Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 43 No. 7
July 2023

Intelligent Android Malware Detection System Using Ensemble Machine Learning

[1]Sumalatha Potteti
Associate Professor, Department of Computer Science and Engineering, Bhoj Reddy Engineering College for Women, Hyderabad, Telangana, India.

[2]Dr. G. S. Mahalakshmi
Associate Professor, Department of Computer Science and Engineering, Anna University, Chennai, Tamandu, India.

Abstract-The privacy and security of users are seriously threatened by the exponential growth of Android devices and the rising prevalence of malware. Because malware is always developing, conventional signature-based malware detection technologies are becoming ineffective. As a result, the demand for sophisticated malware detection systems that can accurately identify fresh and undiscovered malware strains is rising. In order to improve detection accuracy and robustness, this study suggests an intelligent Android malware detection system that makes use of ensemble machine learning techniques. The suggested approach uses an ensemble model created by combining different machine learning algorithms, such as gradient boosting, random forests, support vector machines (SVM), and decision trees. Each base model is trained using a wide variety of features that are taken directly from Android applications, including permissions, API requests, and manifest data. In order to reach a final determination regarding whether an application is malicious or benign, the ensemble model combines the predictions from individual models. An extensive dataset made up of both known and undiscovered malware samples is utilised to assess the system's performance. The experimental findings show that in terms of accuracy, precision, recall, and F1-score, the ensemble model surpasses individual machine learning techniques. The ensemble model successfully lowers false positives and false negatives while achieving a high detection rate for both known and undiscovered malware. The suggested approach also demonstrates remarkable generalisation skills, enabling it to adjust to fresh and undiscovered malware types.

Keywords: Malware detection, machine learning, Ensemble method, SVM, DT, RF

I. Introduction

Smartphones growing popularity and diversity, which includes expensive models with quick CPUs and high-speed Internet connectivity, attract online thieves. Different permissions are frequently required for mobile applications and services to access Android operating system (OS) resources, including as databases, preferences, and files. This, regrettably, allows malware a chance to exploit these permissions and engage in hazardous behaviour. In 2022, more than 3.4 billion Android devices were sold, giving Android an 83.3% market share. The increasing adoption of Android devices and the abundance of mobile applications have fostered a favourable environment for malware attacks.

The Android OS's permissions can be exploited by malware installed on Android devices to carry out unauthorised actions. These actions could involve gaining access to private user information, starting unauthorised communications, adding new malware, or even seizing control of the device. Consequences of such malicious behaviour can include end-user financial losses as well as privacy violations.

Various security measures and remedies have been developed to reduce the hazards brought on by malware on Android devices. These include the deployment of antivirus software, strong authentication systems, and secure application development procedures. In order to reduce the spread of fraudulent apps, Google Play Store, the

official app store for Android, has also put in place rigorous security safeguards. However, malware's dynamic nature presents an ongoing issue. The creators of malware constantly modify their methods in order to avoid detection and take advantage of weaknesses. As a result, sophisticated and intelligent malware detection systems must be created so that the threat posed by constantly changing malware variants may be efficiently identified and reduced.

