



DOI: <https://doi.org/10.71317/kjard.2.10.2026.919>

Kashmir Journal of Academic Research and Development

Journal homepage: <https://rjsaonline.org/index.php/KJARD>



A Baseline Machine Learning Benchmark for Permission-Based Android Ransomware Detection: Comparative Evaluation of Random Forest, XGBoost, Support Vector Machines, Neural Networks, Decision Trees, and Linear Regression

Muhammad Akram Khan

Faculty of ICT, BUITEMS, Quetta, Pakistan

Email: writetoakramkhan@gmail.com

Muhammad Nadeem

Department of CIS, HCT, Abu Dhabi, UAE

Email: mnadeem@hct.ac.ae

Mehrullah Soomro

Faculty of ICT, BUITEMS, Quetta, Pakistan

Email: mehrullahsoomro@gmail.com

Dr. Muhammad Ayub

Department of Electronic Engineering, BUITEMS, Quetta, Pakistan

Email: muhammad.ayub@buitms.edu.pk

Dr. Muhammad Abbas Khan

Department of Electrical Engineering, BUITEMS, Quetta, Pakistan

Email: muhammad.abbas@buitms.edu.pk

Misbah Rani

Department of Computer Science, Faculty of ICT, BUITEMS, Quetta, Pakistan

Email: misbahrani480@gmail.com

ARTICLE INFO

Received:

September 15, 2026

Revised:

October 14, 2026

Accepted:

October 20, 2026

Available Online:

October 27, 2026

Keywords:

Android ransomware, benchmark study, machine learning, Random Forest, XGBoost, support vector machine, neural network, static analysis, decision tree and linear regression, permission analysis.

Corresponding Author:

Muhammad Akram Khan

Email:

writetoakramkhan@gmail.com

ABSTRACT

Permission-based static analysis is still one of the lightest methods for catching ransomware prior to installation, and ransomware is continuing to become more sophisticated on the Android platform. In a previous work, a well-balanced set of 1,990 Android applications (995 for ransomware and 995 for benign) and 365 permission attributes was collected, and Logistic Regression to achieve an accuracy of 97.69% under 10-fold cross validation had been obtained. This paper is along two lines of extension, scale and algorithmic breadth, of this study. The 20,012 applications in the permission dataset are a public-permission version that has been substantially-expanded from the known ransomware/benign dataset included with the publication, and are defined by 2,317 cleaned binary permission/API Usage features used on each sample; we test the performance of six popular supervised learning algorithms (Random Forest, XGBoost, linear-kernel SVC, MLP, a single decision tree, and linear regression). A common preprocessing pipeline was performed across all models and training was performed on a synthesis of a 70/30 split and 10-fold stratified cross-validation. We present accuracy, precision, recall, F1 score, ROC area, false positive rate, confusion matrix, training time and a gini impurity feature ranking. The MLP, Linear SVM, and Random Forest achieved comparable results as Random Forest is the second best (91.72% $\pm$ 0.72% accuracy under cross-validation and 91.72% when performing a held-out test on the train/test split, with an F1-score of 0.922 and an ROC-AUC score of 0.942). The single Decision Tree model was once again the most poor with thresholded Linear Regression being not far behind the dedicated classifiers, meaning that many of the signals in permission space are approximately linear — as seen in our previous study where Logistic Regression performed well. The accuracy of the larger dataset is about six percentage points lower than the accuracy of the curated corpus of, and all models have a false-positive rate of approximately 13–15% and high recall rates (93–98%). In our larger set, we also find a feature-encoding artifact, which is that set ransomware's and set benigns' permissions are named differently, and that this difference gives an inflated sense of the performance and should be dealt with, such as by normalizing permissions names as we did in the benigns in order to do use of that set for production-grade classifiers. Results as well as pipeline is meant to be a repeatable basic, in order to be able to be used within the future in permission to generate detection activities.

Introduction

Ransomware is malicious software that denies the victim access to his/her information, files or device (usually by using encryption methods or by locking the screen) until a ransom is paid [1]. Ransomware has been a growing attack vector affecting Android-the world's most widely deployed mobile operating system-for the last ten years and has proven to be a serious cost-effective attack over time [2] [3]. As the Android permission system is designed to make sure that every application lists the stop API it will use [4] [5] it has become one of the most widely studied static signals for detecting malicious applications and ransomware [6]–[12].

Held in our previous work [1], we gathered benign applications from the Google Play and ransomware from public repositories that are verified by VirusTotal, HelDroid project and Koodous, and decompiled both types of applications to retrieve the permissions requested in the corresponding AndroidManifest.xml file. This balanced set of 995 ransomware and 995 benign applications, each labeled by 365 attributes of binary (file permission) characteristics was tested on four classifiers within WEKA [13]. Based on the results, Logistic Regression had the best accuracy of 97.69%, while Bayes Network, Random Tree and Decision Table methods followed with 96.88%, 96.53% and 95.73% respectively, and MOUNT_UNMOUNT_FILESYSTEMS, SYSTEM_ALERT_WINDOW and RECEIVE_BOOT_COMPLETED were found to be the most discriminative permissions for models [1]. Two issues were unresolved in this study that established classifiers as a standard baseline: first, whether the results may be generalized to a much larger and more diverse set of applications; the second, how the four WEKA classifiers measure up against ensemble, kernel, and neural classifiers, which are now commonly used as standard baselines. Both are covered on a theme in the present paper.

In practice, static, permission-based detection has three advantages. One, permissions do not require the program to be run, so they can be mined prior to execution, unlike dynamic permissions analysis (sandbox-based), the approach is fast, cheap and safe because it does not require any execution [14] [15]. Second, permission sets are naturally expressed as a fixed-length binary feature vector, which can be seamlessly fed into widely-used supervised learning pipelines [6, 10]. Third, many prior studies have demonstrated that only a small number of permissions is responsible for a large portion of the discriminative signal between malicious and benign applications [1, 7, 8, 12] motivating interpretable models as good baselines.

In this study, the baseline benchmark of six classical and widely used supervised learning algorithms—Random Forest [16], XGBoost [17], Support Vector Machines [18], a feed-forward Multi-Layer Perceptron [19, 20], a single Decision Tree [21] and Linear Regression [22, 23] — are trained by using a permission dataset of 20,012 Android applications. The aim is not to design a new algorithm, but to lay out a strict baseline - a common cleaning pipeline, a consistent train/test split for all six models, 10-fold cross validation and a complete list of common evaluation artifacts - precision, recall, F1, roc AUC, confusion matrices, and feature importance - so that future work on this or similar datasets can be compared. The ensemble, kernel and neural methods allow a benchmark to measure the usefulness of ensembling the RF and XGBoost models against a single Decision Tree and the usefulness of having a purpose-built model for classification, instead of using a thresholded regression model.

This paper makes the following main contributions:

- An extension to our previous ransomware permission-based study [1] done with our own curated collection of 1,990 applications and four classifiers to the public application listing of 20,012 apps and six algorithm families in the classifiers, accompanied by a comparison of the results.
- A cleaned and reproducible copy of the dataset of 2,745 Android permissions, which has 2,317 informative binary features after removing 428 constant columns with all zero values.
- A fair comparison and validation through a stratified 70/30 hold-out split using 10-fold stratified cross-validation of six families of classifiers: two ensembles (random forest and boosted trees), a single decision tree, a kernel, a neural network and a linear regression baseline.
- The result of an internal feature-importance analysis to determine the most significant permissions in the larger dataset that predict ransomware that is found on popup files. The result of an internal feature-importance analysis determining which most significant permissions in the larger dataset predict ransomware on popup files.
- Empirical auditor finding of a naming convention artifact between the ransomware versus benign subcategories of the larger set of names; discussion of implications of the finding for the audit accuracy reported in [1] and how the normalization process would remove the artifact.

Related Work

Static, Permission-Based Android Malware Detection

Permission analysis has been around for quite a while in Android-Research. Much of the feature engineering in the literature follows the work of Felts et al. [4] on the creation of one of the first systematic tools to detect over-privilege in permissions—Stowaway—and the work that formed the permission map that he and his co-authors used. Barrera et al. [5] suggested a methodology named Self-Organizing Map (SOM) for an empirical analysis of permission-based security models. Wang et al. [7] have investigated Permission induced risk which lead them to demonstrate that a smaller subset of permissions can be highly predictive of malicious behavior. Zhou and Jiang [24] described the Android malware landscape and its maturation with regard to family types and AndroZoo [25] created one of the big repositories of malware continuously updated with malware applications from which a good deal of this literature relies. The DREBIN system of Arp et al. [6] compiled all its static features from permissions, API calls, intents and manifest components, and evaluated their performance with a linear SVM, which achieve 94% recognition rate with 1% false-positive rate. Recent research on DREBIN [26] and more extensive methodological work on the application of machine learning in security [27], [28] have demonstrated that such results are largely impressionistic and result from how the data was sampled and split (pertinent issue in this paper in Section 8.3 on encoding artifact).

