradually increased. However, these learning-based detection models are often vulnerable to adversarial samples. An automated testing framework is needed to help these learning-based malware detection systems for IoT devices perform security analysis. The current methods of generating adversarial samples mostly require training parameters of models and most of the methods are aimed at image data. To solve this problem, we propose a testing framework for learning-based Android malware detection systems (TLAMD) for IoT Devices. The key challenge is how to construct a suitable fitness function to generate an effective adversarial sample without affecting the features of the application. By introducing genetic algorithms and some technical improvements, our test framework can generate adversarial samples for the IoT Android application with a success rate of nearly 100% and can perform black-box testing on the system.

Keywords: Internet of Things; malware detection; adversarial samples; machine learning

1. Introduction

Since many IoT (Internet of Things) devices run Android systems or Android-like systems, with the popularity of IoT devices, Android malware for IoT devices is also increasing. Meanwhile, machine learning has received extensive attention and has gained tremendous application development in many fields, such as financial economics, driverless, medical, and network security. Thus, there are many learning-based Android malware detection systems [1–6].

However, while machine learning brings us great convenience, it also exposes a lot of security problems [7]. Several papers have studied related Android and IoT security issues [8–13]. Scholars in the security field are increasingly concerned about the security issues associated with the lack of fairness and transparency in machine learning algorithms. An attacker can predict certain sensitive information by observing the model, or recover sensitive data in the data set through existing partial data. A current attack method is called a poisoning attack. Biggio B and Zhu attempted to attack the adaptive facial recognition system by a poisoning attack [14–17]. During the model update, they injected malicious data to offset the central value of the recognition feature in the model, so as to achieve the purpose of verifying the attacker's image. Biggio B and Nelson B also attacked the supervised learning algorithm SVM [18]. Experiments show that the test error of the model classifier can be significantly increased during the gradient rise. However, the injected sample data must meet certain constraints in order to deceive the model, and must be the attacker to control the label of the injection point. Yang et al. conducted an experiment on poisoning attacks against neural network learning algorithms [19]. Compared with the direct gradient algorithm, this proposed method can increase the attack sample generation speed by about 240 times.

In fact, although a poisoning attack can make the model go wrong, the attacker has to work hard on how to inject malicious data. Another common method can let models get the wrong result in a short time, that is, adversarial sample attack. Christian Szegedy et al. first proposed the concept of adversarial samples [20]. By deliberately adding minor changes in the dataset, the perturbed samples will cause the model to output a false result with high confidence. Adversarial samples can increase the prediction error of the model, so that the originally correctly classified sample migrates to the other side of the decision area, thereby being classified into another category.

Existing models are vulnerable to adversarial samples [21–24]. For example, in a malware recognition system, by adding a small perturbation to the original software sample, the result of the sample classification can be changed with a high probability, and even the sample can be classified into an arbitrarily designated label according to the attacker's idea. This makes adversarial samples attack a huge hazard to malware recognition systems [25–27].

All of the learning-based Android malware detection systems for IoT devices have the above problems, so a testing framework is needed to test the robustness of these detection systems. To address this challenge, we propose TLAMD, a testing framework for learning-based Android malware detection systems for IoT Devices. When the test results show that the detection system cannot resist the attack of the adversarial samples, it indicates that this detection system has potential safety hazards, and it needs to be reinforced.

Therefore, how to generate effective adversarial samples is the core issue of the entire testing framework. Our approach to generating adversarial samples for the Android IoT malware detection model is based on genetic algorithms. Without the knowledge of the model parameters, the original sample is used as the input of the approach, and finally the adversarial sample of the specific label is generated. The information used is only the probability of the various types of labels output by the model. We hope that this method can be a robust benchmark for the learning-based Android malware detection model for IoT devices. Our contribution is mainly reflected as follows:

1. We migrated the application of adversarial samples from the image recognition domain to the Android malware detection domain of IoT devices. In this process, simply replacing the model's training data from a picture to an Android application is not possible. On the one hand, the data of the binary program is not continuous like the image data. On the other hand, random perturbation of the binary program may lead to the crash of the program, so special processing is required for the Android application to ensure the validity of the adversarial samples. We borrowed the processing method of Kathrin Grosse [28], which realized the disturbance to the Android application by adding the request permission code in the $AndroidManifest.xml$ file. The difference is that we have made corresponding analysis and restrictions on the types and quantities of permissions that can be added. This method can ensure that the original function of the app is not affected and can be used normally; and the app can be disturbed in the simplest way to achieve the effect of changing the model detection result;
2. We introduce the genetic algorithm into the adversarial sample generation method and implement the black-box attack against the machine learning model. Without knowing the internal parameters such as the gradient and structure of the target network, it is only necessary to know the probability of various types of labels output by the model. Compared to Kathrin Grosse's approach, our approach not only implements black-box attacks, but also has a higher success rate, almost 100%.

The rest of the paper is organized as follows. Section 2 introduces the related background of our approach. Section 3 presents TLAMD (A Testing Framework for Learning-based Android Malware Detection Systems for IoT Devices). Section 4 presents and discusses our experimental results. Finally, further discussions and conclusions are accomplished in Section 5.

2. Related Background

2.1. Neural Network

The essence of the neural network is a function $y = F(x)$, the input x is an n-dimensional vector, and the output y is an m-dimensional vector. The function F implies the model parameter θ. The purpose of the training network is to calculate the value of the parameter θ from the known partial sample information. After the model is completed, the result of predicting x is to solve the value of y by the function F. In this paper, we mainly study the neural network of the m classifier (that is, the output y is an m-dimensional vector). The output of the last layer of the neural network uses a fully connected layer. The classifier outputs the index with the largest value in the output vector dimension as the result, that is:

$$L(x) = arg \max_{j=1} [F(x)]_i, \tag{1}$$

where $L(x)$ is the category of x.