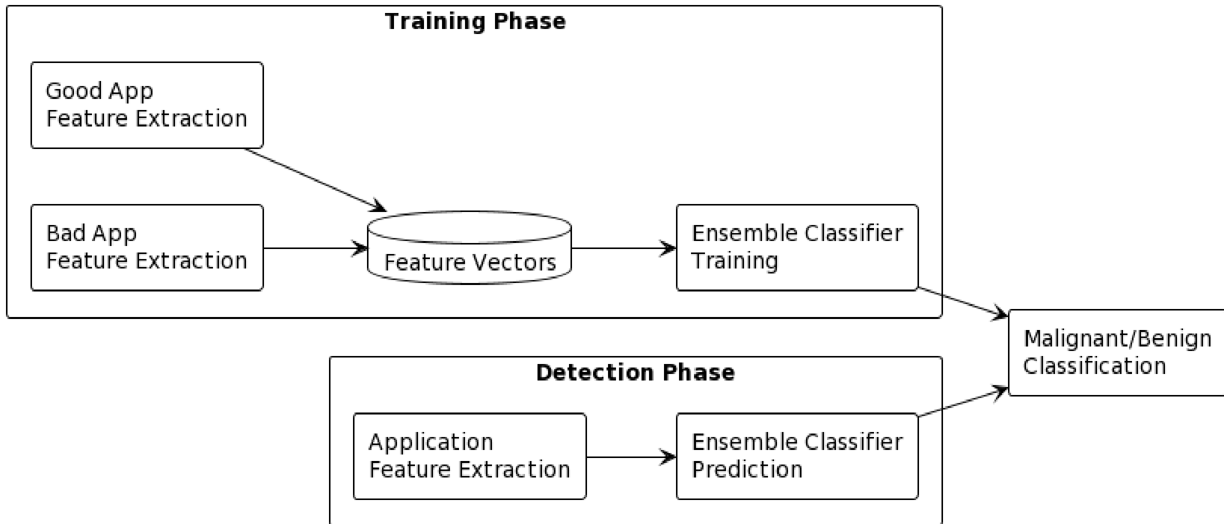

Figure.1 Android Malware Detection System

The Android operating system offers a robust ecosystem of services and applications, and as a result, permissions are required in order to access various resources. However, this also makes it possible for bad actors to take advantage of these permissions and use the Android OS's capabilities for their own evil purposes.A variety of features, including permissions, API calls, and manifest data, are taken from Android applications and used in the proposed intelligent Android malware detection system. The detection system can make deft decisions thanks to these attributes, which offer insightful information about the behaviour and traits of apps. To provide a reliable and effective detection mechanism, the ensemble model is built by mixing different machine learning techniques, such as gradient boosting, decision trees, support vector machines (SVM), random forests, and SVM.

Programme analysis and machine learning approaches are now frequently employed to detect harmful apps. Static and dynamic techniques for programme analysis can be distinguished. Static analysis is a technique used by programmes like Drebin, ICCDetector, and MamaDroid to identify malware by extracting behaviour characteristics from programmes, such as permission requests, API calls, intent kinds, and network addresses. The advantage of static analysis is that it enables speedy scanning and examination of harmful apps.

Dynamic analysis methods, on the other hand, keep an eye on how programmes behave when running in actual or virtual environments. Dynamic analysis can successfully thwart malicious applications' use of code modification techniques by examining how programmes behave while they are being executed. Malicious behaviour that might not be seen during static analysis can be found via dynamic analysis.

The benefit of dynamic analysis is that it can accurately depict an application's actual behaviour regardless of code alterations. Dynamic analysis can identify suspicious activities such unauthorised network communications, sensitive data access, or unexpected resource utilisation by running apps and observing their runtime interactions. This method makes it easier to identify malicious programmes by giving a more

thorough and accurate picture of an application's behaviour.

II. Review of Literature

Due to the quick growth and evolution of Android malware, Android applications are subject to a variety of attacks. Different techniques for analysing and identifying programmes before installation have been developed in order to protect Android applications from malware attacks. The two main categories of these techniques are static analysis and dynamic analysis. Static analysis entails utilising automated tools to look at an Android application's executable file or source code without actually running it. The analysis' findings are acquired by examining the code's organisation, the order of API calls, and how various function calls handle sensitive data. For instance, DroidAPIMiner [1] uses supervised learning methods to identify malware and collects the amount of API calls from applications. In contrast, DroidSIFT [2] concentrates on permission-related API calls and employs weighted API dependency graphs to characterise how applications behave. To detect malware with a high degree of accuracy, machine learning techniques are utilised in conjunction with these graphs as features.

A technique called AppContext [3] is used to describe how Android applications behave when security-sensitive methods are called. These techniques include methods that are permission-protected, source-and-sink, reflection, and dynamic code loading. To understand the motivations underlying these security-sensitive behaviours, AppContext abstracts contexts.AppContext creates a set of features that can be used for accurate malware identification by extracting these contexts. These features give useful information about how the programme behaves and aid in spotting potential security concerns or malicious behaviour.

A flexible taint analysis technique called TaintDroid [4] was created to find unapproved data leaks in Android applications. TaintDroid can find information breaches in Android applications while avoiding false positives by using dynamic taint analysis. It monitors the movement of sensitive data between various components of an application, including location data, device IDs, and phone numbers.The study results from TaintDroid show that a sizable fraction of Android applications contain flaws that allow sensitive data to leak. Such sensitive information includes, for instance, the user's location, phone number, and unique device identification (device ID). TaintDroid's capacity to precisely recognise and report these data leaks contributes to enhancing the security and security of applications for Android.

