JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Android Platform Based Malicious Adware Detection Using SPAARC Tree

[1]Sagar Raj Patil, [2]Rajni Ranjan Singh Makwana,

[1]Mtech Scholar, [2]Asistant Professor,
[1]Dept of CSE/IT, [2]Dept of CSE/IT,
[1]Madhav Institute of Technology and Science, Gwalior, India

Abstract : **Recently, the adoption, as well as usage of smartphones, have risen, making them the primary mode of communication. The majority of Android smartphone applications, or software, are free; but, to earn income, adverts are shown while an app is being accessed. Through numerous types of malware attacks, such as Advertising Software (Adware), attackers are continually watching cellphones to get confidential relevant data from users. Every year, billions of dollars are wasted as a result of adware conducting advertising frauds. Machine learning (ML) approaches are being investigated by security experts and scholars to improve the identification of Android malware. In this research, a novel detection model for protecting smart devices against adware assaults is described, which monitors network traffic to do so. Numerous data pre-processing approaches, feature selection methods, & machine learning techniques are utilized to find adware examples in the dataset that is being provided. Two machine learning techniques, including Dynamic random forest and Split-Point & Attribute Reduced Classifier or SPAARC tree, are employed. Weka is being utilized to preprocess data using Information Gain (IG) & to do ML operations on data. As part of our research, we tested if the SPAARC tree method, when implemented to adware detection systems built on Android, was successful in pattern recognition procedure. The outcomes of the experiments proved that the adoption of the SPAARC tree method outperformed the classic Dynamic Random Forest approach in terms of accuracy in identifying adware attacks, with 95.16 percent accuracy. It identifies a 97.7% true positive rate for benign class 87.4% true positive for adware (adware) class. From this found that, ironically, the binary adware classification problem is easier to solve.**

IndexTerms - **Android, Adware Detection, Machine Learning, SPAARC, Dynamic Random Forest.**

I. INTRODUCTION

Web surfing, email, multimedia, entertainment apps (games, films, and audio), navigating, and stock trading are just a few of the many functions that smartphones can do (Sujithra and Padmavathi 2012). Symbian, Microsoft, Apple, Nokia, and Android are all fighting for a larger part of the smartphone business as the significance of smartphones grows (Sujithra M, Padmavathi G 2013).

Among smartphone platforms, Android adopted an open design with the benefits of system source release & different app distribution pathways for quick market growth, resulting in more than half of the smartphone sales (Sun et al. 2018). Although Android's open design enables harmful code to be readily produced and spread, this stigmatizes Android as being susceptible to malicious malware. As it turns out, Android is the main target of mobile malware, and the number of assaults is increasing rapidly (Xiong et al. 2014). There has also been an increase in the variety and intensity of harmful adware-based hacking attacks targeting Android smartphones that remain fast spreading throughout the globe over time. Most harmful adware assaults target Android devices by utilizing malicious malware to penetrate and manipulate them (Faruki et al. 2015). After that, APK named Root Master is downloaded from a certain URL & user's Android smartphone gets rooted. Install-recovery should be made unalterable so that an attacker's software cannot be detached from an Android device after it has been granted root privileges. When malware infects a user's Android smartphone, it mechanically installs specified programs onto a device or prompts the user to click on APK install option. As a result, the number of applications promoted by attackers that are downloaded and advertising messages are shown to increase gradually over time (Wang et al. 2018). From Android version 2.3.4 to 5.1.1, malicious malware has been found on almost every Android smartphone presently in use (Lee and Park 2020). Android-based devices have become more popular, which has led to the proliferation of harmful software developed by hackers. The answer to this problem is to improve the tracking of malware on Android devices. The use of static or dynamic characteristics derived from Android apps is common in malware detection algorithms. Different ML algorithms are utilized for malware detection to reduce the need for manual updates.

With the rise of ad-supported apps, a new industry emerged: advertising SDKs (Software Development Kits) (Ideses and Neuberger 2014). These SDKs make it very simple for application developers to implement ad income sources into their apps. Due to its simplicity, a new kind of program known as Adware has emerged. Adware apps are simply programming whose primary goal is to generate the most amount of income for the creator while providing the least amount of value to the user.

Adware software is classified as either freeware or shareware. The user is often provided with absolutely nothing. This Adware may cause damage to the people through stealing his information and distributing it to a remote computer, by showing advertisements assertively through screen hijacking, by showing advertisements in the notification area, that is normally applied for essential system occurrences, & also in extreme cases, by hijacking the smartphone speaker.

Once an adware program has been installed on a phone, it is not always possible to track harmful behavior on phone back to the application that was responsible for the installation. Anti-Malware software is being developed by several firms to combat the problem of harmful malware being distributed. Overall, all major desktop anti-malware software providers have started to offer mobile versions of their products, except a few. The study of anti-Malware algorithms has also grown more popular in academic circles. For this work, few researchers (Grace et al. 2012), (Zheng, Sun, and Lui 2013) have discovered malware detection methods. These techniques, as well as systems, often depend on heuristics or signature files, and as a result, they are ill-equipped to cope with newer specimens of malware as well as Adware (Zhou et al. 2012) as they become available. Other research is centered on the application of ML methods for automated categorization of malware; examples of such work may be found in (Aafer, Du, and Yin 2013) (Arp et al. 2014) and (Gascon et al. 2013).

The rest of the paper is structured accordingly. Related work is described in Section 2. Section 3 research methodology. Section 4 also provides details of experiments results and a description of benchmark data sets. Finally, the concluding remarks along with the scope for future work are at the end of this paper in Section 5.

II. LITERATURE REVIEW

Here several types of methods that many researchers have analyzed on malicious adware detection on the android platform are reviewed.

In this work, (Khan et al., 2021) used machine learning to identify wake-lock leaks in Android applications. SMOTE (Synthetic Minority Oversampling) is a method for balancing a dataset by removing data that is not representative of the population and then resampling it. Grid search was utilized to find the optimal settings for the machine learning techniques used to identify wake-lock breaches. For this purpose, the wake-lock leak detection parameters are employed in training. Different machine learning algorithms were used to assess their effectiveness, and they classified them into basic or ensemble methods to do so. Stochastic gradient boosting and Support Vector Machine was the most accurate, with 97% and 98% accuracy, respectively.

