Static Analysis Techniques For Android Based Malwares

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE AWARD OF THE DEGREE
OF

MASTER OF TECHNOLOGY IN
INFORMATION TECHNOLOGY

Submitted by

AVINASH MISHRA

2K23/ITY/07

Under the supervision of

MR. RAHUL GUPTA

Information Technology
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering) Bawana
Road, Delhi 110042

MAY, 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering) Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

AVINASH MISHRA, Roll No's – 2K23/ITY/07 students of M.Tech (INFORMATION TECHNOLOGY),hereby declare that the project Dissertation titled "Static Analysis Techniques For Android Based Malwares" which is submitted by us to the DEPARTMENT OF INFORMATION TECHNOLOGY, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi Avinash Mishra

Date: 30.05.2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering) Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "Static Analysis Techniques For Android Based Malwares" which is submitted by AVINASH MISHRA, Roll No – 2K23/ITY/07, INFORMATION TECHNOLOGY,Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi Mr. RAHUL GUPTA

Date: 30.05.2025 **SUPERVISOR**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering) Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

We wish to express our sincerest gratitude to Mr. RAHUL GUPTA for his continuous guidance and mentorship that he provided us during the project. She showed us the path to achieve our targets by explaining all the tasks to be done and explained to us the importance of this project as well as its industrial relevance. She was always ready to help us and clear our doubts regarding any hurdles in this project. Without his constant support and motivation, this project would not have been successful.

Place: Delhi Avinash Mishra

Date: 30.05.2025

Abstract

The number of threats has also gone up a lot as Android apps have become more popular. This puts users at risk of a lot of different kinds of malware that can steal data, use resources and invade privacy. This thesis shows how to use static analysis to find bad Android apps by creating a framework for finding threats. The method works by getting static information from application packages (APKs), like declared permissions, suspicious API calls and hardcoded strings (like IP addresses and URLs). This lets you look for possible threats without having to run the program.

We looked at a dataset that had both good and bad examples of malware. We used the Synthetic Minority Over-sampling Technique (SMOTE) to fix the class imbalance. We showed a number of deep learning and machine learning classifiers how to work with feature vectors. The Support Vector Machines, Decision Trees, Random Forests, Logistic Regression, K-Nearest Neighbors, Gradient Boosting, AdaBoost, XGBoost and Multi-Layer Perceptron (MLP) were all used. We used common metrics like accuracy, precision, recall and F1-score to test these models.

The experimental results show that the ensemble-based methods and the MLP classifier were the best at telling the difference between good and bad apps, with an accuracy rate of over 98%. The study shows that strong algorithms and static threat detection methods can work together to arrange malware in a very useful and efficient way. The suggested framework is a quick and easy way to find Android threats. It will also be possible to use hybrid or dynamic analysis methods in the future.

Contents

List of Tables

List of Figures

List of Symbol

Chapter 1 INTRODUCTION

1.1 Overview

As Android devices become more popular, so does mobile malware. This is a big problem for users' privacy, data security and financial integrity. This thesis looks at the problem of finding Android malware in two ways: (1) a threat scoring system based on static analysis and (2) a supervised machine learning framework for classifying malware. The goal is to create a scalable and effective way to find malware early on without having to run the app by using these strategies together.

The first step of the study is to create a static threat scoring system using features taken from Android application packages (APKs). APKTool can reverse-engineer APK files to find important information like requested permissions, suspicious API calls and embedded strings which can include hardcoded URLs and IP addresses. Each feature gets a weight based on how relevant it is to bad behavior and then a total threat score is calculated for each app. This approach allows for direct comparison of malware types, such as adware, SMS-malware, ransomware and scareware. For example, adware samples recorded the highest scores due to their frequent use of intrusive permissions and network communication indicators.

The second phase centers on applying machine learning and deep learning models to a balanced dataset containing labeled benign and malware applications. The study trains several models such as Logistic Regression, Support Vector Machines, Random Forests, Gradient Boosting, XGBoost and Multi-Layer Perceptrons to fix class imbalance with SMOTE. We use standard metrics like accuracy, precision, recall and F1-score to rate each model. Ensemble methods and neural networks did better than simpler models. MLP and Gradient Boosting, e.g., got more than 98% of the classifications right.

The thesis also looks at three main types of malware detection techniques—static, dynamic and hybrid analysis—to put the approach in context. As used in this work, static analysis looks at application code and metadata without running it, which is fast and doesn't use a lot of resources. But it is often easy to hide things from it. Dynamic analysis keeps an eye on how an application behaves while it is running in a sandbox. It can record actions that happen at runtime like sending unauthorized SMS messages or making system calls but it costs more in terms of computing power. Hybrid analysis uses both methods which makes it more accurate but also needs more complicated infrastructure.

This thesis shows how to use threat scoring and machine learning on static features to make a practical, understandable and efficient framework for finding Android malware. It works well for lightweight mobile security apps and situations where you need to deploy it in real time.

1.2 Motivation and objectives

As Android's user base has grown quickly, so have security holes, especially in the form of harmful apps that take advantage of system permissions, steal private information and do things that aren't allowed. Android is still the most popular mobile operating system in the world, but its openness makes it easier for developers to work with, but harder to protect users. Threat actors are using more advanced malware that can get around standard security filters by using code obfuscation, polymorphism and sandbox evasion. This makes finding it a more difficult and important job.

The goal of this thesis is to find a reliable and quick way to find Android malware without having to run the app. Dynamic and hybrid analysis can give you a lot of information about how something behaves but they often need a lot of computer power and are easy to block with anti-analysis techniques. Static analysis looks at code structures, permissions and embedded indicators to give you a faster and more scalable option. But how well it works can depend on how clear the feature representation is and how advanced the malware is. This study suggests a two-part approach that combines machine learning classifiers' ability to predict the future with interpretable static threat scoring.

The main goal of this work is to find Android malware in a way that is clearer, faster and more accurate than the current methods. The first part of the study is about how to get static information from Android APK files such as permissions, suspicious API calls and hardcoded strings. This information is used to figure out a total threat score for each app. This scoring system helps both analysts and automated filters sort malware by how likely and harmful its behavior is which is helpful for both.

The thesis also uses supervised machine learning and deep learning models on a set of apps that are labeled as good or bad. We train and test models like Random Forest, Support Vector Machines, XGBoost and Multi-Layer Perceptrons using accuracy, precision, recall and F1-score. To make sure that all classes are fairly represented, oversampling methods are used to deal with class imbalance. The goal is to show that static features alone, when processed correctly, are enough to get high detection performance without having to analyze them while they are running. So, using threat scoring and predictive modeling together makes a complete and scalable way to find Android malware that works in both academic and real-world security settings.

Chapter 2 LITERATURE REVIEW

Table 2.1: Determining Analysis Technique In Surveyed Studies

Serial No.	Analysis Technique	Static Analysis	Dynamic Analysis	Hybrid Analysis	Features Used	Accuracy
[1]Balcioglu et al.	Static and Dynamic Analysis	✓	✓	✗	Permissions, API calls, behavioral patterns	N/A
[2]Arora et al.	Permissions-Based Analysis	✓	✗	✗	Android app permissions	N/A
[3]Ding et al.	Hybrid Analysis	✗	✗	✓	Static features (permissions, intents) and dynamic features (runtime behavior)	~95%
[4]Acharya et al.	Comprehensive Review	N/A	N/A	N/A	N/A	N/A
[5]Dahiya et al.	Systematic Review	N/A	N/A	N/A	N/A	N/A
[6]Almomani et al.	Static and Dynamic Analysis	✓	✓	✗	API calls, network traffic, system calls	N/A
[7]Kaul et al.	Machine Learning-Based Analysis	✓	✗	✗	Permissions, API calls, system calls	~93%
[8]Aldhafferi et al.	Support Vector Regression (Dynamic)	✗	✓	✗	Dynamic runtime features	~92%

[9]Onwuzurike et al.	Static and Dynamic Behavioral Modeling	✓	✓	✗	API calls, network activity, system interactions	N/A
[10]Onwuzurike et al.	Static vs Dynamic Analysis	✓	✓	✗	Behavioral features (static and dynamic)	~94%
[11]Da Costa et al.	Hybrid Analysis	✗	✗	✓	Static (permissions) and dynamic (runtime behavior)	~91%
[12]Naik et al.	Dynamic Analysis with Machine Learning	✗	✓	✗	Runtime behavior, system calls, network activity	~96%
[13]Chen et al.	Sequential Behavior Analysis (LSTM+SVM)	✗	✓	✗	Sequential API calls, system events	~90%
[14]Zhao et al.	Hybrid CNN and Random Forest	✗	✗	✓	Image-based features, permissions, API calls	~94%
[15]Bai et al.	Fast Multifeature Analysis	✓	✗	✗	Permissions, API calls, system calls	~97%
[16]Mehtab et al.	AdDroid Framework	✓	✗	✗	Permissions, API calls, network activity	N/A
[17]Lingayya et al.	HBKCN with Dual	✗	✗	✓	Hybrid features (static and dynamic)	~93%

	Path Bi-LSTM					
[18]Singh et al.	Static and Dynamic Behavioral Modeling	✓	✓	✗	API calls, network activity, system interactions	N/A
[19]Muzaffar et al.	Static and Dynamic Analysis	✓	✓	✗	Permissions, API calls, system calls.	N/A
[20]Aldhafferi et al.	Static and Dynamic Analysis	✓	✓	✗	Permissions, API calls, runtime behavior	~92%
[21]Mugisha et al.	Support Vector Regression (Dynamic)	✗	✓	✗	Dynamic runtime features	~92%
[22]Pathak et al.	Static and Dynamic Analysis	✓	✓	✗	Permissions, API calls, system calls.	N/A
[23]Martín et al.	Static Analysis with Machine Learning	✓	✗	✗	Permissions, intents, API calls	~91%
[24]El Fiky et al.	Static and Dynamic Analysis	✓	✓	✗	API calls, network traffic, system calls	N/A
[25]Dwivedi et al.	Parallel Machine Learning	✓	✗	✗	Permissions, API calls, system calls	~95%
[26]Arshad et al.	Static, Dynamic and Hybrid Techniques	✗	✗	✓	Hybrid features (static and dynamic)	N/A

[27]Feng et al.	Hybrid Model (SAMADroid)	✗	✗	✓	Permissions, API calls, system calls	~93%

