IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 27, Issue 3, Ser. 1 (May. – June. 2025), PP 46-59
www.iosrjournals.org

Android Malware Detection Using Machine Learning

Abdulmunem Alsultan, Sherif Kamel
Computer Science Department – Arab East Colleges – Riyadh - Ksa,
Computer Science Department – Arab East Colleges – Riyadh - Ksa,
Computer Engineering Department – October University For Modern Sciences And Arts (MSA) –Giza - Egypt,

Abstract:
As the popularity and ubiquity of Android devices continue to rise, so does the risk of malicious software targeting these platforms. Android malware poses significant threats to users' privacy, data security, and overall device performance. Therefore, effective detection and mitigation of Android malware have become essential to ensure a safe and secure user experience. In this project, a malware detection system is proposed that extracts permission and intent features from APK files using the SISIK web tool to effectively identify and classify applications as malware or benign without the need to run the application. This is done by incorporating two different Machine Learning (ML) algorithms, which are Random Forest (RF), and Support Vector Machine (SVM). To obtain the best performance in our system, we use a feature selection method. The main contribution of this Research Paper is to enhance the security of Android devices detecting malicious applications before installing them in the devices. Our results show that the RF model, with the use of the Genetic algorithm (GA) to reduce the dataset's dimensions, achieved the highest performance metrics, including accuracy, recall, F1 score, and precision of 98%, 99%, 98%, and 98%, respectively.

Key Word: *Machin Learning, Android Malware, Malware Detection.*

Date of Submission: 22-04-2025 Date of Acceptance: 02-05-2025

I. Introduction

With the development of communication technology and as we live in an interconnected world, mobile devices have become an important part of our daily lives. Android, being one of the most popular mobile Operating Systems (OSs), offers a wide range of features and applications that enhance our productivity and entertainment. As the popularity of Android devices continue to rise, so does the risk of malicious software target these platforms. Therefore, effective detection and mitigation of Android malware have become essential to ensure a safe and secure user experience.

In 2018, Android devices were identified as the most targeted system, experiencing the highest percentage of malware infections at 47.15%. Windows/PCs encountered a threat rate of 35.82%, IoT faced 16.17% threats, and iPhones experienced lower threat rate of 0.85%, as shown in Figure 1. Consequently, there is a strong need for reliable, scalable, and robust malware detection tools for Android devices because malware can compromise user privacy, steal sensitive information stored on the device such as passwords and bank data, and cause financial losses through activities like identity theft or fraudulent transactions. By detecting malware, users can maintain the security and integrity of their personal information and mitigate potential harm.

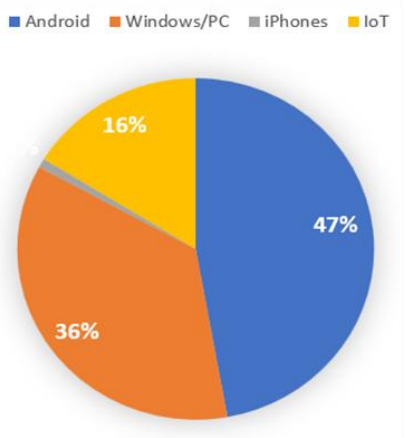

Figure 1. Percentage of Malware Attack

The main focus of this Research Paper is to develop an effective malware detection approach that is designed for Android OS. The detection is based on the static features which are declared in the AndroidManifest.xml file. To achieve this, we start by loading the dataset, which is essential for training and evaluating the detection models. Next, it is important to preprocess the dataset including cleaning it and selecting the most relevant features or characteristics of Android applications that can be used to distinguish between malicious and benign applications. Building the ML models for malware detection is the most important part of this project. The malware detection system is tested and evaluated with both known and unknown malware samples to assess its effectiveness and robustness. This project attempts to achieve the following objectives:

1. Classifying APK files as safe or malicious based on static features found in the APKs.
2. ML models are used in order to analyze the features and predict the class of the applications.
3. Achieving a detection accuracy rate above 95%. This was achieved through the use of ML models. Moreover, the feature selection algorithms participated in increasing the accuracy of the ML models.

The remaining of the paper is structured as follows. In section II, the related background is discussed. Section III presents a survey of the exiting papers and studies. Section IV presents details about the proposed systems. Section V explores the result we got. Finally, Section VI concludes this paper.

II. Background

This section provides an overview of malware detection and malware analysis, the architecture of Android OS and the structure of its applications, and the last section gives a general background related to machine learning (ML).

Structure of Android Application

The extension of Android application files is called Android Package Kit (APK), which stands for Android Package Kit. An APK is a file that is compressed in ZIP format and runs an application installed on the Android OS. It contains multiple files that are needed to run an app, and when unzipped, the APK file's unique structure is revealed [1]. Figure 2 shows the structure of the APK file.

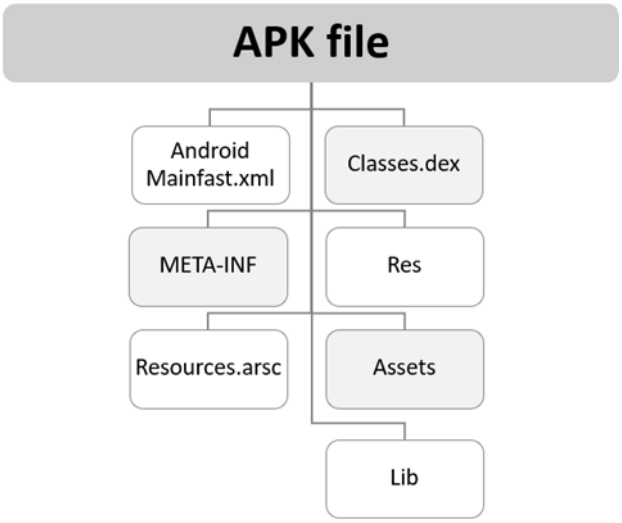

Figure 2: Structure of APK File

The structure is composed of:

1. AndroidManifest.xml: This file is mandatory and contains essential information about the app, such as the permissions required, metadata, hardware and software features, version, package name, etc. Once an application is launched, the first file the android system seeks is the Manifest file [2]. The focus of this project is on this file as it contains most of the required static features.
2. Res folder: This folder includes various sub directories, such as drawable, graphics layout, sound setting, languages, and more.
3. classes.dex: This file contains the compiled bytecode of the app's Java or Kotlin classes. It is a compressed version of the Dalvik EXecutable (DEX) format, which is the format understood by the Android Run-Time (ART or Dalvik).
4. lib/ directory: This directory contains native libraries required by the app, if any.