III. Publically Available Datasets

Malicious software created expressly to target mobile phones or PDAs with wireless capabilities is referred to as mobile malware. Mobile malware's main goals are to damage the system's integrity, which can result in system failures, and to obtain unauthorised access to or leak private data. The dataset selected for the proposed system from the kaggle machine learning [5].

Feature vectors taken from 15,036 applications make up the dataset utilised for creating and testing a multilevel classifier fusion technique for Android malware detection. There are 215 properties in these feature vectors. The dataset consists of 9,476 good apps and 5,560 malicious apps that were gathered via the Drebin project.This dataset's objective is to make it easier to create and assess a reliable Android malware detection solution. With the multilevel classifier fusion method, a classifier model is trained and optimised using feature vectors that were retrieved from the applications. The method seeks to improve the accuracy and reliability of malware detection in the Android ecosystem by merging numerous classifiers at various levels [5].

The collection includes feature vectors that were taken from many different Android applications, including both good and malicious apps from the Drebin project. It contributes to the development of mobile security research by serving as a basis for creating and analysing a multilevel classifier fusion strategy for Android malware detection [5].

IV. Proposed system

In intelligent malware detection the proposed method shown in figure 1, it shows the two phases of the proposed malware detection system: the

Training Phase and the Detection Phase. The goal of the training phase is to identify dynamic behaviour traits by observing the behaviour of both good and bad apps. For each processed application, a feature vector is produced.The feature vectors of good and bad applications are used as input during the training phase of the algorithm, which then trains a number of basic classifiers. The feature vectors are analysed by these basis classifiers, which then yield prediction probabilities for each application. The suggested method uses an ensemble classifier method based on these probabilities to categorise the record as Malignant or Benign.

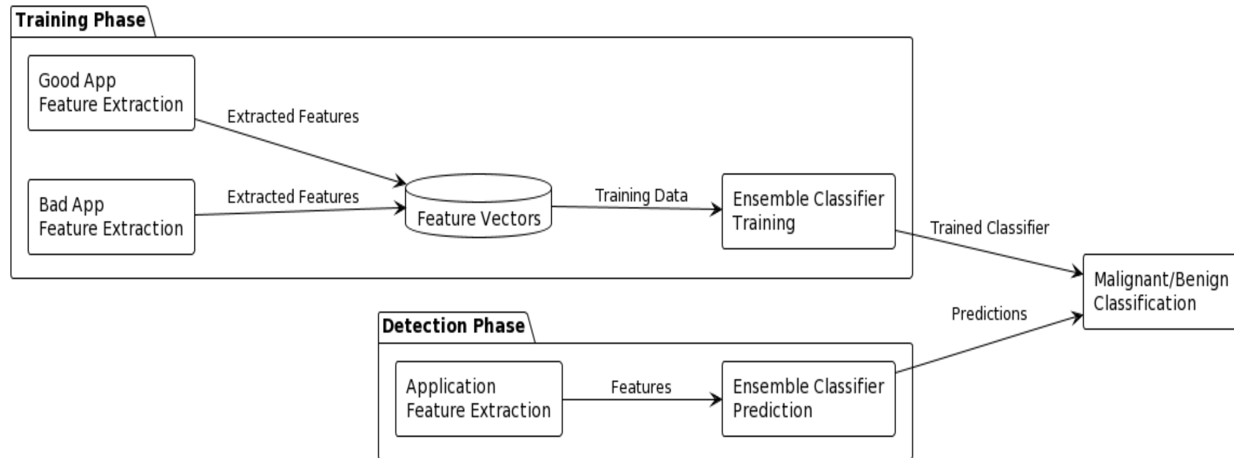

Figure 1: Proposed system architecture

Ensambled function collects runtime behaviour data from a particular application during the initial stage of the training phase. EnDroid makes use of DroidBox, an open-source dynamic analysis tool, to achieve this. TaintDroid, a dynamic taint analysis tool, has more capability thanks to DroidBox, which offers an application sandbox.DroidBox uses dynamic taint analysis methods to track the behaviour of the application and integrates system hooks at the framework layer. It keeps track of a variety of activities carried out by the application, such as processes involving file and network access, data leakage, cryptography, and more. EnDroid can collect thorough runtime behaviour data for the programme under examination using DroidBox.

As in the given dataset there are 215 feature vectored are there, out of it feature selection method select the best suitable feature that participate in the android malware detection mechanism as shown in table 1. The feature collected form the different sources of Android API such as Bluetooth, Camera, Contact etc. and the final version of dataset consist of following feature and more than 36k record for training and testing of ensemble classifier.