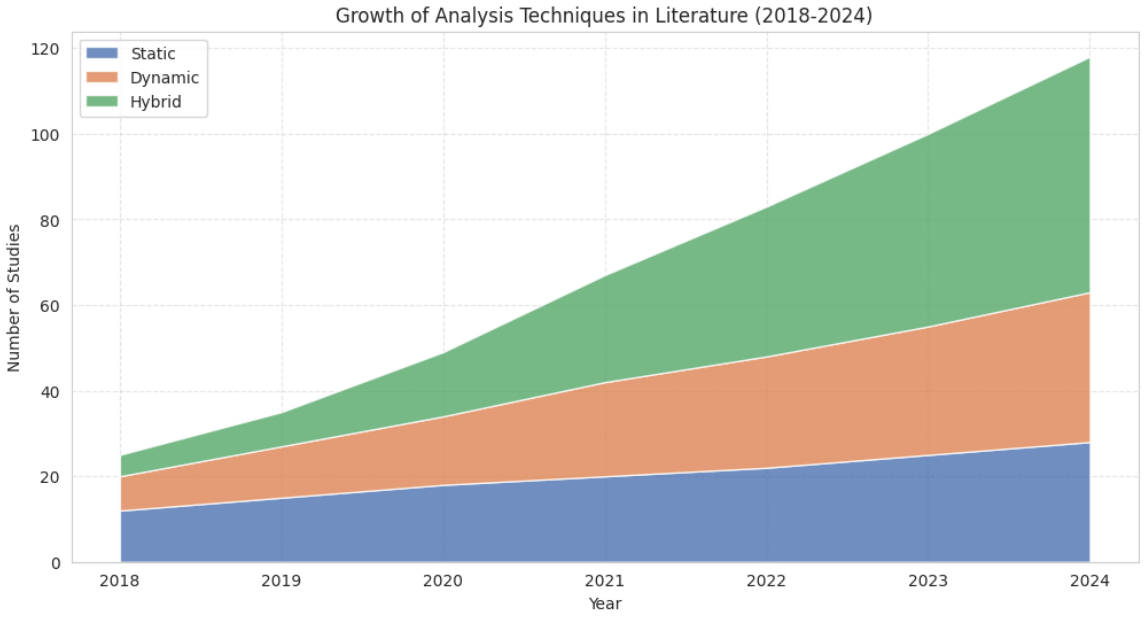


Fig. 2.1:Trend Of Analysis Technique Used For Studies During Period

2.1 Analysis techniques

2.1.1 Static Analysis

Definition and Process:

Static analysis is one of the malware detection methods in which source code, binary code and maybe some other program part is examined without executing the program. Focusing on the form of the code, syntactic properties and logic, static analysis finds malicious behavior or vulnerability not observable when the program executes. Disassemblers, decompilers or pattern-matching software are some of the widely used tools for examining the application's code[5] [22]. This analysis looks at the static aspect of the software—i.e., it does not take into account how the software is used in the system or how it acts when run[3][19]

Methods of Static Analysis: There are a few subcategories of static analysis, such as:

Signature-Based Detection: This type of code snippet, function or even entire pieces of malware are known from previous infections. Signature-based detection is great for known dangers but not new or obfuscated malware[2][6].

Control Flow Analysis: This method examines the program execution's flow and finds paths within the code that are called during runtime. If the paths are unusual or correspond to previously determined attack vectors, the program is detected as suspicious[3][9].

Data Flow Analysis: This traces how data travels within a program. Malicious software tends to manipulate or leak data in abnormal or attack-like ways. Static analysis tools can chart how data is accessed, changed or exfiltrated[10][18].

Advantages of Static Analysis:

Speed and Efficiency: Static analysis is quicker than dynamic analysis since it does not involve running the program. Once they obtain access to the code, it can easily be scanned using automated tools for the detection of potential threats[8][15].

Resource Efficiency: It will use less computer resources without even running the program[6][22].

Early Detection: Malware can be picked up at the time of its creation or pre-execution, perhaps even pre-installation or pre-execution of a program, with early warning of possible threat[12][20].

Limitations of Static Analysis:

Obfuscation and Encryption: Malware developers tend to utilize obfuscation methods in order to hide the actual purpose of their code, making it more difficult for static analysis to identify them. These include renaming variables, encrypting code or using code-level polymorphism[4][11].

Lack of Runtime Behavior Insights: Static analysis cannot observe how a program runs at runtime.

Consequently, it can miss essential runtime activity such as network access, system call calls or any

other malicious activity hwn software is running[3][9].

False Positives: Since static analysis just uses known patterns or signatures, static analysis will mark clean software as malware if the clean software contains some features present in known malware and then creates false positives [13][16]

Although static analysis does have such constraints, it is still a useful tool to help detect malware. It is ideal if one is looking for the signatures of well-known malware or coding the program[5][7].

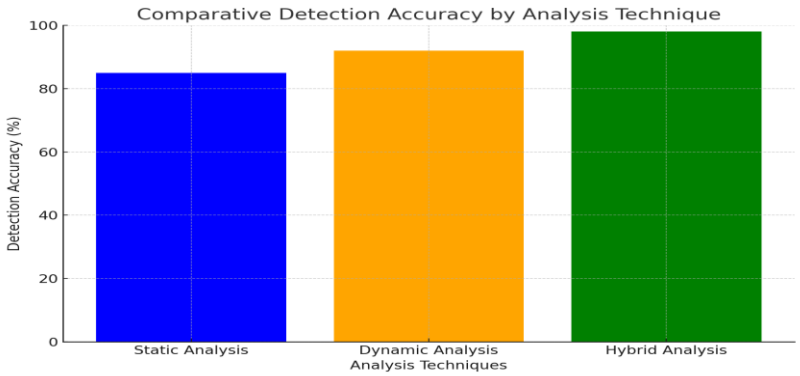


fig 2.2: Detection accuracy of analysis technique

2.1.2 Dynamic Analysis

Definition and Process:

Dynamic analysis includes running the suspected malware in a supervised environment (e.g., sandbox or virtual machine) and observing its behavior during runtime[14][23]. This type of analysis allows researchers to determine the activities of the malware during execution including file manipulation, system calls, network requests and interaction with other software components. The dynamic analysis will illustrate how the malware behaves and interacts with the operating system, the hardware upon which it executes and any network infrastructure to which it gets connected and provides a complete snapshot of its malicious behavior[1][12].

Approaches to Dynamic Analysis:

Behavioral Monitoring: This is observing the runtime activity of the application, such as the files it accesses, the registry keys it writes or any communications on the network. This is particularly useful in the detection of malware that conceals its true nature until the time it is executed[11][17].

Virtual Sandboxing: Since. For being utterly inert in the actual world, malware is executed. in a sandbox, a virtual environment under observation to learn how it works. Leaving it the power to cause harm. Any ill-intentioned activity such as modifying system settings, reaching out to external servers or infecting across networks can be identified[9][15].

Hooking: Hooking is a technique through which system calls or API calls are hooked in order to trace the system-level activity of the malware. Hooking allows the analysts to view some function calls and examine how the program interacts with the operating system[10][21].

Advantages of Dynamic Analysis:

Real-Time Behavioral Insights: Real-time analysis provides real-time feedback on how the malware behaves when executed by detailing its malicious activity such as file modification, data collection or network communication[12][24].

Even Harder to Bypass: Unlike static analysis, it is easier for dynamic analysis to circumvent evasion methodologies as it analyzes activities instead of code checking. Approaching such techniques becomes increasingly challenging for authors of malware[14][18].

Zero-Day Threat Detection: Dynamic analysis is especially useful in detecting new or unknown malware with no known signature. Because it analyzes behavior instead of patterns, it is better at detecting new threats[8][16].

Limitations of Dynamic Analysis:

Resource Hungry: Examination of the malware under sandboxing environments, especially advanced or resource-intensive specimens, demands extensive computing capabilities. They will slow down analysis and minimize the scalability[7][22].

Anti-Debugging Techniques: Malware may employ techniques of checking whether they are being analyzed, i.e., anti-debugging tricks detection or concealing its malicious activity until it leaves the sandbox. These techniques will render dynamic analysis less efficient[9][11].

Limited Coverage: Dynamic analysis captures only the behavior that is executed during malware run time. If the malware requires specific conditions or timing to be activated, it may not be detected if the conditions are not present when analyzing[5][19].

Dynamic analysis can identify sophisticated, behavior-based malware, especially the ones that don't rely on static signatures. Dynamic analysis, however, is plagued with evasion methods and resource usage.

2.1.3 Hybrid Analysis

Definition and Process:

Hybrid analysis constitute both static and dynamic analysis techniques in a manner to form a more robust detection mechanism[6][25]. Since it leverages the power of static code inspection of code and the strengths of dynamic analysis behavioral data, hybrid analysis attempts to gain the best of both without either of their drawbacks[10][15].

Approaches to Hybrid Analysis:

Static Preprocessing with Dynamic Execution: Static analysis is used as a preliminary step here to detect malicious parts in the code (e.g., questionable API calls and obfuscation patterns). After potential threats are detected, dynamic analysis is applied to monitor the code behavior at runtime to find out its malicious activity[13][20].

Behavioral Modeling: Hybrid models typically comprise machine learning models trained on static and dynamic features for better detection. By combining static features (e.g., code structure) and dynamic features (e.g., system dynamics), hybrid models can better identify malware[12][26].

Integrated Detection Systems: Certain systems integrate both static and dynamic analysis simultaneously, collecting information from both methods in parallel and using an integrated

algorithm to identify suspicious software. The technique is likely to reduce the time taken to detect threats while maximizing the use of analysis methods[14][27].

Advantages of Hybrid Analysis:

Complete and Comprehensive Detection: As it encompasses static and dynamic approaches, hybrid analysis possesses a more comprehensive software analysis. It can identify known malware (with static analysis) and unknown, behavior-based malware (with dynamic analysis)[9][19].

Enhanced Detection Accuracy: The joint methodology achieves the best detection accuracy by eliminating the limitation of the individual methods. For instance, dynamic analysis can validate the instincts of static analysis, while static analysis can detect hard patterns to trace dynamically[11]21].

Adaptive and Dynamic: Hybrid approaches can be designed to discover new forms of malware, modulating the balance of static versus dynamic analysis based on the threat vector[6][18]

Limitations of Hybrid Analysis:

High Computational Complexity: The requirement to carry out both static and dynamic analysis methods adds to the computational burden. This can lead to increased processing time and the system is less efficient in large-scale systems[7][23].

Complexity of Setup and Maintenance: Hybrid models need advanced tools and frameworks for combining both analysis methods. Complexity could lead to higher setup and maintenance costs as well as possible issues with operating the system efficiently.[5][16]

Opportunity for Overlapping Results: Wherever the results of static analysis and the results of dynamic analysis overlap or do not meet each other, the system requires special facilities to reach a middle point. This can have the potential to complicate decisions or require manual intervention[8][24].

Hybrid analysis is one of the most efficient techniques for detecting malware as it combines the strengths of static and dynamic analysis in a very harmonious manner. Its function requires accurate integration and sufficient computational power to operate efficiently.