In this study, (Aboosh and Aldabbagh 2021) described that adware assaults on smart devices may be detected and prevented by monitoring network traffic. To steal private information and data from cellphones, hackers are continually monitoring them using different malware assaults, one of which is Advertising Software (Adware). Machine Learning approaches are being developed by security experts and researchers to better identify Android malware. Data pre-processing, feature selection, and machine learning methods are utilized to identify adware samples in this dataset. Adware detection was then tested using RF (Random Forest), k-NN (k-Nearest Neighbor), DT (Decision Tree), XGBoost (Extreme Gradient Boosting), and LR (Logistic Regression) to establish the best classifier for adware identification using 7 performance indicators. Detection accuracies of (98.66%), (98.10%), and (98.05%) were the highest for the suggested methods, which were decision tree, XGBoost, and k-NN.

In this report, (Bagui and Benson 2021) analyzed each adware family individually. Individual adware families have not yet been analyzed. The CICAndMal2017 dataset is used in this study, and feature selection and classification were accomplished utilizing information gain and machine learning. Network traffic samples were used to determine the optimum features for classifying the various types of adware. Overall, the data showed a higher overall classification rate for individual adware families than earlier studies.

In this study, (Dobhal, Das, and Aswal 2020) examined machine learning-based strategies to identify Android malware, specifically Adware. There has been a rise in the number of smartphone users as a result of fast advancements in technology. Android-based systems' security is a major concern that requires more investigation. Some machine learning methods like logistic regression, Linear Discriminant Analysis, k-NN, Classification, And Regression Trees are trained and assessed for two scenarios where one is the two-class scenario, another one is the multi-class scenario. For testing, the remaining 40% of the dataset was used for the machine learning algorithms to learn and the algorithms are tested utilizing a 10-fold cross-validation approach.

In the paper, (Birajdar et al. 2015) offered an adware detection strategy that depends on data mining (DM) on disassembled code This article describes a strategy for developing an accurate adware detection algorithm using adware data collection and ML techniques. They disassembled binary files, generated instruction sequences, and processed their data using various DM & ML methods for feature extraction & reduction to discover dangerous adware. The system identifies both new and known adware instances with high accuracy, although the binary difference between adware and legitimate software is often rather minor.

III. RESEARCH METHODOLOGY

This section focuses on the description of the problems as well as the steps necessary to resolve these problems by using the proposed methods. Our research methodology, as well as a flow chart, are discussed.

A. Problem Identification

The issue of adware is expanding all the time as a result of the enormous monetary gains that adware creators are reaping. Adware is often seen as having a poor level of knowledge among users, who are also unaware of its possible effects. While preparing a data set for ML classification tasks, an issue of class imbalance may occur. An imbalance problem in data collection often happens when one class has considerably more instances than another class or other classes. In addition, according to past research, feature selection techniques can only rank features in terms of their significance, and they are unable to determine the optimal no. of features that are required to train a classification method. Furthermore, the Dynamic Random Forest method (Lee and Park 2020) consumes a significant amount of memory space. As a result, research is being carried out to find solutions to these problems by improving the calculation employed in computing the weight in SPAARC (Split-Point and Attribute Reduced Classifier) method.

B. Proposed Methodology

In this research, a novel detection model for protecting smart devices against adware assaults is described, which monitors network traffic to do so. Various data pre-processing approaches, feature selection methods, & ML procedures are employed to find adware samples in the dataset being provided. For this work, the CIC-AAGM (CIC Android Adware and General Malware) dataset (Lashkari et al. 2017) is used, which captures network traffic by semi-automatically installing android applications on actual android devices. The dataset is classified into 3 classes, contains adware, benign software, & generic malware, as well as it has been discovered that the dataset's composition ratio is substantially unbalanced in one direction. This dataset contains a total of 79 characteristics. Weka analysis is used to analyze the properties of 79 features, after which we undertake feature selection to put on them to modeling. To select features, we adopt the entropy-based information gain approach, which allows us to perform 10-fold validation. A method for feature selection is typically comprised of two parts: a search approach as well as an assessment measure. After this applied the preprocessing step to preprocess the input data, normalization to normalized the data and resampling is used to balance the data, then classification is performed using a machine learning algorithm, namely, SPAARC to identify harmful adware attacks, as well as the rate at which attacks are detected at the instant. The flowchart of this proposed methodology is given in fig. 1.

1) *Data Collection*

First and foremost, it is essential to gather relevant traffic to identify aberrant activity via network traffic. This research makes use of the CICAAGM dataset[1], which records network traffic by semi-automatically installing Android applications on actual Android devices.

2) *Feature Selection*

In this research work, 10-fold validation by using entropy-based information gain technique for feature selection has been applied.

a) K-fold Cross-Validation

When applied in conjunction with the holdout approach, K-Fold Cross Validation (Darapureddy, Karatapu, and Battula 2019) is an improvement. It helps to lower the variation of the final estimate by a significant amount. The training data set is split into K equal-sized random subsets, each of which contains the same amount of data. When the holdout approach is applied to each of these K subgroups, it is performed K times. A subset of K subsets is used as a test set while remaining K-1 subsets are also used to fit the classifier consistently.

b) Information Gain

The IG selection technique is an entropy-based selection approach (Lei 2012), that entails the computation of gain from output data aggregated by feature A, which is symbolized by the letter A. (y, A). The IG (y, A) is denoted by the symbol

$$gain(y, A) = entropy(y) - \Sigma_{C \in vals(A)} \frac{y_c}{y} entropy(yc) \quad (1)$$

Where,

vals (A) = potential rates of attribute A,
yc = a subset of y in which A owns the sum of c.

To add to this, the rule of Equation (1) was total entropy of y, which was proceeded by data segregation, which was predicated on feature A.