Define F as a single-layer fully-connected neural network. The output of the $(n-1)$-th layer is the input of the n-th layer, then:

$$y_n = F_n(y_{n-1}). \tag{2}$$

Typical n-layer fully connected neural networks are:

$$F = F_n * F_{n-1} * ... * F_2 * F_1, \tag{3}$$

$$F_n(x) = \sigma(w_n * x + b_n), \tag{4}$$

where σ is a linear or nonlinear activation function. The commonly used activation functions are RELU [29], tanh [30], sigmoid, etc., ω is the weight of this layer, and b is the layer offset.

2.2. Genetic Algorithm

The idea of the genetic algorithm is to simulate the biological evolution process of natural selection. Using the thought of evolutionary theory, the process of finding the optimal solution of a certain objective function is simulated into the evolution process of the population. Based on the idea of the population, the algorithm uses a population containing information to perform an optimal search in multiple directions and completes the exchange and reconstruction of information in the search process.

The genetic algorithm can be used to search for the feasible solution space of a problem, and then find the possible optimal solution, which is the uncertainty optimization in the optimization problem. Uncertain optimization is to rely on random variables in the search direction, rather than a certain mathematical expression. Compared with other algorithms, the advantage is that, when the optimization converges to the local extremum, the search result can jump out of an optimal solution and continue to search for a better feasible solution.

By choosing the appropriate objective function, the generation of the adversarial sample can be transformed into a solution to the optimization problem. The process of solving the optimal solution corresponding to the objective function is actually the process of generating the adversarial sample. This shows that genetic algorithms can be effectively applied to machine learning and other fields in terms of parameter optimization and function solving. In terms of parameter optimization, Chen et al. used a parallel genetic algorithm to optimize the parameter selection of Support Vector Machine (SVM) [31]. Experiments show that the proposed method is superior to the grid search in classification accuracy, the number of selected features and running time. Phan et al. proposed a GA-SVM model that can effectively improve classification performance based on genetic algorithm and SVM classifier [32]. Alejandre et al. selected features based on machine learning to detect botnets [33]. A genetic algorithm is used in this method to select the set of features that provide the highest detection rate.

2.3. Adversarial Samples

On many machine learning models, the decision boundary of the classifier has a certain margin of error. That is, when the disturbance satisfies $||\eta||_\infty < \epsilon$, the classifier considers that the perturbed input $x' = x + \eta$ is the same as the original input x. Therefore, when the perturbation value on each feature element is less than ϵ, the classifier cannot discern the difference in the sample. However, changes in input characteristics have a cumulative effect on model predictions. Although the perturbation value on each feature element is small, the accumulated error is sufficient to influence the model prediction result.

On each neuron, the adversarial sample will have the following operations:

$$\omega^T x' = \omega^T (x + \eta) \tag{5}$$

although the adversarial sample has no effect on the classification results of the single-dimensional neuron classifier. However, deep learning has a considerable number of neurons. The weight in each neuron has n dimensions. If the average variation of an element in the weight vector is m, the activation effect will increase by $n * m$. Furthermore, in a high dimensional linear classifier, each individual input feature is normalized. The result is that in the process of deep learning, a small change may not be enough to change the input result, but multiple disturbances to the input will cause the classifier to make a wrong classification result.

Many methods of generating adversarial samples need to know the parameters of the learning model to calculate the perturbation values, but some subsequent studies have shown that without knowing the parameters of the learning model [34–37]. The attacker can interact with the black-box learning model to calculate the samples. Specifically, the attacker can estimate the boundary of the decision region of the model according to the difference of the model output brought by different samples, and then use the estimated boundary as a substitute model. Finally, the adversarial samples are calculated by the parameters of the substitute model. Considering that more and more malicious Android application detection methods based on machine learning, how to evaluate the robustness of these detection methods becomes a new problem. Since most machine learning algorithms are vulnerable to adversarial samples, we have thought of using the generated adversarial samples to test the robustness of these detection methods.

3. Methodology

3.1. Framework

The overview of TLAMD is shown in Figure 1.

When the test results show that the detection system cannot resist the attack against the adversarial sample, it indicates that the system has potential safety hazards and it is necessary to implement such reinforcement measures as distillation defense [38] on the detection system. As we can see, how to generate an adversarial sample is the main challenge of this testing framework. Therefore, we will describe the algorithm in detail for generating an adversarial sample for Android malware.