5. META-INF: This directory contains information about the APK's contents and digital signatures mainly for security and integrity purposes.
6. Resources.arsc: This file stores compiled resources used by the app, like strings, layouts, images, and other assets [3].
7. Assets: This directory can optionally contain additional application assets, such as raw data files or custom fonts.

Malware Analysis Approaches

Static, dynamic, and hybrid analysis are the most used techniques in Android malware detection. Static analysis involves examining the code and resources of an Android application without executing the program directly. It is applied by analyzing the AndroidManifest.xml file, smali files (which contain the application's bytecode), and a set of static features, such as permissions, API calls, and Dalvik opcode. These artifacts can be obtained by decompiling the APK files. This approach offers several advantages, including shorter analysis time and lower computational requirements compared to dynamic analysis methods. By leveraging static features, it becomes possible to quickly assess the application's security and detect known malware signatures or common patterns. However, it may be less effective in detecting sophisticated malware that relies on dynamic behavior [4].

Dynamic analysis involves executing the applications and observing their behavior in real time. involves executing applications directly on a real device or within a sandbox environment. This method focuses on monitoring the behavior of the application during run-time. By running the application, analysts can observe its interactions with the device, network, and user data. They capture logs and analyze the network traffic generated by the application. While dynamic analysis offers a more comprehensive understanding of an application's behavior, it can be more time-consuming and computationally demanding compared to static analysis. Nevertheless, it is highly effective in detecting sophisticated malware that employs evasion techniques or exhibits malicious behavior only at run-time [4].

In hybrid analysis, to enhance the overall analysis, the malware is initially examined using static analysis techniques, followed by a dynamic analysis approach. This two-step process involves analyzing the malware's code and structure without execution (static analysis) and then executing the malware and observing its behavior in a controlled environment (dynamic analysis). By combining static and dynamic analysis, a more comprehensive and thorough analysis of the malware is achieved [5].

III. Survey On The Existing Methodologies And Frameworks

With the rise of Android devices, malware targeting this platform has become a significant concern, leading to various research efforts in detection methods. Numerous studies have explored different ML models for Android malware detection, including decision trees (DT), support vector machines (SVM), and deep learning (DL). This section provides a discussion about the recent works related to the detection approaches that were identified and explored in the previous chapter.

Static Analysis

Malware Analysis for Effective Android Malware Detection

In this research paper, the author proposed a method for malware analysis and detection. This method involved examining static attributes including manifest permissions, API call signatures, intent filters, command signatures, and binaries. As shown in Figure 3, this detection approach includes 6 phases. The author used a COLCOM dataset in addition to his dataset which was collected by him. This was done in the first step. The second step, the author collected only the most relevant data from the application that is being analyzed. These features include manifest permissions, API call signatures, intent-filters, command signatures, and binaries.

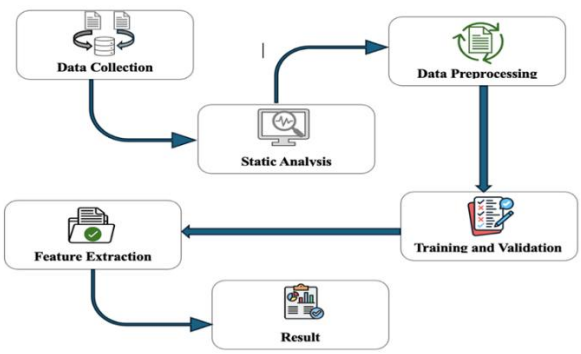

Figure 3: Detection Phases of Effective Android Malware Detection

The preprocessing phases include removing null values and duplication and making a balance between the samples. The training and validation of the dataset is the next step in the process. Training and validation data are separated from the original dataset. To verify that the data used for validation and training are completely different, the Author has utilized cross-validation using a 5-fold fold ratio. Naive Bayes, Key Nearest Neighbors (KNN), and Multi-Layer Perceptron are utilized as ML models that were used in this work for training and validation along with a method called Principal Component Analysis (PCA), which was used in the feature extraction step to reduce the number of attributes required to accurately characterize each app. The conducted experiment showed that MultiLayer Perceptron has the highest level of accuracy but requires a long time for training compared to other ML models used in this experiment. Moreover, the author concluded the two attributes that most effectively distinguish malware from benign applications are the feature to read phone attitude and the Internet. The limitation of this paper is that the best accuracy is obtained with the model that took a long time in the training phase [6].

A Framework for Detection of Android Malware using Static Features

In this paper, the authors proposed an effective framework that combines static features and utilizes ML classifiers. Firstly, they collected the samples and then removed the duplicate apps and performed labelling. After that, static features are extracted using two tools, which are AXMLPrinter and Baksmali Disassembler. Three types of static features were extracted: API calls, permissions, and intents. API calls were extracted from classes.dex, while permissions and intents were extracted from AndroidManifest.xml. Every app is presented in the form of a binary vector, 0s and 1s.