3) *Data Pre-Processing*

Data preprocessing is an important phase in obtaining the maximum value from data collection. Preprocessing have done by the following two methods those are described below subsection:

a) Resampling

Resampling (Ghorbani and Ghousi 2020) is a methodology for improving the accuracy as well as quantifying the uncertainty of a population mean by cost-effectively utilizing a data sample. When used in conjunction with the holdout approach, K-Fold Cross Validation is an improvement. It helps to lower the variation of the final estimate by a significant amount. The training data set is split into K equal-sized random subsets, each of which contains the same amount of data. The holdout method

[1] https://www.unb.ca/cic/datasets/android-adware.html

is repeated K times for each of these K subsets. A subset of K subsets is used as a test set while remaining K-1 subsets are also used to fit the classifier consistently.

b) Normalization

The normalization of data (sometimes referred to as data pre-processing) is an important module of the Data mining procedure. Normalization (also known as Min-Max scaling) is a method for decreasing the size of objects to a more manageable size (Ahsan et al. 2021). It's computed as a new point.

$$X\_new = (X - X\_min)/(X\_max - X\_min) \quad (2)$$

This reduces the range to [0, 1] or [-1, 1] depending on the application.

C. Proposed Approach: SPAARC Tree

This work proposes a method for speeding up the DT induction process named SPAARC (Yates, Islam, and Gao, 2019). Two components the split-point sampling as well as node-attribute subsampling class are used for implementing the SPAARC decision tree classifier. It makes use of a modified version of SimpleCart. In comparison to SimpleCart, SPAARC builds trees with the same classification accuracy as SimpleCart but does so at a rate up to 69 percent faster. As previously stated, Errors Have Been Reduced Pruning Tree is a decision tree learning technique that is both fast and effective. It creates a decision tree based on the data gained and can reduce variance. SPAARC is a classifier that makes use of the Simple CART algorithm. SPAARC is comprised of two components that work together to address the issues associated with decision trees.

As part of our proposed strategy for speeding tree induction, we have combined mechanisms of split-point sampling as well as attribute subset selection into a single unique deployment, which we have designated as SPAARC. Furthermore, this methodology can be employed in some classification models that integrate its arithmetic attribute split-point analysis as well as node-attribute selection recursively as part of its node-attribute selection process, such as a regression analysis technique. Beginning with Node Attribute Sampling (NAS) as well as starting to move on to Split-Point Sampling (SPS), which is covered in the below subsection. These 2 exact components of SPAARC will be thoroughly discussed one by one.

a) Node Attribute Sampling (NAS)

Using our suggested technique, the NAS component eliminates the need of checking all and each non-class property at each tree node while also avoiding the restriction of only picking a subset of attributes beforehand tree induction starts. Our technique is a substitute that dynamically picks attribute space, rotating among complete & subset attribute lists depending on the deep level of a present node under test.

NAS Pseudo-Code:

```
NodeAttributeSample(D, A, M, sortedAtts[]):

Inputs: dataset D, attributes A, tree Depth modulus M,
        Array of sorted attributes sortedAtts[]
Output: attribute to split A, best split point bestSplit

If (treeDepth modulo M) = 1:
| Foreach A_i ∈ A:
| | Splits[] = SplitPointSample(A_i, D, infoGain[])
| End
| sortedAtts[] = sort A by infoGain[]
Else:
| Foreach A_i ∈ sortedAtts[] > median(infoGain[]):
| | splits[] = SplitPointSample (A_i, D, infoGain[])
| End
End
A_s = attribute(maxGain(infoGains[]))
bestSplit = splits[A_s]
Return A_s, bestSplit
```

b) Split-Point Sampling (SPS)

With the help of some measure of information gain, several decision tree techniques search for an ideal splitting point of a mathematical attribute at a node. The Gini index, for illustration, is a statistical metric:

$$Gini\ (R) = 1 - \sum_{i=1}^{m} \quad p_i^2 \quad (3)$$

Where,

pi = probability that record R in dataset D corresponds to class value ci,
ci = class value

It is necessary, though, to test every adjacent pair of the separate attribute value to find the Split Point with the greatest amount of InfoGain.

SplitPointSample pseudo-code:

```
SplitPointSample(A_i, D, infoGain[]):

Inputs: dataset D_sorted, attribute A_i ∈ A, array infoGains[]
Outputs: array of maximum info gains infoGain[],
  candidate split-point splitPoint

If A_i is nominal:
| splitPoint = getNominalSplits(A_i, D, infoGain[])
Elseif A_i is numerical:
| hopStart = D_sorted[0].value(A_i)
| valueRange = D_sorted[last].value(A_i) – hopStart
| hopStep = valueRange/10
| hopPoint = hopStart + hopStep
| Foreach R_i ∈ D_sorted:
| | If R_i.value(A_i) > hopPoint:
| | | m_i = calculateInfoGain(D_sorted, A_i)
| | | If m_i > maxInfoGain:
| | | | maxInfoGain = m_i
| | | | splitPoint = R_i.value(A_i) + R_i-1.value(A_i) / 2
| | | End
| | | hopPoint = hopPoint + hopStep
| | End
| | currentSplitPoint = R_i.value(A_i)
| End
| infoGains[A_i] = maxInfoGain
End
Return splitPoint
```

These set some default hyperparameter values that are derivative through the Cross-Validation concluded grid search technique to enhance the efficiency of SPAARC tree process.

Algorithm: Proposed SPAARC Tree-based Algorithm

Input: CICAAGM dataset
Output: Classification results

Strategy:

Step 1: Collect the CICAAGM dataset of network traffic
Step 2: Dataset separated into 3 classes contains benign, adware, and general malware
Step 3: Remove General malware class
Step 4: Perform feature selection to put on 10-fold validation by using the information gain method
Step 5: Perform resampling
Step 6: Apply data normalization
Step 7: Use machine learning SPAARC tree algorithm with 10-fold CrossValidation to classification
Step 8: Train and test model
Step 9: Perform classification
Step 10: Get classification results as benign or adware
Step 11: End