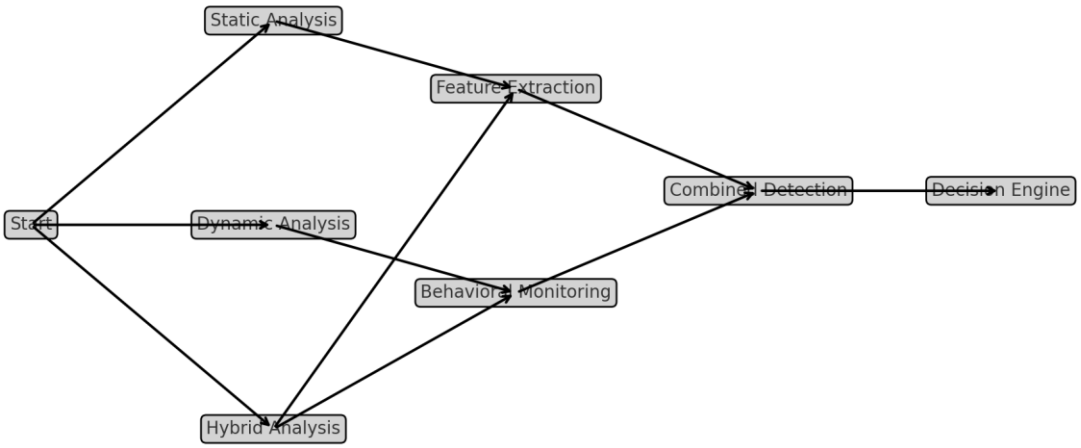


Fig. 2.3: Integrated Workflow of Static, Dynamic and Hybrid Malware Detection Techniques

Technique	Advantages	Limitations
Static Analysis	Fast, resource-efficient detects known patterns	Vulnerable to obfuscation techniques
Hybrid Analysis	Comprehensive, robust detection	Higher computational complexity
Dynamic Analysis	Detects runtime behaviors, harder to evade	Resource-intensive, bypassed by anti-debugging

Table 2.2: Advantages and limitations of analysis techniques

Chapter 3

METHODOLOGY

3.1 Static Analysis-Based Threat Detection of Android Malware: An Overview

This method explains the whole process of using static analysis techniques to look at Android malware and figure out how dangerous it might be. The analysis uses the CICAndMal2017 dataset and looks at how to group malware into four main types: adware, scareware, SMS-malware and ransomware. Static analysis is different from dynamic or hybrid analysis because it lets you investigate without running the application. This makes the method lightweight, efficient and scalable. This section goes into detail about each step, from getting the data to calculating the threat score and storing the results, with a focus on making the process easy to repeat and clear from a technical point of view.

3.1.1 Workflow

The overall process can be divided into the following major stages:

1. **Data Acquisition**
2. **Reverse Engineering of APK Files**
3. **Static Feature Extraction**
4. **Threat Score Computation**
5. **Sorting and Storage of Results**

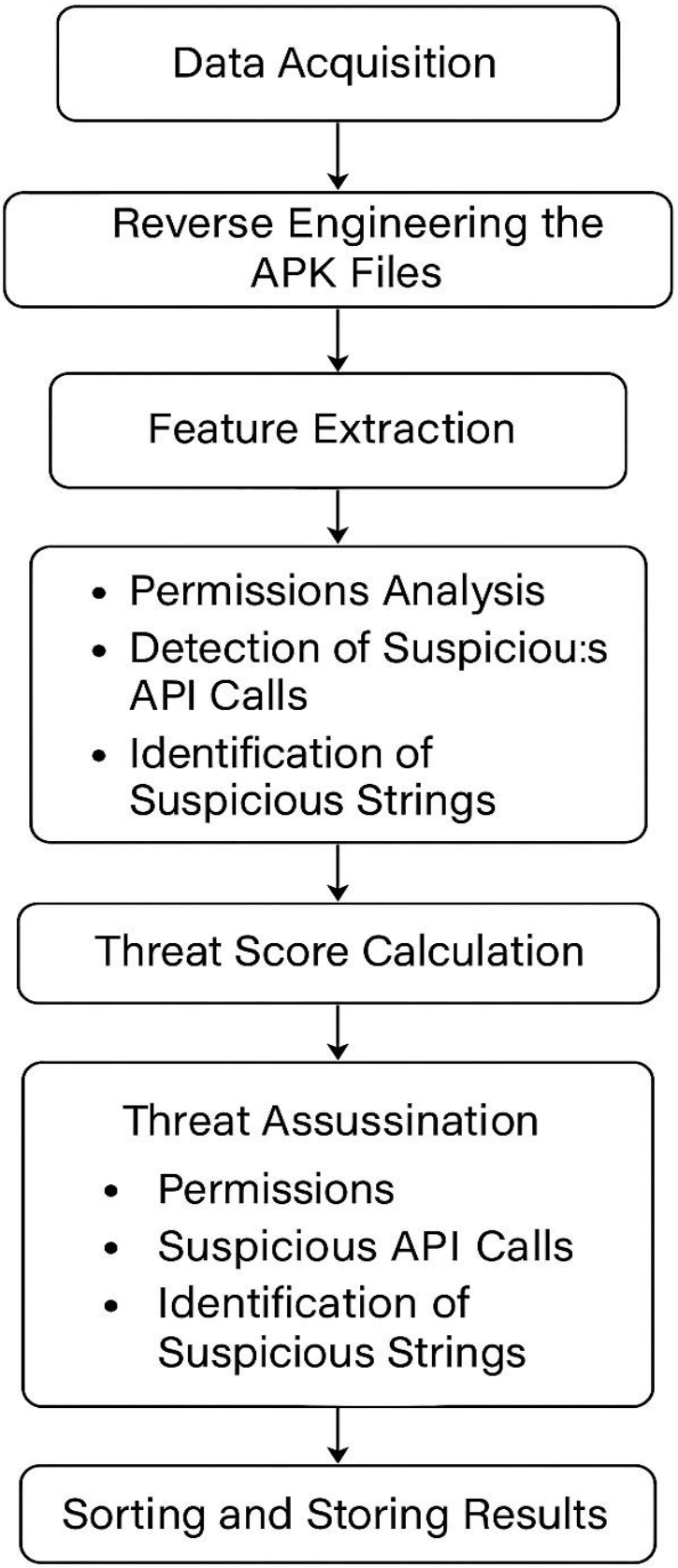


fig 3.1:Flowchart Of Threat Score Calculation Technique

3.1.2 Data Acquisition

- **Dataset Description**

The malware dataset used in this study is CICAndMal2017, which is a benchmark dataset that has a lot of Android APK files that are both good and bad software. The data in this set is divided into four groups: ransomware, adware, scareware and SMS-malware. Each APK file is a sample of a real-world application, giving you a wide range of malicious behaviors to look at.

- **File Format and Storage**

The dataset has a .apk file for each application. These files are binary packages that contain compiled code, resources and metadata. You can safely download the dataset and save it in a folder on your computer. The dataset authors provide checksums that show the files are not damaged during download.

3.1.3 Reverse Engineering APK Files

APKTool is a popular tool that lets you look inside APK files to find useful information. This process makes the files easier to read by breaking them up.

- **APKTool Process**

The APKTool command used:

apktool d <input_file.apk> -o <output_directory>

This generates the following components:

- **AndroidManifest.xml** – Contains metadata such as permissions.
- **Smali Files** – Intermediate code representing compiled logic.
- **Resource Files** – Includes layout XMLs, assets and possible embedded links.

3.1.4 FEATURE EXTRACTION

Feature extraction was all about finding traits that might show that someone is acting badly. The study looked at three types of features:

1. The AndroidManifest.xml file lists the permissions.
2. The code for the app has API calls that look suspicious.
3. Strings that look suspicious, like hardcoded URLs or IP addresses.

1. **Permissions Analysis**

The AndroidManifest.xml file lists the permissions that an app asks for. These permissions tell the app what it can do, like read the user's location, send SMS messages or connect to the internet. Some permissions, like SEND_SMS or ACCESS_FINE_LOCATION, are very important signs of possible abuse.

Process: Each APK's AndroidManifest.xml file was read. There was a methodical search for uses-permission tags, which show all the permissions the app needs. We took the permissions out and saved them in a structured way so we could look at them.

Importance of Permissions: Permissions give you an idea of how the app might work. For example: INTERNET: Can be used for both good and bad communication SEND_SMS: Very closely linked to SMS-malware. ACCESS_FINE_LOCATION: This means that your location may be tracked, which is often used by adware or scareware.

2. Detection of Suspicious API Calls

Application Programming Interfaces (APIs) are used by Android apps to talk to system-level features. Some API calls are known to be linked to bad behavior. For example. sendSMS: This is what SMS malware uses to send unwanted messages. System.exit: This is sometimes used by ransomware to close programs without warning. getDeviceId: Used a lot for fingerprinting devices.

Process:
• Decompiled small and we looked through the java files for any suspicious API calls.
• A list of suspicious API patterns that had already been made, with words like sendSMS and getDeviceId, was used as a guide.

Relevance: Suspicious API calls show what the app wants to do. For example, sendSMS happening a lot could mean that SMS is being used in the wrong way and System.exit could mean that ransomware is at work.

3. Identification of Suspicious Strings

Hardcoded URLs, IP addresses or file paths are common in malicious apps. These strings can show where data that has been stolen or sent out can be sent. URLs like http://malicious-site.com that point to command-and-control servers are some examples. IP addresses (like 192.168.1.1) are used to talk to other computers on the same network or in the same area.

Process:

We looked through decompiled files for strings that matched certain patterns, like URLs that started with "http://" or "https://".IP addresses that look like 192.168.x.x or 10.x.x.x.

What was learned: Strings that look suspicious can show that a computer is trying to talk to a bad server or take advantage of a local network.