Many studies compare classical classifiers based on a list of (dis gó rally) allowed features. Saleem et al. [8] used chi-square and Fisher's exact test to rank permissions with 99.34% accuracy on DREBIN dataset. Arif et al. [9] used 5,000 DREBIN malware and 5,000 AndroZoo benign samples; and, randomly forest was used for achieving a true positive rate of 91.6%. Akbar et al. [10] developed a permission-based detection framework using the classical machine learning models while Syrris and Geneiatakis [11] evaluated the effectiveness of machine learning based on static analysis data and across various machine learning classifiers families, Milosevic et al. [29] showed that permission and source-code features are useful for different families of classifiers. Chrysikos et al. in their work [30] use Random Forest on permission features for answering questions about malware family classification, while Tahreem et al. [31] achieved almost 100% accuracy with Random Forest on 8,078 applications. The theme is that errors reported are more or less sensitive to the manner of collection and categorization of both benign and malicious samples; there are indeed collection-source artifacts that trick into the feature representation, much like those discovered in this paper. Table 1 is a summary of some permission-based studies most similar to this work.

Table 1. Selected permission-based Android malware and ransomware detection studies.

Study	Data	Best model	Reported result
Arp et al. [6]	123,453 benign / 5,560 malware	Linear SVM	94% detection at 1% FPR
Saleem et al. [8]	DREBIN dataset	Feature ranking + classifiers	99.34% accuracy
Arif et al. [9]	5,000 malware / 5,000 benign	Random Forest	91.6% TPR
Alsoghyer and Almomani [12]	Android ransomware / benign apps	Permission-based ML	96.9% detection rate
Tahreem et al. [31]	8,078 APKs	Random Forest	99.11% accuracy
Khan et al. [1]	995 ransomware / 995 benign; 365 permissions	Logistic Regression	97.69% accuracy (FPR 0.023)
This study	10,006 ransomware / 10,006 benign; 2,317 features	XGBoost	91.72% accuracy; ROC-AUC 0.942

Ransomware-Specific Detection

Ransomware Detection Research includes both static analysis and dynamic analysis, or behaviour based, detection research. Moving to the more dynamic aspects we have the Regularized Logistic Regression algorithms in EldeRan [14] which analyze windows API calls, registry operations and activity within the file system; UNVEIL [15] monitors I/O access patterns and

changes in file entropy that occur during a sandbox to detect crypto-ransomware; ShieldFS [32] implements a ransomware-aware file system that self-heals; and CryptoLock [33] monitors user data changes, as a ransomware is detected in a sandbox. To prevent irrevocable loss, Kok et al. [34] proposed a pre-encryption detection algorithm to prevent the crypto-ransomware. With respect to Android, HelDroid [2] has been designed for detection of locking, encryption and ransom-note activities through a combination of taint analysis and text classification and a real-time detector based on user-interface analysis has been suggested by Chen et al. [3] for the detection of Android ransomware. In a similar study carried out on a carefully selected set of ransomware using four WEKA classifiers, we have earlier demonstrated that using application permissions to detect ransomware is a powerful approach [1]. Alsoghyer and Almomani [12] demonstrated the power of application permissions approach to detect ransomware on Android using four classifiers which are included in the WEKA machine learning toolkit. Deep learning models have to date been used to detect Android malware [35] and more recent studies have used Random Forest, and the gradient-boosted tree model for datasets of ransomware [36]–[38] and attention-based LSTM models for detecting ransomware in early stages [39] and for assisting ransomware family classification [40] using explainable models that benefit from recursive feature elimination.

Machine Learning Foundations

The six algorithms are benchmarked in this paper all cover classical statistics, single trees, ensembles, kernel methods or neural networks. The development of Linear Regression goes back to Legendre [22] and Gauss [23] [41] who developed it and extended it into a framework of statistics and correlation by Galton [42]. The base learner for both ensemble methods tested in this paper is a Decision Tree, based on the CART formalism of Breiman et al. [21] which recursively cuts the feature space into axis-aligned regions. Random Forest has been developed by [16] with bootstrap aggregation and random feature subsampling at every split to generate an ensemble of decorrelated decision trees. XGBoost [17] is a second-order gradient boosting algorithm modified for sparse, high-dimensional tabular data which makes it well suited for binary permission vectors. Support Vector Machines [18] which aim to find a hyperplane that best separates the classes and that contains the largest margin, continue to be a good baseline for high dimensional sparse classification as illustrated by DREBIN's own application of linear SVM [6]. The Multi-Layer Perceptrons were developed from the Perceptron [19] and are trained using the back-propagation algorithm [20]. Standard practices for cross validation [46], Evaluation of ROC curves [47] and precision, recall and F1 reporting were adopted in all models implemented using open source scikit-learn [44, 45] and XGBoost [17] libraries.

Dataset Description

In this study an eleven stage pipeline has been developed, the steps of which are summarized in Figure 1. This section summarizes the handling of the data; the last stages are described next in Section 4.

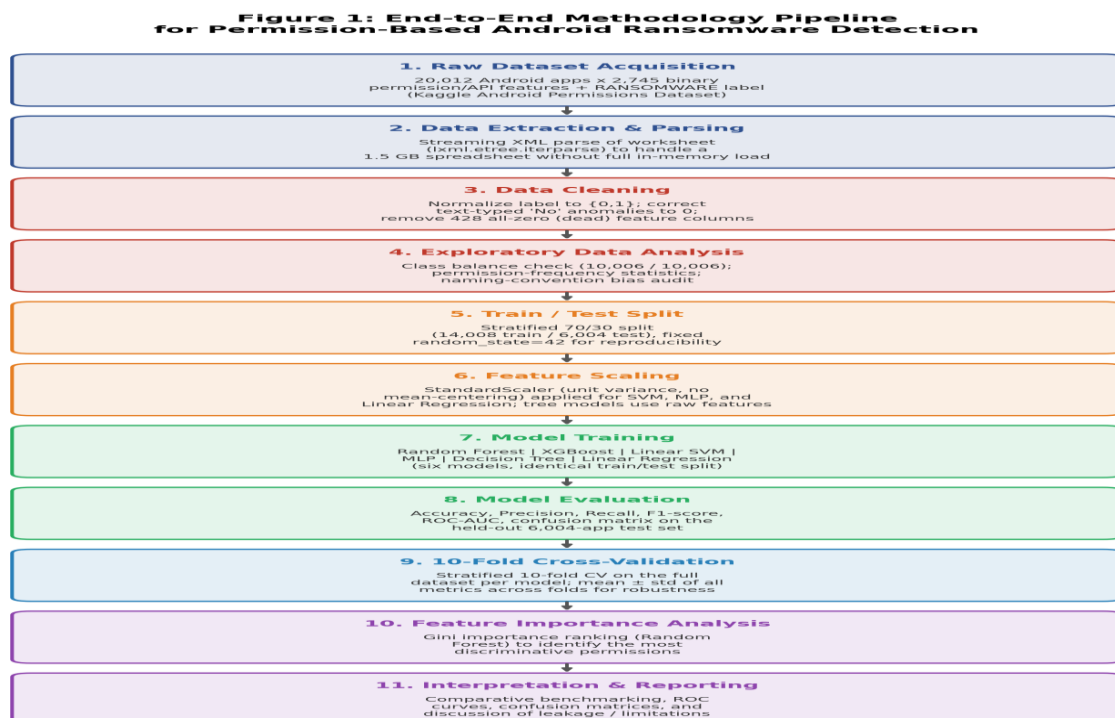


Figure 1. End-to-end (E2E) methodology

In total, the data set that is used in this study comprises 20,012 Android applications; a binary representation of the permissions listed for a particular app, represented with 2,745 binary features; a binary attribute called Google-Play-style permission-group descriptor indicating whether a specific app had a description of a permission group in the Google Play Interface; and a binary attribute called RANSOMWARE (Yes/No), indicating whether a specific app was classified as ransomware. This came from a common Android malware and ransomware detection permissions dataset publicly published [49]. The sample consists of 10,006 ransomware apps and 10,006 normal apps, thus being perfectly class-balanced.

This corpus differs from the one that we sampled and analyzed in our previous work [1]. A comparison of the two is included in Table 2. The larger data set has more applications, more features and a much wider feature space but is not as thoroughly documented in terms of how data was collected or features extracted. The use of it gives those results in [1] (high accuracy using permission features exclusively and high accuracy of linear models) the chance of being transferred to a larger and more diverse set of applications.

Table 2. Examples of the curated corpus from our earlier study compared to data set used in this study.