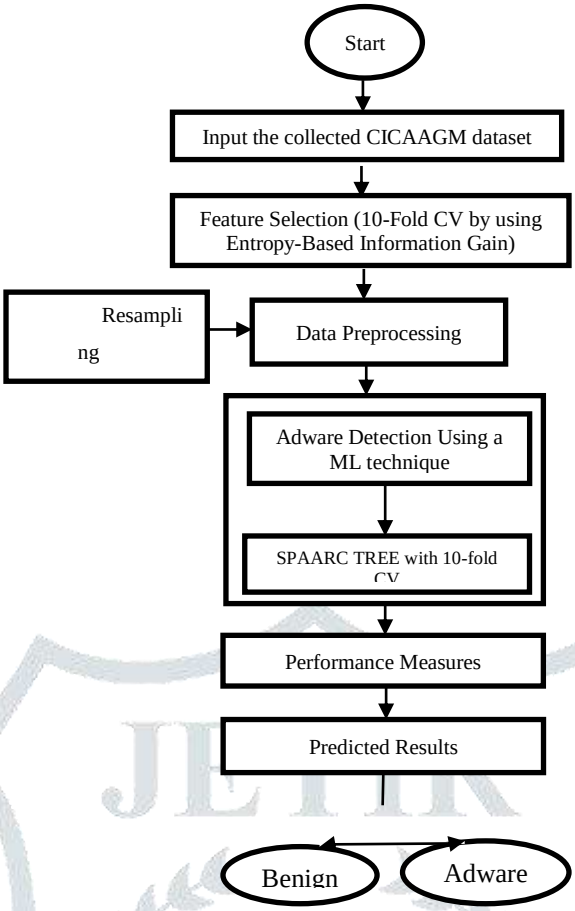

Figure 1: Flow Chart of Proposed Methodology

II.RESULTS ANALYSIS AND DISCUSSION

The descriptions of the dataset, measures, parameters, including experimental findings are all introduced in this section of the paper. The WEKA tool has been used in this research study to conduct experiments. Outcomes of the research for metrics are shown in a variety of graphs and tables.

A. Dataset Descriptions

First and foremost, it is essential to gather relevant traffic to identify aberrant activity via network traffic. This research makes use of the CICAAGM dataset, which collects network traffic via semi-automatically installing Android apps on real-life Android devices. To make this possible, the CIC Center has made an Android sandbox available to the general public for use.

B. Performance Metrics

Model-performance measurements are often calculated as determined as the ratio of the predicted values of predictor variables in a dataset with both the (actual) values of the dependent variable. Predicted values & dependent variable values should be the same in a model. Every machine learning pipeline includes performance measurements as an integral part of its design. They inform you whether or not you are making progress and provide a numerical value for it. Every machine learning problem, just like every performance statistic, may be divided into two categories: regression and classification.

❖ **Classification Metrics**

In the realm of research, classification difficulties are one of the most commonly researched areas. Because classification models provide discrete results, we want a measure that can be used to compare discrete classes in some way. These are applied to assess the efficiency of a technique and to determine how good or terrible the classification is, but each one analyses the model differently. Consequently, to assess Classification models, we'll go through the following measures in more detail:

- **Confusion matrix**

It's a graphical depiction of the parameters listed above in a matrix format. It is usually beneficial to have better visualization. In the confusion matrix, a predicted class is defined by each row, as well as an actual class is given from each column.

For the sake of simplicity, let us consider a binary classification issue in which we must determine if a piece of malware is adware or benign. Many terms need to be understood, including the following:

- **True positives (TP)**: Positive outcomes have been predicted and have occurred.
- **False positives (FP)**: Predicted good outcomes turn out to be negative.
- **True negatives (TN)**: They were predicted to be negative and turned out to be negative.

- **False negatives (FN)**: Forecast to be negative, but turned out to be positive.

1) Accuracy

The most widely applied measure to evaluate a system is not a reliable predictor of the model's efficiency in real-world situations. The worst-case scenario occurs when classes are imbalanced. When it comes to measuring classification accuracy, it is probably the easiest to use and implement. It is described as no. of classifications model appropriately predicts separated by total no. of predictions made.

$$\frac{TP+TN}{TP+FP+TN+FN} \quad (4)$$

2) Precision

It is the proportion of positive cases among the total number of predicted positive instances. Precision is mathematically expressed as no. of correct divided by total no. of correctly identified + no. of incorrectly identified. Considering it a test to determine "to what extent the model is true when it promises to be correct.

$$\frac{TP}{TP+FP} \quad (5)$$

3) Recall

It is the proportion of positive cases among the overall number of positive instances. As a consequence, the denominator (TP + FN) in this case represents the actual no. of positive cases included in the dataset. Consider it as an experiment to see 'how many more correct ones the model missed when it presented the correct ones.'

$$\frac{TP}{TP+FN} \quad (6)$$

4) F-measure

It is determined as harmonics mean of precision & recall, with equal probability for each. It enables a model to be assessed that used a score of 1 that includes both precision and recall, which is useful for evaluating model performance by comparing results.

$$\frac{2*precision*recall}{precision+recall} \quad (7)$$

5) Matthews correlation coefficient (MCC)

MCC is a statistical tool used for model evaluation. Mathematicians use the Matthews correlation coefficient to compute the Pearson product-moment correlation coefficient among actual and potential data using a contingency matrix method. maybe used as an alternate metric that is not influenced by the imbalanced datasets problem. MCC reads as follows in terms of M entries:

$$MCC = \frac{TP*TN-FP*FN}{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)} \quad (8)$$

6) ROC Area

AUC – ROC is the most often used measure in assessment. It is popular because it gives more weight to positive predictions than negative predictions. Furthermore, the change in the fraction of responders does not affect the ROC curve. The ROC is displayed against the TPR and FPR for different threshold values. FPR also rises when TPR increases.

$$TP\ rate\ (TPR) = recall = \frac{TP}{TP+FN} \quad (9)$$

$$FP\ rate\ (FPR) = 1 - specificity = \frac{FP}{TP+FN} \quad (10)$$

7) PRC Area

It represents the relationship between precision and recall for a range of threshold values. The upper right part of the graph is the optimal space in which to achieve high precision and recall. We can select the predictor as well as the threshold value based on the requirements of our application. The AUC for PR is just the area under the curve. Its numerical value should be as high as possible.

8) Kappa statistics

In general, the kappa statistical measure of how well the instances categorized by the machine learning method matches the data labeled as real data, while adjusting for the estimated accuracy of a randomized classifier. To regulate just those occurrences that may have been accurately identified by chance, the kappa statistic is utilized. Attainable accuracy may be evaluated by taking into account both experiential (total) accuracy and random accuracy. Kappa may be computed using the following formula:

$$Kappa = \frac{(total\ accuracy - random\ accuracy)}{(1 - random\ accuracy)} \quad (11)$$

- **Regression Metrics**

The use of regression models allows for the generation of a series of samples. The gap between the anticipated as well as the ground truth must be calculated in some way, but also, we need a measure that does only this. Loss functions indicate the

model's performance. Typically, they're utilized to train a machine learning model, and they're distinguishable in technique parameters. However, if the performance measure is differentiable for certain activities, it may also be utilized as a loss function (possibly with additional regularizations applied), such as the RMSE.