Table 1: Features selection for Proposed Method

Sr. No.	Feature Vector	Input format	Details
1	SEND_SMS	SMS <Contact No>	Send SMS to contact number
2	Phone call	Call<Contact No>	Calling on number
3	Start service	Service <start><API name>	Action on service of API
4	System calls	Read<request srNo>	Reading call for system service file
5	Operation on file	action<Id><file>	Read write operation on file

6	RECEIVE_SMS	SMS <Contact No>	Read received SMS
7	READ Contact	Action <Contact No>	Perform action of reading contact
8	KeySpec	action<input><key>	Input action key reading
9	CAMERA	open<api><camera><input>	Access input data through camera
10	BLUETOOTH	action<service connect>	Action service for communication and transfer exchange of information
11	READ_CALL_LOG	SMS READ<Contact No>	Reading history of call logs
12	IRemoteService	source<sink>	Accessing remote services and api
13	Read Email	Action <Ip><PORT><Dest>	Content reading of Email services, cryptographic functions

a) Cryptographic Operation

Cryptographic operations are commonly used in the world of Android malware to encrypt different malicious components and avoid static detection. Malware frequently encrypts URLs that reach malicious servers, malicious payloads, key method identifiers, target premium SMS numbers, and root vulnerabilities. Some malicious apps relocate a sizable chunk of their functionality into an independent dex file and encrypt it to thwart detection in order to improve their evasive strategies.In contrast, good applications frequently make use of cryptographic APIs to guarantee the security of private data, including passwords. By designating characteristics as cryptographic behaviours in Android applications can be differentiated. refers to the various cryptographic algorithms used in the process of encryption, decryption, or key creation in this context.

b) Call Number

One frequent action of Android malware is to start phone calls on its own without any user input. This functionality enables malware to dial certain numbers without authorization and carry out different destructive activities. These characteristics can be described as call<contact number>, where <contact number> denotes the particular phone number that the malware called.It is feasible to spot possible cases of malware that make unauthorised phone calls by looking at such aspects in the behaviour of an Android application. This study aids in identifying and reducing the dangers brought on by malicious applications that take advantage of this behaviour.

c) Start Service

Malicious actions by Android malware are frequently carried out in the background by service components. This enables the malware to carry out its damaging actions covertly without being seen or directly interacted with by the user. These characteristics can be expressed as background_action, where action is the particular destructive activity carried out by the malware.

d) System Call

Applications must have access to system calls in order to request services from the kernel of the operating system. These calls offer a number of features, including access to hardware assets, handling power, device security, and process-related tasks. Programmes rely on system calls to carry out crucial operations and obtain the resources they need to operate correctly.

e) Read Email

When detecting malware, a detection system examines an application's behaviour to determine whether it tries to access emails without proper authority. This entails keeping track of how the

programme interacts with system resources, network communication channels, and APIs that are linked to email.

f) Bluetooth

Malware detection tools can examine the types of endpoints that are linked to, the services being visited, and the data being shared by observing the behaviour of Bluetooth connections. Malware or a possible breach of security may be indicated by any unusual behaviour or unauthorised access.

g) Read Contact

Since it gives them access to potential victims for various criminal actions like phishing, spamming, identity theft, or social engineering, access to contact information can be a significant resource for attackers. Malware detection systems can spot and warn programmes that may be attempting to gain unauthorised access to the user's contacts by analysing the behaviour of the "READ_Contact" feature in an application.

V. Methodology

a) Learning Phase:

It uses a two-class classification technique during the training phase to learn dynamic behaviour traits from both good and bad applications. It uses supervised learning techniques with input feature vectors produced from these applications. By utilising these techniques, EnDroid creates a classification model that can distinguish between feature vectors coming from malware and those from innocuous applications. The detection phase then uses the classification model that was created during the training phase. This model plays a key role in the detection procedure. EnDroid uses the trained model of classification to analyse feature vectors from unidentified applications in the detection phase to determine if they fall under the benign or dangerous categories.

b) Decision Tree

A well-liked machine learning approach for classification and regression tasks is the decision tree algorithm. The stages involved in creating a decision tree are as follows:Choose the best attribute: The best attribute must be chosen in order for it to serve as the Decision Tree's root node. For each attribute, a measure of impurity, such as the Gini Index or the entropy, is calculated, and the attribute with the biggest information gain is chosen.Make a branch for each value of the attribute: The Decision Tree spreads out after identifying the root node by generating child nodes for each potential value of the chosen attribute. Based on these attribute values, the dataset is divided, and the procedure is repeated recursively for each segment.

