



**ISSN:2229-6107**



**INTERNATIONAL JOURNAL OF  
PURE AND APPLIED SCIENCE & TECHNOLOGY**

**E-mail :**  
**editor.ijpast@gmail.com**  
**editor@ijpast.in**

**www.ijpast.in**

# A NOVEL MACHINE LEARNING APPROACH FOR ANDROID MALWARE DETECTION BASED ON THE CO-EXISTENCE OF FEATURES

R. BHAVANI SANKAR, PALLAPU YESURAJU

---

**Abstract:** The project explores Android malware detection using multiple datasets, including Drebin, Malgenome, and CIC\_MALDROID2020. These datasets provide a rich source of API and Permission data, enabling comprehensive analysis. Various machine learning models have been employed for classification, such as Logistic Regression, Support Vector Machine, K-Nearest Neighbors (KNN), Random Forest, Decision Tree, and a Stacking Classifier that combines Random Forest and Multi-Layer Perceptron (MLP) with LightGBM. The project involves thorough dataset preprocessing, model training, and rigorous performance evaluation. This comprehensive approach aims to develop highly effective models for Android malware detection. By achieving successful model outcomes, this project contributes to advancing mobile security, enhancing our ability to detect and mitigate Android malware threats effectively. The project's findings are valuable for mobile security professionals and researchers. In this paper, we also introduced a Stacking Classifier, incorporating the strengths of Random Forest, Multi-Layer Perceptron (MLP), and LightGBM models for enhanced feature extraction. This ensemble learning approach significantly improves prediction accuracy. Additionally, we developed a user-friendly Flask framework integrated with SQLite for secure signup, signin, and user testing. This streamlined system allows input provision and prediction retrieval, making the overall project more robust and practical for efficient user interactions.

---

*Index terms -co-existence, FP-growth, machine learning, malware.*

---

## INTRODUCTION

Smartphone market is growing immensely. According to the ICD report [1], it is estimated that by 2024, the annual sales of mobile phones will reach more than

351 million units globally. Among the several mobiles operating systems, Android is the dominant operating system with over 2.5 billion active users across over 190 countries [2].

---

Assistant Professor<sup>1</sup>, Dept of CSE, Chirala Engineering College, Chirala,  
[bhavanisankar.cse@cecc.co.in](mailto:bhavanisankar.cse@cecc.co.in)

PG Student<sup>2</sup>- MCA, Dept of MCA, Chirala Engineering College, Chirala,  
[pallapuyesuraju144@gmail.com](mailto:pallapuyesuraju144@gmail.com)

---

The wide range of capabilities offered by smartphones and the rising number of activities carried out by their users, including social networking, online banking, and gaming, has given rise to very serious concerns about device security and personal privacy. Since Android is an open-source platform, it is easy for malware developer to launch their attacks and develop Android malware apps that pose severe harm. Obviously, the impact of Android malware is rising in modern society [3], [4].

The Google Play Store, which is the marketplace for Android applications, has more than 3.43 million apps as of January 2021 [5]. Many third-party and non-Google stores are appearing, such as AppBrain and AppChina, which provide applications to be downloaded by users. The applications provided by the third-party markets have a high risk of being malicious apps that are not monitored or detected. Allix et al. [6] reported that 22% of Google Play applications were identified as malicious applications and 50% of AppChina applications were recognized as malicious apps.

Many defense techniques have been used to detect and prevent malicious applications. Signature-based malware detection is an early method that works as a pattern comparison between already exposed apps and new apps. It works well for known malware but it is easily evaded by obfuscation mechanisms or unknown malware. To evaluate the effectiveness of existing signature-based anti-malware scanners, Zhou and Jiang [7] tested four different mobile security applications against more than 1200 Android applications. The results showed that the existing anti-

malware apps cannot detect obfuscated or repackaged malware apps. Similarly, Scott [8] applied an obfuscation technique to ten different malware applications from different families. The results showed that none of them can detect malicious applications after obfuscation.

Machine learning methods are applied to detect Android malware and distinguish them from benign ones without comparing the patterns of the known Android malware. Machine learning methods work smartly by building a model based on sample data, known as “training data”, to make predictions or decisions without being explicitly programmed. The malware detection techniques using machine learning algorithms are classified into two types, which are static analysis and dynamic analysis. In static analysis, an Android application is examined without running it. In contrast, in dynamic analysis, the application is run in a controlled environment to analyze its behavior.

