Leveraging Co-Existence Features to Improve Android Malware Detection

[1]Dongari Sai, [2]Dr.K.Santhi Sree
[1]*MCA Student, Department of Information Technology, Jawaharlal Nehru Technological University, India*
[2] *Professor of CSE, Department of Information Technology, Jawaharlal Nehru Technological University, India*

Abstract: This project examines Android malware detection using datasets such as Drebin, Malgenome, and CIC_MALDROID2020, which offer extensive API and Permission data for in-depth analysis. We employed several machine learning models for classification, including Logistic Regression, SVM, KNN, Random Forest, Decision Tree, and a Stacking Classifier that blends Random Forest, MLP, and LightGBM. The comprehensive methodology involves data preprocessing, model training, and performance evaluation to create highly effective detection models. These models enhance mobile security by improving malware threat detection and mitigation. Our findings are particularly valuable for professionals and researchers in mobile security. Additionally, we developed a user-friendly Flask framework with SQLite, enabling secure signup, sign-in, and user testing, thereby making the project more practical and robust for efficient user interactions.

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

1. INTRODUCTION

The smartphone market is rapidly growing, with global sales projected to exceed 351 million units by 2024. Android, with over 2.5 billion users, faces significant security challenges due to its open-source nature. By early 2021, the Google Play Store had over 3.43 million apps, and third-party markets like AppBrain and AppChina also host apps, often with higher risks of malware. Research shows 22% of Google Play apps and 50% of AppChina apps were malicious. Traditional signature-based methods struggle with obfuscated malware. Machine learning, utilizing static and dynamic analysis with advanced feature selection, provides a more effective solution for detecting malware.

2.LITERATURE SURVEY

Nearly everyone now uses internet-connected devices, which enhance convenience but also pose security risks. Machine learning has shown promise for malware detection in labs but struggles in real-world scenarios, with performance dropping significantly.
The surge in smartphone use, particularly Android, has increased mobile malware. Traditional defenses are challenged by rapidly evolving threats. Analysis of over 1,200 malware samples revealed their evolution to bypass detection, with existing security solutions detecting only 20.2% to 79.6% of threats.
Our method, which uses permissions and API calls, achieved 97.25% accuracy and 95.87% after feature optimization. DREBIN, a lightweight on-device tool, detected 94% of malware with minimal false positives and analyzed apps in about 10 seconds, proving effective and practical for real-world use.

3.METHODOLOGY

i) Proposed Work:
The proposed machine learning system detects Android malware by integrating permissions and APIs, showing superior accuracy on datasets like Drebin, CIC_MALDROID2020, and Malgenome. It includes a Stacking Classifier combining Random Forest, MLP, and LightGBM for improved detection. A Flask framework with SQLite supports secure user management, enhancing usability and robustness.

ii) System Architecture:
The system starts with a dataset of Android applications, with features extracted from different co-existence combinations. This dataset is divided into a training set and a test set. Machine learning models

(KNN, SVM, RF, DT, LR, and a Stacking Classifier) are trained on the training set to classify apps as malicious or benign, and the test set is used to assess the models' performance on new applications.

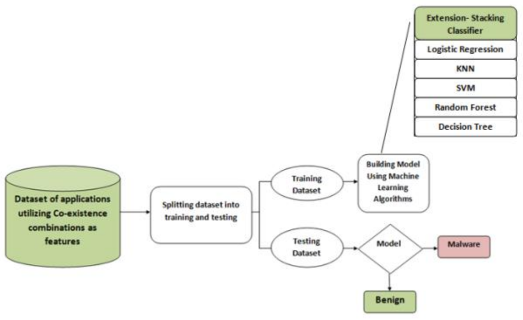
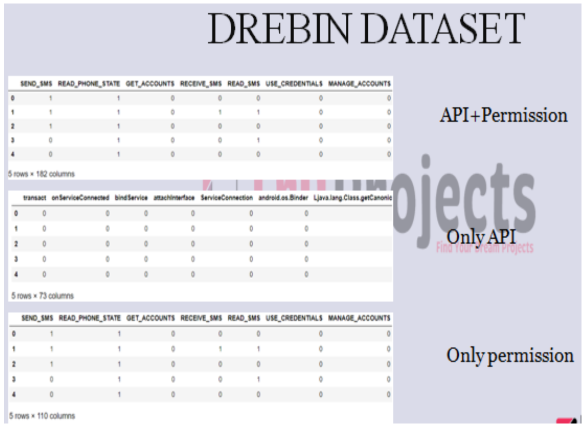

