Enhancing Android Security with Explainable AI: A Hybrid Malware Detection Framework Using Static and Dynamic Features

by

Md. Azaz Ahamed (20301369)
Mitisha Tasnim Rahman (20301327)
Syeda Nahin Farzana (24341274)
Sad Al Sazid (24241343)
Sheikh Zahid Hassan Sameer (21101307)

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
April 2025

Declaration

It is hereby declared that

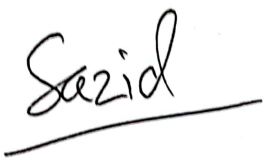
1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Mitisha Tasnim Rahman
20301327

Md. Azaz Ahamed
20301369

Syeda Nahin Farzana
24341274



Sad Al Sazid
24241343

Sheikh Zahid Hassan Sameer
21101307

Approval

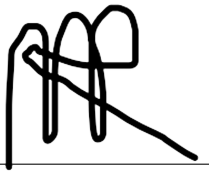
The thesis titled "Enhancing Android Security with Explainable AI: A Hybrid Malware Detection Framework Using Static and Dynamic Features" submitted by

1. Md. Azaz Ahamed (20301369)
2. Mitisha Tasnim Rahman (20301327)
3. Syeda Nahin Farzana (24341274)
4. Sad Al Sazid (24241343)
5. Sheikh Zahid Hassan Sameer (21101307)

Of Spring, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on April 15, 2025.

Examining Committee:

Supervisor:
(Member)



Annajiat Alim Rasel
Senior Lecturer
Department of Computer Science and Engineering
Brac University

Co-Supervisor:
(Member)

Md. Sabbir Ahmed
Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

The proposed paper is an amazing endeavor. Furthermore, the members hereby and genuinely affirm that this was done based on the findings of our thorough investigation. All resources used are properly noted and credited in the report. This research work, in whole or in part, has never been submitted to another university or institution for degree conferring or for any other purpose.

Abstract

Mobile phones have become a major target of cyberattacks because of their vast number of users and open-source nature. Among these many threats, Adware and Trojans are frequent and serious threats that undermine user privacy and all-around system reliability. We investigate the effectiveness of machine learning in classifying malware samples into either Adware or Trojan. We discuss a rigorous data preprocessing pipeline to balance the class distribution by synthetic oversampling, filtering features based on their statistical characteristics, and scaling numeric variables to have zero mean and unit variance. Multiple model evaluation metrics-precision, recall, F1 score, balanced accuracy, and ROC AUC-indicate our models can tell the differences between these classes with Random Forest achieving 96% accuracy in both features. Of particular note are ensemble classifiers, which achieve high performance and demonstrate how integrating various decision boundaries can yield superior results without excessive computational overhead. These findings speak volumes on the practical importance of robust preprocessing, balanced datasets, and interpretable methods for feature importance in order to delve deeper into malicious behaviors. The study gives ample emphasis on evidence-based model selection and shows how such advanced strategies can fortify cybersecurity measures in today's ever-changing world of mobile technologies

Keywords: Android Malware, Adware, Trojan, Machine Learning, Data Preprocessing, Mobile Security

Dedication

This work is dedicated to everyone who inspired and supported me throughout the journey of research and development. I owe my deepest gratitude to my family, whose unwavering belief in my endeavors and continuous encouragement have always been my greatest source of strength and motivation.

I am especially thankful for the guidance and wisdom offered by my mentors and teachers, whose confidence in my potential propelled my academic and professional growth. Their insights and support have shaped not only this research but also my commitment to continuous learning and innovation.

Acknowledgement

To begin, we would want to offer all appreciation to the Almighty, because of whom our thesis was finished without any significant interruptions. Then we want to acknowledge our supervisor without his valuable guidance and feedback we wouldn't be able to come this far.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

During the last decade, there has been a phenomenal proliferation of smartphones, which have brought unparalleled convenience to individual life, organizations, and societies as a whole. Amongst all, Android smartphones hold a considerable share of the mobile market globally. With millions of apps available for download from official stores like the Google Play Store and from various third-party app stores, smartphones have taken over as important and indispensable tools for communication, entertainment, e-commerce, healthcare, and so on. However, the same pervasiveness and capabilities of these gadgets have attracted cybercriminals to them. Especially, the ecosystem of Android has been more and more prone to malicious applications that exploit the vulnerabilities for user data compromise. With the increase in cyber threats, it has raised an alarming need for increasing cybersecurity in mobile devices.

The top issues that face mobile cybersecurity are headed by an unparalleled upsurge in malware affecting Android devices. Estimates have it that there are about 12,000 new instances of malware being generated daily for Android alone; hence, cybersecurity people get a really big headache [7]. These malware applications can be capable of executing a variety of malicious actions: from stealing sensitive user data, disabling key functionalities, and encrypting files for ransom to spreading across other devices through the network. Traditional signature-based detection methods, based on predefined patterns of malware, are insufficient to handle the dynamic nature of current threats. Advanced obfuscation techniques and the use of zero-day vulnerabilities mean that attackers can often successfully evade effective risk detection or mitigation using conventional methods [16].