For example, a decision rule for a node in a Decision Tree can be represented as follows:

```
if (attribute1 <= threshold) {
// Go to left child node
} else {
// Go to right child node
}
```

The Decision Tree algorithm includes phases like attribute selection, attribute-based branching, and classifying leaf nodes. Although the method does not have a precise mathematical equation for building trees, it makes decisions and makes predictions using statistical measures and logical requirements.

c) Support Vector Machine:

A well-liked machine learning approach for regression and classification applications is called Support Vector Machines (SVM). The steps for teaching a support vector machines are as follows:

- Data Pre-processing in step 1
- Choose the Kernel Function in Step 2
- Define the SVM Model in Step 3
- Train the SVM Model in Step 4
- Solve the optimisation problem in Step 5
- Calculate the decision in Step 6 Boundary
- Make predictions in Step 7

Solving a convex optimisation problem is necessary for the mathematical formulation of SVM. The goal is to identify the best hyperplane that maximises the margin between the classes given a training dataset with labelled examples. The mathematical expression is as follows:

Let (x_i, y_i) be the training cases for a binary classification problem, where x_i denotes the feature vector and y_i is the matching class label (+1 or -1). The issue with SVM optimisation is that:

$$\text{minimize: } \left(\frac{1}{2}\right) * ||w||^2 + C * \Sigma\left(\max\left(0, 1 - y_i * (w^T * x_i + b)\right)\right)$$

d) Random Forest:

An ensemble learning technique called the Random Forest algorithm mixes various decision trees to produce predictions. Here are some mathematical calculations and step-by-step explanations of the Random Forest algorithm.

The aggregate of predictions is the primary focus of the mathematical calculations in Random Forest. The projected class labels for classification tasks can be created by tallying the votes and choosing the class with the highest total as the final forecast. The aggregate can be expressed numerically as:

$$\text{Prediction} = \text{argmax}\left(\text{count}(\text{class}_{\text{label}})\right)$$

The projected values from the decision trees can be combined for regression tasks by averaging them.

e) Linear SVM:

Assume that application x is located on the hyperplane's class Ce side and that its distance from the hyperplane is O(x). The likelihood is then,

$$\text{PCe}(x) = 1\,1 + \exp\left(-O(x)\right)$$

$$\text{Pc}(x) = 1 - \text{PCe}(x), \text{for } c\ 6 = \text{Ce}.$$

Let Q be the total number of decision trees used in Extremely Randomised Trees, Random Forest, and Boosted Trees ensemble methods.

Logistic regression is used as the meta-classifier in the ensemble approach outlined to integrate the prediction probabilities produced by the basic classifiers. The choice of logistic regression is based on how easy it is to interpret. Based on the accuracy of its predictions, this meta-classifier distributes weights to the basis classifiers, reflecting the relative value of each base classifier for various prediction classes. The meta-classifier can then determine which classifiers are more dependable and improve the ensemble's performance as a whole.

$$\text{Pc}(x) = \sigma\left(\omega^0 + \omega^1 P^1 c(x) + \omega^2 P^2 c(x) + \ldots + \omega_n P_n c(x)\right)$$

Here, Pc(x) stands for the probability of a prediction for class c of the application x, is the sigmoid function that converts the linear combination of the weighted prediction probabilities to a probability value between 0 and 1, and ω_0, ω_1, ω_2, ..., ω_n stands for the trained parameters of the logistic regression model.

f) Staking:

Using the training dataset, the stacking strategy trains multiple base classifiers. Prediction probabilities are generated for the supplied input data by each basic classifier. The meta-classifier, which is trained to generate the final prediction, receives these prediction probabilities from the base classifiers.Stacking's performance is not necessarily better than that of individual classifiers. The classifiers used as base classifiers and the functions used to combine the output prediction probabilities determine how effective it is. The classifiers selected and the aggregation method used to combine their predictions both have a significant impact on how well the Stacking ensemble performs overall.Algorithm 1 outlines the process of ensemble learning using Stacking, which involves the following steps:

- Split the training dataset into multiple subsets.
- Train each base classifier using a different subset of the training dataset.
- Generate prediction probabilities for the training dataset using each base classifier.
- Combine the prediction probabilities from the base classifiers.
- Train the meta-classifier using the combined prediction probabilities as input features.
- Generate the final prediction using the trained meta-classifier.