Extension- Stacking Classifier
Logistic Regression
KNN
SVM
Random Forest
Decision Tree
Dataset of applications utilizing Co-existence combinations as features
Splitting dataset into training and testing
Training Dataset
Building Model Using Machine Learning Algorithms
Testing Dataset
Model
Malware
Benign

Fig 1 Proposed architecture

iii) Dataset collection:

DREBIN

- • Drebin is a key dataset in Android malware research, featuring a wide array of benign and malicious apps. Its extensive and varied nature makes it a valuable benchmark for robust malware detection models. • We used the Drebin dataset with multiple feature combinations. • The top 5 rows of data for each feature combination are shown below, including the number of 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 × 182 columns

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

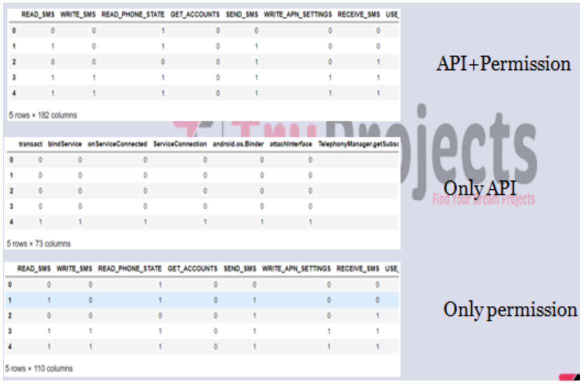
5 rows × 110 columns

Fig 2 Drebin dataset

MALGENOME

Malgenome specializes in Android malware, offering a targeted set of malware samples that complements the Drebin dataset. It aids in research and model development with its diverse malware instances.

We utilized the Malgenome dataset with various feature combinations and show the top 5 rows of data for each, along with the number of columns



	READ_SMS	WRITE_SMS	READ_PHONE_STATE	GET_ACCOUNTS	SEND_SMS	WRITE_APN_SETTINGS	RECEIVE_SMS	USE_
0	0	0	1	0	0	0	0	
1	1	0	1	0	1	0	0	
2	0	0	0	0	1	0	1	
3	1	1	1	0	1	1	1	
4	1	1	1	0	1	1	1	

5 rows × 182 columns

	transact	bindService	onServiceConnected	ServiceConnection	android.os.Binder	attachInterface	TelephonyManager.getSubsc
0	0	0	0	0	0	0	
1	0	0	0	0	0	0	
2	0	0	0	0	0	0	
3	0	0	0	0	0	0	
4	1	1	1	1	1	1	

5 rows × 73 columns

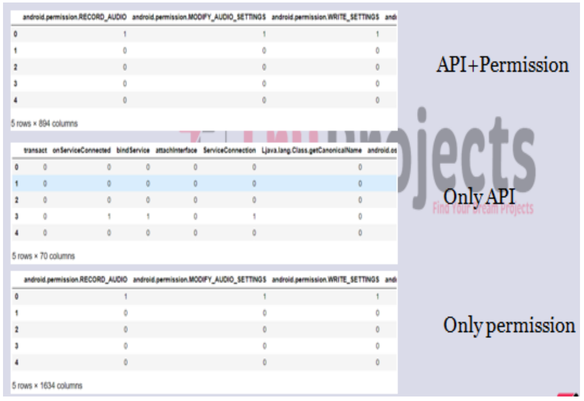
	READ_SMS	WRITE_SMS	READ_PHONE_STATE	GET_ACCOUNTS	SEND_SMS	WRITE_APN_SETTINGS	RECEIVE_SMS	USE_
0	0	0	1	0	0	0	0	
1	1	0	1	0	1	0	0	
2	0	0	0	0	1	0	1	
3	1	1	1	0	1	1	1	
4	1	1	1	0	1	1	1	

5 rows × 110 columns

Fig 3 Malgenome

CIC_MALDROID2020

- CIC_MALDROID2020, from the Canadian Institute for Cybersecurity, is notable for its extensive size, recent updates, and comprehensive diversity.
- We used the CIC_MALDROID2020 dataset with different feature combinations.
- The top 5 rows of data for each feature combination are shown below, along with the number of columns..