Android malware is not an issue that solely concerns technology; it also affects user privacy and data security on a larger digital scale [2]. An individual user can suffer from stolen private photos, messages, financial information, and location data in case of a malware attack. In an organizational network, compromised devices can lead to data breaches, financial losses, and reputational damage. Moreover, with mobile devices increasingly being used with critical infrastructure, such as healthcare and banking systems, the potential for disastrous consequences is rising. These increasing risks raise the bar to call for more sophisticated, scalable, and adaptable methodologies for malware detection and prevention [3].

Recent advances in machine learning and artificial intelligence have pointed to new

directions for solving Android malware challenges. Unlike traditional methods, machine learning models scan through massive datasets of applications to identify complicated patterns and generalize their understanding to find previously unseen threats. Some of these models use static features, such as permissions and API calls, while others depend on runtime behaviors and system logs for holistic malware detection. Deep learning, has gained particular prominence in this domain, enabling superior performance in classifying malware into different categories, while the detection of zero-day attacks is also made possible.

1.2 Problem Statement

The increase of mobile technologies has been complemented by growth in cyber threats that are targeted towards mobile devices; and a large fraction targets the operating system of Android. The open nature of Android, complemented by high market share, presents it as a prime target for cyber criminals looking to exploit vulnerabilities. It probably represents the most urgent problem in this sphere: the rapid proliferation of malware for Android devices grew into a complex and highly resistant threat. This can be perpetrated through seemingly innocuous apps that use social engineering, exploit kits, or unauthorized application stores. An issue like this is not only technical but also an important concern for user privacy and data security in general.

The malware complexity on Android has increased exponentially, where cyber criminals use advanced methods of code obfuscation, encryption, and polymorphism techniques to defeat detection [6]. These are very hard for traditional mechanisms to keep up with, such as signature-based and heuristic methods. Conventional approaches commonly fail at finding new or modified malware strains, and have particularly fallen short about zero-day threats that take advantage of previously unknown vulnerabilities. This very limitation has left millions of users of Android in the wide open to risks including unauthorized access to data, financial theft, and disruption of the systems [1].

The diversity of Android malware amplifies the challenge of detection and classification. Malware regarding Android appears in various forms such as adware, ransomware, trojans, and spyware-all having different behaviors and attack vectors. Many malware samples also fall into families of certain characteristics but differ so slightly that their classification and response are not accurate. Furthermore, the extremely large number of benign apps and dynamic nature of app updates contribute to a high risk of false positives, undermining user trust in cybersecurity solutions.

The urgency of this problem is underscored by the sheer scale of Android malware incidents reported globally. Considering that an estimated 12,000 new malware samples emerge daily, existing detection systems pose scalability issues. This automatically overwhelms the existing security infrastructures and delays the identification of critical vulnerabilities, leaving devices for longer periods without protection. These challenges demand further understanding of the characteristics of malware, scalable methods for analyzing large datasets, and new ways to enable detection with increased accuracy without compromising user experience or device performance. However, building defenses remains daunting without robust datasets and a comprehensive taxonomy of malware families.

1.3 Scope of the study

The study will dwell on the critical and ever-expanding area of mobile cybersecurity, especially in Android smartphones, which have become the most widely used mobile operating systems globally. This study also considers the vulnerabilities inherent in mobile devices, making them easy targets for cybercriminals. This openness of Android, combined with the broad spectrum of its application ecosystem, creates a hostile environment where malicious actors can manipulate these gaps for security reasons. Exploring this problem from an underlying perspective, the research tried to show how such vulnerabilities are used in launching attacks, disrupting operations, and stealing sensitive user information.

Behaviors and strategies adopted by malware of Android compromise devices and their security, too, have been discussed in the study. It examines the different modes of malware distribution, such as unofficial app stores, phishing attacks, and malicious advertisements, besides manipulating app permissions, system processes, and communication networks to achieve the desired result. The research will investigate all aspects of threats that face Android users, from adware and ransomware to more sophisticated malware types such as trojans and spyware.

The broader implications of malware on Android for individual users, organizations, and critical infrastructure are also discussed here. The attack of malware does not stop with affecting the performance of a device and further inconveniencing users, but it could lead to very serious results: financial theft, identity fraud, data breaches, and even disruptions in essential services. This research considers these implications and, as such, demands holistic solutions that address not only the technical aspects of malware detection but also the socio-economic risks of mobile cybersecurity threats.

The study will, therefore, review the current practices and methodologies in mobile cybersecurity. The research critically reviewed various tools and techniques to identify shortcomings in the traditional methods of tackling problems, hence highlighting areas of further innovation: detecting obfuscated malware, managing large-scale threats, and tactics that evolve almost every other day. The research also takes into consideration aspects related to the role of users' behavior and awareness in reducing risks, with a view to informed decision-making when interacting with mobile devices and applications.

It therefore expects to contribute to the general area of cybersecurity, with implications and recommendations that might be useful for scholars, practitioners, and policy analysts. It advocates collective efforts towards mobile device security hardening through the development of advanced detection mechanisms, enhancing security protocols, and offering user-centered education programs. This paper therefore scopes the boundaries within which the study will be performed and pinpoints the most pertinent challenges in malware detection and prevention in Android; it thus sets the foundation for future work in encouraging continuous exploration within the mobile cybersecurity domain.