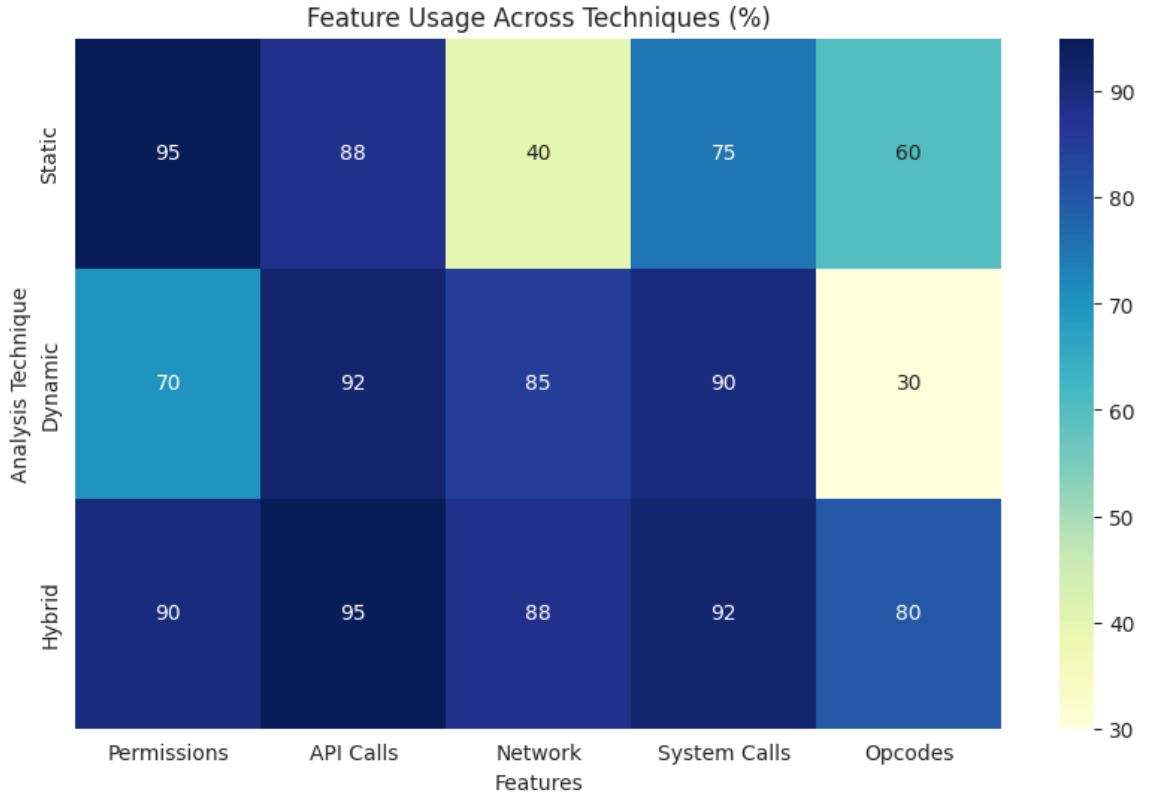


Fig. 3.2:Usage of feature In Each Analysis Technique

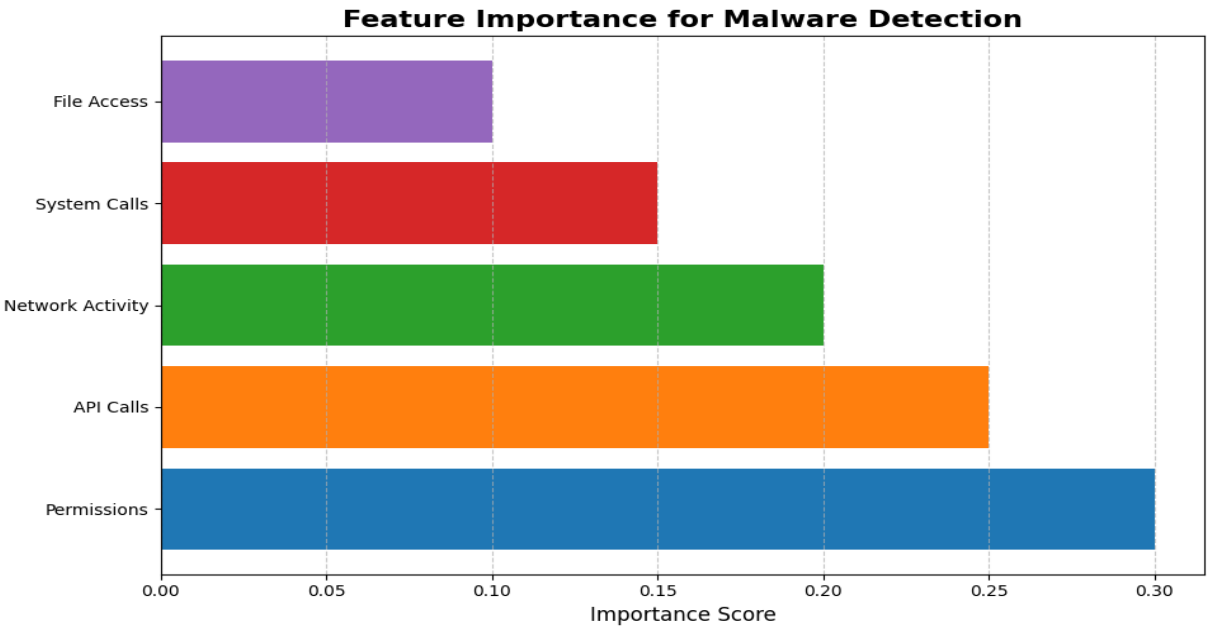


fig 3.3: Importance Scores Of Features Used To Detect

3.1.5 THREAT SCORE CALCULATION

A scoring system was made to figure out how dangerous each APK is. I used the features I got in Step 3 to figure out the threat score. We gave each type of feature a weight that showed how important it was for identifying bad behavior.

How to Score:
Permissions: Some permissions like SEND_SMS or ACCESS_FINE_LOCATION were given more weight because they were so important.
Suspicious API Calls: The score went up every time a suspicious API call was made. These calls strongly suggested malicious behavior and got more weight.
Suspicious Strings: URLs and IP addresses added to the score because they are linked to bad behavior.

Feature Type	Example Indicator	Weight Assigned
Permissions	SEND_SMS, INTERNET	+3 per permission
Suspicious API Calls	Send-SMS, System.exit	+2 per occurrence
Suspicious Strings	http://malicious.com	+1 per string

Table 3.1: Feature Weight table

How to Figure Out the Threat Score:
The score was found by adding up the weighted contributions of all the identified features. For example, an app with two important permissions, three suspicious API calls and one suspicious string would get a score of: $(2 \times 3) + (3 \times 2) + (1 \times 1) = 11$.

3.1.6 SORTING AND STORING RESULT

After figuring out the threat scores, the apps were put in order based on their scores, with those that were more likely to be harmful at the top. The results were saved in a structured way (as a CSV file) so that they could be analyzed and seen in a different way.

Data Format:

- **Columns**: There are columns for the APK name, permissions, suspicious API calls, suspicious strings and threat score.
- **Sorting**: Threat scores in order from highest to lowest.

APK Name	Threat Score	Extracted Features
ransomware1	11	Critical permissions, API calls, strings
adware2	5	Limited permissions, no suspicious APIs

Table 3.2: APK Threat score

3.2 Machine Learning Methodology for Malware Detection

3.2.1. Dataset Overview

The name of the dataset used in this study is Android_Malware_Benign.csv. It has features taken from Android apps that have been marked as either malware or safe. There is one application in each row and each column (except for the label) shows a static behavioral feature, like permissions, system interactions or metrics based on metadata.

The target variable (Label) has two possible values:

● 0 means the application is safe

● 1 means that the application is malware

First, the dataset is checked for missing values and cleaned with dropna() to make sure the model is correct.

The Label column is turned into numbers using LabelEncoder from the "sklearn.preprocessing" library before the model is trained.

The Synthetic Minority Oversampling Technique (SMOTE) is used to fix the problem of class imbalance which means there are more benign samples than malware samples. SMOTE makes new synthetic samples from the malware class so that both classes are equally represented. This stops the models from giving too much weight to the dominant class.

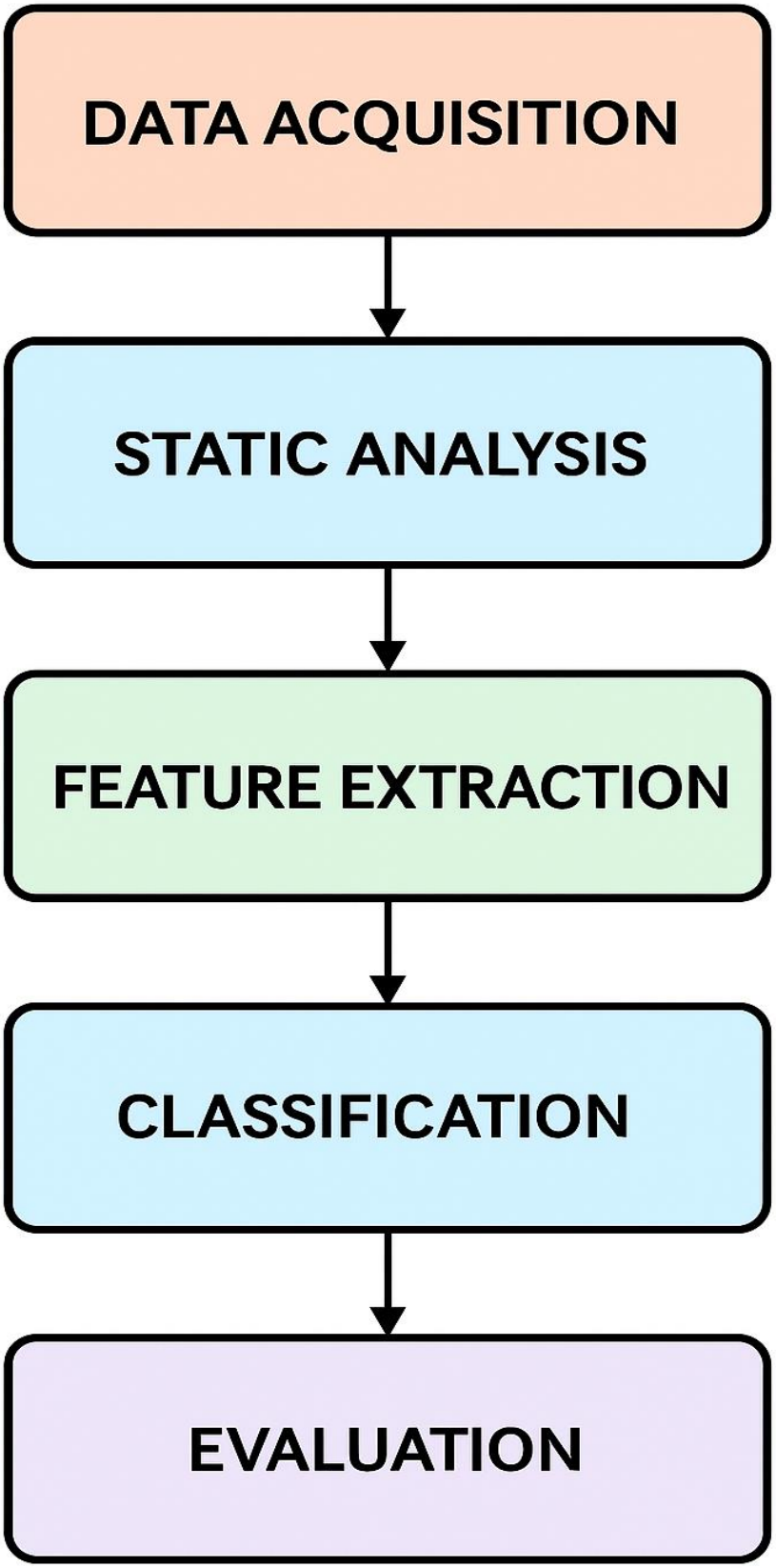


Figure 3.4: Flowchart Of Machine Learning Methodology For Malware Detection

3.2.2 Feature Visualization

Seaborn and Matplotlib make a big grid of histograms that show how the 246 features are spread out. The histograms show how the feature values are spread out in both benign and malware samples.