	android.permission.RECORD_AUDIO	android.permission.MODIFY_AUDIO_SETTINGS	android.permission.WRITE_SETTINGS	and
0	1	1	1	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

5 rows × 894 columns

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

5 rows × 70 columns

	android.permission.RECORD_AUDIO	android.permission.MODIFY_AUDIO_SETTINGS	android.permission.WRITE_SETTINGS	and
0	1	1	1	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

5 rows × 1634 columns

Fig 4 CIC_MALDROID2020

iv) Data Processing:

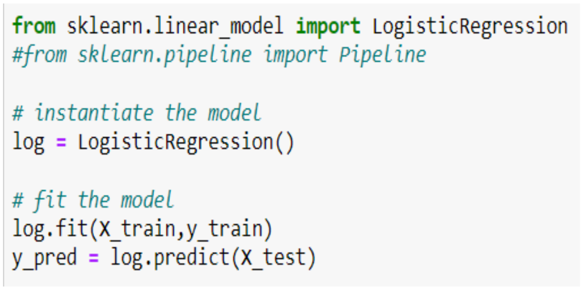
Data processing converts raw data into valuable information through collection, organization, cleaning, analysis, and conversion into formats like graphs. Methods include manual, mechanical, and electronic approaches. Automated solutions, such as software, are essential for turning large data volumes into actionable insights to improve decision-making and operations

v) Feature selection:

Feature selection identifies the most relevant and non-redundant features for model building. With growing dataset sizes, reducing dimensions is crucial to improve model performance and lower computational costs. This process, a key part of feature engineering, involves eliminating unnecessary features to enhance the efficiency and accuracy of machine learning models.

vi) Algorithms:

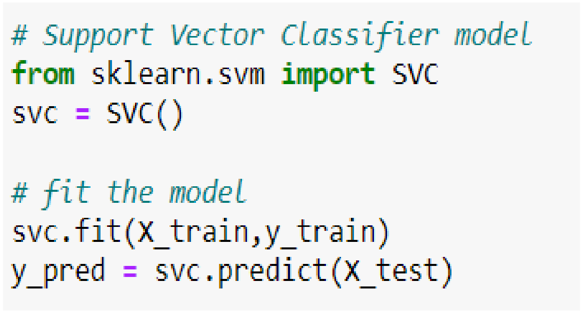
Logistic Regression is a classification algorithm that uses the sigmoid function to convert input features into a probability score between 0 and 1. A threshold is then applied to classify inputs into categories, with the model adjusting coefficients during training to optimize accuracy.



```
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

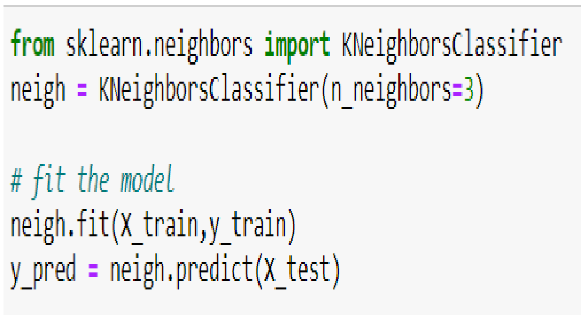
A Support Vector Classifier (SVC) finds the optimal hyperplane to separate data classes while maximizing the margin between them. It uses support vectors to ensure accurate classifications for both binary and multi-class problems.