1.4 Significance of the Study

In the modern connected world, mobile phones have become important for communication, business, education, and even entertainment. While the smartphone has turned into a container of sensitive personal and corporate information, its security

has become a prime issue. On the other hand, the recent surge in cyberattacks targeting mobile platforms-especially Android-is seriously threatening the privacy, safety, and well-being of users. This research is important because it addresses the pressing need for the understanding, mitigation, and response to ever-changing challenges in mobile cybersecurity, especially regarding the detection and classification of malware.

Android malware has emerged as one of the most chronic and pervasive cybersecurity threats facing the modern world. A malware attack can cause an awful lot of damage, such as financial theft, identity fraud, hijacking of your device for jamming, and surveillance. This study is important since it adds to the increasing knowledge about Android malware nature and behavior, helping researchers and cybersecurity practitioners build more efficient strategies for such threats. This research aims to try and demarcate the attack vectors of malware, exploiting the vulnerabilities it uses to proliferate, with a view to opening the way to harder defenses that can prevent mobile users from harm.

Beyond the immediate threats posed by malware, the broader implications of mobile cybersecurity are just as important. With the increased use of mobile devices in important systems such as health, banking, and smart cities, the impact of an outbreak of malware will be immense. For instance, the successful breach of a single mobile device integrated with other networks could be all that is needed to create major shutdowns in those networks. This study is crucial in that it tries to make mobile systems more resilient to these contexts, thus securing not only individual but also general digital exposure.

Another important dimension of this study regards its role in informing policy and industry practices. This research consequently has the potential to aid governments, businesses, and consumers in understanding the dynamic landscape of mobile cybersecurity threats and necessary countermeasures. The mapping of key vulnerabilities and proffering of evidence-based solutions through this study can help shape the development of regulations, standards, and best practices that foster a safer and more secure mobile environment. It also highlights the cooperation that will be required among all stakeholders-technology developers, cybersecurity experts, and regulatory bodies-in arriving at a common approach to respond to mobile threats.

This research is important to further the field of cybersecurity research and innovation. By indicating the shortcomings of the existing detection methods and highlighting the need for more adaptive, scalable, and intelligent solutions, it inspires future developments in the domain. This welcomes novel ideas in techniques for the detection and mitigation of malware while improving the knowledge base on challenges related to mobile cybersecurity. Finally, this research contributes to a safer digital future by arming the cybersecurity community with knowledge and tools to deal with the ever-increasing threat that Android malware poses.

1.5 Research Objectives

The primary objectives of this study are as follows:

- **Identify Key Vulnerabilities:** Analyze the major vulnerabilities in Android devices that malware exploits to compromise system security and user privacy.

- **Understand Malware Behaviors:** Examine the common patterns, behaviors, and attack vectors used by Android malware to target devices and users.
- **Explore Advanced Detection Methods:** Investigate contemporary techniques and approaches for detecting and classifying Android malware, focusing on scalable and adaptable solutions.
- **Evaluate the Effectiveness of Current Measures:** Critically assess the limitations of existing cybersecurity practices and solutions for protecting Android devices.
- **Enhance Cybersecurity Awareness:** Highlight the importance of user education and best practices to reduce the risks associated with Android malware.
- **Recommend Future Directions:** Provide actionable insights and recommendations for researchers, practitioners, and policymakers to strengthen mobile device security.

1.6 Contribution

The key contributions of this research are outlined below:

- **Hybrid Dataset Design and Robust Evaluation:** A dual-dataset strategy was developed by integrating static and dynamic features to support a comprehensive cybersecurity framework for mobile devices. This design enabled robust evaluation of model performance across different feature types, allowing for a deeper understanding of how various data dimensions contribute to threat detection.
- **Incorporation of Explainable AI (XAI):** SHAP (SHapley Additive exPlanations) analysis was employed to interpret the predictions of the machine learning models. This allowed for identification of the most influential features contributing to classification decisions, promoting model transparency and interpretability.
- **Model Refinement through Interpretability:** Although SHAP does not directly affect accuracy, the insights it provided supported informed decisions in feature selection and hyperparameter tuning. This interpretability contributed to a more refined and effective model development process.
- **High Classification Accuracy:** The final models achieved a classification accuracy of at least 98 percent, surpassing the performance of existing approaches applied to the same datasets.
- **Enhanced Vulnerability Detection:** The results indicate a notable improvement in detecting security vulnerabilities in mobile devices, underscoring the effectiveness of the proposed hybrid and interpretable machine learning framework.

Chapter 2

Literature Review

With the constant rise of mobile phone users, mobile devices have become an attractive target for cyber criminals as this small device carries all our sensitive information. It is important to ensure the privacy of all mobile devices. In this paper, we are focusing on a hybrid approach which is a combination of multiple detection techniques to leverage their strengths and mitigate their weaknesses. It integrates static analysis, dynamic analysis, and machine learning methods to create a more robust and comprehensive detection system of different detection and defensive strategies to produce a strong and complete security system. It decreases false positives and negatives by improving the detection rate using machine learning, It understands complex patterns and correlations among the features. Dynamic analysis detects real-time dangerous activity whereas statics analysis swiftly filters programs.