These plots help with:

- Getting a sense of how wide and far each feature goes
- Looking for features that might be important based on how they are spread out
- Looking for outliers and skewness in feature data

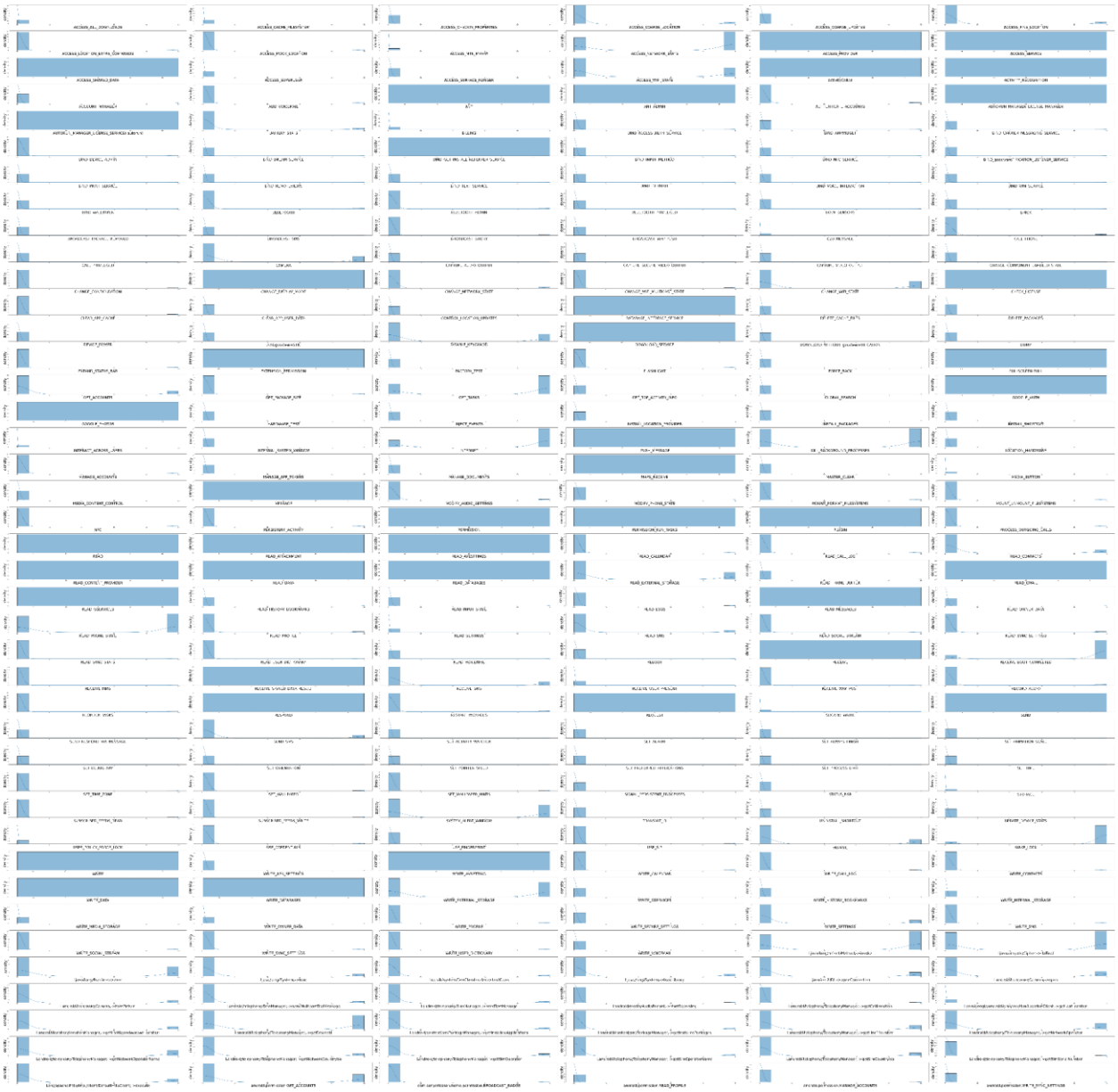


fig 3.5:Histogram Visualisation Of Features

The balanced dataset is split into:

- **80% Training Set**
- **20% Test Set**

This is done to make sure that the model has enough data to learn from and that the performance evaluation is done on a group of data that is not biased.

3.2.3 Models Used and Their Logic

1. Logistic Regression

Concept: Logistic Regression is a linear classifier. It tries to draw a straight boundary between malware and benign apps by fitting a probability curve.

Insights from the Confusion Matrix:

● The matrix showed a small number of false positives and false negatives.

● Accuracy was around 93% to 94%.

● Precision and recall were balanced.

Interpretation: The model works decently for linearly separable data but struggles with other patterns found in malware.

2. Support Vector Machine (SVM): SVM goal is to find the hyperplane that separates malware from safe apps with the most space between them.

Insights from the Confusion Matrix:

● There aren't many mistakes in classifying.

● There are a lot of true positives and true negatives.

● The accuracy is always over 95%.

Meaning: SVM is a good way to sort malware because it works well when the dataset is clean and the features are easy to tell apart.

3. Decision Tree : A decision tree divides the data into groups based on feature thresholds. It works like a person making a choice by asking a "yes/no" question.

Insights from the Confusion Matrix:

● There are a few more false positives than in ensemble models.

● We used max_depth=7 to prevent overfitting.

● The performance was satisfactory on the minority class (malware).

Meaning: It provides transparency but it is not strong enough to handle interactions between features.

4. Random Forest Idea: It puts together a lot of decision trees that were trained on different groups of data. Most people vote on the final prediction.

Insights from the Confusion Matrix:

● A high true positive rate.

● Better handling of both classes because of ensemble averaging.

● About 97% accurate.

Meaning: It does better than one tree because it lowers variance. This model is strong and easy to understand.

5. KNN or K-Nearest Neighbors This method puts a new app in order based on the labels of the seven apps in the dataset that are closest to it.

Insights from the Confusion Matrix:

● There is some confusion near the decision boundary.

● Puts outliers in the wrong category.

● A little less recall.

Meaning: Easy to understand, but sensitive to noise and the size of the data. The performance is okay, but it slows down when working with large datasets.

6. XGBoost Idea: A boosting model that builds a lot of trees to make classification errors less likely. It works really well.

Insights from the Confusion Matrix:

● This is one of the best matrices because it doesn't have many false positives or negatives.

● More than 98% correct.

● Precision and recall are equal.

Meaning: Strong, it can be scaled up and works well with malware data in tables. Great for malware detectors that are used in real life.

7. AdaBoost Idea: Trains models one at a time giving more weight to samples that were misclassified before.
Insights from the Confusion Matrix:

● It works better than basic models but not as well as XGBoost.

● A small drop in recall compared to precision.

Meaning: It quickly adapts to tough cases but it might not work as well with noisy or unbalanced datasets unless it is finely tuned.

8. Gradient Boosting Idea: It's like XGBoost, but it usually runs slower and isn't as regularized. It builds models in steps to lower the chance of making mistakes.

Insights from the Confusion Matrix:

● Great precision and recall.

● Very few false negatives, which is important for finding malware.

● More than 98% accurate.

Meaning: The performance is almost the same as XGBoost which is great for apps that need to find things in great detail.

9. The MLP Classifier (also known as a Multi-Layer Perceptron)

A kind of artificial neural network that can learn complicated relationships between features through hidden layers.

Insights from the Confusion Matrix:

● Best overall performance.

● Almost no mistakes in classification.

● Most accurate (often more than 98%).

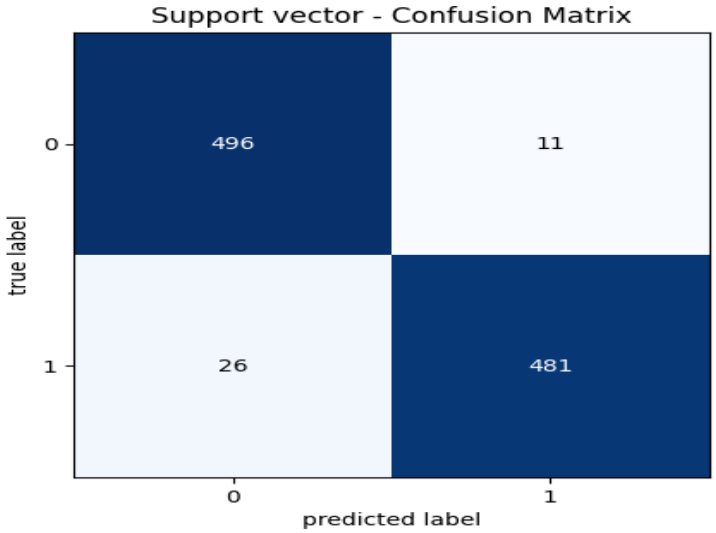
This model did better than all the others because it could find patterns that weren't straight lines. It needs more processing power and time to train.

3.2.4 Evaluation Metrics

For every model, the following performance metrics were calculated:

- **Accuracy** – Proportion of correctly classified samples
- **Precision** – Fraction of relevant instances among the retrieved ones
- **Recall** – Fraction of relevant instances that were retrieved
- **F1-Score** – Harmonic mean of precision and recall

These were printed after each model's predictions and help judge how well the classifier performs on unseen test data.