1) MAE (Mean Absolute Error)

MAE is distinct as an average of variance among both ground truth and anticipated values over a positive period. It may be stated numerically in the following way:

$$\frac{1}{N}\sum_{j=1}^{N} |y_j - \hat{y}_j| \qquad (12)$$

2) RMSE (Root Mean Squared Error)

RMSE is usually expressed in form of the average of the squared deviations between both the target value and the predicted value predicted by the regression model. It is an Sqrt is of MSE. It can be represented by the following mathematical formula:

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N} \left(y_j - \hat{y}_j\right)^2} \qquad (13)$$

3) RAE (Relative Absolute Error)

The RAE normalizes the overall absolute error via dividing this by the simple predictor's overall mean absolute. The Ei, of an individual model I is calculated numerically using the formula:

$$E_i = \frac{\sum_{j=1}^{N} |P_{ij} - T_j|}{\sum_{j=1}^{N} |T_j - \underline{T}|} \qquad (14)$$

4) RRSE (Root Relative Squared Error)

RRSE is expressed as a percentage of the value obtained using a simple predictor. To be more explicit, this straightforward predictor is the average of actual values. According to calculation, the RRSE Ei of a single model i is determined using the continuity formula for the paradigm i:

$$E_i = \frac{\sum_{j=1}^{N} (P_{ij} - T_j)^2}{\sum_{j=1}^{N} (T_j - \underline{T})^2} \qquad (15)$$

C. Experimental Results

In this subsection, experimental outcomes are described in further detail. The assessment of each of the measurements contained in the practical and recommended resolution was performed in the testing experiments carried out. In this part, we give the proposed method's outcome with a dataset.

1) Dataset Preparation Results

This subsection consists of several results for preparing the dataset with some steps. In this CICAAGM dataset is considered that is imbalanced so needs to be processed according to requirements.

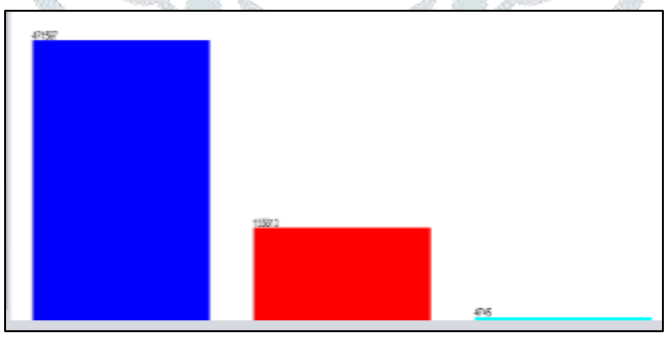

Figure 2: Visualize Dataset

Figure 2 shows the visualization of the dataset distribution count in all three categories benign, adware, and GeneralMalware. In this, benign has 471597 counts, adware has 155613 counts, and GeneralMalware has 4745 counts.

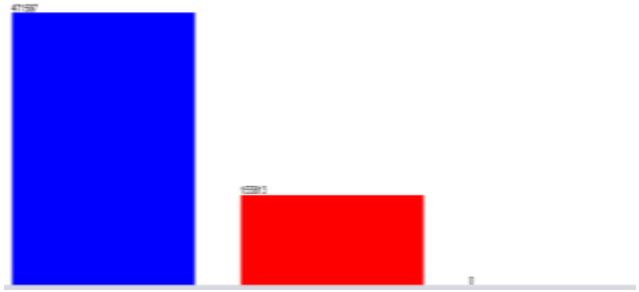

Figure 3: Remaining dataset after removal of records with values

Figure 3 visualize the left data after removing the records values of GeneralMalware to perform binary classification. This is done by applying some filter instances according to the value of attributes. In this the split point is 3, nominal indices and attribute index is last while remaining parameters are set as false. From this visualization, we can see that here GeneralMalware has 0 counts.

2) Feature Extraction Results

Once data has been prepared so next step is to extract the features from this given dataset that are displayed in this subsection.

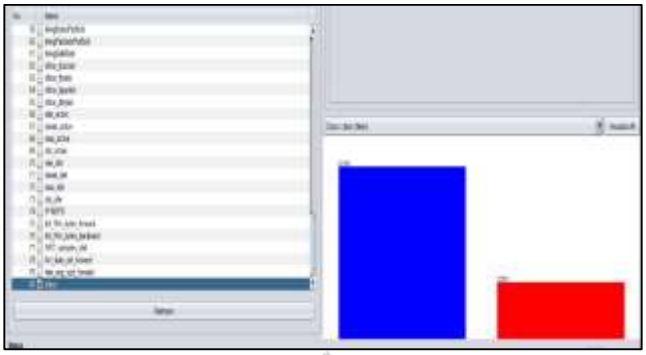

Figure 4: Feature selection using Information Gain

Now show all features/attributes generated by CIC-flowmeter in the CICAAGM dataset which has a total of 627210 instances with 80 attributes (79 attributes and 1 class). This only visualizes the benign and adware category because the GeneralMalware that had 0 counts deleted.

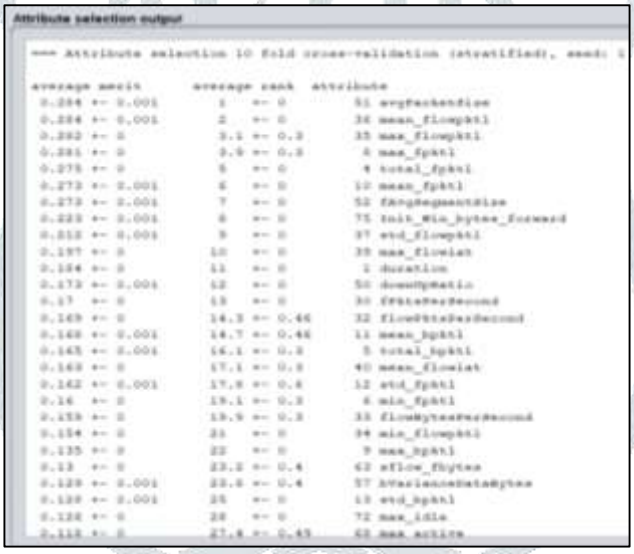

Figure 5: 10-fold cross-validation with InfoGain

Figure 5 displayed the results after applying 10-fold cross-validation with seed value 1 in the InfoGain feature selection method. it shows the average merits, average rank, and attribute.