Property	Khan et al. [1]	This study
Applications	1,990	20,012
Ransomware / benign	995 / 995	10,006 / 10,006
Features	365 permissions	2,745 raw; 2,317 after cleaning
Feature type	Declared uses-permission entries, names normalized	Permissions, API-usage flags and permission-group descriptors
Benign source	Google Play Store	Public permission dataset [49]
Ransomware source	HelDroid, Koodous, public repositories (VirusTotal-verified)	Public permission dataset [49]
Classifiers	LR, Random Tree, Decision Table, Bayes Network (WEKA)	RF, XGBoost, Linear SVM, MLP, DT, Linear Regression (scikit-learn, XGBoost)
Evaluation	10-fold cross-validation	Stratified 70/30 hold-out and 10-fold stratified cross-validation

Data Cleaning

The feature matrixes of this study were pre-processed in 3 stages prior to training. First, NaN was identified by Normalizing the 965 cells in 1 feature column to 0, when they actually were stored as the text string "No" which is 0 on a numeric scale. Second, 428 of the 2,745 on the screen are all the same (zero across all the 2,745 applications), so none of the applications in the dataset requested the permission, and therefore they were considered as uninformative and were removed. This reduced the number of features from 2,745 binary features to 2,317 binary features. Third, the classification was reworked as a binary integer (1 = ransomware, 0 = benign) classification. The class distribution is represented graphically in figure 2.

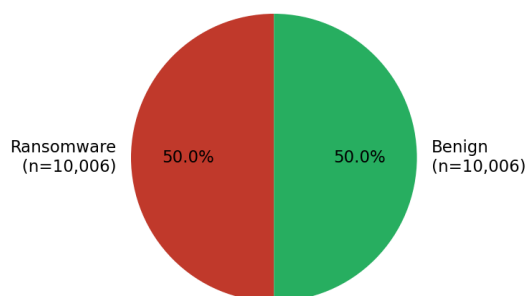


Figure 2. Class distribution (n = 20,012)

Descriptive Statistics

Each request on average requests 6.7 permissions from the cleaned feature set, and the median request is just 3. The distribution is highly right skewed: A few applications ask for just a handful of permissions, while some ask for dozens and - in rare instances - over 280. This data sparsity is typical of features being in a sparse binary format, as in permission based feature matrices [6], [7] and is responsible for selecting a high dimensional data type (high dimensional sparse binary) for classifiers like Random Forest or XGBoost that they can handle without dimensionality reduction.

Methodology

Understanding the Data Structure and Analysis of an XML Document

This was provided as a spreadsheet with the worksheet XML being more than 1.5 GB (about 55 million cell values) when compressed. In order to keep the peak amount of memory greater than 100 Mb as low as possible, a streaming XML parser (lxml.etree.iterparse) was used to process the data row by row, discarding each row element after parsing it so that it stays in memory as little as possible.

Data Cleaning and Feature Reduction

As mentioned in the Section 3.1, text-typed anomaly was normalized, all-zero columns were removed and the label was binarized. The benefit of removing constant columns is that there is no predictive information being thrown away, as there is no mutual information between a constant column and the target and they will have no impact on training time.

Exploratory Data Analysis

To determine the class balance, the distribution of permission counts per application, and the distribution of application-usage per-class permission counts (Sections 3 and 8), we looked at the distribution of students in the classes and the number of permissions they used (app, and class, respectively) relative to the total number of students in the classes. This step uncovered an artifact in the naming convention, which is reported in Section 8.3, revealing the need to audit feature provenance before assuming reliability on the validity of headline accuracy numbers [26] & [28].

Train/Test Split

The data was cleaned, and then partitioned into a training set (70% = $n = 14,008$), a test set (30% = $n = 6,004$) using stratified sampling, with the ratio of classes characterized by the same ratio across both sets, a fixed random seed (random_state = 42) for repeatability. There are no applications listed in both partitions. Furthermore, 10-fold stratified cross validation was used for validation of each model (in 4.7).

Feature Scaling

The single decision tree actually learned directly from the binary feature matrix, as were the tree-based models (Random Forest, XGBoost) since decision-tree splits are invariant to monotonic feature scaling. The Linear SVM, the MLP and the Linear Regression scaled the features using scikit-learn's StandardScaler (with the mean not being subtracted, so as to not lose sparsity and variance being set to 1).

Evaluation Metrics

The six models have been trained on the same training partition and tested once on the same held-out test partition. Considering ransomware as true-positive, with TP, TN, FP and FN denoting true positive, true negative, false positive and false negative, respectively, we report accuracy = $(TP + TN) / (TP + TN + FP + FN)$; precision = $TP / (TP + FP)$; recall, or true-positive rate (TPR) = $TP / (TP + FN)$; false-positive rate (FPR) = $FP / (FP + TN)$; the F1-score, the harmonic mean of precision and recall [48]; and the area under the ROC curve (ROC-AUC) [47]. The native output of Linear Regression consists of a continuous value instead of a probability of class; the predictions were thresholded at 0.5 to yield a binary decision, and directly the raw continuous output from the model was used to compute ROC-AUC. To reveal error trade-offs for each approach - important in a security context that a ransomware is often not malicious and a legitimate app might appear malicious to the casual user - confusion matrices were recorded.

10-Fold Stratified Cross-Validation

The single hold-out split was complemented by a re-evaluation of each model using 10-fold stratified cross-validation (CV) on the whole cleaned set of data [46]. Data were divided into ten disjoint, class stratified folds, each containing roughly 2001 applications, the data for each model were subsequently retrained 10 times using a different fold of the data each time, and the

final scores reported are the average and standard deviation over the 10 folds that were not included in the train set. This is the same protocol used in our previous study [1] which allows comparison in a direct manner to the cross validations results of the two studies. The same preprocessing steps as for the hold-out experiment were used; thus, the results of the two experiments consist of an internal consistency check.

Machine Learning Algorithms

The following section explains algorithms used and their hyperparameters, which were benneched in this study.

Random Forest

Random Forest [16] is an ensemble method where multiple models are trained as a collection of decision trees then predict the most frequently occurring class over the varying models. Each tree is grown with a resampling of the training set and the subset of features fed into the learner at each split point is a random subset to remove the correlation between the trees and minimize variance when compared with a single tree [16] [50]. Random Forest works well with high-dimensional, sparse binary feature matrices, implicit feature selection is performed at the split, and it is robust to the presence of irrelevant or redundant features. We used scikit-learn's implementation [44] with 300 trees, unrestricted depth and the Gini impurity criterion.

XGBoost (Extreme Gradient Boosting)

The regularized version of gradient boosting [43] is XGBoost [17] where trees are added one-by-one and each new tree is trained to minimize the error of the previous ensemble. It adds second order gradient information to the classical gradient boosting, regularization of leaf weights by L1 and L2 norms, and implements a sparsity-aware split-finding algorithm which natively supports zero valued features inherently – a feature that directly applies to sparse permission vectors. XGBoost was defaulted to 300 boosting iterations, max depth of 6 and learning rate 0.1 and the training evaluation method was set to log-loss.

Support Vector Machine (Linear Kernel)

Support Vector Machines [18] classify data by learning a hyperplane that gives the big gap between the margins of the two classes. As shown by DREBIN's implementation of a linear SVM [6] in binary permission vectors of high dimensional sparse data, it can be very efficient to have a linear kernel for such data, and sometimes give a good separation between the two classes. The needed class-probability values to get the ROC-AUC estimates used the sigmoid (Platt) scaling procedure with 3-fold internal cross-validation [52] over LinearSVC configuration with C = 1.0 written in scikit-learn [51].

Multi-Layer Perceptron (Neural Network)

Multi-Layer Perceptron (MLP) is a feed-forward neural network with one or more fully connected hidden layers and one or more sigmoid activation functions that are trained using back-propagation [20] which minimizes a cross-entropy loss. It can model more complex decision boundaries than a linear SVM and is based upon the learning architecture of the Perceptron [19]. We employed a maximum of 200 epochs, ReLU activation [53] and the Adam optimizer [54], internal validation split with a patience of 10 epochs, and the training stopped after 21 epochs because of the early stopping criterion.

Decision Tree

Random Forest is an ensemble of Decision Trees where they are created recursively, and Gradient Boosted Trees, like XGBoost, are also built from a single Decision Tree called a Decision Tree Classifier. The algorithm is based on the CART framework [21]: For each split at an internal node, it compares several features and thresholds to choose the best features, which, based on the Gini index based error measure, are used to split the training data with respect to the chosen threshold as it minimizes the class impurity. The algorithm is as follows: It splits the training data at each internal node with respect to the feature and threshold that minimize the class impurity (the Gini index based error measure). When new samples appear, they were sorted using the made decisions dictating from root to leaf. Despite the relative simplicity of the single tree, it is the most idiosyncratic learner in this study since an unrestricted tree may split until it conforms to the peculiarities of the training set [50]. We implemented scikit-learn's algorithm [44] with the Gini criterion, with no restrictions in its depth (not using pruning model) and trained it as a baseline to be used as a reference to assess both the interpretability and the performance of the two ensembles.