The last step is the classification process in which these features are trained using four ML classifiers which are SVM, Random Forest (RF), KNN and DT and then compare the results. They divided the dataset into a ratio of 80% and 20%. 80% of the data is used for training and the rest 20% is used for testing purposes. In the training phase, they applied a 5-fold cross validation technique in which the complete data is split into five equal parts. At each run, four parts are used for training purposes and the rest is utilized for testing purposes. This was repeated five times. The authors tested the models with the use of permission, intent, API calls, and a combination of all previous static features. The results from the test indicate that the combination of features outperforms individual features, As appears in Figure 4. Additionally, it was found that RF and KNN classifiers achieved the best accuracy rate [7].

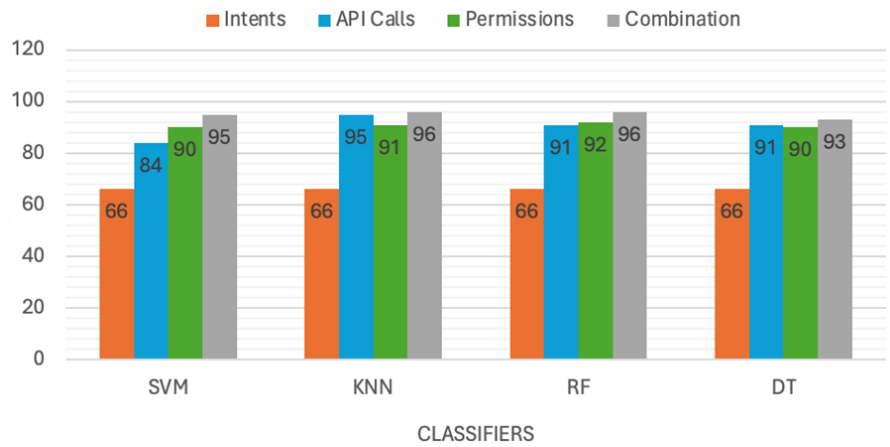

Figure 4: Accuracy Result of [7]

Dynamic Analysis

Mldroid Framework for Android Malware Detection using Machine Learning Techniques

In this paper, the authors proposed a framework that detects malware from Android applications by performing dynamic analysis. As shown in figure 5, they started by collecting .apk files from different trusted sources, such as Google's play store. A total of 55,000 malware samples are collected from three different data sets. The model is trained by using the dynamic behavior of real-world applications that were collected from different promised repositories and the experiment was performed on more than 500,000 Android applications. After collecting samples of .apk files from various repositories, they extracted permission and API calls from each of the .apk files. The authors extracted features from the tested application while running them in an emulator. Each application is represented as an 1844-dimensional Boolean vector, where '1' implies that the application requires the specified features and '0' implies that features are not required.

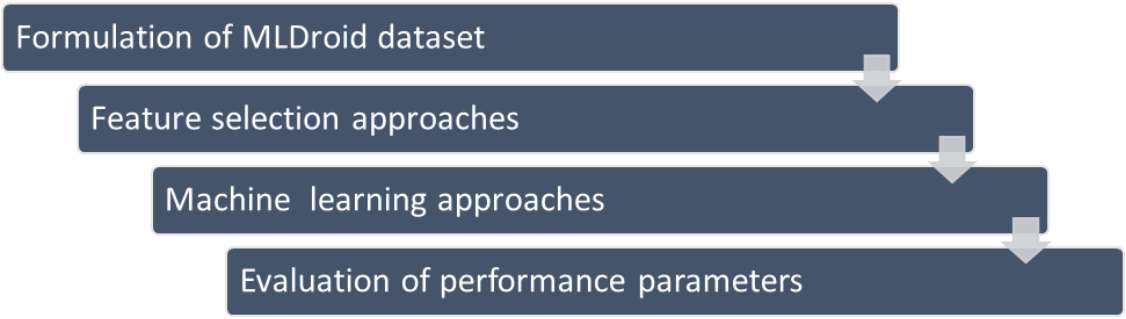

Figure 5: Mldroid Framework Methodology

In features, selection and ranking, the authors used multiple algorithms and compared their performance. They used chi-Squared, information-gain, oneR feature selection, PCA, and logistic regression. They explored four types of ML models that are not widely used, which are the farthest first clustering, nonlinear ensemble decision tree forest approach, Multilayer Perceptron and the last one is DL algorithm. In this work, distinct performance matrix, which are F-measure, accuracy, intra-cluster and inter-cluster distance, were utilized for measuring the performance of malware detection approaches. The con of the approach is that the model performed better when only trained with a few numbers of malware families. They achieved a detection rate of 98.8% to detect malware from real world applications [8].

Hybrid Analysis

Two Anatomists Are Better Than One Dual-Level Android Malware Detection

In this paper, the authors introduced an automated hybrid analysis tool that extracted groups of static and dynamic features to analyze the behavior of an application in the Android platform. This approach includes two main subsets, one for static analysis and one for dynamic analysis, as appear in figure 6. Static features were extracted with the use of reverse engineering techniques. In this step the author utilized APKtool to get relevant information from AndroidManifest.xml" and "classed.dex".

In Dynamic feature extraction, they used a tool to extract dynamic feature during runtime. A total of six feature categories are investigated, including permissions, intents, API calls, network traffic, inter-app communication, and Java classes. Among these categories, only permissions, intents, and API calls apply to static analysis. The authors found out that the API calls category showed greater influence and tends to enhance the performance of the hybrid methods. Additionally, the Java classes category also achieved notably high average importance scores. The experiment for this work was done over three different datasets in order to show that this hybrid analysis of Android applications can greatly improve the detection capabilities of a detection model [9].