Static Analysis uses static features that are extracted from the manifest (i.e. permissions requested by apps), the source code (i.e. API calls), and intents [19,21]. Many of the extracted features may be disruptive features. For example, many permissions are requested by both benign and malware applications. Therefore, different feature selection methods are applied to select the most relevant features that accurately classifies malware apps. Predefined feature selection algorithms utilize statistical methods to score the correlation or dependency between input variables and output or class variables. Most of these algorithms score each feature alone. In the contrary, this paper relies on the co-existence of features to detect Android malware.

## 1. LITERATURE SURVEY

In today's world most human carry at least one electronic computing device, which has a connection to the internet [4]. Internet starting to have influence in our everyday life. Other than computers and mobile devices, traditionally standalone equipment and devices are too now connected to the internet to make them smart. Critical infrastructure of cities, healthcare and other industries (SCADA) has been connected to internet to make it smarter. Growth of internet helps to make human life easier to live.

The issue of malware detection through large sets of applications, researchers have recently started to investigate the capabilities of machine-learning techniques for proposing effective approaches. So far, several promising results were recorded in the literature, many approaches being assessed with what we call in the lab validation scenarios. [6] This paper revisits the purpose of malware detection to discuss whether such in the lab validation scenarios provide reliable indications on the performance of malware detectors in real-world settings, aka in the wild. To this end, we have devised several Machine Learning classifiers that rely on a set of features built from applications' CFGs. We use a sizeable dataset of over 50 000 Android applications collected from sources where state-of-the art approaches have selected their data. We show that, in the lab, our approach outperforms existing machine learning-based approaches. However, this high performance does not translate in high performance in the wild. The performance gap we observed—F-measures dropping from over 0.9 in the lab to below 0.1 in the wild—

raises one important question: How do state-of-the-art approaches perform in the wild?

The popularity and adoption of smart phones has greatly stimulated the spread of mobile malware, especially on the popular platforms such as Android [10,15,18]. In light of their rapid growth, there is a pressing need to develop effective solutions. However, our defense capability is largely constrained by the limited understanding of these emerging mobile malware and the lack of timely access to related samples. [7] In this paper, we focus on the Android platform and aim to systematize or characterize existing Android malware. Particularly, with more than one year effort, we have managed to collect more than 1,200 malware samples that cover the majority of existing Android malware families [7,13,38], ranging from their debut in August 2010 to recent ones in October 2011. In addition, we systematically characterize them from various aspects, including their installation methods, activation mechanisms as well as the nature of carried malicious payloads. The characterization and a subsequent evolution-based study of representative families reveal that they are evolving rapidly to circumvent the detection from existing mobile anti-virus software. Based on the evaluation with four representative mobile security software, our experiments show that the best case detects 79.6% of them while the worst case detects only 20.2% in our dataset. These results clearly call for the need to better develop next-generation anti-mobile-malware solutions.

As the number of smartphone users increases in terms of billions every year, and users now store personal and sensitive information on their mobile device

which gives a huge platform for a hacker to steal users sensitive information [10]. We have proposed a method to detect android malware using permissions and API. We have generated two types of feature vector named as common and combined feature vector. We obtained 97.25% accuracy for common and 96.56% accuracy for combined features using logistic regression. Further, to reduce training and testing time of classification we have optimized the feature to 131 by removing low variance features with which we have achieved 95.87% accuracy.

Malicious applications pose a threat to the security of the Android platform. The growing amount and diversity of these applications render conventional defenses largely ineffective and thus Android smartphones often remain unprotected from novel malware. [13] In this paper, we propose DREBIN, a lightweight method for detection of Android malware that enables identifying malicious applications directly on the smartphone. As the limited resources impede monitoring applications at run-time, DREBIN performs a broad static analysis, gathering as many features of an application as possible. These features are embedded in a joint vector space, such that typical patterns indicative for malware can be automatically identified and used for explaining the decisions of our method. In an evaluation with 123,453 applications and 5,560 malware samples DREBIN outperforms several related approaches and detects 94% of the malware with few false alarms, where the explanations provided for each detection reveal relevant properties of the detected malware [7,13,38]. On five popular smartphones, the method requires 10 seconds for an

analysis on average, rendering it suitable for checking downloaded applications directly on the device.

## 2. METHODOLOGY

### i) Proposed Work:

The proposed system, a specialized machine learning model for Android malware detection, highlights the significance of co-existing permissions and APIs in distinguishing malware from benign applications. It excels in accuracy, surpassing existing models across datasets, including Drebin, CIC\_MALDROID2020, and Malgenome [9,13,25]. Its primary objective is to enhance Android security and protect user privacy. In this paper introduced a Stacking Classifier, incorporating the strengths of Random Forest, Multi-Layer Perceptron (MLP), and LightGBM models for enhanced feature extraction. This ensemble learning approach significantly improves prediction accuracy. Additionally, we developed a user-friendly Flask framework integrated with SQLite for secure signup, signin, and user testing. This streamlined system allows input provision and prediction retrieval, making the overall project more robust and practical for efficient user interactions.