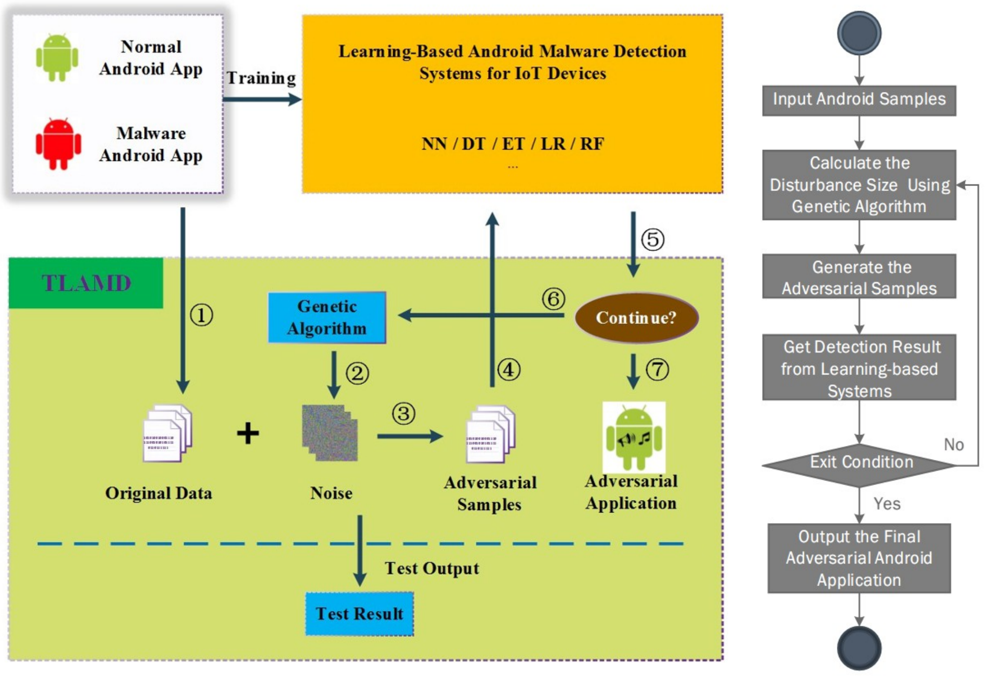
Normal Android App
Malware Android App
Training
Learning-Based Android Malware Detection Systems for IoT Devices
NN / DT / ET / LR / RF
...
TLAMD
①
Genetic Algorithm
②
③
④
⑤
⑥
Continue?
⑦
Original Data
Noise
Adversarial Samples
Adversarial Application
Test Output
Test Result
Input Android Samples
Calculate the Disturbance Size Using Genetic Algorithm
Generate the Adversarial Samples
Get Detection Result from Learning-based Systems
Exit Condition
No
Yes
Output the Final Adversarial Android Application

Figure 1. Overview of our testing framework for learning-based Android Malware detection systems for IoT devices. (1) Original Sample Input; (2) Calculate the disturbance size; (3) Generate the adversarial samples; (4) Get detection result from learning-based systems; (5) Determine if the exit condition is met; (6) If not, calculate the new disturbance size using genetic algorithm; (7) If yes, output the final adversarial android application.

3.2. Algorithm

Our goal is to add minor perturbations to the malware without changing the malware functionality, so that the previously trained detection model misidentifies it as normal software. Therefore, our approach generates an adversarial sample by adding permission features to the $AndroidManifest.xml$, and in order not to affect the function of the original malware, the disturbance does not reduce the existing permission features. For a single input sample X, the classifier returns a two-dimensional vector $F(X) = [F_0(X), F_1(X)]$, where $F_0(X)$ indicates the probability that the software is a normal software, $F_1(X)$ indicates the probability that the software is a malware, and satisfies the constraint $F_0(X) + F_1(X) = 1$. We aim to add a perturbation δ to make the classification result $F_1(X+\delta)$ is less than $F_0(X+\delta)$. At the same time, the smaller the δ, the better, that is, the fewer the number of permission features added in the manifest file, the better. For example, for a specific malware x, we use a genetic algorithm to find out which permission features δ are added to x, and finally make x detected as normal software with minimum number of permission added.

From a mathematical point of view, the process of misjudging the detection model by adding the permission features is regarded as a problem to be solved. The feasible solution space of the problem is the disturbance if the detection model is successfully misjudged. The optimal solution is to minimize the disturbance value, that is, add the least permission feature. A genetic algorithm is a type of algorithm that finds the possible optimal solution by searching for a feasible solution space of a problem. Our approach is to use genetic algorithms to search for the minimum perturbation value that causes the detection model to be misjudged. The pseudo code of our approach is shown in Algorithm 1.

Algorithm 1 Generating an adversarial sample.

```
Require: Popluation Size $pop\_size$
  $\delta \leftarrow initialization()$
  for $i = 0 \rightarrow pop\_size$ do
    $P_i \leftarrow Crossover\_Operator()$
    $P_i \leftarrow Mutation\_Operator()$
    Compute $\rightarrow S(\delta)$
    if $F(X+\delta) > 1 - F(X+\delta)$ then
      Continue
    else
      Output $\rightarrow \delta$
    end if
  end for
```

The specific steps are as follows:

(1) Randomly generate the population $\delta = P_1, P_2, ..., P_M$. M is the number of individuals, the individual $P_i \in \{0,1\}^n$ refers to the permission characteristics to be added in the category, and n is the number of permission features in the category. In addition, 1 means to add the corresponding permission; otherwise, 0 means not to add. Our strategy is to only add permissions and not reduce permissions. Therefore, if the original malicious sample has a certain permission feature, the permission cannot be removed, that is, the disturbance is 0.

(2) Determine the fitness function.

$$S(\delta) = \min\ w_1 \cdot F(X+\delta) + w_2 \cdot num(\delta), \tag{6}$$

where w_1 and w_2 represent the two weights, δ is the added small disturbance, $F(X+\delta) \in [0,1]$ means that the probability of original malicious sample is still detected as a malware, $num(\delta_i)$ indicates the number of permission features added.

When w_1 is much larger than w_2, the sample after the addition of the disturbance must be detected as normal by the detection model to survive, and the individual detected as a malicious sample will be eliminated. The surviving individual must meet the minimum number of added permission features; otherwise, it will also be eliminated. The fitness function defined in this way searches for an optimal solution that can successfully cause the detection model to be misjudged.

(3) Perform mutation operations according to a certain probability to generate new individuals. The mutation refers to adding a disturbance to the corresponding category according to a certain probability, that is, changing the value from 0 to 1, and satisfying the constraint proposed in step (1).

(4) Generate a new generation of the population from the mutation and return to step (2). If the preset number of iterations is reached, the loop is exited.