The paper titled "Android malware detection:A literature review" by Sabbah,Taweel and Zein from Birzeit University offers a thorough overview of current android malware detection methods.The study highlights the increasing use of mobile apps in important areas like health, logistics and banking, which makes android devices more vulnerable to attacks (Sabbah et al., 2023)[18].By analyzing existing research, the authors identify three main approaches to identify android malware: analyzing network traffic, examining internal interactions and checking app permissions.Additionally, the review examines different analysis techniques such as static, dynamic and hybrid methods, stressing the importance of identifying vulnerabilities early to prevent malware attacks.It discusses the challenges in detecting android malware, pointing out the high frequency of attacks on android devices and the lack of malware detection for iOS devices.The authors used a dataset in their study that covers 12 years of android history (2008-2020) and contains hybrid features from the android malware dataset.It was gathered using emulators and real devices and includes more than 70,000 malware and 72,000 benign apps.In summary, this literature review is a valuable tool for cyber security researchers and professionals,providing valuable insights into the various methods and technologies used to detect and prevent android malware threats.It not only summarizes existing information but also suggests future research paths to improve malware detection mechanisms in the constantly changing field of mobile devices.

A paper titled 'A Comprehensive study of Malware Detection in Android operating systems' published in Asian Journal of Research in Computer Science illustrated

different malware detection techniques of android phones, especially focusing on Machine Learning based groupings which is a research chapter of artificial intelligence(Hamdi et al., 2021) [10]. One of the key characteristics of android is that each application is absolutely apartheid from other applications. This is why the accuracy of machine learning algorithms is considered to be high in detecting malwares. The moment a malware-corrupted operating system is installed, users are susceptible to several threats like- losing database, stealing personal information, prying users, controlling devices remotely and ransomware which causes economic loss. Machine learning models are initially structured in a way so that it can deliver algorithms which permit the computer to learn on its own demanding stats and probability. If we deeply analyze the wide range of classifiers for machine learning, we may find- KNN(K-nearest neighbor), SVM(Support Vector Machine), Decision Tree, Neural Networks, Naive bias and many more which can be effective to recognize malwares. To recognize the malwares in terms of static and dynamic analysis, machine learning classification methods give higher precision than others. The paper also highlighted two deep learning methods- DexCNN and DexCRNN for training the preprocessed segments. Here DexCNN can reach up to 93.4% accuracy and DexCRNN gives accuracy of 95.8%. KNN, Decision Tree and Random forest are the three discrete categorizations for autonomous learning which gives less expensive storage representation and also accelerates the learning procedure.

The study "AndroPack" is a novel hybrid technique that finds malware kept hidden in packed applications for Android by combining static analysis and dynamic analysis. This system extracts features from an Android application and trains a machine-learning model to identify the malicious application (M & Chandran, 2024)[20]. The author of this study used the KronoDroid dataset which is a hybrid dataset containing static and dynamic features of Android spyware. It has a large number of malware samples and benign applications for training and testing and also it provides a wealth of information including permissions, intents, system calls and other metadata. The output of their study found after implementing the hybrid methods accuracy is 95% in detecting packed Android malware. This increased accuracy highlights the usefulness of combining static and dynamic analysis with ensemble learning. The first limitation is relying on a single dataset, which could not accurately capture the vast diversity of malware variations. On the other hand, the article discussed the importance for packed malware detection, but it doesn't provide an in-depth review of certain packing methods and how these things are handled. Besides, it requires continuous updates and adaptations to stay effective because of the dynamic behavior of Android malware and the potential for rapid evolution which can pose challenges.

Another paper illustrated an ensemble learning technique employing three machine learning algorithms- Least square support vector machine (LS-SVM), Regularized random vector functional link neural network (RRVFLN) and Kernel extreme learning machine optimization (KELM) for android malware detection (Alamro et al., 2023)[15]. This Automated Android Malware Detection applying Optimal Ensemble Learning Approach for Cybersecurity i.e. AAMD-OELAC method executes the preprocessing of data at the initial phase. The Hunter-prey optimization, shortly known as HPO algorithm has been also used for better malware detection rate. The

amalgamation of these algorithms is highly effective to recognize harmful patterns and demeanor in android apps. In the data preprocessing stage, the vital features have been selected, developing a data frame where columns indicate APIs and rows imply the types of files(benign or malware). The AAMD-OELAC technique has been experimented on Andro-Autopsy dataset where the number of benign classes is 5000 and malware is 2500 among 7500 samples. This technique is able to categorize the two classes correctly on multiple epochs and also gives the higher accuracy, precision, recall, F-score and MCC compared to other models.