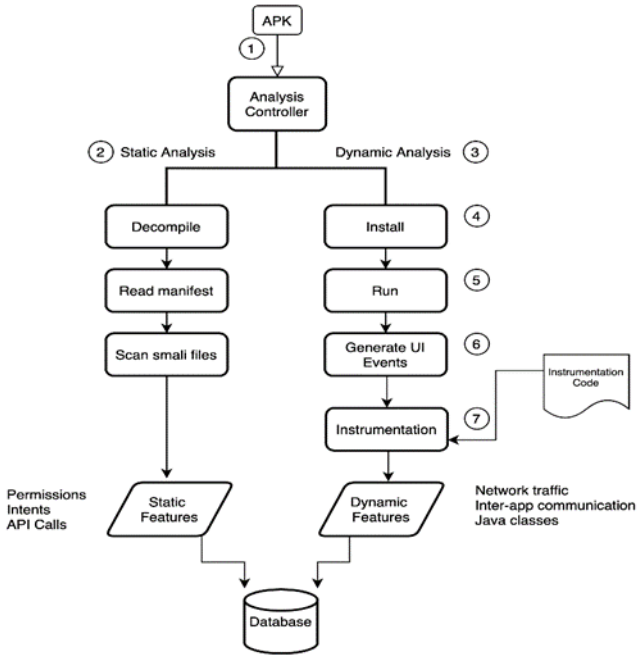

Figure 6: Automated Hybrid Analysis Approach

Comparison of previous literature review

This section provides a discussion about the recent related works to the detection approaches that were identified and explored in the previous chapter. Table 1 provides a summary of related studies' surveys regarding the detection analysis approaches, used ML models, Advantages, and Limitations found in each paper.

Table 1: Summery of Related Works

Paper	Pub. Year	Analysis Approach	ML Model	Contribution	Limitations
Malware Analysis for Effective Android Malware Detection	2023	Static Analysis	Naive Bayes, KNN, and Multi-Layer Perceptron	Presenting the top 10 features that have a significant impact on the model's decision.	1- The dataset is old, containing information that is over five years old. 2- The best accuracy is obtained with the model that took a long time in the training phase.
A Framework for Detection of Android Malware using Static Features	2020	Static Analysis	SVM, KNN, RF, and DT.	The authors conducted a comparative analysis to evaluate the performance of using individual features compared to the use of a combination of features.	The times taken to train and test the models are not mentioned at all.
Mldroid Framework for Android Malware Detection using Machine Learning Techniques	2021	Dynamic Analysis	DL algorithm, farthest first clustering, Multilayer Perceptron and nonlinear ensemble decision tree forest	The use of rarely used ML models.	The model performed better when only trained with a few numbers of malware families.
Two Anatomists Are Better Than One Dual-Level Android Malware Detection	2020	Hybird Analysis	Many ML models.	Applying the models on three datasets and comparing their result.	He uses of outdated dataset

Summery

In a study conducted by Gorment et al. [10] on malware detection for different platforms, it was found that 53.3% of the studies focused on static analysis. In contrast, dynamic analysis accounted for 28.9% and hybrid analysis for 17.8%, as illustrated in the Figure 7. This also highlights the significant effectiveness and attractiveness of static analysis for malware detection across various platforms. For that reason, we will use static analysis for malware detection in this project.

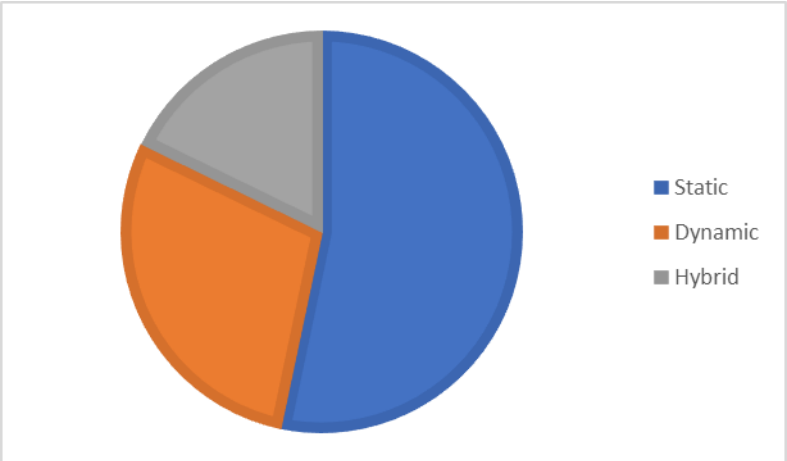

Figure 7: Usage of Analysis Approaches Based on [10]

Even though the recently proposed works by researchers have great accuracy, but some did not take onto consideration the time taken to build and the ML models. For example, BALCIOG ̆LU [6], the accuracy achieved was close to 99%, but the time taken was very long. In research paper, a new Android malware detection approach will be proposed using static features with a balance between the accuracy of the model and the time taken to train and test the ML models.

IV. Proposed System

The objective of this section section, we focus on the phases for building the proposed system including preprocessing the dataset, training and testing ML models, APK file analyzing, and classifying new samples. Figure 8 shows a flow diagram of malicious and benign APKs classification using ML models.