Algorithm Ensemble Staking

Step 1: Let $\text{predict}_{\text{proba(x)return}}$ the prediction probabilities on test se
Step 2: Let fit(x)return the objective function on training set x
Step 3: procedure Stacking(Mtrain, Mtest, H, k)
Step 4: # constructing base classifiers, SVM/KNN
/Adaboost
Step 5: # splitting dataset into k subsets
Step 6: {M1 train,· · · , Mk train} ← split(Mtrain, k)
Step 7: for each algorithm Hi in H do
Step 8: for each subset Mj train in Mtrain do
Step 9: M − j train ← Mtrain M j train

Step 10: Y j Hi (x) ← Hi . fit(M − j train)
Step 11: P Hi M j train
← Y j Hi (x). $\text{predict}_{\text{proba(M j train)}}$
Step 12: P Hi j Mtest
← Y j Hi (x). $\text{predict}_{\text{proba(Mtest)}}$
Step 13: end for
Step 14: P Hi Mtrain
← rbind(P Hi M1 train ,· ·
· , P Hi Mk train)
Step 15: P Hi Mtest
← average(P Hi1 Mtest ,· ·
· , P Hik Mtest)
Step 16: end for
Step 17: # constructing new dataset for second stage
Step 18: Trainsetnew
←
∪ (P H1 Mtrain ,· · · , P Hr Mtrain)
Step 19: Testsetnew
←
∪ (P H1 Mtest ,· · · , P Hr Mtest)
Step 20: YLR
← Hlogstic regression. fit(Trainsetnew)
Step 21: Ltest ← YLR. predict(Testsetnew)
Step 22: end procedure

VI. Result and Discussion

The feature selection algorithm's selection of cryptographic operations' recurrence rates for both harmful and legitimate applications. The y-coordinate shows the percentage of applications that conduct a given cryptographic operation out of all the applications that use cryptographic operations, while the x-coordinate shows the crucial cryptographic operations selected by the feature selection algorithm.It is clear from the observations in Figure 2 that malware writers frequently choose less secure encryption techniques like DES over more secure encryption algorithms like AES. The reason for this phenomena is that malware tries to use encryption techniques that are less likely to arouse suspicion and elude detection by security systems. However, to preserve the secrecy of data, benign applications frequently use encryption techniques.

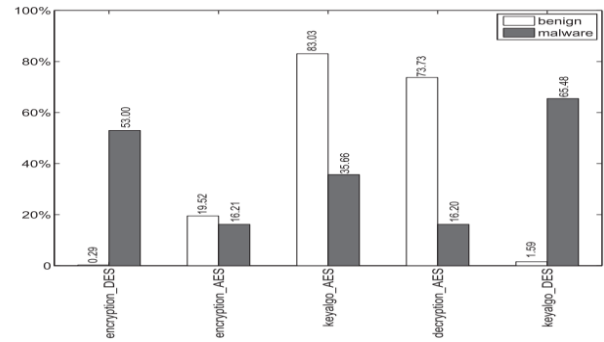

Figure 2: Cryptographic Operation occurrence

The frequency rates of specific information leak procedures between harmful and good applications are shown in Figure 3. We can see that both categories of applications frequently exhibit information leak behaviours. IMSI_SMS, PHONE_NUMBER_Network, IMSI_File, and PHONE_NUMBER_File are examples of information leak behaviours, but they tend to be more prevalent in malicious programmes. The frequency of information leak behaviours in benign applications, such as LOCATION_Network and LOCATION_GPS_Network, is higher. These results show that various information leak behaviours are connected to various application categories.

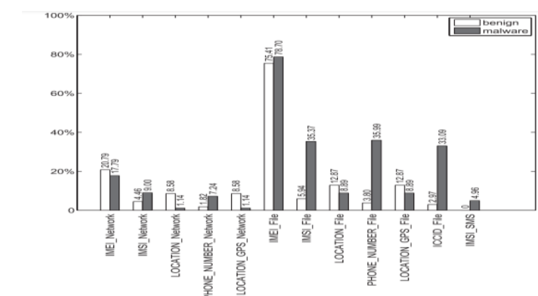

Figure 3: Specific information leak procedures between harmful and good applications

Figure 4 shows the difference in system call frequency between legitimate and malicious programmes. Compared to good apps, malware uses system calls more frequently. Malicious apps tend to use particular system calls more frequently, like fork, fchmod, and wait4. These system calls imply harmful operations like changing the ownership of files containing sensitive data or making child processes to carry out covertly harmful deeds.