Linear Regression

Linear Regression, the classical (ordinary least squares) method dates back to the work of Legendre [22] and Gauss [23], [41] on least squares, and was later expressed statistically by Galton [42]. The model consists of $w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$, where the values of the coefficients w are chosen to minimize the sum of squared residuals (the errors squared and added

together) from the equation. Linear Regression is used on continuous targets and is provided as a neat and simple linear baseline: the output of the continuous baseline was thresholded at 0.5 to get a ransomware/benign decision and the unthresholded output was used to calculate ROC-AUC. It does not represent a probabilistic decision boundary and is not required to be 'clipped' to lie between 0 and 1, so it is not necessarily as suitable for the task as are the dedicated classifiers in Sections 5.1–5.5, and its inclusion merely measures the gain that comes from having a dedicated classifier in place that uses the same feature matrix. A closed-form OLS solver provided by scikit-learn [44] was used on the features in the vector representation that were scaled in the same way as for the SVM and MLP (Section 4.5).

Experimental Setup

All experiments were coded in Python and ran on Random Forest with scikit-learn [44] and scikit-learn [45] for linear SVM, the MLP, the Decision Tree and the linear regression, and graduated by using the official library for XGBoost [17] for gradient boosting. Table 3 shows the configuration.

Table 3. Experimental configuration and hyperparameters.

Parameter	Value
Dataset size	20,012 applications × 2,317 features (after cleaning)
Class balance	10,006 ransomware / 10,006 benign (50% / 50%)
Train / test split	70% / 30%, stratified, random_state = 42
Training set size	14,008 applications
Test set size	6,004 applications (3,002 per class)
Random Forest	300 trees, unrestricted depth, Gini criterion
XGBoost	300 rounds, maximum depth 6, learning rate 0.1
Linear SVM	LIBLINEAR solver, C = 1.0, Platt-calibrated (3-fold)
MLP	Hidden layers (128, 64), ReLU, Adam, early stopping (patience 10)
Decision Tree	Gini criterion, unrestricted depth (CART)
Linear Regression	Closed-form OLS, 0.5 decision threshold
Feature scaling	StandardScaler (no mean-centering) for SVM, MLP and Linear Regression only
Evaluation metrics	Accuracy, precision, recall, F1-score, ROC-AUC, FPR
Cross-validation	10-fold stratified, shuffled, random_state = 42

Results

Overall Performance Comparison

In Table 4 the accuracy, precision, recall, f1 score and ROC-AUC, as well as wall clock speed for training are shown on the same held out test-set (n = 6004) for all six models.

Table 4. Performance of all six models on the held-out test set (n = 6,004).

Model	Accuracy	Precision	Recall	F1-score	ROC-AUC	Train time (s)
Random Forest	90.94%	87.02%	96.24%	91.40%	0.9329	39.7
XGBoost	91.72%	87.00%	98.10%	92.22%	0.9416	16.4

Linear SVM	90.92%	86.93%	96.34%	91.39%	0.9384	3.0
MLP (neural network)	90.57%	86.45%	96.24%	91.08%	0.9282	17.6
Decision Tree	89.76%	87.36%	92.97%	90.08%	0.9132	3.1
Linear Regression	90.89%	87.07%	96.04%	91.34%	0.9308	10.3

At training speed, XGBoost was slightly slower (about 2.4 times) than Random Forest, and it was slightly faster than all other models, with the exception of its precision (87.00%) which was slightly worse than the single Decision Tree. Random Forest, the Linear SVM and the MLP had very similar accuracy (90.6% – 90.9%), implying that the sequential error correction of the gradient boosting provides a small amount of benefit compared to either bagging (Random Forest) or a linear or shallow nonlinear decision boundary (linear SVM or MLP) on this feature representation. It took less than 3 seconds to train the Linear SVM compared to more than 13 seconds in the case of Random Forest (which is also quite impressive), and achieved the same F1 score, and thus would be a good baseline lowlatency classification solution to deploy on-device or in a resource constrained setup.

The single Decision Tree was the poorest classifier with an accuracy of 89.76%, an F1 score of 0.9008, and an ROC-AUC value of 0.9132. It learned the small details of the training data, as is typical of fetching an unensembled "very ill-equipped tree" in an unrestricted, 2,317-dimensional sparse feature space, rising to 160 levels, 1,321 leaves. The gap illustrates the real gains accomplished by an ensemble of 300 trees, namely, the bagging on 300 trees (Random Forest) gained approximately 1.2 percentage points of accuracy and 0.020 ROC-AUC; the boosting on 300 trees (XGBoost) gained about 2.0 percentage points of accuracy and 0.028 ROC-AUC.

Interestingly, Linear Regression, although not a specially designed classifier, performed competitively with some of the classifiers, including accuracy 90.89%, F1-score 0.9134, and ROC-AUC 0.9308, better than the single Decision Tree for every metric, except for precision, and better than the MLP for every metric except recall. This means that much of the discriminative signal in this permission space is close to being linearly separable: if we fit a thresholded least-special model of the data, we will already get most of what the MLP and the single tree get. This trend was also observed in the previous survey [2] that gave superior performance to the linear Logistic Regression model, compared to the tree-based, rule-based and Bayesian models.

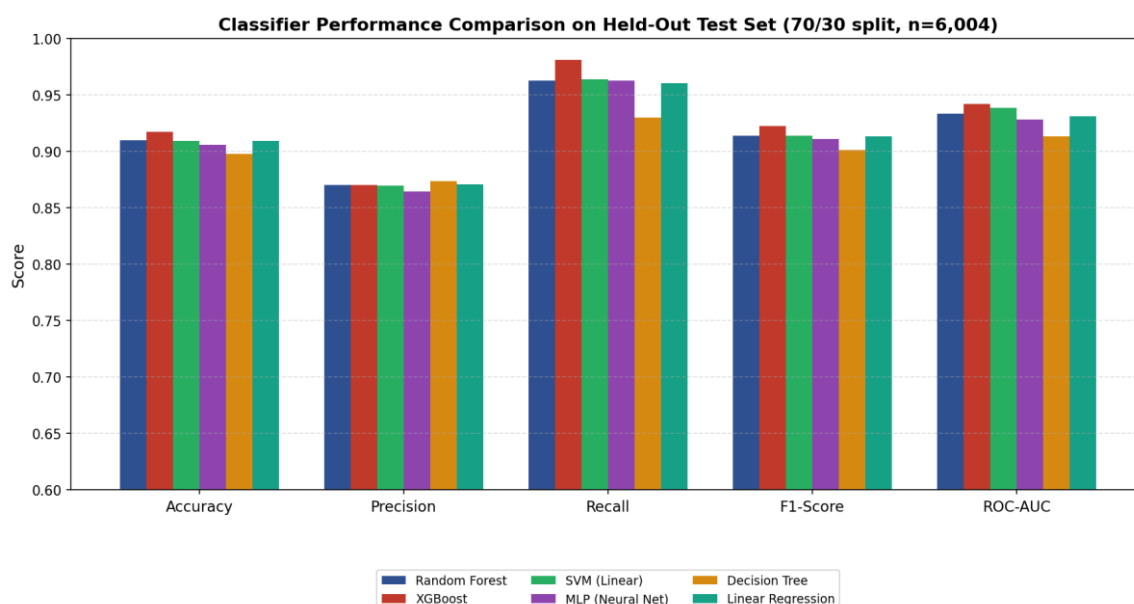


Figure 3. Correctness, precision, recall, F1-score and ROC-AUC results for all six models on test set not included in training.

ROC Curve Analysis

All the ROC curves of the six models are displayed in figure 4 [47]. All 6 models clearly outperform random guessing (AUC is 0.500) with the highest score from ROC-AUC 0.913 to 0.942 for the XGBoost model. With the five top curves tightly grouped,

it is clear that some kind of signal is learned from the permission feature set, and that this signal seems to be the same across the model families (consistent with the results reported in Section 8.3 - a small number of highly discriminative features dominate the decision, regardless of specific model type). The red Decision Tree looks overfitted, as seen in the visibly reduced curve.

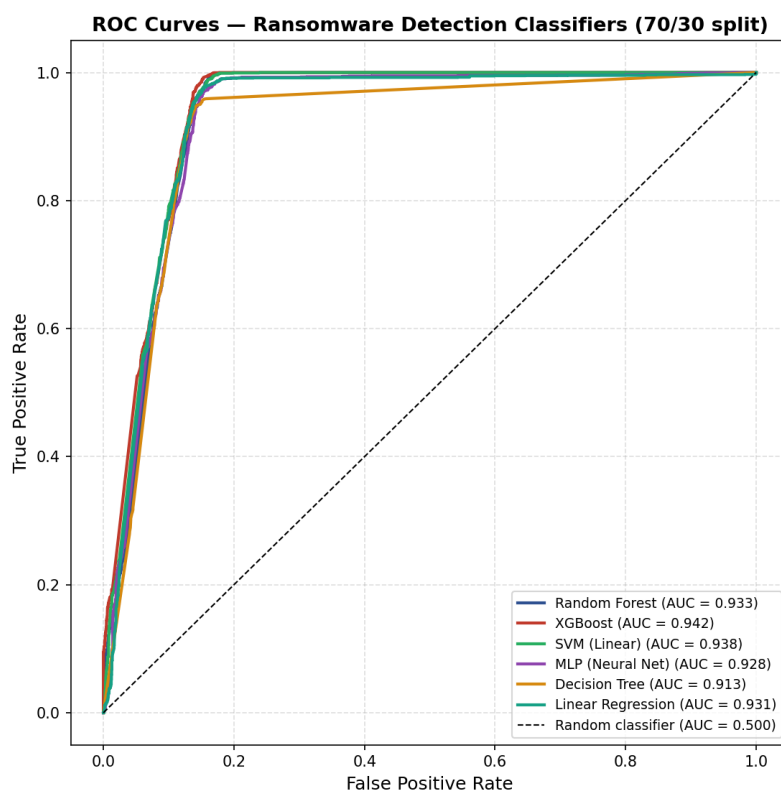


Figure 4. The ROC curves for all six models trained on the held-out test set (70/30 split).