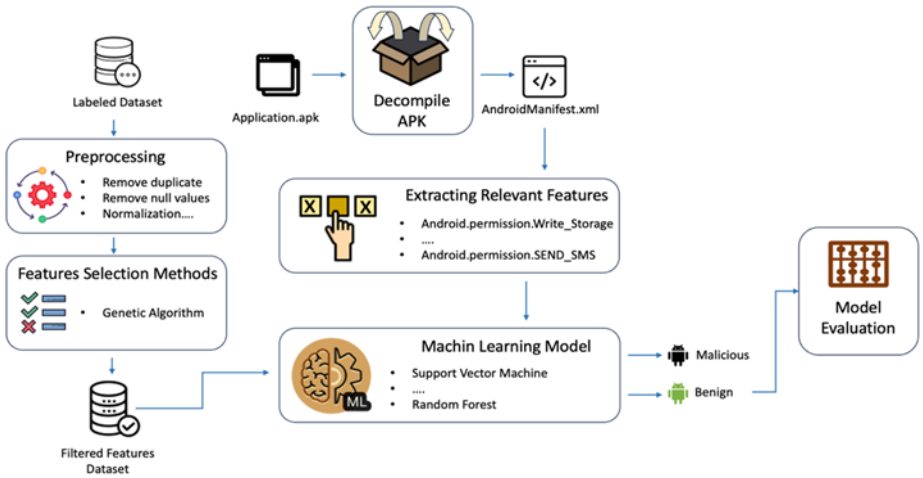

Figure 8: Overall System Architecture

Data loading and preprocessing

First, started by loading and exploring the dataset. We used the Panda Python library to load the datasets and store them in a data frame object. Then, to check how many rows and columns there are in each dataset, we used the shape function. As we mention in the previous section, the number of features in the dataset is 532 and number of samples is 847. Figure 9 shown the distribution of each class, where ”0” represents the Malicious samples and ”1” represents Benign. Figure 10 shows the distribution of features based on group on the dataset.

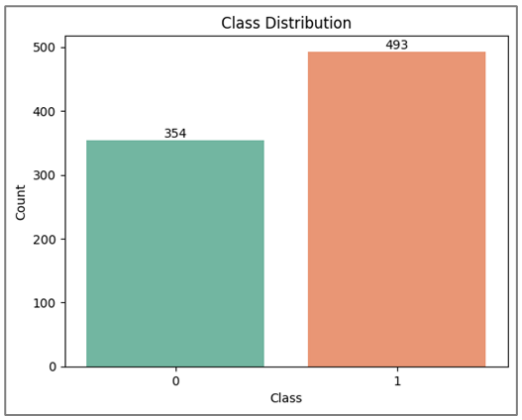

Figure 9: Distribution of each class.

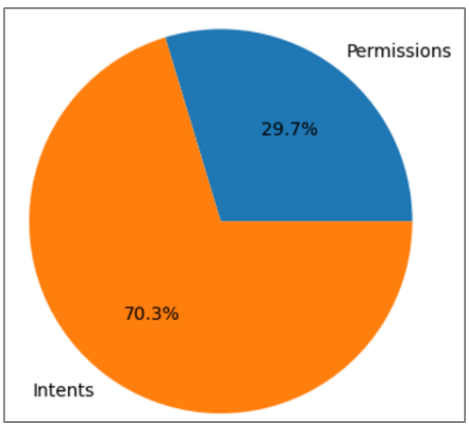

Figure 10: Distribution of features.

Data preprocessing refers to the steps and techniques applied to transform raw data to a suitable format to be used for training and testing ML models. It involves transforming and cleaning the data to ensure its quality, consistency, and suitability for further analysis. For performing data preprocessing in this research project we have used Python programming language due to its extensive libraries and tools that could be used for this purpose. The steps applied for data preprocessing include data cleaning, which involves handling missing data and removing duplicate rows because they can affect the result of classification and lead to unreliable output. Moreover, this step involves dealing with inconsistencies in the dataset. We will address any inconsistencies or errors in the data.

Removing Null values

The null() function will be used to check if our dataset contains any missing values. Figure 11 shows when we checked the number of samples with null values. We got 847, which corresponded to the total number of samples in the dataset. Then we realized that two columns contain null values for all samples. The second line in the figure shows when we deleted these columns. After that, we again checked the number of features and found that there were now 529 features and one column for the label.

```
print(data.isnull().sum().sum())
data.drop(columns=['Unnamed: 0', 'Unnamed: 530'], inplace=True)
data.shape

✓ 0.0s
847

(847, 530)
```

Figure 11: Removing null values.

Shuffling the Datasets

For shuffling the dataset, we use the sample() function with the parameter frac=1 to ensure that the entire dataset is sampled. Data shuffling or randomization is used because the samples in the datasets are often collected in a specific order which may introduce some bias. By shuffling the data, we ensure that the order of the data points does not influence the model's training and testing process.

Feature Selection

Feature selection is a technique that involves reducing the number of features in a dataset by selecting the most important and effective ones. It aims to identify and choose the features that have the most significant impact on the target variable or the overall performance of a model. This can be done based on statistical measures or correlation analysis. For this research paper, a Genetic Algorithm (GA) is applied to reduce the number of features. After applying each technique, the accuracy of the models is measured to compare their performance. GA is an algorithm that simulates the natural process of evolution. It involves a crossover process where multiple generations are combined and iterated until the best generations are obtained. The algorithm aims to optimize a solution by mimicking the principles of natural selection and genetic inheritance. It functions by employing bioinspired operators such as crossover, mutation, and selection. Table 2 shows the final number of selected features by GA feature selection methods in the dataset, which enhances the accuracy of the ML models and speeds the detection process.

Table 2: No. of selected features

No. of all features	529
No. of features after applying GA	280

Splitting the Dataset

For training and testing our ML models, we have to split the data set into features and labels, for that, we removed the result columns, which include the labels, and stored them in a variable, named "y" and the remaining columns which are the features are stored in a variable "X", where we store the selected features by GA, as shown in Figure 12 Then we divided our dataset into two sets: training and testing sets using the train test split() function with four parameters. The first and second parameters are the features and labels which we declared above and the third parameter is test size=0.2, which means 20% of the data will be used for testing, while the remaining 80% will be used for training. The fourth parameter is random state=42, which sets