Indicating their desire to carry out unauthorised actions and change sensitive data, malicious apps use system calls more frequently and exhibit a higher prevalence of specific information leak behaviours. In contrast, benign programmes focus on safe and secure operations and display different information leak behaviours and use system calls less frequently.

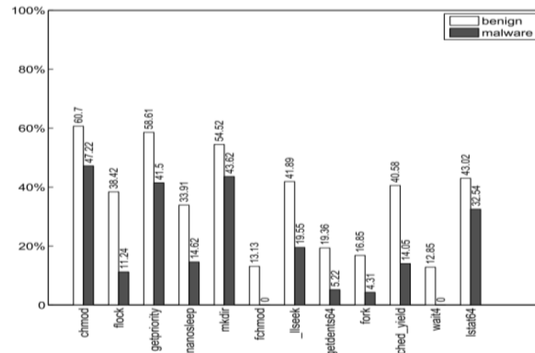

Figure 4: Difference in system call frequency

between legitimate and malicious programmes

The evaluation results of the ensembled approach employing several machine learning algorithms are shown in Table 1. These methods are implemented using Python scripts that make use of the Sklearn module. It is important to emphasise that substantial experimentation, which is outside the purview of this study, is needed to fine-tune the ideal parameters for any machine learning method. With the main goal of demonstrating the efficacy of Stacking, we compare the performance of machine learning algorithms using their default settings in this experiment.

The findings show that, with the exception of Naive Bayes and K-Nearest Neighbours (KNN), the majority of learning algorithms function satisfactorily. Compared to the other algorithms, these two perform less well.

Table 2: Summary of different machine learning algorithms

Method	Features	Accuracy in (%)	Precision Call(%)	Recall in (%)	F1 ScoreVlaue (%)
Decision Tree	Max Depth: 6	81.3	81.5	85.6	81.5
Random Forest	Num of Trees: 120, Max Depth: 20	86.6	87.2	89.9	88.5
Logistic Regression	Regularization: L2, C: 1	88.1	88.8	91.2	85.8
Support Vector Machine	Kernel: RBF, C: 1	87.7	89.1	90.6	87.8
Naive Bayes	-	75.4	79.9	74.8	74.8
Ensemble Method	-	89.3	91.5	90.2	89.8

VII. Conclusion

The ensemble technique, a dynamic analytic framework created for efficient malware detection in Android applications, is introduced in this study. The ensemble method makes use of automated techniques to extract various kinds of dynamic behaviour traits, which are essential for spotting unsafe actions taken by actual applications. The chi-square feature selection algorithm, which helps to reduce irrelevant or noisy features and concentrates on extracting key features, is used into the Ensemble method to improve detection accuracy.The Ensemble approach uses the potent ensemble learning technique of stacking for the detection mechanism. In order to detect Android malware, Stacking improves classification performance by integrating the predictions of numerous basic classifiers. On two separate datasets, the effectiveness of the ensemble approach is assessed while taking into account variances in the feature space, the use of machine learning algorithms, and selecting features algorithms.The testing findings support Stacking's excellent performance in detecting Android malware. The Ensemble technique with Stacking consistently

shows the best classification accuracy when compared to alternative methods. These results demonstrate the Ensemble method's promise as a cutting-edge tool for efficiently identifying and reducing Android malware threats.