The authors exhibited a distinct phonological approach named Apposcopy to diagnose malwares that rob users private information(Feng et al., 2014)[4]. Taint analysis and signature-based detectors are the two general ways for instinctively recognizing malevolent applications. Since there are several benign apps which require access to delicate data for operating their updated components, we cannot classify every app that disclose user details as malware. As a result , taint analysis is unable to spontaneously differentiate between benign apps and malicious apps. On the other hand, signature detectors categorize an application as virus if it finds similarities with typical expressions of malware. But the only drawback is that these malicious characteristics need to be renovated regularly as alternate mutations of the identical malware groups become apparent. Apposcopy encompasses static analysis along with signature specification language that authorizes two semantic attributes of android app- control flow and data flow. This study used an interpreted version of GoldDream which is a known malware family. One of the main features of these malwares is that they generate a recipient upon text messages or phone calls. Whenever any of these incidents occur, the malwares incorporates their background services which include- transferring user information (android's IMEI number, subscription id) to a secluded server. In order to correctly identify a malware as a member of this family, lines of code are being written which comprises all the signature behavior of GoldDream. Applying all the three predicates (Component type, Control flow, Data flow) and matching the subqueries with the provided specifications, Apposcopy is able to determine the malware of that specific class.

Dhalaria and Gandotra (2021) introduced a hybrid route that obtains multiple features employing static and dynamic analysis for inspecting, recognizing and categorizing android malwares [8]. Firstly, they created their own two datasets. The first one is created for detecting 'benign' and 'malware' relating to binary classification and the second one is for detecting malware families that share similar characteristics and source codes referring to multiclass classification. The number of static and dynamic features is 352 and 323 respectively for both datasets. Then an information gain method has been implemented to remove outliers, missing values and unimportant features from the datasets. Selecting 110 static features and 99 dynamic features in the first dataset and 47 static features and 35 dynamic features in the second dataset, several machine learning classifiers have been applied for detecting malwares.Considering only static features, the precision provided by RF in dataset-1 is 96.5% and 97.01% if only dynamic features are intended alone. But it shows the maximum accuracy 98.53% in the hybrid approach which indicates that better result and performance can be achieved in this analysis compared to scenarios where static or dynamic analyses are counted alone.

The research paper titled "Understanding research trends in Android Malware Research" presents a comprehensive analysis of research trends in the field of android malware using latent semantic analysis (LSA)(J. Singh et al., 2021)[11].The study begins by highlighting android's leading position in the smartphone market and the rise in malware attacks targeting android devices,stressing the need for strong security measures.It emphasizes that researchers are focusing on understanding different types of malware attacks,how they spread and ways to stop them, using previous studies in this field.The paper presents a new method using Latent Semantic Analysis(LSA) to study android malware literature,identify key research areas and trends.It analyzes 843 articles published between 2009 and 2019.The process involves preparing the articles,Creating a matrix of documents and terms, and using Singular Value Decomposition (SVD) to simplify the data and reveal hidden concepts.The research shows five main areas and twenty trends in android malware,helping us understand the changing threats of android devices.The study highlights that the static analysis is very important for android security and will remain so.It also point out the need for quicker and better ways for android malware detection and harmful applications.In summary,this paper presents how important information modeling techniques,especially LSA are for discovering key research areas for android malware research.It provides insights for researchers,cybersecurity experts and smartphone users helping them to understand the changing challenges of android security and protecting android devices against malware threats.

Khan and Sharma (2023) explained that Android's open-source nature and global popularity have made it an attractive target for cybercriminals, leading to the proliferation of SMS malware and riskware attacks[17]. These attacks compromise user privacy and data security, often involving malicious URLs sent via text messages. Traditional malware detection methods include static and dynamic analyses, with static analysis examining code without execution and dynamic analysis monitoring runtime behavior. Hybrid detection and machine learning approaches have been developed to enhance detection, leveraging advanced machine learning techniques. Ensemble learning, combining models like Random Forest, Gradient Boosting, Support Vector Machines, and Decision Trees, has proven effective in aggregating diverse model predictions. The CICMalDroid 2020 dataset was used to train multiple machine learning algorithms, with the Random Forest classifier demonstrating superior performance and the Naive Bayes algorithm showing reliability. A proposed methodology integrates a machine learning-trained chip for real-time detection of riskware and SMS malware, providing immediate responses to potential threats. However, the dynamic nature of cyber threats necessitates continuous updates and robust cybersecurity practices.

The rise of Android as the dominant mobile operating system has led to the development of Android applications, which have attracted malicious activities. This has necessitated the development of robust malware detection mechanisms to safeguard users and their data(Abhishek et al., 2024) [14]. Traditional malware detection methods are basically static analysis techniques, which use a signature-based algorithm but this method is struggling with zero-day vulnerabilities and novel malware variants. To break this tradition, experts are attempting to use dynamic analysis methods, which is worked by monitoring the runtime behavior of applications, which

is more effective way than static analysis to address malicious activities. In dynamic analysis, Machine learning plays a crucial role for enhancing malware detection, because it analyzes the data from history to identify patterns of applications malicious activities. A study investigated several ML models to find the efficiency of dynamic analysis of malware detection. In the result of the investigation, Random Forest model output is the highest accurate, which is around 91.22%. The methodology of this investigation is the accurate data preprocessing to handle inconsistencies, redundancies, and missing values, and used performance metrics, apart that to get the value of accuracy, precision, and recall which will sharply evaluate the model's capabilities. For future research, researchers can reduce the limitations of ensemble learning methods and add advanced feature selection techniques to improve the detection accuracy and resilience against evolving threats.