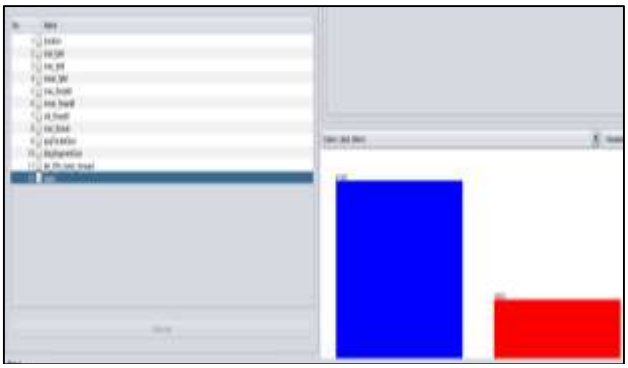

Figure 6: Data after getting 11 Attributes

3) Proposed Method Results: SPAARC Tree

This subsection displayed the tables and graphs of all results achieved by the proposed method SPAARC tree. Table 1 shows the statistical summary of the data.

Table 1: Statistics summary

Statistics	Value
Minimum	0
Maximum	1
Mean	0
StdDev	0.004

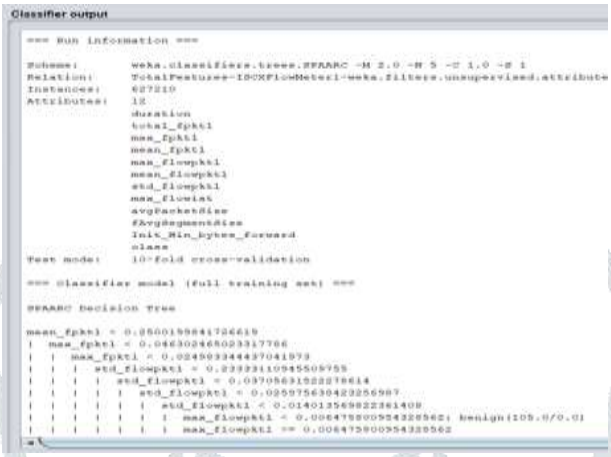

Figure 7: Tree structure of SPAARC tree

Figure 7 displayed the execution of the tree structure for the SPAARC tree. It has been applied 10-fold cross-validation with the SPAARC tree.

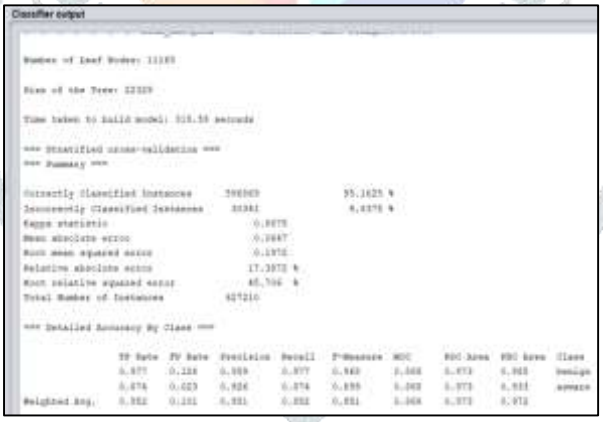

Figure 8: Classification results of SPAARC tree

Figure 8 shows the classification performance of the SPAARC tree using different performance metrics and simulation time to build the model. It displayed the stratified cross-validation results. Here, the total number of leaf nodes is 11165, and the size of the tree is 22329. It also shows the detailed accuracy by class benign and adware (adware). In this, correctly classified instances are 596869 for cross-validation whereas incorrectly classified instances are 30341 for cross-validation. The time taken to build this model is 319.59 seconds.

Table 2: Confusion matrix of SPAARC tree

a	b	Classified as
461361	10775	a=benign
19566	135508	b=asware

Table 2 demonstrates the confusion matrix of the SPAARC tree for the 10-folds cross-validation test. In that 461361 instances are correctly predicted as benign values, 10775 instances are incorrectly predicted as benign values, 19566 instances are correctly predicted but misclassified as adware values and 135508 instances are incorrectly predicted as adware values.

4) Comparison Results and Discussion

The interpretation of results is more focused on what we have evaluated the mean and how reliable or valid they are. However, the discussion part uses those interpretations to answer the research questions and compare these findings with what another baseline method scholar has found. In this number of tabular and graphs are used to compare and validate the proposed method in contrast to the baseline Dynamic Random forest (DRF) method. It shows the most relevant information in graphs, figures, and tables.

Table 3: Performance comparison between baseline DRF and Proposed SPAARC tree

Performance metrics	DRF	SPAARC Tree
MAE	8.58	6.47
RMSE	20.39	19.72
RAE	23.00	17.38
RRSE	47.20	45.70

Table 3 represents the number of performance metrics to compare the baseline and proposed methods. A comparison has been made between dynamic random forest and the SPAARC tree machine learning algorithm. It shows the regression metrics for performance to show the error or loss values. Here, the MAE, RMSE, RAE and RRSE are 8.58%, 20.39%, 23% and 47.2% for DRF and 6.47%, 19.72%, 17.38% and 45.70% for SPAARC tree, respectively.