References

1. Y. Aafer, W. Du, and H. Yin, ''DroidAPIMiner: Mining API-level features for robust malware detection in Android,'' in Proc. Int. Conf. Secur. Privacy Commun. Syst. Cham, Switzerland: Springer, 2013, pp. 86–103.
2. M. Zhang, Y. Duan, H. Yin, and Z. Zhao, ''Semantics-aware Android malware classification using weighted contextual API dependency graphs,'' in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2014, pp. 1105–1116
3. W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, ''AppContext: Differentiating malicious and benign mobile app behaviors using context,'' in Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng. (ICSE), vol. 1, May 2015, pp. 303–313.
4. W. Enck et al., ''TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones,'' ACM Trans. Comput. Syst., vol. 32, no. 2, p. 5, 2010
5. Yerima, Suleiman (2018): Android malware dataset for machine learning 2. figshare. Dataset. https://doi.org/10.6084/m9.figshare.5854653.v1Data Source https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653
6. J. Garcia, M. Hammad, and S. Malek, ''Lightweight, obfuscation-resilient detection and family identification of Android malware,'' ACM Trans. Softw. Eng. Methodol., vol. 26, no. 3, 2016, Art. no. 11.
7. W.-C. Wu and S.-H. Hung, ''DroidDolphin: A dynamic Android malware detection framework using big data and machine learning,'' in Proc. Conf. Res. Adapt. Convergent Syst., 2014, pp. 247–252.
8. Y. Yang, Z. Wei, Y. Xu, H. He, and W. Wang, ''DroidWard: An effective dynamic analysis method for vetting Android applications,'' Cluster Comput., vol. 19, pp. 1–11, Dec. 2016.
9. F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, ''Deep ground truth analysis of current Android malware,'' in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment. Cham, Switzerland: Springer, 2017, pp. 252–276.
10. A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, ''MADAM: Effective and efficient behavior-based Android malware detection and prevention,'' IEEE Trans. Dependable Secure Comput., vol. 15, no. 1, pp. 83–97, Jan./Feb. 2018
11. D. H. Wolpert, ''Stacked generalization,'' Neural Netw., vol. 5, no. 2, pp. 241–259, 1992.
12. M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, ''Evaluation of Android malware detection based on system calls,'' in Proc. ACM Int. Workshop Secur. Privacy Anal., 2016, pp. 1–8.
13. Y. Aafer, W. Du, and H. Yin, ''DroidAPIMiner: Mining API-level features for robust malware detection in Android,'' in Proc. Int. Conf. Secur. Privacy Commun. Syst. Cham, Switzerland: Springer, 2013, pp. 86–103.
14. M. Zhang, Y. Duan, H. Yin, and Z. Zhao, ''Semantics-aware Android malware classification using weighted contextual API dependency graphs,'' in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2014, pp. 1105–1116.
15. C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, ''Droidminer: Automated mining and characterization of fine-grained malicious behaviors in Android applications,'' in Proc. Eur. Symp. Res. Comput. Secur. Cham, Switzerland: Springer, 2014, pp. 163–182.

16. W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, ''AppContext: Differentiating malicious and benign mobile app behaviors using context,'' in Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng. (ICSE), vol. 1, May 2015, pp. 303–313.
17. W. Enck et al., ''TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones,'' ACM Trans. Comput. Syst., vol. 32, no. 2, p. 5, 2010.
18. L. Chen, M. Zhang, C.-Y. Yang, and R. Sahita. (2017). ''Semisupervised classification for dynamic Android malware detection.'' [Online]. Available: https://arxiv.org/abs/1704.05948?context=cs
19. M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, ''ANDRUBIS–1,000,000 apps later: A view on current Android malware behaviors,'' in Proc. 3rd Int. Workshop Building Anal. Datasets Gathering Exper. Returns Secur. (BADGERS), Sep. 2014, pp. 3–17.
20. M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, ''Mobile-sandbox: Having a deeper look into Android applications,'' in Proc. 28th Annu. ACM Symp. Appl. Comput., 2013, pp. 1808–1815.
21. M. Lindorfer, M. Neugschwandtner, and C. Platzer, ''MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis,'' in Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf. (COMPSAC), vol. 2, Jul. 2015, pp. 422–433
22. Z. Sun, N. Ampornpunt, M. Varma, and S. Vishwanathan, ''Multiple kernel learning and the smo algorithm,'' in Proc. Adv. Neural Inf. Process. Syst., 2010, pp. 2361–2369.
23. G. Meng, Y. Xue, Z. Xu, Y. Liu, J. Zhang, and A. Narayanan, ''Semantic modelling of Android malware for effective malware comprehension, detection, and classification,'' in Proc. 25th Int. Symp. Softw. Test. Anal., 2016, pp. 306–317.
24. K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, ''AndroZoo: Collecting millions of Android apps for the research community,'' in Proc. IEEE/ACM 13th Working Conf. Mining Softw. Repositories (MSR), May 2016, pp. 468–47
25. N. McLaughlin et al., ''Deep Android malware detection,'' in Proc. 7th ACM Conf. Data Appl. Secur. Privacy, 2017, pp. 301–308.
26. H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, ''Structural detection of Android malware using embedded call graphs,'' in Proc. ACM Workshop Artif. Intell. Secur., 2013, pp. 45–54.