```
# 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

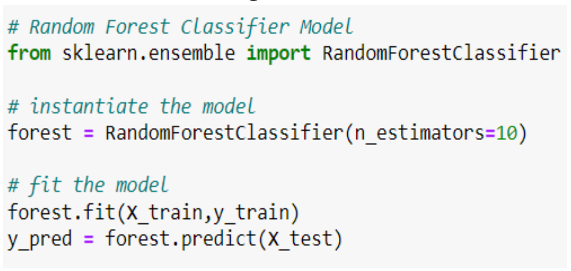
K-Nearest Neighbors (KNN) is a simple, flexible algorithm for classification and regression. It predicts outcomes based on the majority vote or average of the K nearest data points, but can be sensitive to the choice of K and high-dimensional data.



```
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

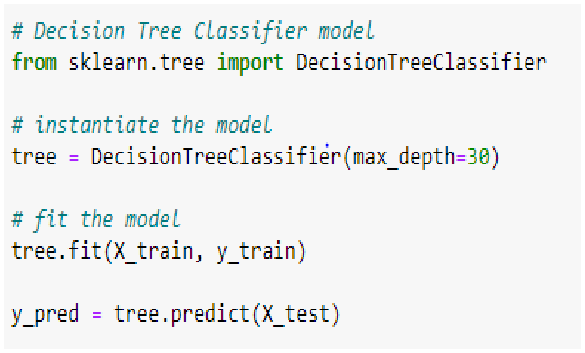
Random Forest is an ensemble method that improves predictions by combining multiple decision trees. It trains trees on random data subsets and averages their predictions, enhancing accuracy, reducing overfitting, and offering robust performance for both classification and regression tasks.



```
# 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

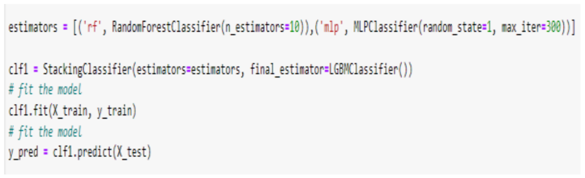
A Decision Tree is a machine learning model that classifies or predicts outcomes by splitting data based on key features, creating an interpretable tree-like structure.



```
# 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 enhances prediction accuracy by combining multiple base models, like Random Forest, MLP, and LightGBM. Predictions from these models are used as inputs for a meta-learner, which integrates them to produce a final, more precise prediction.



```
estimators = [('rf', RandomForestClassifier(n_estimators=10)),('mlp', MLPClassifier(random_state=1, max_iter=300))]