Confusion Matrix Analysis

The confusion matrix of each of the six models is shown in Tables 5 and Figure 5. Note that each model tends to overclassify in clearly higher than overclassify results — that is, recall of the ransomware class (93.0%–98.1%) is very high when compared to precision, meaning that there is a significant chance that the classifiers will classify an innocuous application as ransomware. The false-positive rates are from 13.5% to 15.1%. A false-positive rate this big would be way too bad to pass for an application-store verification pipeline without a second-stage review in a security operations scenario, but it would be quite good in a basic security scenario. The single Decision Tree had the highest rate of false negatives (211), indicating that, despite the interpretability, it is the least reliable of all models to trap a ransomware.

Table 5. Confusion-matrix counts and false positive on held-out test-set (6,004 samples totalling 3,002 samples per class).

Model	TN	FP	FN	TP	FPR
Random Forest	2,571	431	113	2,889	14.36%
XGBoost	2,562	440	57	2,945	14.66%
Linear SVM	2,567	435	110	2,892	14.49%
MLP (neural network)	2,549	453	113	2,889	15.09%
Decision Tree	2,598	404	211	2,791	13.46%
Linear Regression	2,574	428	119	2,883	14.26%

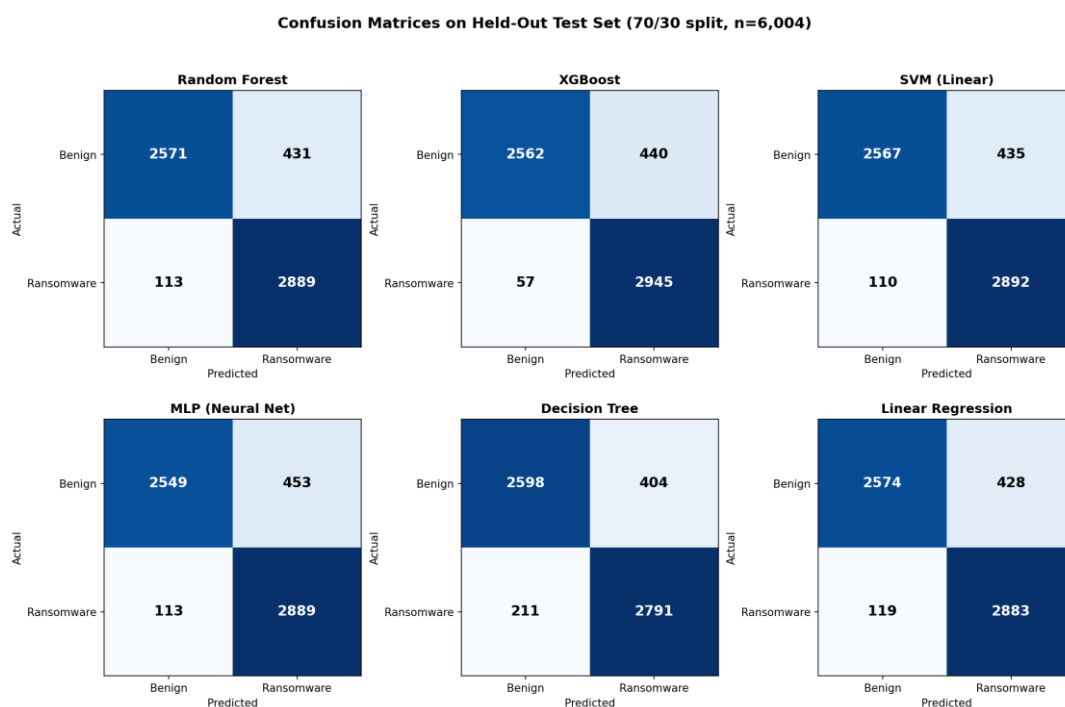


Figure 5. All the six models have been validated on a held-out test set using confusion matrices.

Training Time Comparison

Figure 6 compares wall-clock training time on identical hardware (single-core CPU, no GPU). The Linear SVM and the Decision Tree were fastest (3.03 s and 3.07 s), followed by Linear Regression (10.3 s), XGBoost (16.4 s) and the MLP (17.6 s); Random Forest was slowest (39.7 s) because it builds 300 unrestricted-depth trees over 2,317 features. The single Decision Tree — the base learner that Random Forest replicates 300 times — trains almost 13 times faster than the ensemble while giving up about 1.2 percentage points of accuracy, illustrating the classic accuracy/compute trade-off of ensembling.

Another comparison is made on identical hardware (single core CPU, without GPU), as shown in Figure 6. Linear SVM and the Decision Tree were fastest (taking 3.03 s and 3.07 s respectively), followed by Linear Regression (10.3 s), XGBoost (16.4 s) and the MLP (17.6 s); Random Forest took the longest time (39.7 s), as it creates 300 “unrestricted” trees, each having a maximum depth of 1,268 features. The single Decision Tree, base learned by Random Forest, has almost 13 times the speed gain when training with the same accuracy loss of ~1.2%, demonstrating the classic trade-off between accuracy and compute of ensembling.

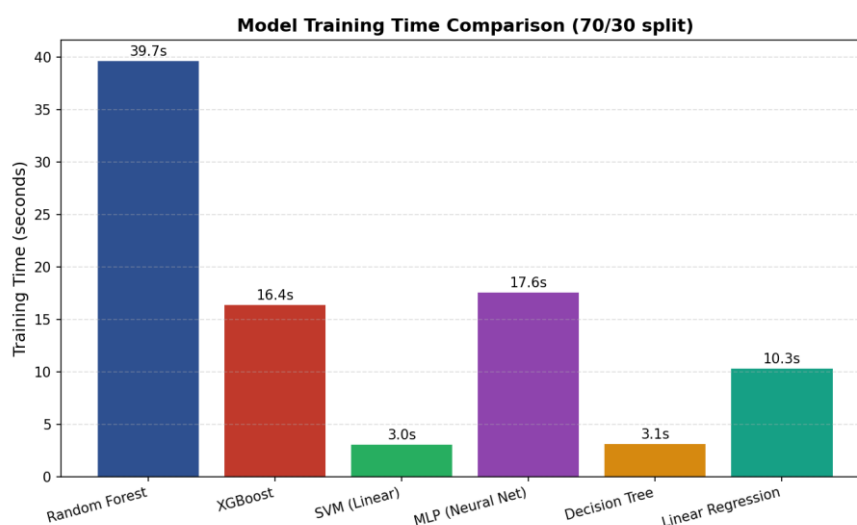


Figure 6. Wall-clock training time of all six models.

Feature Importance Analysis

In Table 6 and Figure 7, the list of 15 most important features in the Random Forest model (70/30 split) is shown. While the ability to tap into the Internet for information has a significant impact, this input accounts for approximately 38.9% of total importance, over 3 times more important than the next most important feature, so that the classifiers rely mostly on a few important features instead of a wide range of permissions.

Table 6. Random forest Gini importance is used to obtain the top 15 important features.