### ii) System Architecture:

The system architecture initiates with a dataset comprising Android applications, where the features are derived from various co-existence combinations[4,5,7]. This dataset is then divided into two subsets: a training set and a test set. The training set serves as the foundation for constructing machine learning models (knn, svm, rf, dt, lr and extension-

stacking classifier). These models are trained to distinguish between malicious and benign applications. Subsequently, the test set is employed to

assess the performance and effectiveness of these models in classifying new, unseen applications.

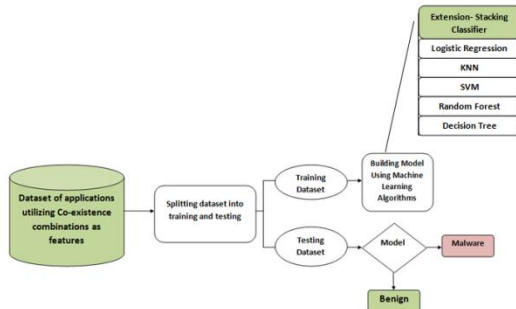


Fig 1 Proposed architecture

**iii) Dataset collection:**

**DREBIN**

- Drebin is a well-known dataset in the field of Android malware research. It contains a large collection of Android applications, including both benign and malicious apps. It is widely used as a benchmark dataset for Android malware detection due to its size and diversity, making it suitable for building robust models [9,13,25].
- So, we have used drebin dataset with these feature combinations.
- We are displaying top 5 rows of the data with each feature combination here. and we can see the no. columns present.

**DREBIN DATASET**

	SEND_SMS	READ_PHONE_STATE	GET_ACCOUNTS	RECEIVE_SMS	READ_SMS	USE_CREDENTIALS	MANAGE_ACCOUNTS
0	1	1	0	0	0	0	0
1	1	1	0	1	1	0	0
2	1	1	0	0	0	0	0
3	0	1	0	0	1	0	0
4	0	1	0	0	0	0	0

5 rows • 102 columns

	transact	onServiceConnected	bindService	attachInterface	ServiceConnection	android.os.Binder	Ljava.lang.Class.getCanonicalName
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0

5 rows • 73 columns

	SEND_SMS	READ_PHONE_STATE	GET_ACCOUNTS	RECEIVE_SMS	READ_SMS	USE_CREDENTIALS	MANAGE_ACCOUNTS
0	1	1	0	0	0	0	0
1	1	1	0	1	1	0	0
2	1	1	0	0	0	0	0
3	0	1	0	0	1	0	0
4	0	1	0	0	0	0	0

5 rows • 110 columns

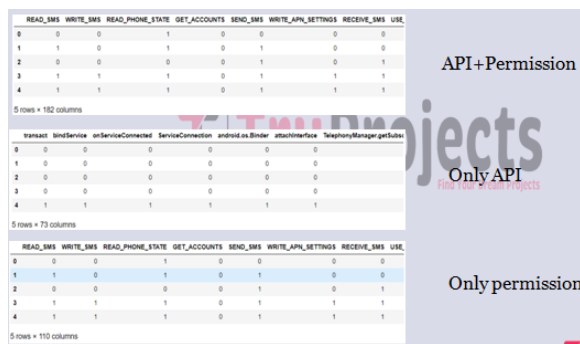
projects  
 Only API  
 Only permission

Fig 2 Drebin dataset



## MALGENOME

- Malgenome contains Android applications, specifically focusing on malware samples. It includes a variety of Android malware instances. It complements Drebin by offering a specialized set of Android malware samples for research and model development.
- We utilized the Malgenome dataset in conjunction with different feature combinations [917,36]. And.. we are presenting the top 5 rows of data for each feature combination, allowing us to observe the number of columns in each case.



The figure displays three data snippets from the Malgenome dataset, each showing the top 5 rows of data for a specific feature combination. The columns represent various Android permissions and services.

Feature Combination	Columns
API+Permission	152
Only API	73
Only permission	110

Fig 3 Malgenome

## CIC\_MALDROID2020

- CIC\_MALDROID2020 is provided by the Canadian Institute for Cybersecurity (CIC) and is known for its size, recency, diversity, and comprehensiveness.
- Similarly, we have used CIC\_MALDROID2020 dataset with these feature combinations. We are displaying top 5 rows of the data with each feature combination here. and we can see the no. columns present.