clf1 = StackingClassifier(estimators=estimators, final_estimator=LGBMClassifier())
# fit the model
clf1.fit(X_train, y_train)
# fit the model
y_pred = clf1.predict(X_test)
```

Fig 10 Stacking classifier

4.EXPERIMENTAL RESULTS

Precision: Precision measures the proportion of true positive predictions among all positive predictions made. It is calculated using the formula:

Precision = True positives/ (True positives + False positives) = TP/(TP + FP)

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

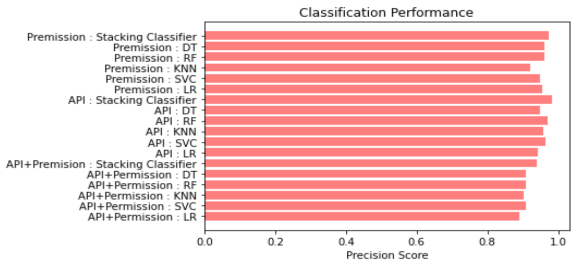


Fig 11 Precision comparison graph

Recall: Recall measures a model's ability to identify all relevant instances of a class. It is the ratio of true positives to the sum of true positives and false negatives, indicating the model's completeness in capturing the target class.

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

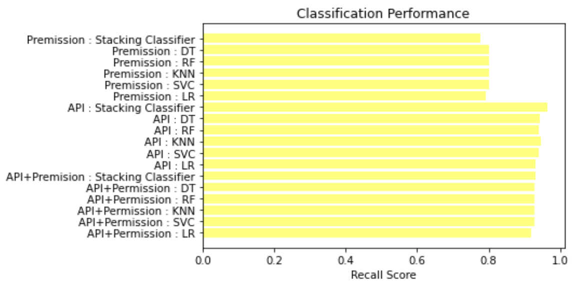


Fig 12 Recall comparison graph

Accuracy: Accuracy is the ratio of correctly predicted outcomes to the total predictions, reflecting a model's overall performance in a classification task.

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

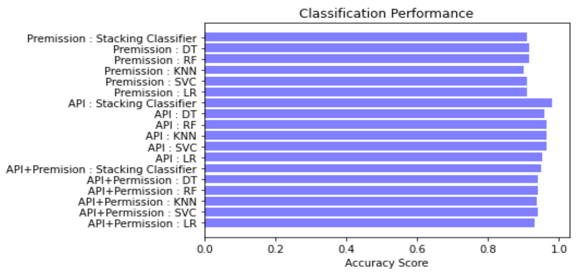


Fig 13 Accuracy graph

F1 Score: The F1 Score is the harmonic mean of precision and recall, balancing false positives and negatives, and is ideal for imbalanced datasets.

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

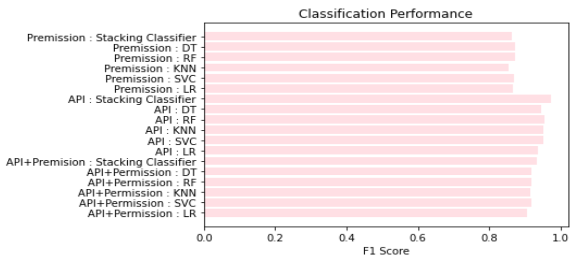
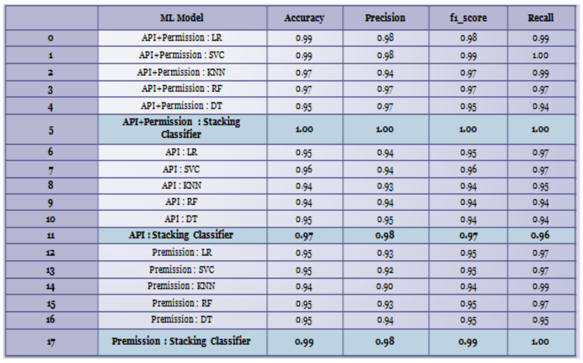
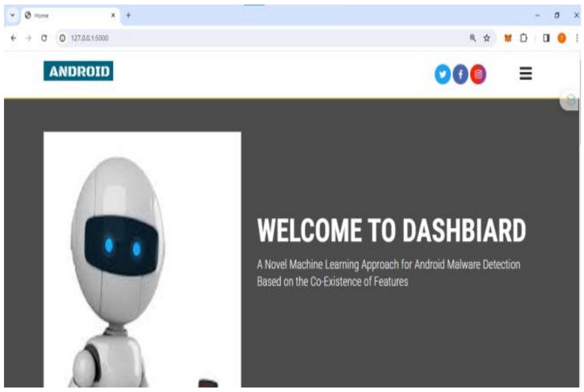


Fig 14 F1Score



	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	Premission : LR	0.95	0.93	0.95	0.97
13	Premission : SVC	0.95	0.92	0.95	0.97
14	Premission : KNN	0.94	0.90	0.94	0.99
15	Premission : RF	0.95	0.93	0.95	0.97
16	Premission : DT	0.95	0.94	0.95	0.95
17	**Premission : Stacking Classifier**	**0.99**	**0.98**	**0.99**	**1.00**

Fig 15 Performance Evaluation



WELCOME TO DASHBIARD

A Novel Machine Learning Approach for Android Malware Detection Based on the Co-Existence of Features

Fig 16 Home page

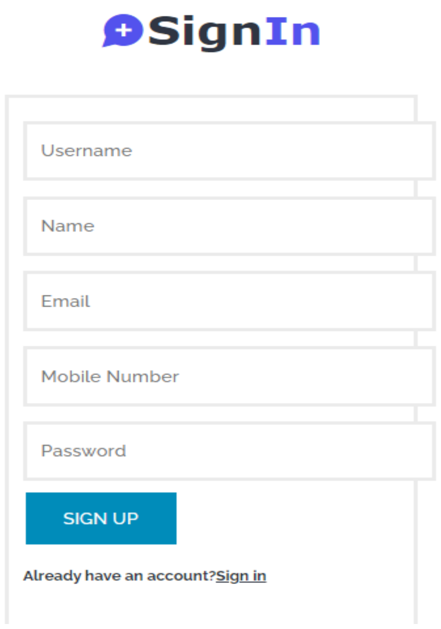
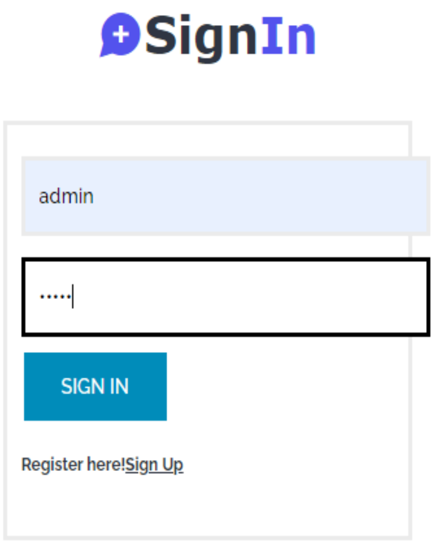


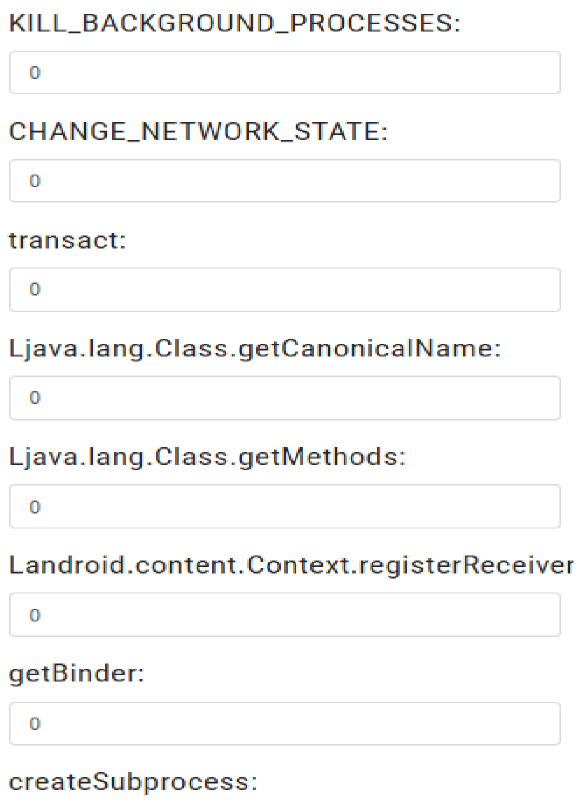