Rank	Feature (permission / API)	Gini importance
1	Network_communication_full_Internet_access	0.3889
2	Network_communication_view_network_state	0.1020
3	Phone_calls_read_phone_state_and_identity	0.0400
4	Storage_modify_delete_USB_storage_contents_modify_delete_SD_card_contents	0.0370
5	ACCESS MOCK LOCATION	0.0265
6	Your_location_fine_GPS_location	0.0250
7	Your_location_coarse_network-based_location	0.0247
8	Hardware_controls_control_vibrator	0.0239
9	ACCESS_WEAUTHERCLOCK_PROVIDER	0.0149
10	GOOGLE_blogger	0.0133
11	System_tools_prevent_device_from_sleeping	0.0123
12	ACCESS_NOTIFICATION_POLICY	0.0113
13	ABACK	0.0100
14	Network_communication_view_Wi-Fi_state	0.0092
15	ACCESS_NOTIFICATIONS	0.0083

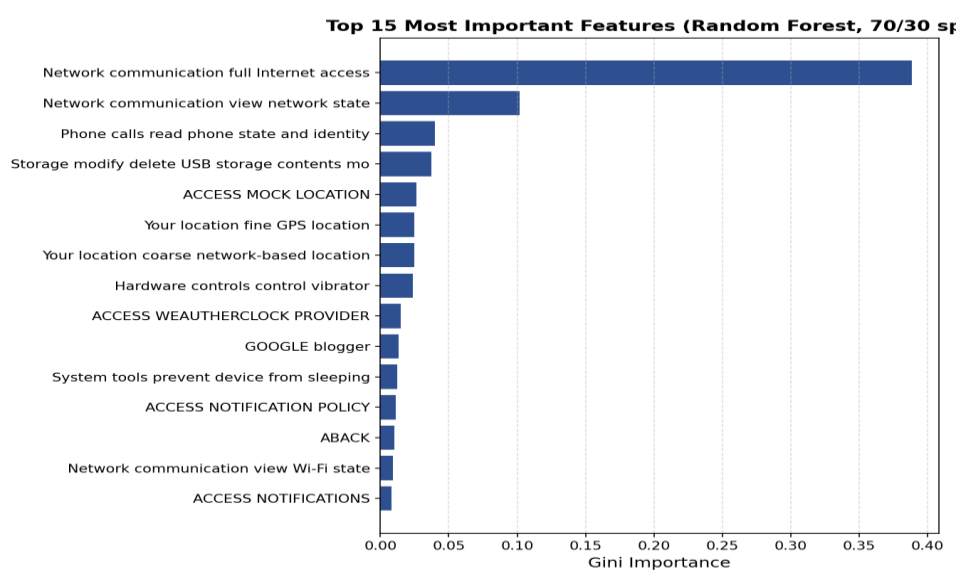


Figure 7. Top 15 features using Random Forest Gini importance.

10-Fold Stratified Cross-Validation

In order to gauge the sensitivity of the performance of each of these models to the specific 70/30 ratio, each of these models was also tested by 10-fold stratified cross validation [46] over the entire dataset. There were 10 runs and each model was trained to 9 folds (18,011 applications) and tested on the 10th fold (2,001 applications). Since all applications are executed as test data only once, this provides a more reliable estimate of the generalization performance than a single split and the standard deviation gives an indication about the variation in performance depending on the partition.

Table 7. Results with 10-fold cross validation (mean $\pm$ std within folds).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	ROC-AUC
Random Forest	90.95 $\pm$ 0.66	86.80 $\pm$ 0.95	96.61 $\pm$ 0.45	91.44 $\pm$ 0.58	0.9328 $\pm$ 0.0056
XGBoost	91.65 $\pm$ 0.72	86.90 $\pm$ 1.04	98.12 $\pm$ 0.48	92.17 $\pm$ 0.64	0.9406 $\pm$ 0.0046
Linear SVM	90.47 $\pm$ 0.54	84.40 $\pm$ 0.76	99.30 $\pm$ 0.26	91.25 $\pm$ 0.46	0.9153 $\pm$ 0.0082
MLP (neural network)	90.89 $\pm$ 0.76	86.71 $\pm$ 0.97	96.60 $\pm$ 0.77	91.38 $\pm$ 0.69	0.9274 $\pm$ 0.0077
Decision Tree	89.61 $\pm$ 0.77	86.93 $\pm$ 1.00	93.25 $\pm$ 0.57	89.98 $\pm$ 0.71	0.9142 $\pm$ 0.0071
Linear Regression	90.54 $\pm$ 0.78	86.79 $\pm$ 0.90	95.65 $\pm$ 0.83	91.00 $\pm$ 0.72	0.9249 $\pm$ 0.0041

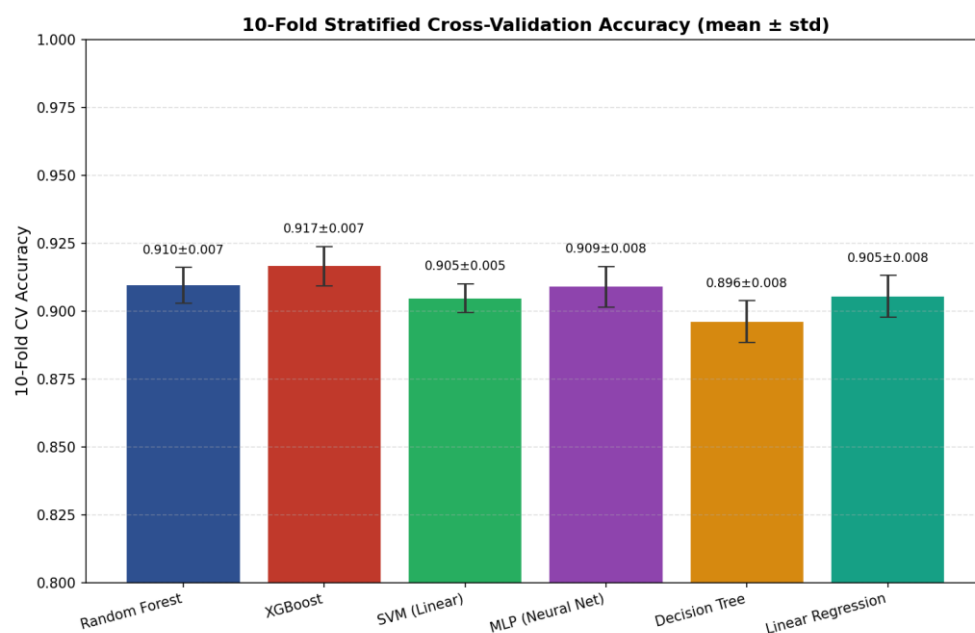


Figure 8. 10-fold stratified cross-validation accuracy

The cross-validated results (Table 7) correspond well with the results obtained in the hold-out section (Table 4), which also serves as a cross-validated result. XGBoost again achieved the highest mean accuracy (91.65% $\pm$ 0.72%) and ROC-AUC (0.9406 $\pm$ 0.0046), followed by Random Forest (90.95% $\pm$ 0.66%) and the MLP (90.89% $\pm$ 0.76%), with the single Decision Tree again weakest (89.61% $\pm$ 0.77%). All standard deviations are below $\pm$ 1%, meaning the performance of each model is not the result of a biased split, and the rank of the 6 algorithms were unchanged across the folds. The highest average (99.30%) and lowest

variance cross-validated recall (0.26%) of the Linear SVM trades the lowest cross-validated precision (84.40%) making it appealing when minimizing ransomware is important.

Discussion

Comparison with Our Earlier Study

With the same 10-fold cross validation strategy, Logistic Regression (best model on the curated corpus [1]) achieved 97.69% accuracy and a false positive rate of 0.023 on the trained set, while the best model on the big set XGBoost had an fpr of ~14.7% on the hold out set under the cross validation setup. The gap is likely to be due to the following reasons. Larger data set is ten times larger and more varied, which means that it will probably have a larger number of borderline applications, whose access profiles intersect. Secondly, all apps in the curatorised corpus were directly decompiled and all of the manifest files were extracted, while the process of picking out apps and assigning a label to ransomware is more sketchy for the larger set. Third, they have different spaces of interfaces, where [1] has 365 normalized permission names, while the larger one combines 2,317 permissions with permission-group descriptors and flags representing its usage with 2 different interface names (Section 8.3). Lastly, approximately 73% of the samples in [1] that were decompiled were candidate samples of ransomware; the ransomware class might be over-represented by the easier-to-decompile samples and perhaps the easier-to-classify ones, too.

The two studies have some striking commonalities. In both, a model containing only permission was significantly better at distinguishing ransomware from the benign applications than by chance, and both Logistic Regression and a thresholded Linear Regression were highly competitive: Logistic Regression was best on the hold-out split of the four classifiers examined in [1] and here a thresholded Linear Regression was only 1 percent removed from the best model on the hold-out split. Unlike the 15 most important features found in [1], the most discriminatory permissions fail to appear here (Table 6): RECEIVE_BOOT_COMPLETED, SYSTEM_ALERT_WINDOW and MOUNT_UNMOUNT_FILESYSTEMS. This difference should not be interpreted until both the naming conventions have been unified, since it is driven by mainly descriptive style features which are confounded with class, in the larger dataset (Section 8.3).

Comparison with the Broader Literature

This accuracy range, (89.6%–91.7%), is lower than the highest such accuracy found for permission-based Android malware detection, for example, 99.34% accuracy in the evaluation of Arif et al. by RF using feature ranking [8] albeit with greater number of samples and 8,078 applications used for evaluation, compared to 231 in this case, that yielded a 99.11% accuracy [31]. This is expected because studies that add additional static information (API calls, intents, code structure, as done by DREBIN [6]) or use a powerful permission selection method to select a compact, high-discriminative feature subset [7] [8] generally are more accurate than the permission-only baseline performed here, which is weakly discriminative because the features were used individually. There are also known problems of sample and division of reported accuracy making comparison between datasets difficult [26, 27].