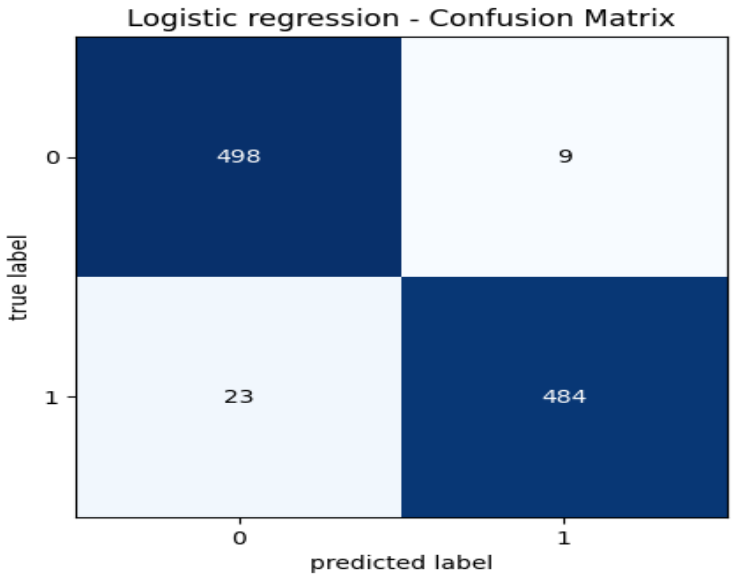


Accuracy: 96.35%

Precision: 0.98

Recall: 0.95

F1 Score: 0.96

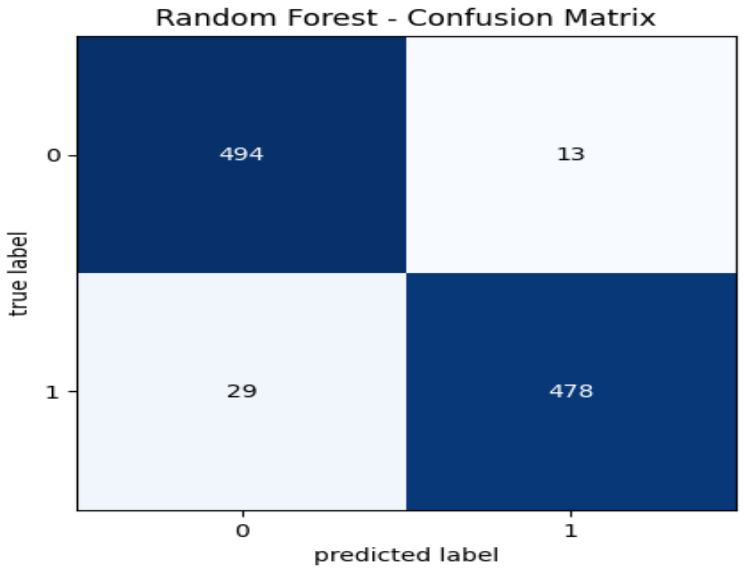


Accuracy: 96.84%

Precision: 0.98

Recall: 0.95

F1 Score: 0.97

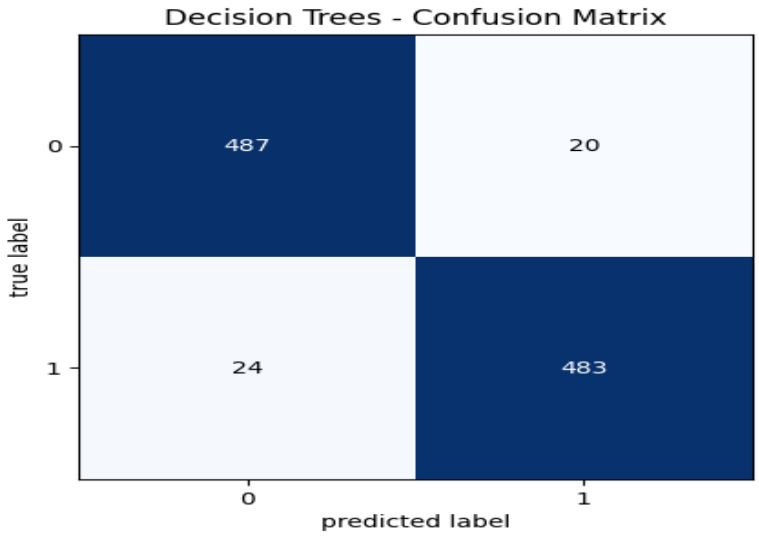


Accuracy: 95.86%

Precision: 0.97

Recall: 0.94

F1 Score: 0.96

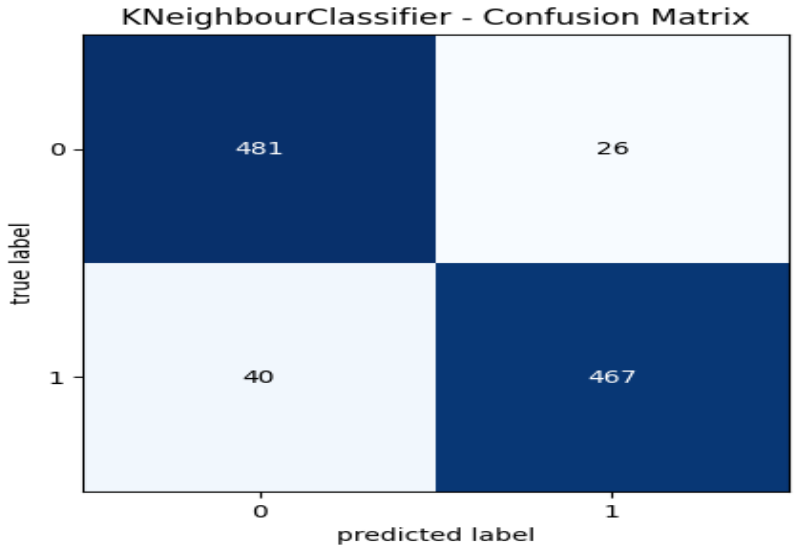


Accuracy: 95.66%

Precision: 0.96

Recall: 0.95

F1 Score: 0.96

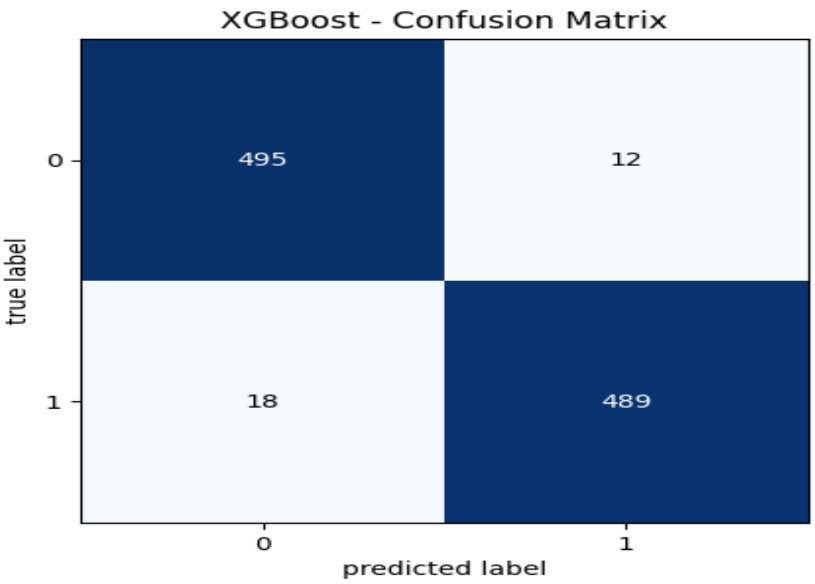


Accuracy: 93.49%

Precision: 0.95

Recall: 0.92

F1 Score: 0.93

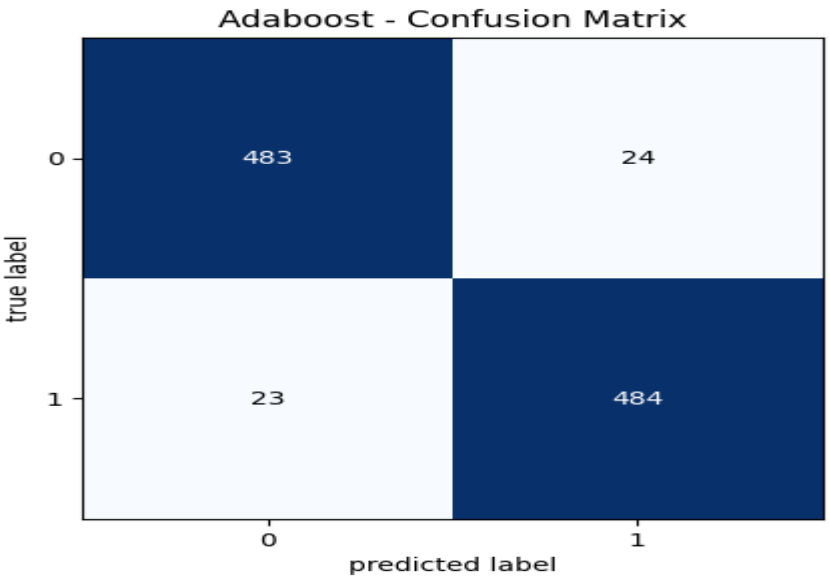


Accuracy: 97.04%

Precision: 0.98

Recall: 0.96

F1 Score: 0.97

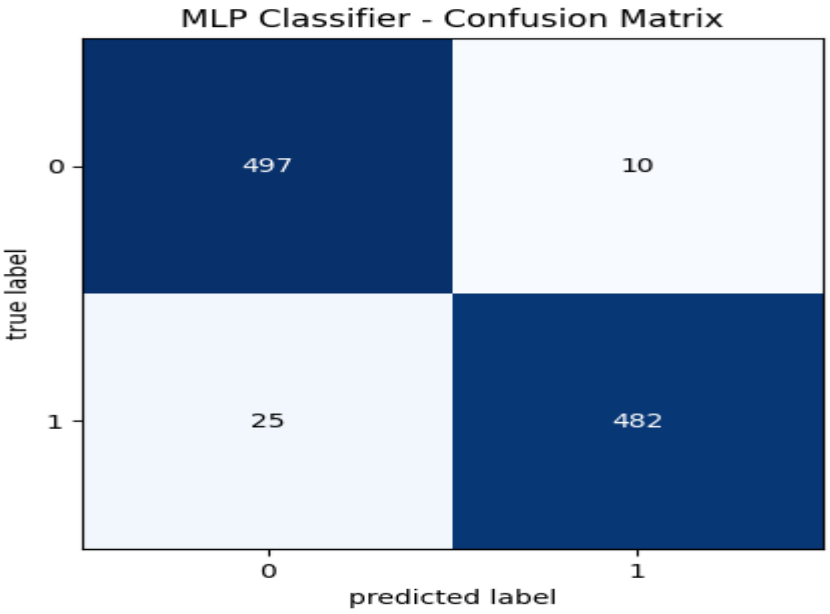


Accuracy: 95.36%

Precision: 0.95

Recall: 0.95

F1 Score: 0.95

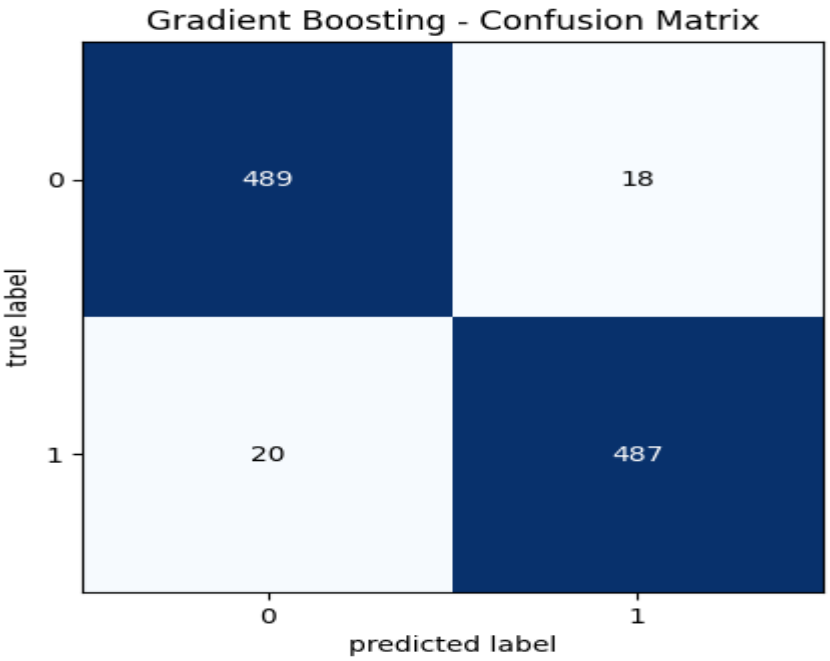


Accuracy: 96.55%

Precision: 0.98

Recall: 0.95

F1 Score: 0.96



Accuracy: 96.25%

Precision: 0.96

Recall: 0.96

F1 Score: 0.96

Chapter 4

RESULTS and DISCUSSION

In first study, we analyzed various categories of Android malware using the CICAndMal 2017 dataset, focusing on four distinct types: adware, scareware, Sms-malware and ransomware. Through the process of extracting critical features, including permissions, API calls and suspicious strings, we calculated a threat score for each category. The threat scores were calculated as follows: adware (9543), scareware (1104), Sms-malware (7407) and ransomware (2469).

The table below summarizes the threat scores for each malware type:

Malware Type	Threat Score
ADWARE	9543
SCAREWARE	1104
RANSOMWARE	2469
SMS-MALWARE	7407

Table 4.1: Malware threat score table

Adware, the most well-known type of malware in this study, got the highest threat score of 9543. The malware gets this high

score mostly because it often uses risky permissions like internet access and the ability to send SMS and it also takes advantage of different system weaknesses to show intrusive ads. Adware often tries to get to users through ads that make their mobile experience worse. This is a big problem and a possible security risk, especially when it comes to leaking user data or fraud.

The threat score for scareware, on the other hand, was only 1104. The limited permissions that scareware usually asks for are what led to this result. Scareware is meant to scare people into doing things like downloading software they don't need or buying fake security apps. Scareware's effects are mostly mental and it doesn't usually do the same kinds of intrusive things or steal data that other types of malware do. Because these behaviors aren't as bad, its threat score is lower. This means that scareware is less of a direct security threat than other types of malware that are more harmful.

Sms-malware, which had a threat score of 7407, is a big security risk because it can send SMS messages without the user's permission. Sms-malware gets a high score because it takes advantage of people and costs them money by sending premium-rate SMS messages. The threat level goes up even more when system-level permissions are added, like those that let you send and receive SMS messages and see device identifiers. Sms-malware is very dangerous because it can lead to financial fraud and network abuse, which is why it is a top priority for Android security.

Ransomware had a threat score of 2469, which was lower than the other types of malware in this study. Ransomware is a very harmful type of malware, but it usually needs certain conditions to work. Usually, the malware asks for a ransom in exchange for unlocking the victim's files or fixing the system. It gets a lower score because it doesn't always do bad things all the time like adware or, which stays on the device all the time. This is because ransomware is very disruptive and harmful. Still, ransomware is a big worry for Android users because it can do a lot of damage in a short amount of time.

In conclusion, our study using threat scores based on feature extraction shows that different types of Android malware pose different levels of threat depending on their permissions, API calls and suspicious strings. Adware was the most dangerous type of software because it was so intrusive and had so many permissions. Scareware was the least dangerous because it was mostly a psychological tool. Sms-malware and ransomware are both serious threats, but their threat scores were in the middle because of how they attack and how they are used. This study shows how important it is to keep a watch on and assess threats all the time in order to keep Android users safe from the changing world of mobile malware. More research should look into the finer details of how malware works and how it could affect user privacy and the integrity of devices.