The figure displays three data snippets from the CIC\_MALDROID2020 dataset, each showing the top 5 rows of data for a specific feature combination. The columns represent various Android permissions and services.

Feature Combination	Columns
API+Permission	894
Only API	70
Only permission	1634

Fig 4 CIC\_MALDROID2020

#### iv) Data Processing:

Data processing involves transforming raw data into valuable information for businesses. Generally, data scientists process data, which includes collecting, organizing, cleaning, verifying, analyzing, and converting it into readable formats such as graphs or documents. Data processing can be done using three methods i.e., manual, mechanical, and electronic. The aim is to increase the value of information and facilitate decision-making. This enables businesses to improve their operations and make timely strategic decisions. Automated data processing solutions, such as computer software programming, play a significant role in this. It can help turn large amounts of data, including big data, into meaningful insights for quality management and decision-making.

#### v) Feature selection:

Feature selection is the process of isolating the most consistent, non-redundant, and relevant features to use in model construction. Methodically reducing the size of datasets is important as the size and variety of datasets continue to grow. The main goal of feature selection is to improve the performance of a predictive model and reduce the computational cost of modeling.

Feature selection, one of the main components of feature engineering, is the process of selecting the most important features to input in machine learning algorithms. Feature selection techniques are employed to reduce the number of input variables by eliminating redundant or irrelevant features and narrowing down the set of features to those most relevant to the machine learning model. The main benefits of

performing feature selection in advance, rather than letting the machine learning model figure out which features are most important.

#### vi) Algorithms:

**Logistic Regression** is a classification algorithm that predicts the probability of an input belonging to a specific category. It employs the sigmoid function to map the input features to a probability score between 0 and 1, and a threshold is applied to classify the input into one of two or more categories based on this probability. The model learns coefficients during training to best fit the data and make accurate classifications.

```
from sklearn.linear_model import LogisticRegression
#from sklearn.pipeline import Pipeline

# instantiate the model
log = LogisticRegression()

# fit the model
log.fit(X_train,y_train)
y_pred = log.predict(X_test)
```

Fig 5 Logistic regression

A **Support Vector Classifier (SVC)** is a machine learning model that finds the best possible boundary (hyperplane) to separate different classes of data while maximizing the margin between them. It identifies key support vectors to make accurate classifications, making it effective for both binary and multi-class classification tasks.



```
# Support Vector Classifier model
from sklearn.svm import SVC
svc = SVC()

# fit the model
svc.fit(X_train,y_train)
y_pred = svc.predict(X_test)
```

Fig 6 SVM

**K-Nearest Neighbors (KNN)** is a versatile machine learning algorithm used for classification and regression tasks. It finds the K nearest data points to a given input and makes predictions based on majority voting or averaging. KNN is non-parametric and simple to implement but can be sensitive to the choice of K and may not perform well in high-dimensional data without preprocessing [30].

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)

# fit the model
neigh.fit(X_train,y_train)
y_pred = neigh.predict(X_test)
```

Fig 7 KNN

**Random Forest** is an ensemble learning method that combines multiple decision trees to make predictions. It works by training a collection of decision trees on random subsets of the data and then averaging their predictions. This ensemble approach enhances accuracy, reduces overfitting, and provides robust

performance for both classification and regression tasks.

```
# Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(n_estimators=10)

# fit the model
forest.fit(X_train,y_train)
y_pred = forest.predict(X_test)
```

Fig 8 Random forest

A **Decision Tree** is a machine learning model that makes decisions by recursively splitting data into subsets based on the most significant feature, aiming to classify or predict outcomes. It creates a tree-like structure where each node represents a feature and each branch represents a possible decision, making it interpretable and effective for various tasks.

```
# Decision Tree Classifier model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth=30)

# fit the model
tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)
```

Fig 9 Decision tree

A **Stacking Classifier** is an ensemble learning method that combines the predictions of multiple base models, such as Random Forest (RF), Multi-Layer Perceptron (MLP), and LightGBM, to create a more accurate final prediction. It leverages the strengths of these

individual models to improve overall predictive performance. In this process, the base models are first trained on the training data, and their predictions are then used as input features for a meta-learner, which learns how to combine these predictions to make the final prediction. Stacking is a powerful technique used to enhance predictive accuracy and is commonly employed in machine learning for various applications.