4. Experiments

4.1. Data Set and Environment

In order to verify the effectiveness of the adversarial sample, we attempt to train five different classifier models, including logistic regression (LR), decision tree (DT), and fully connected neural network (NN) and so on. The hardware environment and software environment of all experiments are shown in Table 1:

All the data we use in the experiments come from the DREBIN dataset [39,40]. The DREBIN dataset has a total of 123,453 sample data for Android applications, including 5560 malicious samples and contains as many as 545,333 behavioral features.

The features of the Android app in this dataset consist of eight categories and are shown in Table 2:

(S1) Hardware components, which are used to set the hardware permissions required by the software.
(S2) Requested permissions, which are granted by the user at the time of installation and allow the application software to access the corresponding resources.
(S3) App components, which include four different types of interfaces: activities, services, content providers and broadcast receivers.
(S4) Filtered intents, which are used for process communication between different components and applications.
(S5) Restricted API (Application Programming Interface) calls, access to a series of key API calls.
(S6) Used permissions, a subset of permissions that are actually used and requested in S5.
(S7) Suspicious API calls, API calls for allowing access to sensitive data and resources.
(S8) Network addresses, the IP addresses accessed by the application, including the hostname and URL.

Table 1. The environment of all experiments.

CPU	Inter(R) Core(TM) i5-7400 CPU @ 3.00GHz
Memery	8 GB
Video Card	Inter(R) HD Graphics 630
Operating System	Windows 10
Programming Language	Python 3.6
Development Platform	Jupyter Notebook
Dependence	Tensorflow, Keras, numpy etc.

The first four classes are extracted from the manifest file, and the last four classes are extracted from the disassembly code. Since our method only adds permission requests to the $AndroidManifest.xml$ file, we only cover the features in S1 to S4. In Section 4.2.1, we further reduce the feature categories used.

Table 2. Eight features in the DREBIN dataset.

Class	Name	Numbers	Rate (/Total)
S1	Hardware Components	72	0.013%
S2	Requested Permissions	3812	0.704%
S3	App Components	218,951	40.488%
S4	Filtered Intents	6379	1.178%
S5	Restricted API Calls	733	0.136%
S6	Used Permissions	70	0.013%
S7	Suspicious API Calls	315	0.058%
S8	Network Address	310,447	57.4%

4.2. Android Malware Detection Model

First, a detection model is trained to determine whether an Android sample is malware. When the detection model reaches a certain accuracy, our approach is used to generate an adversarial sample for the model.

4.2.1. Feature Extraction

We use a random forest approach to measure the importance of features in the feature extraction phase. The number of features is effectively reduced without affecting the accuracy of detection.

Random forest is an integrated learning in machine learning. It is an integrated classifier composed of multiple sets of decision trees: $h(X, \theta_k), k = 1, 2, ...,$ where θ_k is a random variable subject to independent and identical distribution, and k represents the number of decision trees. The principle is to generate multiple decision trees and let them learn independently and make corresponding predictions. Finally, observe which category is selected the most and get the result.

The specific steps are as follows:

(1) Select out of bag (OOB) to calculate the corresponding out-of-bag data deviation $error_1$ for each decision tree.
(2) Add random noise, perturb all samples of OOB, and then calculate the out-of-bag data deviation $error_2$ again.
(3) Define and calculate the importance of the features:

$$I = \sum(error_1 - error_2)/N, \tag{7}$$

where N is the number of forest decision trees.

If $error_2$ is greatly increased after adding random noise, the OOB accuracy rate decreases, indicating that this type of feature has a greater impact on the prediction result, that is, the importance is higher.

The sorting result of feature importance is shown in Figure 2.

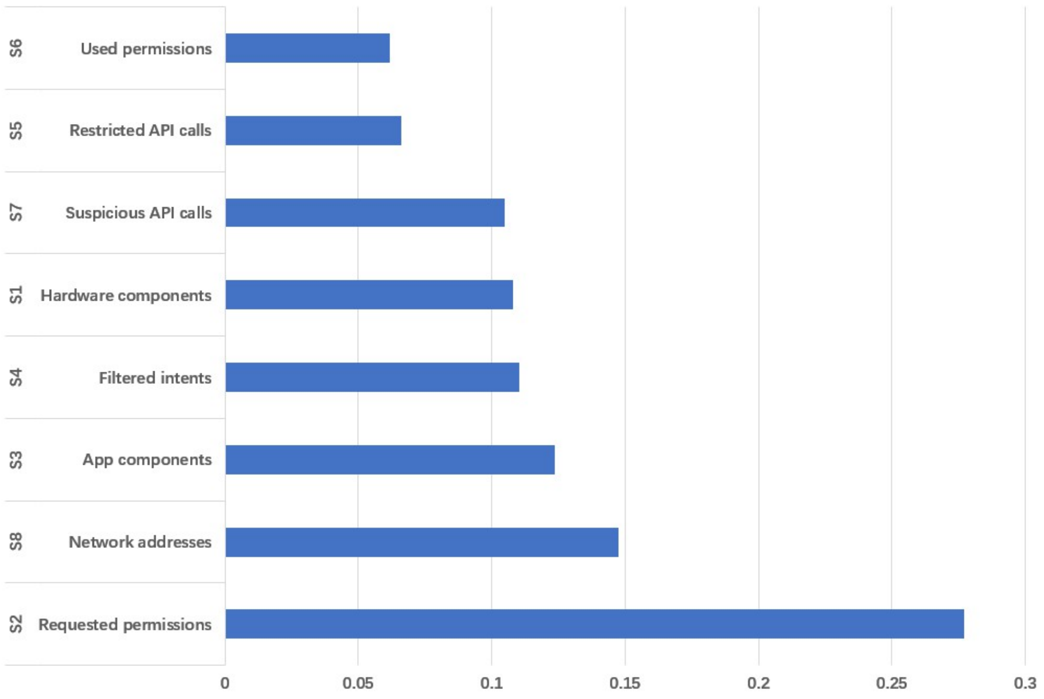

Figure 2. The sorting result of feature importance. The ordinate represents different behavioral feature categories and the abscissa represents the proportion of importance.