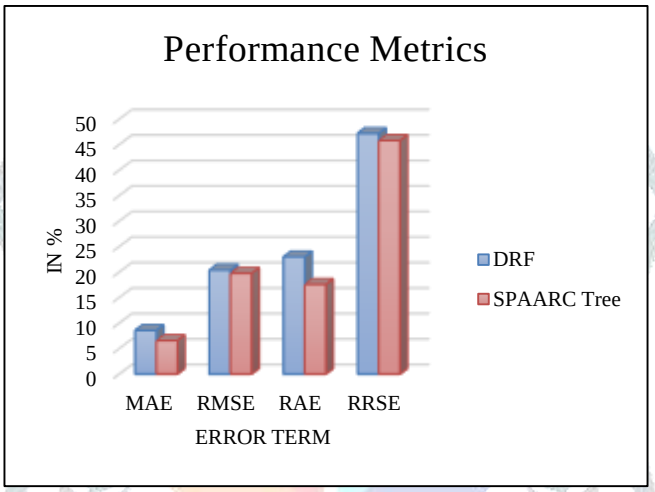

Figure 9: Comparative performance metrics for the error term

The comparative graphical portrayal has depicted in figure 9 to show the error term using the SPAARC tree. The comparison has been done between DRF (represents by blue color) and SPAARC tree (represents by pink color). The x-axis displays error terms and the y-axis displays their respective values in %. From this depiction, we can see that the SPAARC tree has taken minimized error than the baseline DRF method.

Table 4: Accuracy comparison between baseline DRF and Proposed SPAARC tree

Classification metrics	DRF (in%)	SPAARC Tree (in%)
Accuracy	94.31	95.16
Kappa statistics	84.41	86.75
Precision	94.2	95.1
Recall	94.3	95.2
F-measure	94.2	95.1
MCC	84.5	86.8

Table 4 represents the number of classification performance metrics to compare the baseline DRF and proposed SPAARC tree machine learning methods. It shows the accuracy classification metrics for performance to show the accuracy. Here, the accuracy, kappa statistics, precision, recall, f-measure and MCC are 94.31%, 84.41%, 94.2%, 94.3%, 94.2% and 84.5% for DRF and 95.16%, 86.75%, 95.1%, 95.2%, 95.1% and 86.8% for SPAARC tree, respectively.

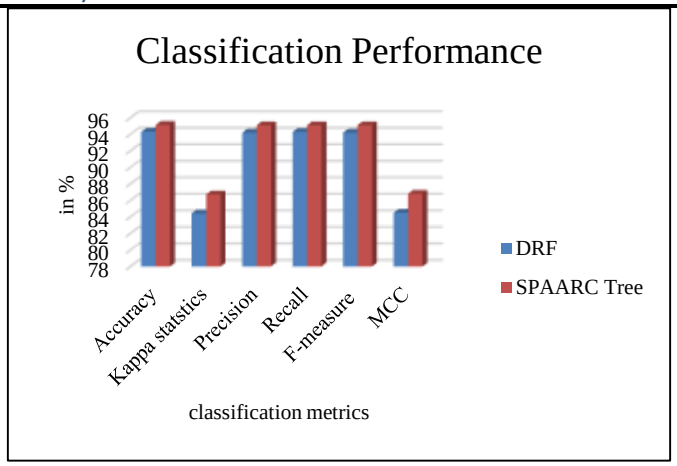

Figure 10: Comparative classification metrics for accuracy term

The DRF method had average accuracy results of 94.31 percent, while the SPAARC tree method had a cross-validation accuracy results of 95.14 percent, indicating that the SPAARC tree method had somewhat superior performance than the DRF method by around 0.83 percent on average. Especially by the F-measure, the SPAARC tree approach has been shown to beat the DRF method by an average of around 0.9 percentage points. According to these findings, the SPAARC tree method has good efficiency even when used for a network traffic monitoring system built on Android, as seen in Fig.10.

Table 5: Area curve comparison between baseline DRF and Proposed SPAARC tree

Area curve	DRF	SPAARC Tree
ROC AUC	0.984	0.973
PRC	0.985	0.972

Table 5 represents the comparison table for the area curve to show a comparison between baseline DRF and Proposed SPAARC tree methods. These tabular representations revealed that the SPAARC tree has minimized the ROC AUC and precision-recall curve than the baseline DRF method.

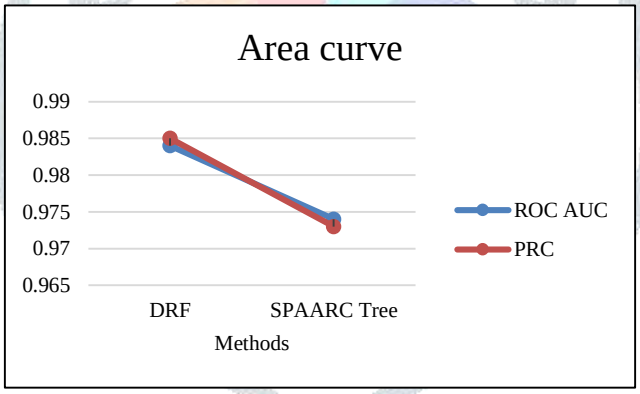

Figure 11: Comparative area curve between DRF and SPAARC tree

Figure 11 illustrated the comparative line graph for the area curve between the baseline DRF method and the proposed SPAARC tree method. The x-axis signifies machine learning methods while the y-axis signifies their respective values. In this line graph, the ROC AUC are 0.984 and 0.973 for DRF and SPAARC tree, respectively and the precision-recall curve (PRC) are 0.985 and 0.972 for DRF and SPAARC trees, respectively.

III.CONCLUSION AND FUTURE WORK

In this study, two distinct supervised learning strategies, namely Dynamic Random forest and SPAARC, were used to produce the comparative comparison of machine classifiers. The various performance metrics were used to test the efficiency of the classification methods developed with the help of the WEKA data mining tool. The investigation found that there are a large number of classifiers available, all of which, if thoroughly investigated, would provide more precise findings for adware identification. The SPAARC tree was determined to be a good classifier, with the highest accuracy of 95.16 percent as well as the lowest false positive rate of 0.023 percent of all classifiers tested. Because of the low false-positive rate achieved in this study, anti-adware application developers should consider implementing the machine learning classification automated system that was explored to be best in this research to obtain the feature of adware attack recognition and classification, according to the results of the comparative analysis.

New developments in this study and current classifiers may depend on more than one type of feature or modalities of data to identify malware or adware, indicating that they are more versatile. It may classify multimodal techniques into three categories, based on how the various data modalities are combined: When using early-fusion techniques, the unimodal feature

vectors are combined into a joint interpretation. When using late-fusion methods, one prototype per modality is trained, with its results merged. When using intermediate-fusion techniques, the intermediate features acquired by independent factors are combined into a sharable recognition.