```
estimators = [('rf', RandomForestClassifier(n_estimators=10)), ('mlp', MLPClassifier(random_state=1, max_iter=300))]
clf1 = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
# fit the model
clf1.fit(X_train, y_train)
# fit the model
y_pred = clf1.predict(X_test)
```

Fig 10 Stacking classifier

### 3. EXPERIMENTAL RESULTS

**Precision:** Precision evaluates the fraction of correctly classified instances or samples among the ones classified as positives. Thus, the formula to calculate the precision is given by:

$$\text{Precision} = \frac{\text{True positives}}{(\text{True positives} + \text{False positives})} = \frac{TP}{TP + FP}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

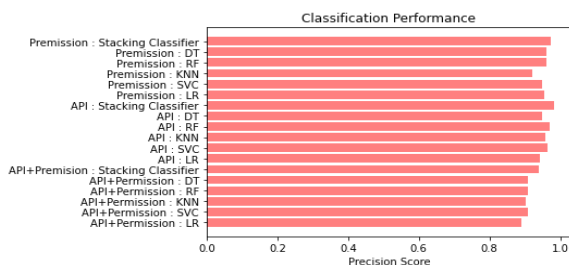


Fig 11 Precision comparison graph

**Recall:** Recall is a metric in machine learning that measures the ability of a model to identify all relevant instances of a particular class. It is the ratio of correctly predicted positive observations to the total actual positives, providing insights into a model's completeness in capturing instances of a given class.

$$\text{Recall} = \frac{TP}{TP + FN}$$

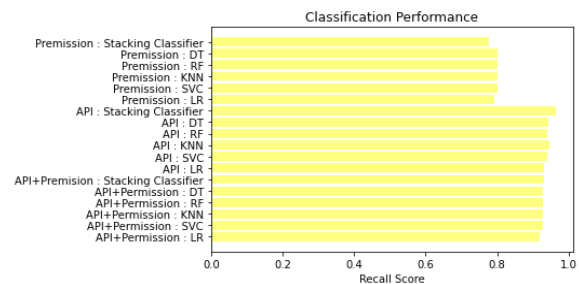


Fig 12 Recall comparison graph

**Accuracy:** Accuracy is the proportion of correct predictions in a classification task, measuring the overall correctness of a model's predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

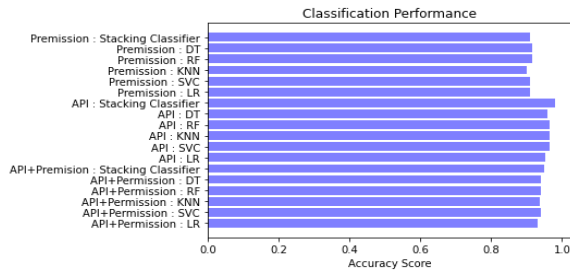


Fig 13 Accuracy graph

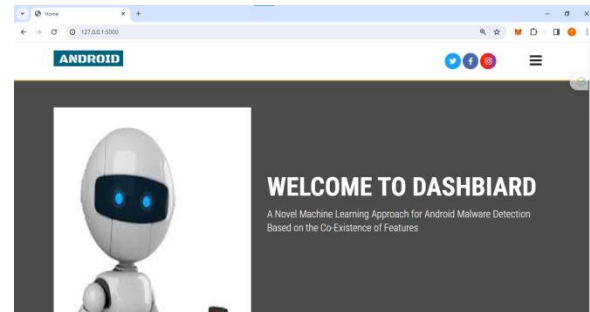


Fig 16 Home page

**F1 Score:** The F1 Score is the harmonic mean of precision and recall, offering a balanced measure that considers both false positives and false negatives, making it suitable for imbalanced datasets.

$$F1 \text{ Score} = 2 * \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} * 100$$

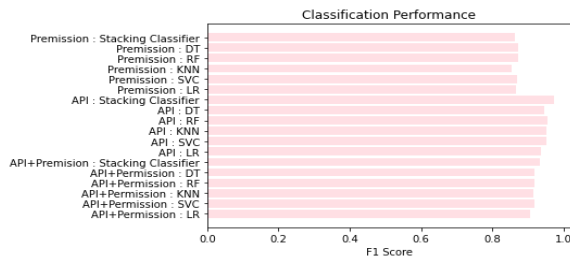


Fig 14 F1Score

	ML Model	Accuracy	Precision	f1_score	Recall
0	API+Permission : LR	0.99	0.98	0.98	0.99
1	API+Permission : SVC	0.99	0.98	0.99	1.00
2	API+Permission : KNN	0.97	0.94	0.97	0.99
3	API+Permission : RF	0.97	0.97	0.97	0.97
4	API+Permission : DT	0.95	0.97	0.95	0.94
5	API+Permission : Stacking Classifier	1.00	1.00	1.00	1.00
6	API : LR	0.95	0.94	0.95	0.97
7	API : SVC	0.96	0.94	0.96	0.97
8	API : KNN	0.94	0.93	0.94	0.95
9	API : RF	0.94	0.94	0.94	0.94
10	API : DT	0.95	0.95	0.94	0.94
11	API : Stacking Classifier	0.97	0.98	0.97	0.96
12	Permission : LR	0.95	0.95	0.95	0.97
13	Permission : SVC	0.95	0.94	0.95	0.97
14	Permission : KNN	0.94	0.90	0.94	0.99
15	Permission : RF	0.95	0.95	0.95	0.97
16	Permission : DT	0.95	0.94	0.95	0.95
17	Permission : Stacking Classifier	0.99	0.98	0.99	1.00

Fig 15 Performance Evaluation

 **Sign In**

**SIGN UP**

Already have an account? [Sign in](#)

Fig 17 Signin page

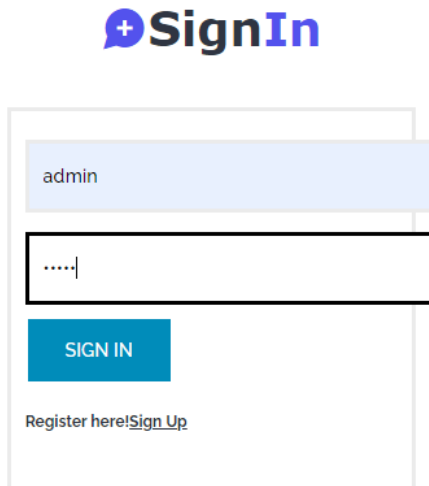


Fig 18 Login page

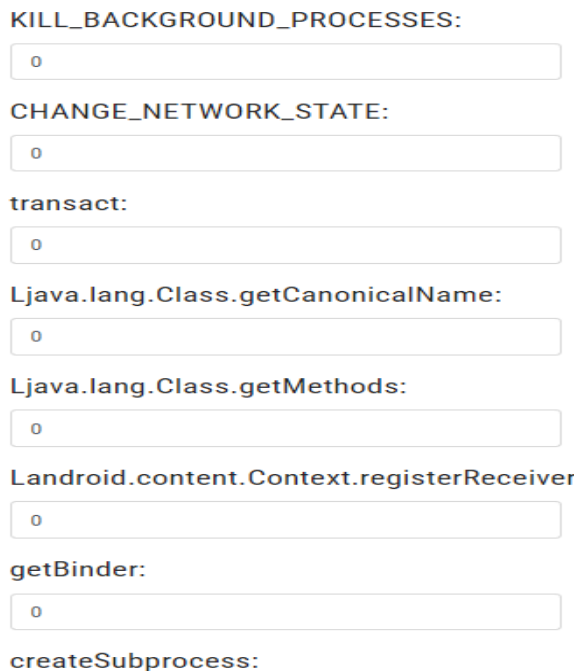


Fig 19 User input



Result: **Malware Android Attack is Detected!**

Fig 20 Predict result for given input