A Naming-Convention Encoding Artifact

During exploratory analysis (Sections 3 and 4.3) we found that the 2,317 retained feature columns use two distinct naming styles: 236 "descriptive" columns use Google-Play-style human-readable phrasing (e.g., Network_communication_full_Internet_access), and the remaining columns use raw upper-case Android manifest permission constants (e.g., ACCESS MOCK_LOCATION). These two families are not evenly distributed across the classes: 86.5% of benign applications activate at least one descriptive-style feature, compared with only 13.5% of ransomware applications, while 60.4% of ransomware applications activate at least one upper-case feature, compared with 29.5% of benign applications. This strongly suggests that the two class subsets were collected or feature-extracted through different pipelines or tools, and that a systematic naming difference — rather than purely behavioral differences between ransomware and benign applications — contributes measurable signal to the classifiers. This is a form of data leakage [55], a pitfall that is common in machine learning for security [28].

The feature-importance ranking (Table 6, Figure 7) corroborates this finding: the most important feature by a wide margin (Gini importance 0.389) is descriptive-style, and seven of the eight top-ranked features are descriptive-style. A classifier that can separate ransomware from benign applications largely by the naming convention in which a permission is encoded is not learning a robust behavioral signature of ransomware; it is partly learning a data-collection artifact. Similar evaluation biases have been documented in re-analyses of DREBIN [26], which reinforces that accuracy figures for permission-only classifiers should be read alongside an audit of feature provenance. The remedy is straightforward: in our earlier study, extracted permission names were normalized by stripping standard and vendor-specific prefixes and consolidated into a single vocabulary

[1]. Applying an equivalent normalization – mapping each descriptive-style feature to its manifest constant – would remove this artifact.

We report this finding transparently because it affects how the accuracy figures in Tables 4 and 7 should be read: the artifact-free detection performance of these classifiers is likely somewhat lower than reported here. Re-running this benchmark after collapsing both naming families into a single taxonomy, to isolate the purely behavioral signal, is a priority for future work (Section 10).

In exploratory analysis (Sections 3 and 4.3) two different naming styles of these feature columns were identified: 236 "descriptive" feature columns are named using a Google-Play style human-readable, natural language statement (e.g., `Network_communication_full_Internet_access`); the rest of the feature columns are named as raw upper-cased Android manifest permission constants (e.g., `ACCESS MOCK_LOCATION`). Only 13.5% of ransomware applications are detected at all of defensive capabilities, compared to 86.5% of benign ones, and only 29.5% of ransomware applications activate at least one upper-case feature, versus 60.4% of benign applications. It shouldn't be surprising that the two subsets of classes have likely been collected or featureextracted using separate pipelines or tools; the presence of some systematic naming difference, and not only behavior difference between ransomware and legitimate applications, imparts measurable signal to the classifiers. This is a type of data leakage [55] which is frequently found in machine learning in security applications [28].

Indeed, the feature importance ranking (Table 6, Figure 7) confirms this observation and the most important feature, by more than a factor of 2, is descriptive-style, with 7 of the 8 most important features being descriptive-style. Displaying a list of ransomware versus other applications largely by the naming convention used for a permission is not a learning of a robust ransomware behavioral signature, it's partially a re-learning of a data-collection artifact. These types of evaluation bias have been reported in re-analyses of DREBIN [26], which further underscores the need to read accuracy numbers in conjunction with a feature provenance audit for permission-only classifiers. The solution is simple: In our previous work, we extracted the permission names and removed both standard and vendor-specific prefixes, and combined them into a single vocabulary [1]. This artifact could be eliminated if an equivalent normalization (in which each feature of the descriptive string is created into a manifest constant) were applied.

This result is reported candidly since it impacts the interpretation of the accuracy in Tabs. 4 and 7: the actual accuracy of these classifiers in artifact-free detection will tend to be slightly lower than reported here. Future works should include re-running this benchmark after collapsing both naming families into a single taxonomy, to separate the pure behavioral signal as listed in Section 10.

Security Implications of the Top Discriminative Permissions

The top-ranked upper case permissions can be plausibly associated to these ransomware attacks mentioned in literature [2],[3] apart from the naming artefact set aside. Accesses to notification behavior are provided by `ACCESS_NOTIFICATION_POLICY` and `ACCESS_NOTIFICATIONS`, both of which can be exploited by the locker ransomware to keep the ransom message always in front of the user or prevent them from getting an alarm, and access for supplying false location data is given by `ACCESS MOCK_LOCATION`. These observations come with the naming artifact and they are directional rather than definitive – they should be revisited on a leakage-corrected feature set, in conjunction with the persistence, overlay and storage permissions observed in [1].

Limitations and Threats to Validity

There are limitations of this study which must be taken into consideration when interpreting the results. First, due to naming conventions, the two classes will converge, leading to an inflated accuracy when considering the output (see Section 8.3), and hence the results can be seen as an upper bound, and should be redone after correcting for the leakage (see Section 8.3). Second, there is a temporal and spatial bias in evaluation. The fact that random splits include examples from all time periods makes them inappropriate when the Android version of the model is deployed in newer applications than the versions it was trained on to limit transferability [27] while the number, and type, of permissions granted at each Android API version has changed over time [56] further reduces transferability. Thirdly, no systematic optimization of hyperparameters – e.g., grid search or Bayesian optimization – was conducted, with models run using default or slightly optimized parameters to represent a baseline for the purpose of the study. Fourth, unlike other systems such as EldeRan [14], UNVEIL [15] and ShieldFS [32] rely solely on the statically extracted behavioral features that include API call sequences, activity in the file system and network usage. Sixth, researchers didn't test the system for robustness against evasion attacks despite the fact that permissions may be granted or revoked without altering the malicious behavior of the application. Last, the 50/50 category split is not a true representation of how ransomware actually comprises apps in real app stores, as it does not represent a significant proportion.

Conclusion and Future Work

This paper enhanced our previous permission-based study on Android ransomware in [1] by using a common dataset of 20,012 applications, which were class-balanced, and introducing six supervised learning algorithms: Random Forest, XGBoost, a Linear SVM, a Multi-Layer Perceptron, a single Decision Tree and Linear Regression with a reproducible baseline benchmark. The dedicated classifiers, namely XGBoost, Random Forest, Linear SVM (0.927 ± 0.017) and the MLP (0.926 ± 0.011) fared better than the single Decision Tree (0.927) due to overfitting, and the linear regression (91.21) with the threshold remained close to the dedicated classifiers, suggesting high-scale linear separability in the permission space also, consistent with the strong performance of Logistic Regression in [1]. The margin-based decision boundaries and ensembling algorithms perform a consistent advantage when compared to decision tree or naive linear fit on this task while the quality of the features is more important for the choice of the algorithm. The accuracy on the larger dataset ranges from about 6 percentage points lower than what is found for our curated dataset, with all models having excessive false-positive rates, in excess of 13-15%, for deployment on its own. A further artifact that we found in the larger data set was one that involved the naming of the classes, which most likely accounts for an inflated accuracy for all of the six models that we took into consideration and should be corrected prior to use of the data set in the production classifiers.

Further work that builds upon the baseline should take the following directions:

- Simply combine the permission naming (descriptive and upper-case) families into a unified taxonomy, similar to normalisation performed in [1] and rerun this benchmark to assess the detection performance in artifact-free mode.
- Assess generalization in a different dataset, using one to train and the other to test, with a common "Permission Vocabulary" (PV) obtained through a standard normalization procedure over a common set of permissions (1,990-20,012).
- Then, add hyperparameter optimization within each CV fold (nested cross-validation) to get CI over tuned instead of default configurations [46].
- Use dynamic/behavioural signals in the form of API-invocations, file-system events, and network activity as shown in EldeRan [14] and ShieldFS [32] to determine if a model which combines static and dynamic models can regain this loss of accuracy once the encoding artifact has been removed.
- Based on previous permission-ranking efforts[7] and [8] perform feature selection to identify an artifact-free, minimal and accurate set of permissions, using feature selection methods such as mutual information, recursive feature elimination, SHAP-based selection.
- Compare pipeline performance in realistic, unbalanced setting where ransomware is prevalent, resorting to resampling techniques like SMOTE [58] as needed, and with time-aware train/test splits [27].

References

- M. A. Khan, M. Nadeem, and J. Muhammad, "A permission-based machine learning model for Android ransomware detection," *Spectrum of Engineering Sciences*, vol. 4, no. 3, pp. 4818-4831, 2026, doi: 10.5281/zenodo.21818735.
- N. Andronio, S. Zanero, and F. Maggi, "HelDroid: Dissecting and detecting mobile ransomware," in *Research in Attacks, Intrusions, and Defenses (RAID 2015)*, *Lecture Notes in Computer Science*, vol. 9404. Cham, Switzerland: Springer, 2015, pp. 382-404, doi: 10.1007/978-3-319-26362-5_18.
- J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du, and G.-J. Ahn, "Uncovering the face of Android ransomware: Characterization and real-time detection," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1286-1300, 2018, doi: 10.1109/TIFS.2017.2787905.
- A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. 18th ACM Conf. Computer and Communications Security (CCS)*, Chicago, IL, USA, 2011, pp. 627-638, doi: 10.1145/2046707.2046779.
- D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to Android," in *Proc. 17th ACM Conf. Computer and Communications Security (CCS)*, Chicago, IL, USA, 2010, pp. 73-84, doi: 10.1145/1866307.1866317.
- D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. Network and Distributed System Security Symp. (NDSS)*, San Diego, CA, USA, 2014, doi: 10.14722/ndss.2014.23247.