As we mentioned before, we only cover the four types of features from S1 to S4. Taking into account the number and importance of various features, we finally choose the two characteristics of S1 and S2.

4.2.2. Training Detection Model

To test the effectiveness of our method for different detection models, we trained five kinds of detection models.

a. Neural Network

Our neural network chooses a two-layer fully connected model with 200 neurons in each connected layer and the activation function is RELU. The output layer of the last layer has two neurons and is the $soft$ max activation unit. Furthermore, no dropout operation is performed on each layer. To train our network, we used the gradient descent training method with a batches size of 256. All data was trained five times per iteration.

b. Logistic Regression

Since there are only two types of target predictions, we adopt a two-class logistic regression model. The penalty term selects the L2 paradigm, and the model parameters satisfy the Gaussian distribution, that is, the parameters are constrained so that they do not over-fitting. Considering that the solution problem is not a linear multi-core, and the number of samples is selected to be larger than the number of features, the dual method is not set. Set the condition for stopping the solution is that the loss function is less than or equal to $1 \times e^{-4}$; the category weight defaults to 1. The maximum number of iterations of the algorithm convergence is set to 10.

c. Decision Tree

The decision tree is a tree structure used for classification. The maximum depth of the decision tree is set to 15 to prevent overfitting. The *min_impurity_decrease* is set to 0. The *min_samples_split* is set to 2, indicating the minimum number of samples required for internal node subdivision. The $min\_samples\_leaf$ is set to 10, indicating the minimum number of samples in the leaf node. The max_leaf_nodes is set to None, which is expressed as the maximum number of leaf nodes in the decision tree. The $min\_weight\_fraction\_leaf$ is set to 0, which represents the minimum value of all sample weights and sums of leaf nodes.

d. Random Forest

Random forest is an integrated learning. Through the bootstrap resampling technique, a number of sample inputs are randomly selected from the original training set with repeated iterations. In this way, a new training set is obtained, and then several decision trees are generated to form a random forest. The $max\_feature$ is set to auto, that is, a single decision tree can utilize all permission features. The *n_estimators* is set to 20, which means there are 20 decision trees to form the random forest to be trained. The min_sample_leaf is set to 20, that is, the minimum number of sample leaves in each decision tree is 20.

e. Extreme Tree

Extra Tree is equivalent to a variant of the random forest. Compared with random forests, the randomness is further calculated when dividing the local best, that is, the selection of the division points is calculated. Most of its parameters are the same as those of random forests, except that *n_estimators* is set to 10 and *max_depth* is set to 50.

Finally, when the five detection models are trained, we test 42,570 samples and the results are shown in Table 3.

Table 3. The detection results of five models.

Models	TP[a]	FP [a]	FN [a]	TN [a]	Accuracy	Precision	Recall
NN (Neural Network)	40770	0	74	1726	99.83%	1	95.95%
LR (Logistic Regression)	40770	0	234	1566	99.45%	1	96.32%
DT (Decision Tree)	40770	0	60	1740	99.86%	1	95.91%
RF (Random Forest)	40770	0	32	1768	99.92%	1	95.85%
ET (Extreme Tree)	40770	0	16	1784	99.96%	1	95.81%

[a] TP = True Positive, FP = False Positive, FN = False Negative, TN = True Negative.

4.3. Simulation Experiments

After getting the trained detection models, we will generate adversarial samples for the five models. The features we add to the $AndroidManifest.xml$ file are from S1 or S2. The parameters of the generation algorithm are also different depending on the permission category. The details are as shown in Table 4.

Table 4. The parameters of our approach.

Features	S1: Hardware Components	S2: Requested Permissions
Initialize Probability	1%	0.01%
Mutation Probability	30%	0.5%
Iterations	50	50
Population	150	150
Attacked Samples	1000	1000

The final experimental results are shown in Table 5. In the ten sets of adversarial sample generation experiments for the five detection models, the success rates are above 80%, and most of them are close to 100%. In order to generate these adversarial samples, the average number of permission features added is less than three. On the one hand, it shows that the adversarial sample generated by our method is very effective and our approach is able to be a robust benchmark for the learning-based Android malware detection model for IoT devices; on the other hand, it shows that the existing machine learning algorithms are very vulnerable to the adversarial sample. Our TLAMD test framework is very necessary.

In subsequent experiments, we also performed a reinforcement method for the distillation defense of these models. However, the reinforced model is still unable to resist the attack of adversarial samples, and the success rate of our approach is still close to 100%. This means that, when we want to reinforce existing machine learning models, common methods such as distillation defenses work poorly. We need to find a more effective defense method.

Figure 3 shows the most frequently added permissions in the ten sets of adversarial sample generation experiments for the five kinds of detection models. Compared to other permission features, these permissions are mostly permissions that involve sensitive privacy. In order to verify whether these features have a decisive influence on the model discrimination results, we have conducted further experiments. In the new experiment, we will not allow the algorithm to add the features listed in the figure. However, the success rate of the generated adversarial samples is consistent with the previous one in Table 5, and the number of permission features added is slightly increased. It can be seen that those features that are added more frequently only have greater weight, but have no decisive influence on the results.