#### 4. CONCLUSION

The project demonstrated the effectiveness of machine learning models in detecting Android malware. These models showed strong capabilities in identifying malicious apps, which is crucial for protecting Android users. The stacking classifier, proved to be more effective than individual algorithms. This approach of combining multiple models enhanced the overall detection accuracy, showcasing the power of ensemble learning. Flask and SQLite create a user-friendly interface, enabling broader accessibility. The design supports user testing, input validation, and seamless model predictions, enhancing practical usability and adoption. The project highlighted the significance of combining both API and Permission features. This combination was found to be critical for improving malware detection, emphasizing the importance of considering multiple static features in analysis. The performance of machine learning models varied across different datasets, such as Drebin, Malgenome, and CIC\_MALDROID2020 [36]. This underscores the importance of careful dataset selection and understanding for developing accurate detection models. The models achieved a balance between accuracy and minimizing false positives. This is

essential as it ensures that while detecting malware, legitimate apps are not mistakenly flagged as threats, reducing inconvenience for users. The outcomes of this project have broad implications. Security professionals can use these improved detection techniques to enhance cybersecurity. App developers can better safeguard their apps against potential threats, and end users benefit from increased protection against Android malware, ultimately leading to a safer mobile experience.

## 5. FUTURE SCOPE

Further research could focus on improving the real-time detection capabilities of the proposed system by continuously monitoring and analyzing dynamic features. This would enable the system to respond more effectively to evolving Android malware threats. Exploring techniques to select the most relevant dynamic features for malware detection can lead to more efficient and accurate models. Feature selection methods, such as mutual information or recursive feature elimination, could be investigated. Extending the system to detect behavioral anomalies in Android applications can provide an additional layer of security. This involves identifying deviations from expected behavior, which could be indicative of malware. [7] As new types of Android malware emerge, the system could be designed to adapt and update its models and detection strategies. Regularly incorporating new threat intelligence and updating the system is essential for long-term effectiveness. Expanding the capabilities of the system to encompass cross-platform malware detection, including iOS and other mobile operating systems, can provide a more holistic solution for mobile security.

## REFERENCES

- [1] H. Menear. (2021). IDC Predicts Used Smartphone Market Will Grow 11.2% by 2024. Accessed: Oct. 30, 2022. [Online]. Available: <https://mobile-magazine.com/mobile-operators/idc-predicts-usedsmartphone-market-will-grow-112-2024?page=1>
- [2] D. Curry. (2022). Android Statistics. Accessed: Oct. 30, 2022. [Online]. Available: <https://www.businessofapps.com/data/android-statistics/>
- [3] O. Abendan. (2011). Fake Apps Affect Android Os Users. Accessed: Oct. 30, 2022. [Online]. Available: <https://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/72/fake-apps-affect-android-osusers>