the random seed to 42 to obtain the same train test split every time. Figure 13 show the number of samples for training and testing after splitting.

```
x= df2[tt]
y = df2['Result']
RF = RandomForestClassifier(random_state=7)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Figure 12: Splitting the dataset

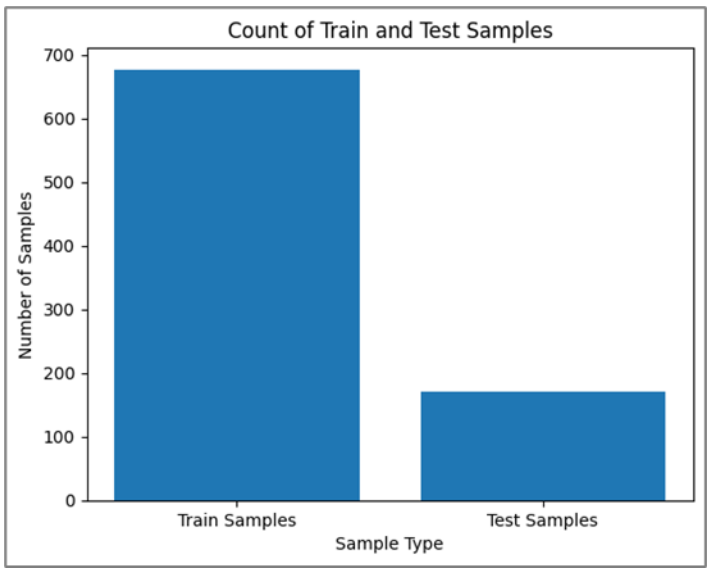

Figure 13: Number of training and testing samples

Machine Learning Classifiers

The system applies ML models for classification, including SVM and RF. These models are commonly employed in the field of malware detection due to their effectiveness in handling complex and high-dimensional feature spaces. These ML models will be trained using labeled data extracted from APK files. The training process involves adjusting the models' parameters to minimize prediction errors and optimize their performance. Once trained, the models will be capable of classifying new APK files as either malware or benign based on the learned features. All the used ML models are extracted from scikit-learn library. Then, we fit the models to the training data. Fitting the model involves training it on the training set to learn the relationships between the features and the label.

- **RF**: It is a classification algorithm that combines multiple uncorrelated decision tree classifiers. Each tree independently predicts the class for input values, and the final prediction is made by voting or averaging the predictions of all the trees. This supervised learning algorithm uses the bagging method to train multiple decision trees and combines their outputs to achieve more accurate and stable predictions that the data point falls into that category [11]. Figure 14 shows the code for using RF model.

```
RF = RandomForestClassifier(random_state=7)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

starttime = time.time()
RF.fit(X_train, y_train)
end = time.time()
RF_train_time = end - starttime
print("Random Forest Train Time:", RF_train_time)

starttime = time.time()
Y_predict = RF.predict(X_test)
end = time.time()
RF_test_time = end - starttime
print("Random Forest Test Time:", RF_test_time)
```

Figure 14: Code for using RF

- **SVM**: It utilizes a hyperplane in n-dimensional space to separate data into two distinct regions, corresponding to different classes. The objective is to maximize the margin between these regions. The margin is determined by the distance between instances of the two classes, with particular emphasis on the closest instances, known as support vectors. This approach allows us to effectively classify new samples based on their position relative to the learned hyperplane [12]. Figure 15 shows the code for using SVM model.

```
from sklearn import svm
SVM = svm.SVC()

starttime = time.time()
SVM.fit(X_train, y_train.values.ravel())
end = time.time()
svm_train_time = end - starttime
print("Support VM  Train Time:", svm_train_time)