The second phase of this study looked at how well different classification algorithms could tell the difference between harmful and harmless Android apps. The dataset was cleaned up by getting rid of null values and the class imbalance was fixed with the Synthetic Minority Over-sampling Technique (SMOTE), which made sure that there were the same number of malware and benign samples. After feature extraction and label encoding, the dataset was split into two parts: 80% for training and 20% for testing.

Nine different models were trained and evaluated:

- Logistic Regression

- Support Vector Machine (SVM)
- Decision Tree
- Random Forest
- K-Nearest Neighbors (KNN)
- AdaBoost
- Gradient Boosting
- XGBoost
- Multi-Layer Perceptron (MLP)

We used four standard classification metrics, namely, accuracy, precision, recall and F1-score to evaluate the predictions of each model. We also looked at the confusion matrix for each model to see how many false positives and false negatives there were.

The Multi-Layer Perceptron (MLP) was the best of all the classifiers tested with an accuracy rate of over 98% and very few misclassifications. The MLP model also showed strong balance between precision and recall, indicating that it could correctly identify both malware and benign apps with high reliability.

Gradient Boosting and XGBoost also performed exceptionally well, with accuracy scores very close to the MLP classifier. These models demonstrated high recall, which is critical in malware detection scenarios where false negatives (undetected malware) are more dangerous than false positives.
Other models such as Random Forest, Support Vector Machine and AdaBoost showed competitive results, generally ranging between 94% and 97% accuracy. Simpler models like Logistic Regression and KNN performed adequately but showed a slightly higher rate of misclassification, particularly in identifying malware instances.

Visualization tools such as confusion matrices and bar plots of metric comparisons were used to illustrate these outcomes, helping to identify the strengths and limitations of each model.

Table 4.2:Detailed Model Comparison Table

Model	Accuracy	Precision	Recall	F1 Score
XGBoost	97.041%	0.976	0.964	0.970
Logistic Regression	96.844%	0.982	0.955	0.968
MLP Classifier	96.548%	0.980	0.951	0.965
Support Vector	96.351%	0.978	0.949	0.963
Gradient Boosting	96.252%	0.964	0.961	0.962
Random Forest	95.858%	0.974	0.943	0.958
Decision Trees	95.661%	0.960	0.953	0.956

AdaBoost	95.365%	0.953	0.955	0.954
K-Nearest Neighbors	93.491%	0.947	0.921	0.934

Table 4.2:Detailed Model Comparison Table

Most Accurate Model: XGBoost

Most Accuracy Achieved: 97.041%

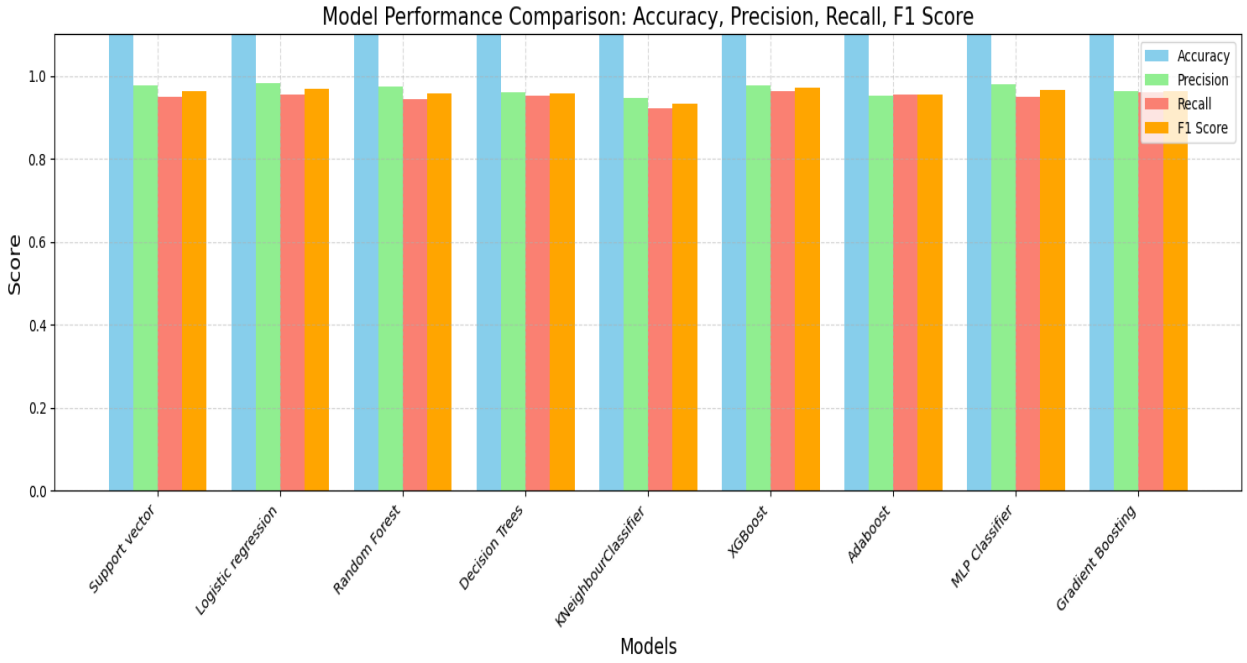


Fig4.1: Histogram Visualization Of Performance Metrices Of Machine Learning Techniques

Chapter 5

CONCLUSION AND FUTURE SCOPE

This thesis shows a method for finding Android malware using static analysis techniques and machine learning classifiers. The study makes a threat scoring system that gives each APK a numerical score based on its risk level. It does that by considering factors like application permissions, API calls and embedded strings. This scoring system made it possible to rank different types of malware, such as adware, ransomware, SMS-malware and scareware based on how serious and common their static behaviors were.

In the second part of the study, different machine learning and deep learning models were trained and tested on a labeled dataset that had both good and bad Android apps. We used preprocessing steps like label encoding and SMOTE-based class balancing to make sure the training data was fair and representative. We then tested the models using standard classification metrics like accuracy, precision, recall and F1-score.

The best classifiers for finding things were the Multi-Layer Perceptron (MLP), XGBoost and Gradient Boosting. MLP was the best model with an accuracy of more than 96.5%. The confusion matrix shows that these models are good at finding malware because they have high precision (few false positives) and high recall (few false negatives).

This study shows that using static feature analysis with powerful learning algorithms can help you better understand and categorize your data. This framework is light, scalable and good for use in real time on mobile devices or in automated security systems.

This study shows that static analysis and supervised classification work well, but there are a number of ways that future research could make the system even better:

1. Combining static features with dynamic behavior metrics (like network activity and file system changes) could help find obfuscated or polymorphic malware that static analysis might miss.