Sharma and Kapoor (2022) discussed the advancement of mobile malware detection techniques which is an advanced hybrid approach of the combination of machine learning and optimization algorithms to change the traditional method of static and dynamic analysis [12]. The Adaptive Neuro-Fuzzy Inference System (ANFIS) is a hybrid intelligent system which includes neural networks and fuzzy logic principles to handle uncertainties and imprecisions in real-world data. Optimization algorithms are so complicated for enhancing the performance of ANFIS, as traditional methods like Particle Swarm Optimization (PSO) have limitations. To identify these, new algorithms have been proposed which are nature-inspired metaheuristic optimization algorithms, like the Firefly Algorithm (FA) which is integrated with ANFIS to create a hybrid model (ANFIS-FA) for mobile malware detection. The ANFIS-FA model is spread into other hybrid optimization approaches to achieving higher R^2 values and lower RMSE values in both training and testing phases for increasing the chances of accuracy and reliability. The study concluded that the integration of ANFIS with the Firefly Algorithm provides a robust framework for mobile malware detection, showing significant improvements in prediction accuracy and error reduction. The improvement of ANFIS with the Firefly Algorithm shows a robust framework for mobile malware detection, which is significantly improved in the prediction of accuracy and error reduction. For future research directions, this paper includes refining input variables sets to enhance the model's performance and addressing issues related to the selection and elimination of unnecessary variables. Exploring other nature-inspired optimization algorithms and their integration with ANFIS could provide further improvements in detecting sophisticated mobile malware[12].

The authors of the study suggested a feature selection method to overcome the issues related to permission models in Android operating systems and develop an effective malware detection framework called PermDroid(Mahindru et al.,2024)[21]. The study focuses on creating a dynamic analysis-based methodology that uses permission, API calls, app ratings, and user download counts as features. They gathered data from 70000 .apk apps from Google play store, more than 300000 .apk files from third-party application stores like Softonic, Android Authority, CNET. 70000 apps that are malware infected from SandDroid. The authors achieved high malware detection rates using DNN. It reaches 98.8% accuracy and F-measure close to 0.94. Their framework successfully detected both known and unknown malware

families. It achieved better detection rates than many antivirus scanners. For future work, the author aims to enhance the models ability to detect zero-day attacks by training their model with more diverse datasets. Additionally, future studies can also concentrate on figuring out how few permissions are required for effective malware evaluation.

Huang et al. (2024)[19] have proposed MicDet to solve the issue of unauthorized use of mobile device's microphones. Their main focus is detecting unauthorized microphone accessibility by harmful softwares running on Android tablets, and smartphones. Whether the software is running in the background or foreground they aim to detect it and alert users to such threats. They gathered data in two methods. First method is normal usage data collection where the users were asked to perform normal operations on different voice-enabled apps, capturing the time stamps of request and response for each app. Second method is illegal access data collection where they have developed an app "EarSpy" that could access the microphone without user authorization. The authors installed this app on many devices to collect illegal access data, user interactions were recorded over 30 minutes. The MicDet strategy achieved high accuracy in most of the cases, over 90% for detecting unauthorized microphone access in different devices. Volunteers who took part in a survey believed the program practical, with more than 96% prepared to use it on their devices. One limitation of the study is each app's usual microphone access is initiated by a user touch event, which may not be true for some apps with voice wake-up capabilities. Secondly, the injection attack approach used to create illegal applications may not work with all apps, particularly those with security features. In future works the authors aim to improve injection attack methods to work on more applications and bypass existing notification systems without being detected. They also aim to investigate more stealthy attack strategies and build easy solutions to automatically prevent unwanted access without user interaction. They are interested in analyzing user behavior to develop more subtle illegal access methods, as well as expand the scheme to include applications with voice wake-up functions.

R. Feng et al. (2021)[9] suggested MobiTive which is an implementation-sensitive Android malware detection solution created as an automatically installed application on mobile devices. Their main goal is to provide a mobile-friendly Android malware detection system that can challenge the current server-based solution. Their main concern was balanced performance and accuracy to provide a realistic usable solution for the end customer. The researchers of this paper gathered 70000 android applications and 29010 malware samples including apps from Drebin, VirusShare, Genome project, Drebin, KuaDet and Pwnzen infotech. While removing duplicates and unsuccessful occurrences,18,000 benign and 18,000 chose malicious samples to test. These samples were categorized as (80%) training, (10%) validation, and (10%) testing sets. The author's main goal in the future is to enhance MobiTive's ability to identify new malware families by adding more effective features with a minimal performance impact. They plan on experimenting with adaptive algorithms that mix both static and dynamic analytical methods to increase zero-day threat detection rates.