Table 5. The results of our approach. [b]

Model	Category	Success Rate	Average of num (δ)
NN	S1	1	2.25
	S2	1	2.33
LR	S1	0.998	2.66
	S2	0.995	1.94
DT	S1	0.896	1.05
	S2	0.992	1.68
RF	S1	0.866	2.89
	S2	0.995	9.54
ET	S1	0.833	2.81
	S2	0.945	9.36

[b] Each line of data in the table is the average of the 1000 sample tests results.

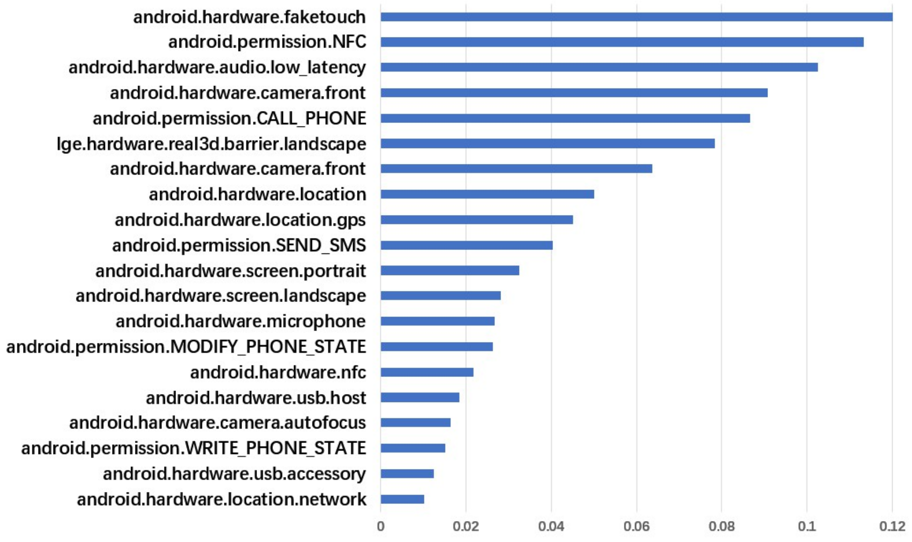

Figure 3. The most frequently added permissions in our adversarial sample generation experiments. The data is the average of 5 × 2 × 1000 samples test results.

Figure 4 is a trend graph of fitness function values as a function of the number of iterations. As the number of iterations increases, the value of the fitness function decreases rapidly. It shows that it is very effective to use the genetic algorithm to solve the problem of generating adversarial samples.

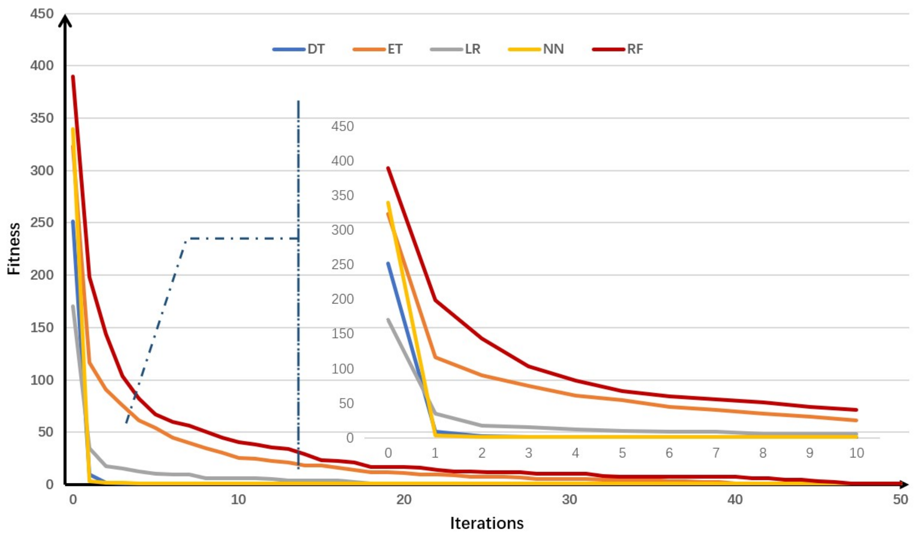

Figure 4. Trend graph of fitness function values with number of iterations.

By comparing the individual models, it can be found that the more complex the detection model, the better the effect of the adversarial samples generated for the model. This phenomenon may be different from what we expected. We believe that one possible reason is that the more complex the model, the more times the feature is processed. This makes small changes in features easily magnified, making the model very sensitive to adversarial samples.

Figure 5 is a box plot of the fitness function values of adversarial samples for five detection models with S1 permission features and Figure 6 is with S2 permission features. As can be seen from the figures, the adversarial samples generated by our approach is very stable. There are only a very small number of divergence points out of 1000 samples. By comparing Figures 5 and 6, the stability of the adversarial sample generated by S2 is better. The reason is that the number of permission features in the S2 list is much larger than the number in the S1 list. This is equivalent to finding the optimal solution of the objective function in a larger space, so there is a greater probability of finding a better solution. Combined with Figure 3, it also provides us with an idea of how to strengthen the learning-based detection model. It is not useful to improve the defense of high-weight permission features. It is necessary to optimize the detection model so that it is not sensitive to small disturbances of all sample features.