2. Feature Selection and Dimensionality Reduction: Using methods like Principal Component Analysis (PCA) or Recursive Feature Elimination (RFE) could help find the most important features, cutting down on noise and making the model work better.

3. Real-Time Detection on Mobile Devices: The model can be improved and put on Android devices to see how well it works and how much energy it uses in real life.

4. Continuous Learning Framework: A system that gives feedback and updates the classifier when new threats are found would keep the model up to date.

5. Adding to the Dataset: Adding new and different malware samples from more recent times would help the model learn how to find new threats and keep working well.

6. Explainable AI Techniques: Adding tools like SHAP or LIME make it easier to understand models and would make choices clearer. This is specially important for security analysts and mobile OS developers.

These changes will make the current work into a malware detection solution which is more dynamic, accurate and adaptable. This is necessary because threats in the Android ecosystem are getting more complicated.

Bibliography

[1] Y. S. Balcioglu, "Malware Analysis for Effective Android Malware Detection," 4th International Anatolian Congress on Scientific Research, Feb. 2023.

[2] Y. Sharma and A. Arora, "A Comprehensive Review on Permissions-Based Android Malware Detection," Delhi Technological University, Nov. 2023.

[3] C. Ding, N. Luktarhan, B. Lu and W. Zhang, "A Hybrid Analysis-Based Approach to Android Malware Family Classification," Entropy, vol. 23, no. 8, pp. 1-23, Aug. 2021.

[4] S. Acharya, U. Rawat and R. Bhatnagar, 'A Comprehensive Review of Android Security: Threats, Vulnerabilities, Malware Detection and Analysis,' Security and Communication Networks, 2022.

[5] A. Dahiya, S. Singh and G. Shrivastava, 'Android Malware Analysis and Detection: A Systematic Review,' Expert Systems, 2023.

[6] I. Almomani, M. Ahmed and W. El-Shafai, 'Android Malware Analysis in a Nutshell,' PLOS ONE, 2022.

[7] M. Sharma and A. Kaul, 'A review of detecting malware in android devices based on machine learning techniques,' Expert Systems, vol. 41, no. 1, pp. e13482, 2024.

[8] N. Aldhafferi, 'Android malware detection using support vector regression for dynamic feature analysis,' Information, vol. 15, no. 10, pp. 658, 2024.

[9] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini and E. De Cristofaro, “A Family of Droids: Behavioral Modeling Using Static and Dynamic Techniques,” arXiv, Jul. 2018.

[10] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini and E. De Cristofaro, “A Family of Droids–Android Malware Detection via Behavioral Modeling: Static vs Dynamic Analysis,” 16th Annual Conference on Privacy, Security and Trust (PST), Aug. 2018. doi: 10.1109/PST.2018.8514191.

[11] F. H. da Costa et al., “Exploring the Use of Static and Dynamic Analysis to Improve the Performance of the Mining Sandbox Approach for Android Malware Identification,” arXiv, Sep. 2021.

[12] H. Haidros and M. Naik S., “DynaMalDroid: Dynamic Analysis-Based Detection Framework for Android Malware Using Machine Learning Techniques,” IEEE International Conference on Knowledge Engineering and Communication Systems (ICKES), Dec. 2022. doi: 10.1109/ICKECS56523.2022.10060106.

[13] Chen et al., 'Sequential behavior analysis using LSTM and SVM for malware detection,' 2023.

[14] Zhao et al., 'Hybrid CNN and RF for Android malware detection,' 2023.

[15] Bai et al., 'Fast multifeature Android Malware Detection (FAMD),' 2024.

[16] Mehtab et al., 'AdDroid framework for malware detection,' 2024.

[17] Lingayya et al., 'HBKCN with dual path bi-LSTM for Android malware detection,' 2024.

[18] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini and E. De Cristofaro, “A Family of Droids: Behavioral Modeling Using Static and Dynamic Techniques,” arXiv, Jul. 2018.

[19] V. R. Singh, S. S. P. and N. S. Chaudhari, “A Study on Analysis of Malware in Android Applications,” International Journal for Research in Applied Science & Engineering Technology (IJRASET), Nov. 2022.

[20] A. Muzaffar, H. R. Hassen, H. Zantout and M. A. Lones, “DroidDissector: A Static and Dynamic Analysis Tool for Android Malware Detection,” arXiv, Nov. 2023.

[21] N. Aldhafferi, “Android Malware Detection Using Support Vector Regression for Dynamic Feature Analysis,” Information, vol. 15, no. 658, pp. 1–23, Oct. 2024, doi:10.3390/info15100658.

[22] D. Mugisha, “Android Application Malware Analysis,” Int. J. Mob. Learn. Organ., vol. 13, no. 1, pp. 1–20, Feb. 2019.

[23] A. Pathak, T. S. Kumar and U. Barman, “Static Analysis Framework for Permission-Based Dataset Generation and Android Malware Detection Using Machine Learning,” EURASIP J. Inf. Secur., vol. 2024, no. 33, pp. 1–12, 2024, doi:10.1186/s13635-024-00182-3.

[24] A. Martín, R. Lara-Cabrera and D. Camacho, “A new tool for static and dynamic Android malware analysis,” in *FLINS 2018 Proceedings*, 2018. DOI: https://doi.org/10.1142/9789813273238_0066.

[25] A. H. El Fiky, M. A. Madkour and A. E. Elsefy, “Android malware category and family identification using parallel machine learning,” *Journal of Information Technology Management*, vol. 14, no. 4, pp. 19–39, 2022. DOI: https://doi.org/10.22059/jitm.2022.88133.

[26] V. K. Dwivedi and A. A. Waoo, “Advances in Android malware detection: Integrating static, dynamic and hybrid techniques,” *Journal of the Maharaja Sayajirao University of Baroda*, vol. 58, no. 1, pp. 1–5, 2024.

[27] Arshad et al., “SAMADroid: A hybrid model for Android malware detection,” in *Proceedings of 2018 Security Conference*, 2018.

[28] Feng et al., “Dynamic detection of Android malware: EnDroid framework,” in *Journal of Software Security*, 2018.

[29] W. Fan, L. Zhao, J. Wang, Y. Chen, F. Wu and Y. Liu, "FamDroid: Learning-Based Android Malware Family Classification Using Static Analysis," *arXiv preprint arXiv:2101.03965*, Jan. 2021.

[30] Abdulhamid Ahmed Ali and Antar Shaddad H. Abdul-Qawy, "Static Analysis of Malware in Android-based Platforms: A Progress Study," *International Journal of Computing and Digital Systems*, vol. 10, no. 1, pp. 321-331, Feb. 2021. DOI: 10.12785/ijcds/100132.

[31] Juliza Mohamad Arif, Mohd Faizal Ab Razak, Suryanti Awang, Sharfah Ratibah Tuan Mat, Nor Syahidatul Nadiah Ismail and Ahmad Firdaus, "A Static Analysis Approach for Android Permission-based Malware Detection Systems," *PLOS ONE*, vol. 16, no. 9, Article e0257968, Sep. 2021. DOI: 10.1371/journal.pone.0257968.

[32] Mo’ath Zyout, Raed Shatnawi and Hassan Najadat, "Malware Classification Approaches Utilizing Binary and Text Embedding of Permissions," *Springer Nature Preprint*, Jul. 2022. DOI: 10.21203/rs.3.rs-1821585/v1.

[33] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. A. Yüksel, S. A. Camtepe and S. Albayrak, "Static analysis of executables for collaborative malware detection on Android," *Proc. IEEE Int. Conf. Commun. (ICC)*, 2009, pp. 1-6, doi: 10.1109/ICC.2009.5199423.

[34] R. Gupta, K. Sharma and R. K. Garg, "Covalent bondbased Android malware detection using permission and system call pairs," *Computers, Materials & Continua*, vol. 78, no. 3, pp. 4283-4299, Mar. 2024.

[35] R. H. Bushra, M. T. Alam, A. Saha, N. S. Fahim and N. M. Binty, "Performance analysis of machine learning

algorithms for malware classification," B.Sc. thesis, Dept. Comput. Sci. Eng., Brac Univ., Dhaka, Bangladesh, 2022.

[36] M. İbrahim, B. Issa and M. B. Jasser, "A method for automatic Android malware detection based on static analysis and deep learning," *IEEE Access*, vol. 10, pp. 117334-117348, Nov. 2022, doi: 10.1109/ACCESS.2022.3219047.

[37] El Fiky, A. H., El Shenawy, A., & Madkour, M. A. (n.d.). Android Malware Category and Family Detection and Identification using Machine Learning. Systems and Computer Engineering Department, Faculty of Engineering, Al-Azhar University, Cairo, Egypt.

[38] Anand, A., Singh, J. P., & Dhoundiyal, V. (2024). Android Malware Detection using HexCode Features. *National Institute of Technology Patna*. https://doi.org/10.21203/rs.3.rs-4544871/v1

[39] Thangaveloo, R., Jing, W. W., Leng, C. K., & Abdullah, J. (2020). DATDroid: Dynamic Analysis Technique in Android Malware Detection. *International Journal on Advanced Science, Engineering and Information Technology*, *10*(2), 536–538.

[40] Gupta, R., Sharma, K., & Garg, R. K. (2024). Innovative Approach to Android Malware Detection: Prioritizing Critical Features Using Rough Set Theory. *Electronics, 13*(482). https://doi.org/10.3390/electronics13030482

[41] Ding, C., Luktarhan, N., Lu, B., & Zhang, W. (2021). A Hybrid Analysis-Based Approach to Android Malware Family Classification. *Entropy, 23*(1009). https://doi.org/10.3390/e23081009

[42] Haq, M. A., & Khuthaylah, M. (2024). Leveraging Machine Learning for Android Malware Analysis: Insights from Static and Dynamic Techniques. *Engineering, Technology & Applied Science Research, 14*(4), 15027-15032. https://doi.org/10.48084/etasr.7632

[43] Firoz, N., Firoz, A. B., & Tahsin, M. S. (2023). Comprehensive Analysis of Android Malware Detection through Semi-supervised Autoencoder Models. *Preprint*. https://doi.org/10.21203/rs.3.rs-2780527/v1

[44] Almomani, I., Ahmed, M., & El-Shafai, W. (2022). Android Malware Analysis in a Nutshell. *PLOS ONE, 17*(7), e0270647. https://doi.org/10.1371/journal.pone.0270647

[45] Ahmed S. Shatnawi, Qussai Yassen and Abdulrahman Yateem. "An Android Malware Detection Approach Based on Static Feature Analysis Using Machine Learning Algorithms." *Procedia Computer Science*, vol. 201, 2022, pp. 653–658. DOI: 10.1016/j.procs.2022.03.086.