REFERENCES

Aafer, Yousra, Wenliang Du, and Heng Yin. 2013. "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android." In *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*. https://doi.org/10.1007/978-3-319-04283-1_6.

Aboosh, Omar Sh. Ahmed, and Omar Abdulmunem Ibrahim Aldabbagh. 2021. "Android Adware Detection Model Based on Machine Learning Techniques." In *2021 International Conference on Computing and Communications Applications and Technologies (I3CAT)*, 98–104. https://doi.org/10.1109/I3CAT53310.2021.9629400.

Ahsan, Md Manjurul, M. A. Parvez Mahmud, Pritom Kumar Saha, Kishor Datta Gupta, and Zahed Siddique. 2021. "Effect of Data Scaling Methods on Machine Learning Algorithms and Model Performance." *Technologies* 9 (3): 52. https://doi.org/10.3390/technologies9030052.

Arp, Daniel, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. 2014. "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket." In . https://doi.org/10.14722/ndss.2014.23247.

Bagui, Sikha, and Daniel Benson. 2021. "Android Adware Detection Using Machine Learning." *International Journal of Cyber Research and Education*. https://doi.org/10.4018/ijcre.2021070101.

Birajdar, Neeta D, Madhav N Dhuppe, Trupti M Hegade, Nikita S Jadhav, and Manoj D Shelar. 2015. "Review Paper On Adware Detection Using Instruction Sequence Generation." *International Journal of Engineering and Techniques* 1 (6): 25–28. http://www.ijetjournal.org.

Darapureddy, Nagadevi, Nagaprakash Karatapu, and Tirumala Krishna Battula. 2019. "Research of Machine Learning Algorithms Using K-Fold Cross Validation." *International Journal of Engineering and Advanced Technology* 8 (6 Special issue): 215–18. https://doi.org/10.35940/ijeat.F1043.0886S19.

Dobhal, Dinesh C, Purushottam Das, and Kiran Aswal. 2020. "Detection of Android Adwares by Using Machine Learning Algorithms." *International Journal of Engineering and Advanced Technology* 8 (4S): 17–21. https://doi.org/10.35940/ijeat.d1005.0484s19.

Faruki, Parvez, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. 2015. "Android Security: A Survey of Issues, Malware Penetration, and Defenses." *IEEE Communications Surveys and Tutorials*. https://doi.org/10.1109/COMST.2014.2386139.

Gascon, Hugo, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. 2013. "Structural Detection of Android Malware Using Embedded Call Graphs." In *Proceedings of the ACM Conference on Computer and Communications Security*. https://doi.org/10.1145/2517312.2517315.

Ghorbani, Ramin, and Rouzbeh Ghousi. 2020. "Comparing Different Resampling Methods in Predicting Students' Performance Using Machine Learning Techniques." *IEEE Access* 8: 67899–911. https://doi.org/10.1109/ACCESS.2020.2986809.

Grace, Michael, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. 2012. "RiskRanker: Scalable and Accurate Zero-Day Android Malware Detection." In *MobiSys'12 - Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. https://doi.org/10.1145/2307636.2307663.

Ideses, Ianir, and Assaf Neuberger. 2014. "Adware Detection and Privacy Control in Mobile Devices." In *2014 IEEE 28th Convention of Electrical and Electronics Engineers in Israel, IEEEI 2014*. https://doi.org/10.1109/EEEI.2014.7005849.

Lashkari, Arash Habibi, Andi Fitriah A.Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A. Ghorbani. 2017. "Towards a Network-Based Framework for Android Malware Detection and Characterization." In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, 233–23309. https://doi.org/10.1109/PST.2017.00035.

Lee, Kyungmin, and Hyunhee Park. 2020. "Malicious Adware Detection on Android Platform Using Dynamic Random Forest." In *Advances in Intelligent Systems and Computing*, 994:609–17. https://doi.org/10.1007/978-3-030-22263-5_57.

Lei, Shang. 2012. "A Feature Selection Method Based on Information Gain and Genetic Algorithm." In *Proceedings - 2012 International Conference on Computer Science and Electronics Engineering, ICCSEE 2012*. https://doi.org/10.1109/ICCSEE.2012.97.

Sujithra M, Padmavathi G, Sathyanarayanan S. 2013. "A Survey on Mobile Application Development Using Android OS." *IEEE International Conference on Research and Development Prospects on Engineering and Technology (ICRDPET-2013)*, no. March 2013: 269–75.

Sujithra, M., and G. Padmavathi. 2012. "Next Generation Biometric Security System, An Approach for Mobile Device Security." In *ACM International Conference Proceeding Series*. https://doi.org/10.1145/2393216.2393280.

Sun, Caijun, Hua Zhang, Sujuan Qin, Nengqiang He, Jiawei Qin, and Hongwei Pan. 2018. "DexX: A Double Layer Unpacking Framework for Android." *IEEE Access*. https://doi.org/10.1109/ACCESS.2018.2875694.

Wang, Wei, Zhenzhen Gao, Meichen Zhao, Yidong Li, Jiqiang Liu, and Xiangliang Zhang. 2018. "DroidEnsemble: Detecting Android Malicious Applications with Ensemble of String and Structural Static Features." *IEEE Access*. https://doi.org/10.1109/ACCESS.2018.2835654.

Xiong, Ping, Xiaofeng Wang, Wenjia Niu, Tianqing Zhu, and Gang Li. 2014. "Android Malware Detection with Contrasting Permission Patterns." *China Communications*. https://doi.org/10.1109/CC.2014.6911083.

Yates, Darren, Md Zahidul Islam, and Junbin Gao. 2019. "SPAARC: A Fast Decision Tree Algorithm." In *Communications in Computer and Information Science*. https://doi.org/10.1007/978-981-13-6661-1_4.

Zheng, Min, Mingshen Sun, and John C.S. Lui. 2013. "Droid Analytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware." In *Proceedings - 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013*. https://doi.org/10.1109/TrustCom.2013.25.

Zhou, Yajin, Zhi Wang, Wu Zhou, and Xuxian Jiang. 2012. "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets." In *NDSS*.