- W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in Android applications for malicious application detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869–1882, 2014, doi: 10.1109/TIFS.2014.2353996.
- M. S. Saleem, J. Mišić, and V. B. Mišić, "Android malware detection using feature ranking of permissions," *arXiv preprint arXiv:2201.08468*, 2022.
- J. M. Arif, M. F. Ab Razak, S. Awang, S. R. Tuan Mat, N. S. N. Ismail, and A. Firdaus, "A static analysis approach for Android permission-based malware detection systems," *PLOS ONE*, vol. 16, no. 9, Art. no. e0257968, 2021, doi: 10.1371/journal.pone.0257968.
- F. Akbar, M. Hussain, R. Mumtaz, Q. Riaz, A. W. A. Wahab, and K.-H. Jung, "Permissions-based detection of Android malware using machine learning," *Symmetry*, vol. 14, no. 4, Art. no. 718, 2022, doi: 10.3390/sym14040718.
- V. Syrris and D. Geneiatakis, "On machine learning effectiveness for malware detection in Android OS using static analysis data," *Journal of Information Security and Applications*, vol. 59, Art. no. 102794, 2021, doi: 10.1016/j.jisa.2021.102794.
- S. Alsoghyer and I. Almomani, "On the effectiveness of application permissions for Android ransomware detection," in *Proc. 6th Conf. Data Science and Machine Learning Applications (CDMA)*, Riyadh, Saudi Arabia, 2020, pp. 94–99, doi: 10.1109/CDMA47397.2020.00022.
- M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009, doi: 10.1145/1656274.1656278.
- D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection," *arXiv preprint arXiv:1609.03020*, 2016.
- A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A large-scale, automated approach to detecting ransomware," in *Proc. 25th USENIX Security Symp.*, Austin, TX, USA, 2016, pp. 757–772.
- L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.
- T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD)*, San Francisco, CA, USA, 2016, pp. 785–794, doi: 10.1145/2939672.2939785.
- C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995, doi: 10.1007/BF00994018.
- F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, doi: 10.1037/h0042519.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, doi: 10.1038/323533a0.
- L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA, USA: Wadsworth, 1984.
- A. M. Ahmed, A. Rizaner, and A. H. Ulusoy, "A decision tree algorithm combined with linear regression for data classification," in *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCEEE)*, 2018, pp. 1–5.
- S. Pathak, I. Mishra, and A. Swetapadma, "An assessment of decision tree based classification and regression algorithms," in *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*, 2018, pp. 92–95.
- Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Security and Privacy (S&P)*, San Francisco, CA, USA, 2012, pp. 95–109, doi: 10.1109/SP.2012.16.
- K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting millions of Android apps for the research community," in *Proc. 13th Int. Conf. Mining Software Repositories (MSR)*, Austin, TX, USA, 2016, pp. 468–471, doi: 10.1145/2901739.2903508.
- N. Daoudi, K. Allix, T. F. Bissyandé, and J. Klein, "A deep dive inside DREBIN: An explorative analysis beyond Android malware detection scores," *ACM Transactions on Privacy and Security*, vol. 25, no. 2, 2022, doi: 10.1145/3503463.

- F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in Proc. 28th USENIX Security Symp., Santa Clara, CA, USA, 2019, pp. 729–746.
- D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in Proc. 31st USENIX Security Symp., Boston, MA, USA, 2022, pp. 3971–3988.
- N. Milosevic, A. Dehghantanha, and K.-K. R. Choo, "Machine learning aided Android malware classification," *Computers & Electrical Engineering*, vol. 61, pp. 266–274, 2017, doi: 10.1016/j.compeleceng.2017.02.013.
- N. Chrysikos, P. Karampelas, and K. Xylogiannopoulos, "Permission-based classification of Android malware applications using random forest," in Proc. 22nd European Conf. Cyber Warfare and Security (ECCWS), vol. 22, no. 1, 2023, pp. 132–142, doi: 10.34190/eccws.22.1.1212.
- M. Tahreem, B. Z. Hussain, I. Andleeb, and A. Hameed, "Machine learning-based Android intrusion detection system," arXiv preprint arXiv:2412.03894, 2024.
- A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barengi, S. Zanero, and F. Maggi, "ShieldFS: A self-healing, ransomware-aware filesystem," in Proc. 32nd Annu. Computer Security Applications Conf. (ACSAC), Los Angeles, CA, USA, 2016, pp. 336–347, doi: 10.1145/2991079.2991110.
- N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and drop it): Stopping ransomware attacks on user data," in Proc. IEEE 36th Int. Conf. Distributed Computing Systems (ICDCS), Nara, Japan, 2016, pp. 303–312, doi: 10.1109/ICDCS.2016.46.
- S. H. Kok, A. Abdullah, N. Z. Jhanjhi, and M. Supramaniam, "Prevention of crypto-ransomware using a pre-encryption detection algorithm," *Computers*, vol. 8, no. 4, Art. no. 79, 2019, doi: 10.3390/computers8040079.
- A. Alhussen, "Advanced Android malware detection through deep learning optimization," *Engineering, Technology & Applied Science Research*, vol. 14, no. 3, pp. 14552–14557, 2024, doi: 10.48084/etasr.7443.
- K. Kunku, A. N. K. Zaman, and K. Roy, "Ransomware detection and classification using machine learning," in Proc. IEEE Symp. Computational Intelligence in Cyber Security (CICS), 2023; arXiv:2311.16143.
- M. Masum, M. J. H. Faruk, H. Shahriar, K. Qian, D. Lo, and M. I. Adnan, "Ransomware classification and detection with machine learning algorithms," in Proc. IEEE 12th Annu. Computing and Communication Workshop and Conf. (CCWC), Las Vegas, NV, USA, 2022, doi: 10.1109/CCWC54503.2022.9720869.
- P. Azugo, H. Venter, and M. W. Nkongolo, "Ransomware detection and classification using random forest: A case study with the UGRansome2024 dataset," arXiv preprint arXiv:2404.12855, 2024.
- A. Alqahtani, M. O. Ohemeng, and F. T. Sheldon, "An intelligent sensing framework for early ransomware detection using MHSA-LSTM machine learning," *Sensors*, vol. 26, no. 3, Art. no. 952, 2026, doi: 10.3390/s26030952.
- R. A. Mowri, M. Siddula, and K. Roy, "A comparative performance analysis of explainable machine learning models with and without RFECV feature selection technique towards ransomware classification," arXiv preprint arXiv:2212.04864, 2022.
- S. M. Stigler, "Gauss and the invention of least squares," *The Annals of Statistics*, vol. 9, no. 3, pp. 465–474, 1981, doi: 10.1214/aos/1176345451.
- F. Galton, "Regression towards mediocrity in hereditary stature," *Journal of the Anthropological Institute of Great Britain and Ireland*, vol. 15, pp. 246–263, 1886, doi: 10.2307/2841583.
- J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001, doi: 10.1214/aos/1013203451.
- F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- L. Buitinck et al., "API design for machine learning software: Experiences from the scikit-learn project," arXiv preprint arXiv:1309.0238, 2013.

- R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in Proc. 14th Int. Joint Conf. Artificial Intelligence (IJCAI), Montreal, QC, Canada, 1995, pp. 1137–1143.
- T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006, doi: 10.1016/j.patrec.2005.10.010.
- D. M. W. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.
- "A. Alomar, A. Aljarullah, and S. Abu-Ghazalah, "Permissions-based Android malware detection using machine learning," *Neural Computing and Applications*, vol. 37, no. 6, pp. 5255–5270, 2025, doi: 10.1007/s00521-024-10950-4.
- J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1993.
- R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.
- J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *Advances in Large Margin Classifiers*, A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, Eds. Cambridge, MA, USA: MIT Press, 1999, pp. 61–74.
- V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in Proc. 27th Int. Conf. Machine Learning (ICML), Haifa, Israel, 2010, pp. 807–814.
- D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in Proc. 3rd Int. Conf. Learning Representations (ICLR), San Diego, CA, USA, 2015; arXiv:1412.6980.
- S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 4, Art. no. 15, 2012, doi: 10.1145/2382577.2382579.
- X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the Android ecosystem," in Proc. 28th Annu. Computer Security Applications Conf. (ACSAC), Orlando, FL, USA, 2012, pp. 31–40, doi: 10.1145/2420950.2420956.
- K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *Computer Security – ESORICS 2017, Lecture Notes in Computer Science*, vol. 10493. Cham, Switzerland: Springer, 2017, pp. 62–79, doi: 10.1007/978-3-319-66399-9_4.
- N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002, doi: 10.1613/jair.953.



2026 by the authors; *Journal of Kashmir Journal of Academic Research and Development*. This is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (<http://creativecommons.org/licenses/by/4.0/>).