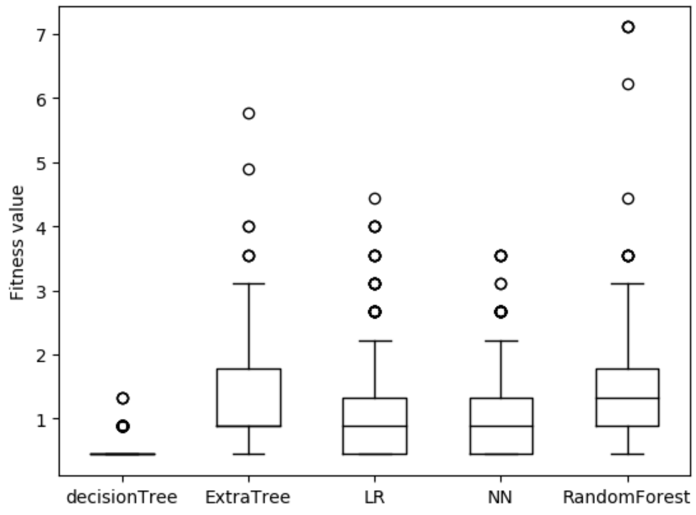

Figure 5. The fitness function values of adversarial samples for five detection models with S1 permission features.

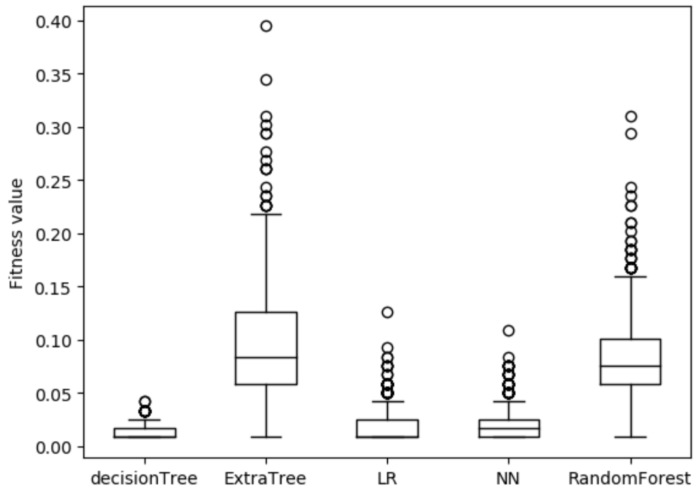

Figure 6. The fitness function values of adversarial samples for five detection models with S2 permission features.

5. Conclusions

To address the challenge of the lack of the testing framework for learning-based Android malware detection systems for IoT devices, we approach TLAMD. Our experimental results show that our approach generates high-quality adversarial samples with a success rate of nearly 100% by adding permission features. In the technical implementation of the TLAMD algorithm, the selection of feature and the range of disturbance are the keys to have a good result. We hope TLAMD can be a benchmark for learning-based IoT Android malware detection model. The limitations of TLAMD is our black-box approach need frequent model requests and our future work includes reducing the requesting times and designing an effective defense approach to reinforce the malware detection model.

Author Contributions: Data curation, H.W.; Formal analysis, X.L.; Investigation, X.L.; Methodology, X.L.; Supervision, Q.Z.; Validation, M.G.; Writing—original draft, X.L.; Writing—review and editing, X.Z. and X.D.

Funding: This research was funded by the National Natural Science Foundation of China under Grant No. 61672170, No. 61871313 and No. 61572115, in part by the National Key R&D Plan under Grant CNS 2016QY06X1205.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ham, H.S.; Kim, H.H.; Kim, M.S.; Choi, M.J. Linear SVM-based android malware detection for reliable IoT services. *J. Appl. Math.* **2014**, *2014*, 594501. [CrossRef]
2. McNeil, P.; Shetty, S.; Guntu, D.; Barve, G. SCREDENT: Scalable Real-time Anomalies Detection and Notification of Targeted Malware in Mobile Devices. *Procedia Comput. Sci.* **2016**, *83*, 1219–1225. [CrossRef]
3. Aswini, A.; Vinod, P. Towards the Detection of Android Malware using Ensemble Features. *J. Inf. Assur. Secur.* **2015**, *10*, 9–21.
4. Odusami, M.; Abayomi-Alli, O.; Misra, S.; Shobayo, O.; Damasevicius, R.; Maskeliunas, R. Android Malware Detection: A Survey. In Proceedings of the International Conference on Applied Informatics Applied Informatics, Bogota, Colombia, 1–3 November 2018; Florez, H., Diaz, C., Chavarriaga, J., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 255–266.
5. Misra, S.; Adewumi, A.; Maskeliūnas, R.; Damaševičius, R.; Cafer, F. Unit Testing in Global Software Development Environment. In Proceedings of the International Conference on Recent Developments in Science, Engineering and Technology, Gurgaon, India, 13–14 October 2017; Springer: Berlin, Germany, 2017; pp. 309–317.
6. Alhassan, J.; Misra, S.; Umar, A.; Maskeliūnas, R.; Damaševičius, R.; Adewumi, A. A Fuzzy Classifier-Based Penetration Testing for Web Applications. In Proceedings of the International Conference on Information Theoretic Security, Libertad City, Ecuador, 10–12 January 2018; Springer: Berlin, Germany, 2018; pp. 95–104.
7. Barreno, M.; Nelson, B.; Joseph, A.D.; Tygar, J.D. The security of machine learning. *Mach. Learn.* **2010**, *81*, 121–148. [CrossRef]
8. Wu, L.; Du, X.; Wu, J. Effective defense schemes for phishing attacks on mobile computing platforms. *IEEE Trans. Veh. Technol.* **2016**, *65*, 6678–6691. [CrossRef]
9. Cheng, Y.; Fu, X.; Du, X.; Luo, B.; Guizani, M. A lightweight live memory forensic approach based on hardware virtualization. *Inf. Sci.* **2017**, *379*, 23–41. [CrossRef]
10. Hei, X.; Du, X.; Lin, S.; Lee, I. PIPAC: Patient infusion pattern based access control scheme for wireless insulin pump system. In Proceedings of the 2013 Proceedings IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 3030–3038.
11. Du, X.; Xiao, Y.; Guizani, M.; Chen, H.H. An effective key management scheme for heterogeneous sensor networks. *Ad Hoc Netw.* **2007**, *5*, 24–34. [CrossRef]
12. Du, X.; Guizani, M.; Xiao, Y.; Chen, H.H. Transactions papers a routing-driven Elliptic Curve Cryptography based key management scheme for Heterogeneous Sensor Networks. *IEEE Trans. Wirel. Commun.* **2009**, *8*, 1223–1229. [CrossRef]
13. Du, X.; Chen, H.H. Security in wireless sensor networks. *IEEE Trans. Wirel. Commun.* **2008**, *15*, 60–66.
14. Biggio, B.; Fumera, G.; Roli, F.; Didaci, L. Poisoning adaptive biometric systems. In Proceedings of the Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Hiroshima, Japan, 7–9 Novembe 2012; Springer: Berlin, Germany, 2012; pp. 417–425.
15. Biggio, B.; Didaci, L.; Fumera, G.; Roli, F. Poisoning attacks to compromise face templates. In Proceedings of the 2013 International Conference on Biometrics (ICB), Madrid, Spain, 4–7 June 2013; pp. 1–7.