Fei Tong and Zheng Yan's work "A hybrid approach of mobile malware in Android"

addresses restrictively to regular Android devices for which over 97% of all the mobile attacks are aimed at [5] by employing machine learning algorithms on both standard and recovered static features. The study underscores the critical importance of next-generation approaches to detect malware, highlighting that cellphones are ubiquitous and often contain sensitive personal information. The growth in app usage of mobile apps, and vulnerabilities in the operating systems on which they run has brought diffused focus to security upgrade waiting for months. The first group of malware detection technologies both fall into what are known as static analysis and dynamic analysis techniques. Static analysis: which refers to the investigation of application code without actually running it, meaning that this is a suitable tool for finding statically known malware and as well identification of threats by their signature. Although this method can be applied to program analysis without consuming resources at run-time, and it serves as a basis for examining many types of malware (e.g., obfuscated code or encrypted strings), there are some limitations when dissecting advanced malwares that rely on complex obfuscation methods (pure text-string encryption) or medium-dependent behavior coordination layers providing no signature within the static codes. Whereas dynamic analysis evaluates the behavior of an app while it is running. This one monitors system calls, network traffic etc that may suggest you are under an attack. If 0day vulnerabilities and also runtime threats that static method might overlook, this comes efficient as well in discovering vulnerable aspects of a product by dynamic analysis.

Ullah et al. (2022) and Pak et al [13] who proposed a novel technique of exploiting call graphs with image visualization features for malware detection/classification working at an accuracy level of 99.27%. The broad availability of assets in the android environment makes it one major target for malware and demands advanced detection techniques with a challenging goal. Traditional malware detection techniques involve a system of dynamic analysis — observing the behavior of an application while it is running (also static analysis—examining code without executing). Of course, both the techniques have their own cons as well — dynamic analysis will cost too many resources and static analysis is cleverly obfuscated. According to our knowledge, machine learning has speculation related power in malware detection as far as they can be massive dataset based trend observers. To model app behavior and detect potential fraudulent activity, graph-based methods like API-Call Graphs (ACGs) are utilized as well as Control Flow Graphs (CFGs). These methods capture the structure of Android apps, and are capable of identifying whether an app is benign or malicious.

Chapter 3

The Dataset

3.1 Overview of the Dataset

The basic ingredient of a sound and representative dataset is what has been in most malware detection studies. In fact, from which any decent development and effective evaluation could begin for any machine learning model. Considering an Android security perspective, large-scale datasets are allowed not only to explore the wide range of diversified samples but also a wide range of kinds of malicious behaviors with diversified benign applications. This dataset was developed in co-operation with the Canadian Centre for Cyber Security and the Canadian Institute for Cybersecurity; it will be a good basis for training and benchmarking advanced approaches for detection. It attempts to solve some of the most important problems that researchers face, such as poor sample diversity, outdated malicious applications, and poor coverage of real-world attack vectors.
The dataset provided is balanced, having an equal number of benign and malicious samples. There are 200,000 benign applications sourced from trusted sources and 200,000 malware samples. Thus, this study can introduce equal numbers of both benign and malicious apps, that is, 400,000 samples in total, to the analysis in order to reduce class imbalance bias that may skew the performance of machine learning algorithms toward those classes that have higher population counts. These datasets are balanced, and this goes a long way toward helping the classifiers derived thereof not to overlook the smaller classes-less common malware families-or overfit to the dominant categories.
Malware categorization involves 14 prominent malware categories that cut across a wide range of real-world threats. These include, among others, adware, backdoor, file infector, ransomware, and trojans-even those more specific ones like trojan-banker and trojan-spy. It covers a wide range of malicious behaviors, thus exposing the detection algorithms to various attack strategies, from stealthy data exfiltration to outright disruption or extortion. The dataset aggregates 191 well-known malware families, enabling finer-grained analyses of how specific families evolve, inherit traits, or make use of distinct attack vectors. Based on the characteristics of the features, two parquet files of dataset have been created that contain static and dynamic features respectively. Every static feature that has been derived from the original dataset contains values. The features include Package, Services, Activities, Meta-Data, Permissions, Intent Actions, Receivers and Providers. The runtime changes and behavioral signs of malwares are listed in the dynamic features combining to the

fundamental six categories- Network, API, Process, Memory, Logcat and Battery. API features describe data that the two applications use while communicating with one another by requests and responses.
The reasons for making this sure with the reliability in its labeling include its use of multi-engined scanning through the VirusTotal against a 70% consensus threshold. This, in context, means that an application can thus be tagged malicious only when, under live conditions, more than 70 percent of the capable AVs used by Virus Total flag it off as malicious. This threshold significantly reduces the chances of labeling any benign samples as malicious, hence improving the quality of ground truth. The applications labeled as benign have been screened in such a way that the possibility of false negatives is minimal. A tight labeling strategy provides not only more confidence among the researchers for the dataset but also minimizes further experimentation as now, the manual verification or cleansing for the mislabeled samples is not that cumbersome.
The dataset from a taxonomic point of view gives a fine-grained classification of malware families. Each malicious sample has a category which represents its principal malicious functionality; for example, surveillance, credential theft, device hijacking. This high-level categorization is complemented, in the case of each malware family, by information such as shared characteristics ranging from APIs they hook to permissions they request, that support researchers in finding similarities among different samples. Indeed, this grain of observation is special, being highly necessary for finding minor variations within a malware family or observing the temporal evolution of an attack.
The dataset has been curated to also represent recent and legacy threats. Inclusion of older samples gives researchers a chance to test the backward compatibility and robustness of their detection methods, so that tools do not miss already established families. The data also covers the current threat landscape, represented by the inclusion of newer, hence more sophisticated, strains of malware. This breadth and depth in coverage, especially in this respect, enhance the usefulness for a dataset in longitudinal studies where one may want to compare several generations of performances of malicious code detection strategies.
This dataset represents an important point related to the standardization of Android malware research by providing such a large and well-balanced collection carefully labeled. Matching this with the rich coverage of both categories and families, along with strict labeling, is a good fit to diversity and the fast-paced evolution that characterize the Android threat landscape. It allows rich analytics ranging from malware family clustering to assessment of zero-day detection methods.