- [4] C. D. Vijayanand and K. S. Arunlal, "Impact of malware in modern society," *J. Sci. Res. Develop.*, vol. 2, pp. 593–600, Jun. 2019.
- [5] M. Iqbal. (2022). App Download Data. Accessed: Oct. 30, 2022. [Online]. Available: <https://www.businessofapps.com/data/app-statistics/>
- [6] K. Allix, T. Bissyand, Q. Jarome, J. Klein, R. State, and Y. L. Traon, "Empirical assessment of machine learning-based malware detectors for android," *Empirical Softw. Eng.*, vol. 21, pp. 183–211, Jun. 2016.
- [7] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 95–109.

- [8] J. Scott. (2017). Signature Based Malware Detection is Dead. Accessed: Oct. 30, 2022. [Online]. Available: <https://icitech.org/wpcontent/uploads/2017/02/ICIT-Analysis-Signature-Based-MalwareDetection-is-Dead.pdf>
- [9] Q. M. Y. E. Odat. Accessed: Dec. 27, 2022. [Online]. Available: <https://github.com/esraacell28/a-novel-machine-learning-approach-forandroid-malware-detection-based-on-the-co-existence>
- [10] S. R. Tiwari and R. U. Shukla, “An Android malware detection technique based on optimized permissions and API,” in Proc. Int. Conf. Inventive Res. Comput. Appl. (ICIRCA), Jul. 2018, pp. 258–263.
- [11] (2018). Dex2jar—Tools To Work With Android.dex & Java.Class Files. Accessed: Oct. 30, 2022. [Online]. Available: <https://kalilinuxtutorials.com/dex2jar-android-java/>
- [12] Androzo. Accessed: Jul. 30, 2022. [Online]. Available: <https://androzo.uni.lu/>
- [13] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, “Drebin: Effective and explainable detection of Android malware in your pocket,” in Proc. NDSS, Feb. 2014, pp. 23–26.
- [14] Virusshare. accessed: Jul. 30, 2022. [Online]. Available: <https://virusshare.com/>
- [15] H. Cheng, X. Yan, J. Han, and C.-W. Hsu, “Discriminative frequent pattern analysis for effective classification,” in Proc. IEEE 23rd Int. Conf. Data Eng., Apr. 2007, pp. 716–725.
- [16] M. Parkour. Contagio Mini-Dump. accessed: Jul. 30, 2022. [Online]. Available: <http://contagiominidump.blogspot.it/>
- [17] Malgenome Project. accessed: Jul. 30, 2022. [Online]. Available: <http://www.Malgenomeproject.org>
- [18] C.-F. Tsai, Y.-C. Lin, and C.-P. Chen, “A new fast algorithms for mining association rules in large databases,” in Proc. IEEE Int. Conf. Syst., Man Cybern. San Francisco, CA, USA: Morgan Kaufmann, Oct. 1994, pp. 487–499.
- [19] A. Lab. (2017). Amd Dataset. Accessed: Oct. 30, 2022. [Online]. Available: <https://www.kaggle.com/datasets/blackarcher/malware-dataset>
- [20] V. Avdiienko, “Mining apps for abnormal usage of sensitive data,” in Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng., vol. 1, May 2015, pp. 426–436.
- [21] Y. Aafer, W. Du, and H. Yin, “Droidapiminer: Mining api-level features for robust malware detection in android,” in Security and Privacy in Communication Networks, T. Zia, A. Zomaya, V. Varadharajan, and M. Mao, Eds. Cham, Switzerland: Springer, 2013, pp. 86–103.
- [22] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. De Geus, “Identifying Android malware using dynamically obtained



features,” J. Comput. Virology Hacking Techn., vol. 11, no. 1, pp. 9–17, 2015.

[23] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, “DroidScribe: Classifying Android malware based on runtime behavior,” in Proc. IEEE Secur. Privacy Workshops (SPW), May 2016, pp. 252–261.