16. Biggio, B.; Fumera, G.; Roli, F. Pattern recognition systems under attack: Design issues and research challenges. *Int. J. Pattern Recognit. Artif. Intell.* **2014**, *28*, 1460002. [CrossRef]
17. Zhu, X. Super-class Discriminant Analysis: A novel solution for heteroscedasticity. *Pattern Recognit. Lett.* **2013**, *34*, 545–551. [CrossRef]
18. Biggio, B.; Nelson, B.; Laskov, P. Poisoning attacks against support vector machines. *arXiv* **2012**, arXiv:1206.6389.
19. Yang, C.; Wu, Q.; Li, H.; Chen, Y. Generative poisoning attack method against neural networks. *arXiv* **2017**, arXiv:1703.01340.
20. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. *arXiv* **2013**, arXiv:1312.6199.
21. Carlini, N.; Wagner, D. Towards evaluating the robustness of neural networks. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 39–57.
22. Moosavi-Dezfooli, S.M.; Fawzi, A.; Fawzi, O.; Frossard, P. Universal adversarial perturbations. *arXiv* **2017**, arXiv:1610.08401.
23. Warde-Farley, D.; Goodfellow, I. 11 adversarial perturbations of deep neural networks. In *Perturbations, Optimization, and Statistics*; MIT Press: Cambridge, MA, USA, 2016; p. 311.
24. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The limitations of deep learning in adversarial settings. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), Saarbrücken, Germany, 21–24 March 2016; pp. 372–387.
25. Chen, S.; Xue, M.; Fan, L.; Hao, S.; Xu, L.; Zhu, H.; Li, B. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *Comput. Secur.* **2018**, *73*, 326–344. [CrossRef]
26. Chen, L.; Hou, S.; Ye, Y.; Xu, S. Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks. In Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), Barcelona, Spain, 28–31 August 2018; pp. 782–789.
27. Al-Dujaili, A.; Huang, A.; Hemberg, E.; O'Reilly, U.M. Adversarial deep learning for robust detection of binary encoded malware. In Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018; pp. 76–82.
28. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial examples for malware detection. In Proceedings of the European Symposium on Research in Computer Security, Oslo, Norway, 11–15 September 2017; pp. 62–79.
29. Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier nonlinearities improve neural network acoustic models. In Proceedings of the 30th International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; Volume 30, p. 3.
30. Mishkin, D.; Matas, J. All you need is a good init. *arXiv* **2015**, arXiv:1511.06422.
31. Chen, Z.; Lin, T.; Tang, N.; Xia, X. A parallel genetic algorithm based feature selection and parameter optimization for support vector machine. *Sci. Progr.* **2016**, *2016*, 2739621. [CrossRef]
32. Phan, A.V.; Le Nguyen, M.; Bui, L.T. Feature weighting and SVM parameters optimization based on genetic algorithms for classification problems. *Appl. Intell.* **2017**, *46*, 455–469. [CrossRef]
33. Alejandre, F.V.; Cortés, N.C.; Anaya, E.A. Feature selection to detect botnets using machine learning algorithms. In Proceedings of the 2017 International Conference on Electronics, Communications and Computers (CONIELECOMP), Cholula, Mexico, 22–24 February 2017; pp. 1–7.
34. Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z.B.; Swami, A. Practical black-box attacks against machine learning. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Abu Dhabi, UAE, 2–6 April 2017; pp. 506–519.
35. Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z.B.; Swami, A. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv* **2016**, arXiv:1602.02697.
36. Papernot, N.; McDaniel, P.; Goodfellow, I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv* **2016**, arXiv:1605.07277.
37. Narodytska, N.; Kasiviswanathan, S.P. Simple Black-Box Adversarial Attacks on Deep Neural Networks. In Proceedings of the CVPR Workshops, Honolulu, HI, USA, 21–26 July 2017; pp. 1310–1318.

38. Papernot, N.; McDaniel, P.; Wu, X.; Jha, S.; Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 23–25 May 2016; pp. 582–597.
39. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 23–26 February 2014; Volume 14, pp. 23–26.
40. Spreitzenbarth, M.; Freiling, F.; Echtler, F.; Schreck, T.; Hoffmann, J. Mobile-sandbox: having a deeper look into android applications. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, Coimbra, Portugal, 18–22 March 2013; pp. 1808–1815.