starttime = time.time()
Y_predict = SVM.predict(X_test)
end = time.time()
svm_test_time = end - starttime
print("Support VM  Test Time:", svm_test_time)
```

Figure 15: Code for using SVM

Evaluation

Evaluation metrics are calculated using four parameters: True Positive (TP), False Negative (FN), False Positive (FP), and True Negative (TN). These evaluation metrics provide a comprehensive assessment of the performance of our proposed system by taking into account both correct predictions (TP and TN) and incorrect predictions (FP and FN). Accuracy measures the overall correctness of the system's predictions. It is calculated as shown in the below equation.

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

Precision represents the percentage of correctly classified malware samples out of all samples predicted as malware. It is calculated as shown in the below equation.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall measures the ability of the malware detection system to correctly identify malware samples. It represents the percentage of actual malware samples that are correctly classified as malware. It is calculated as shown in below equation.

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

Saving and Loading ML Models

Because the RF model has the highest accuracy value, we planned to use it for classifying new samples. To save and load our model, we use the serialization and deserialization techniques provided by the Pickle and Joblib libraries. After we trained and tested the model, we used the dump() function which saves the model in a pickle file with a .pkl extension. Joblib.dump() function serializes the model object and writes it to the specified file. For loading and using the ML model we used joblib.load() function with passing the pkl file. We can then make predictions on new samples. This is done in order to save time by avoiding the need to retrain the model every time you want to make predictions on new samples.

Features Extractions

Two types of features are needed for this project from APK files which are shown in Table 3.

Table 3: Types of features

Permissions features	Android applications request various permissions to access certain sensitive resources or data in order to perform their functional requirements and complete specific actions on a mobile device. Users have to allow the use of the requested permissions by applications [13]. Malicious apps often request excessive or unnecessary permissions, which indicates suspicious behavior.
Intents	Intents are a fundamental component of Android's inter-application communication system [14]. When one activity within an application needs to communicate with another activity, it generates an intent. An intent serves as a message or request that encapsulates the important information that

	the sending activity wants to communicate to the receiving activity [15]. By analyzing intent features, we can explore the communication patterns and potentially malicious activities of an application.

By combining permission features with intent features, we can leverage the information from both aspects to enhance the accuracy of your malware detection model. These features are extracted from the APK files using the SISIK web tool. It is a free online tool that provides a wide range of functionalities and methods to help in feature extraction and analysis. it extracts the manifast.xml from the APK file. Then we extract permission and intent from manifast.xml. We used Selenium which is an open-source framework for allowing Python to interact with web elements such as SISIK.

APK Classification

After extracting the relevant features from the APK file using the SISIK tool, we used them as input to the ML models. The predict proba() functions is used the final result of the system. It provides the probability of the input APK file belonging to each class. By using the predict proba() function, we can obtain additional information beyond just the class labels. For example, if the predicted probabilities for a given APK file are [0.8, 0.2], it indicates that the model is 80% sure that the file is malware) and 20% confident that it is safe.

GUI

Figure 16 shows the home interface of the system. The only thing that a user does is uploading an APK file and press the DETECT Button.

Figure 16: Home Interface of the system.

After that, the system will send the APK file to SISIK to decompile and extracting the needed features, which forces a new web window to open automatically showing the analysis result page in SISIK, as shown in Figure 17 Then, this window will close, and the final result well be shown to the user.

Figure 17: Analysis result page in SISIK

When the APK file has high probability of being malware, the system will show a warning with the percentage, as shown in Figure 18. On the other hand, when the APK file looks safe, Figure 19 will be shown.

Figure 18: Malicious APK File

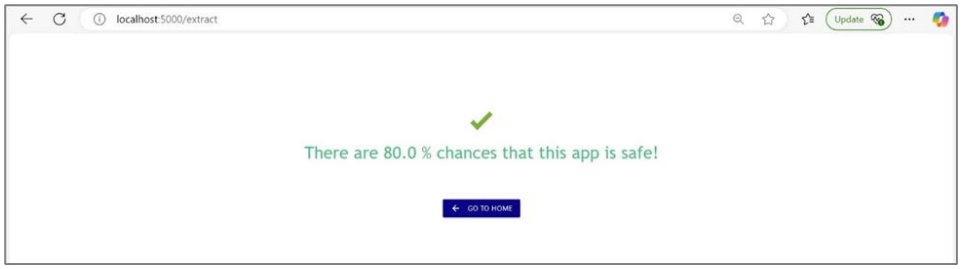

Figure 19*:* Safe APK File

V. Result And Findings

In this section, we explore the result we got during the implementation of research paper. We Compare the result obtained using bot ML models.

ML Models Evaluation

In this section, we present, compare, and discuss the overall result of the evaluation metric; accuracy, recall and precision, for both ML models. Moreover, we compare the time taken to train and test by both ML models. Figure 20 shows the comparison between the accuracy, recall, and precision of RF and SVM models. RF performs better than SVM, this is because RF is considered ensemble learning that combines multiple DT models to make a prediction and due to its robustness over overfitting.

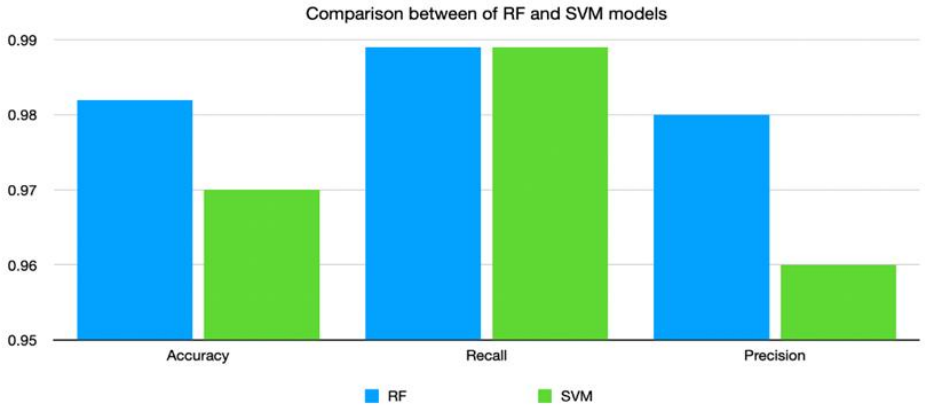

Figure 20: Comparison between of RF and SVM models.

After conducting this comparison, we decided to choose the RF model, as it has shown better performance than SVM. Hence, RF model is saved as pickle file and to used it to detect new samples. Figure 21 shows the time taken by both ML models to train and test. As it appears RF models has the longest time to train, this is because the training process of the RF model involves building and combining these multiple decision trees, which can be computationally intensive and result in a longer training time. However, during the test, the model can make predictions by simply averaging the outputs of the individual decision trees, which require a shorter time.

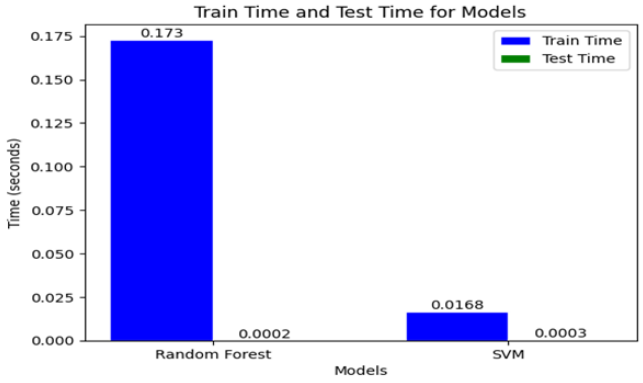

Figure 21: Time taken to train and test by both ML

Testing the Overall Performance of the System