Fig 17 Signin page



admin

•••••

Fig 18 Login page



KILL_BACKGROUND_PROCESSES: 0

CHANGE_NETWORK_STATE: 0

transact: 0

Ljava.lang.Class.getCanonicalName: 0

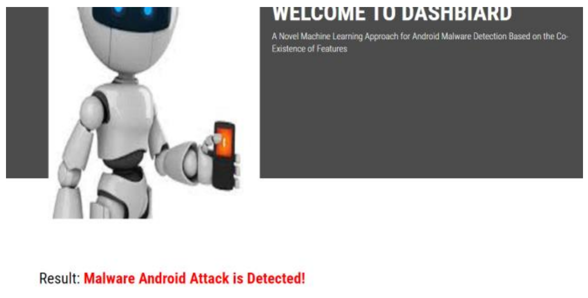
Ljava.lang.Class.getMethods: 0

Landroid.content.Context.registerReceiver 0

getBinder: 0

createSubprocess:

Fig 19 User input


WELCOME TO DASHBIARD

A Novel Machine Learning Approach for Android Malware Detection Based on the Co-Existence of Features

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

Fig 20 Predict result for given input

5.CONCLUSION

The project highlighted the effectiveness of machine learning in detecting Android malware, with the stacking classifier outperforming individual models and enhancing accuracy. Flask and SQLite were integrated to offer a user-friendly interface for efficient testing and predictions. The project emphasized the need to combine API and Permission features for better detection and noted varying model performance across datasets like Drebin, Malgenome, and CIC_MALDROID2020, underscoring the importance of careful dataset selection. The models achieved a balance between accuracy and minimizing false positives, reducing the risk of misclassifying legitimate apps.

6.FUTURE SCOPE

Further research should aim to enhance real-time detection by continuously analyzing dynamic features to better address evolving malware threats. Techniques like mutual information or recursive feature elimination can refine feature selection, improving model efficiency and accuracy. Adding behavioral anomaly detection could provide extra security by identifying deviations from normal app behavior. Adapting the system to incorporate new threat intelligence and update models is crucial for maintaining effectiveness. Expanding to cross-platform malware detection, including iOS, would offer a more comprehensive mobile security solution.

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/esraa-cell28/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] Androzoo. Accessed: Jul. 30, 2022. [Online]. Available: https://androzoo.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://contagiomini dump. 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.