[24] H. Cai, N. Meng, B. G. Ryder, and D. Yao, “DroidCat: Effective Android malware detection and categorization via app-level profiling,” IEEE Trans. Inf. Forensics Security, vol. 14, no. 6, pp. 1455–1470, Jun. 2019.

[25] A. F. A. kadir, N. Stakhanova, and A. Ghorbani, “An empirical analysis of Android banking malware,” Tech. Rep., Nov. 2016.

[26] M. Sun, M. Zheng, J. C. S. Lui, and X. Jiang, “Design and implementation of an Android host-based intrusion prevention system,” in Proc. 30th Annu. Comput. Secur. Appl. Conf. New York, NY, USA: Association for Computing Machinery, 2014, pp. 226–235, doi: 10.1145/2664243.2664245.

[27] Google Play Store. Accessed: Jul. 30, 2022. [Online]. Available: <https://play.google.com/store/games>

[28] D. Son. (2019). Apktool—Tool For Reverse Engineering Android Apk Files. Accessed: Oct. 30, 2022. [Online]. Available: <https://securityonline.info/apktool-reverse-engineering-android-apkfiles/>

[29] McAfee. Accessed: Jul. 30, 2022. [Online]. Available: <https://www.mcafee.com/>

[30] G. Baldini and D. Geneiatakis, “A performance evaluation on distance measures in KNN for mobile malware detection,” in Proc. 6th Int. Conf. Control, Decis. Inf. Technol. (CoDIT), Apr. 2019, pp. 193–198.

[31] Y. Zhang, Y. Yang, and X. Wang, “A novel Android malware detection approach based on convolutional neural network,” in Proc. 2nd Int. Conf. Cryptogr., Secur. Privacy, Mar. 2018, pp. 144–149.

[32] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, “Static malware detection and attribution in Android byte-code through an end-to-end deep system,” Future Gener. Comput. Syst., vol. 102, pp. 112–126, Jan. 2020, doi: 10.1016/j.future.2019.07.070.

[33] M. Lindorfer, M. Neugschwandtner, and C. Platzer, “MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis,” in Proc. COMPSAC, vol. 2, Jul. 2015, pp. 422–433.

[34] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, “MADAM: Effective and efficient behavior-based Android malware detection and prevention,” IEEE Trans. Depend. Sec. Comput., vol. 15, no. 1, pp. 83–97, Jan./Feb. 2018.

[35] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, “StormDroid: A streaming machine learning-based system for detecting Android malware,” in Proc. 11th ACM Asia Conf. Comput. Commun. Secur. York, NY, USA: Association for Computing

Machinery, May 2016, pp. 377–388, doi:  
10.1145/2897845.2897860.

[36] (2020). CIC\_MALDROID2020 Dataset,  
Canadian Institute for Cybersecurity. Accessed: Oct.  
30, 2022. [Online]. Available:  
<https://www.unb.ca/cic/datasets/maldroid-2020.html>

[37] S. Y. Yerima and S. Sezer, “DroidFusion: A  
novel multilevel classifier fusion approach for  
Android malware detection,” IEEE Trans. Cybern.,  
vol. 49, no. 2, pp. 453–466, Feb. 2019.

[38] H. L. Thanh, “Analysis of malware families on  
Android mobiles: Detection characteristics  
recognizable by ordinary phone users and how to fix  
it,” J. Inf. Secur., vol. 4, no. 4, pp. 213–224, 2013.

[39] E. B. Karbab, M. Debbabi, A. Derhab, and D.  
Mouheb, “MalDozer: Automatic framework for  
Android malware detection using deep learning,”  
Digit. Invest., vol. 24, pp. 48–59, Mar. 2018.

[40] Z. Aung and W. Zaw, “Permission-based  
Android malware detection,” Int. J. Sci. Technol.  
Res., vol. 2, no. 3, pp. 228–234, 2013.