After the comparison performed in the previous subsection, we conclude that the RF model has shown better performance than SVM. Our system is considered to be easy to use since the end user does not do any configuration and should only upload the APK file. The user should not have to worry about the underlying processes or switch between different tools. The system automatically sends the APK to the SISIK web tool, extracts the relevant data from the APK file, and then Selenium will take over to return the extracted information to the system. The ML models will use this information to detect the APK file. In this section, we will test real and new APK files.

For testing samples, we have collected APK files that were published by developers for test purposes on some websites. We have tested 2 samples: 1 benign and 1 malware. We send each sample to the SISIK web tool to be decompiled and analyzed. After that, it returned with a set of features that the APK file requested. We used them as input to the pickled ML model, which is RF model. Table 4 provides a comparison of the results we obtained for the real APK file we have tested with the correct type of the file, either malware or safe. The results indicate the chances of an application being classified as malware or safe. The average time taken to predict a new sample is 15 seconds. This involves sending the APK file to the SISIK tool to be decompiled and then analyzed. Then the extracted features are sent back to the system to the pickle file to make a prediction and present the result to the user.

Table 4: System Result

APK file	Actual Type	Result obtained for the system
File.apk	Malware	There are 60 % chances that this app is malware!
SurlyProject.apk	Safe	There are 99.14 % chances that this app is safe!

VI. Conclusion

With the increasing popularity and widespread use of Android devices, the risk of malicious software targeting these platforms is also on the rise. Android malware presents substantial dangers to user privacy, data security, and the overall performance of the devices. In this paper, we proposed an efficient system for identifying malicious applications based on the features extracted using static malware analysis. Efficient and accurate Android malware detection not only protects individual users' privacy and data security but also preserves the integrity of enterprise networks and sensitive information. By leveraging detection approaches, promoting user awareness, and implementing security best practices, we can mitigate the risks posed by Android malware and ensure a secure mobile experience for users.

In this research paper, we leveraged ML models along with permission and intent features extracted from the APK file to check the probability of being malicious or benign, which could help to either install that APK file or not. Our finding shows that RF outperforms SVM in accurately classifying APK files. The RF model achieved an accuracy rate of 98%, indicating its effectiveness in accurately classifying APK files. Our research paper contributes to the enhancement of malware detection methods for APK files. The high accuracy and performance of the RF model and the emphasis on testing real-world data highlight the potential effectiveness and practical applicability of our system.

In the future, we are planning to incorporate the use of explainable ML models in the Android malware detection field which provides interpretability in the decision-making process and enables users to understand why a specific prediction or classification is made by ML models. explainable ML models address the block box issues introduced by some ML models where the users do not know why specific prediction is taken by the ML models. However, it is important to take into consideration making a balance between interpretability and the performance of the models.

We believe that there are still more features to be explored and analyzed from Android applications for malware detection. we will explore the use of other Android features that could be extracted from Android applications without installing them, such as the inclusion of specific strings in the code comments. Some developers and programmers leave comments that may contain hints related to their malicious intent.

References

[1]. J. Lee, H. Jang, S. Ha, And Y. Yoon, "Android Malware Detection Using Machine Learning With Feature Selection Based On The Genetic Algorithm," Mathematics, Vol. 9, No. 21, 2021.
[2]. N. Peiravian And X. Zhu, "Machine Learning For Android Malware Detection Using Permission And Api Calls," In 2013 Ieee 25th International Conference On Tools With Artificial Intelligence, Pp. 300–305, 2013.
[3]. M. Bhatt, H. Patel, And S. Kariya, "A Survey Permission Based Mobile Malware Detection," International Journal Of Computer Technology And Applications, Vol. 6, P. 2, 10 2015.
[4]. J. Lee, H. Jang, S. Ha, And Y. Yoon, "Android Malware Detection Using Machine Learning With Feature Selection Based On The Genetic Algorithm," Mathematics, Vol. 9, No. 21, P. 2813, 2021.
[5]. N. Tarar, S. Sharma, And R. Challa, "Analysis And Classification Of Android Malware Using Machine Learning Algorithms," Pp. 738–743, 11 2018.

[6]. Y. Balcioglu, "Malware Analysis For Effective Android Malware Detection," In International Anatolian Congress On Scientific Research, Pp. 1112–1119, Mar. 2023.
[7]. M. Dhalaria And E. Gandotra, "A Framework For Detection Of Android Malware Using Static Features," In 2020 Ieee 17th India Council International Conference (Indicon), Pp. 1–7, Ieee, 2020.
[8]. A. Mahindru And A. Sangal, "Mldroid—Framework For Android Malware Detection Using Machine Learning Techniques," Neural Computing And Applications, Vol. 33, No. 10, Pp. 5183–5240, 2021.
[9]. V. Kouliaridis, G. Kambourakis, D. Geneiatakis, And N. Potha, "Two Anatomists Are Better Than One—Dual-Level Android Malware Detection," Symmetry, Vol. 12, No. 7, 2020.
[10]. N. Z. Gorment, A. Selamat, L. K. Cheng, And O. Krejcar, "Machine Learning Algorithm For Malware Detection: Taxonomy, Current Challenges And Future Directions," Ieee Access, 2023.
[11]. J. Lee, H. Jang, S. Ha, And Y. Yoon, "Android Malware Detection Using Machine Learning With Feature Selection Based On The Genetic Algorithm," Mathematics, Vol. 9, No. 21, P. 2813, 2021.
[12]. N. Peiravian And X. Zhu, "Machine Learning For Android Malware Detection Using Permission And Api Calls," In 2013 Ieee 25th International Conference On Tools With Artificial Intelligence, Pp. 300–305, 2013
[13]. S. Ramachandran, A. Dimitri, M. Galinium, M. Tahir, I. V. Ananth, C. H. Schunck, And M. Talamo, "Understanding And Granting Android Permissions: A User Survey," In 2017 International Carnahan Conference On Security Technology (Iccst), Pp. 1–6, Ieee, 2017.
[14]. A. Pathak, "Exploring Android Intents." Https://Medium.Com/@Myofficework000/Intents-In-Android-713da59ee700. [Accessed March 8, 2024].
[15]. M. W. Afridi, T. Ali, T. Alghamdi, T. Ali, And M. Yasar, "Android Application Behavioral Analysis Through Intent Monitoring," In 2018 6th International Symposiumon Digital Forensic And Security (Isdfs), Pp. 1–8, Ieee, 2018.