3.2 Data Preprocessing

Here is the data processing technique that we done:

- **Filtering and Label Selection:** The code begins by filtering the dataset to include only two specific categories, “Adware” and “Trojan.” This step is crucial for narrowing down the analysis to a binary classification problem. By subsetting `df` using `df['Label'].isin(['Adware', 'Trojan'])`, the dataset is reduced to samples that represent these two distinct malware types. This focused approach simplifies later stages of feature engineering and

model training, while also offering clearer performance metrics when comparing classification results.

- **Statistical Feature Screening:** Before moving on to more complex transformations, the code conducts a one-sample t-test (`ttest_1samp`) against zero for each numeric column. Columns that exhibit a mean significantly different from zero (i.e., having a p-value less than 0.05) are retained for subsequent steps. This procedure acts as a basic feature-filtering method, potentially removing columns that do not offer meaningful variance relative to zero. Although unconventional compared to traditional feature-selection techniques, this approach helps reduce dimensionality and may mitigate the impact of noisy or irrelevant features.

- **Label Encoding:** To facilitate binary classification, the labels in the "`Label`" column are mapped to numerical values. Specifically, "Adware" is mapped to 0, and "Trojan" is mapped to 1. This numerical encoding allows machine learning models to process the target variable effectively and distinguishes the two malware types in all subsequent training steps.

- **Addressing Class Imbalance with SMOTE:** An inherent issue in malware datasets is the imbalance between different classes, which can bias models toward the majority class. To counter this, the `SMOTE` (Synthetic Minority Over-sampling Technique) algorithm is employed. SMOTE synthesizes new minority-class samples by interpolating existing samples, thus enhancing the dataset's balance without merely duplicating entries. By improving the representation of the minority class, SMOTE can lead to better generalization and reduce the risk of the model overlooking the Trojan category (if it happens to be the minority class in the initial dataset).

- **Feature Scaling with MinMaxScaler:** After balancing the dataset, numeric features are scaled to the range $[0, 1]$ using `MinMaxScaler`. This transformation normalizes all features to a uniform scale, preventing any single feature with a large numerical range from dominating the model training process. Scaling is particularly beneficial for distance-based or gradient-based algorithms, as it ensures more stable and efficient convergence, ultimately contributing to more robust classification performance.

3.3 Data Splitting for Model Training

- **Rationale for Train–Test Split:** To effectively assess model performance on unseen data and avoid overfitting, the dataset is divided into training and testing subsets. This separation enables unbiased evaluation of the model's predictive capabilities, ensuring that metrics reflect performance on data not used during the training process.

- **Implementation Details:** In the code, data splitting is implemented using the `train_test_split` function from the `scikit-learn` library. The parameter `test_size=0.4` designates 40% of the data for testing, leaving the remaining 60% for training. This ratio was chosen to provide a substantial

training set for model learning, while also retaining a sufficiently large test set for reliable performance evaluation.

- **Random State and Reproducibility:** By specifying `random_state=42`, the split is made reproducible; anyone re-running the code will obtain the same division of training and testing samples. This consistency is vital for comparing results across different experiments or parameter configurations.

- **Effects on Class Distribution:** Even though `SMOTE` was employed to balance the classes before splitting, it is still important to confirm that each subset retains a proportional distribution of the two classes (`Adware` and `Trojan`). Post-split checks ensure that the training and test subsets reflect similar class ratios, thereby reducing sampling bias.

- **Alternatives and Extensions:** While a single train–test split is a widely accepted approach, additional techniques such as cross-validation or stratified sampling could further enhance model evaluation. For instance, *Stratified K-Fold Cross-Validation* can provide more reliable metrics by cycling through multiple train–test partitions. However, for clarity and computational efficiency, this experiment retains the simpler 60/40 train–test split scheme.

3.4 Ethical Considerations

Any research into the development of malware detection systems will involve dealing with data that is somewhat sensitive, perhaps even malicious code samples originating from real-world attacks. Researchers should make sure that their work cannot harm others by accident, either through spreading the malicious software themselves or by letting others access this malicious software. Any dataset involving Android malware needs to be properly and securely stored with access strictly provided to authorized people only. Furthermore, all information that may lead to the identification of individuals, organizations, or specific victims has to be anonymized and sanitized upon results disclosure or publication. This confidentiality is of utmost importance in cases when malware samples are collected from either public or private repositories, which might contain identifying metadata or unique device-specific artifacts.
The major ethical principle driving cybersecurity research involves the concept of "do no harm." Therefore, malware detection and analysis should be performed within an isolated environment, such as a secure sandbox or a dedicated virtual machine, to avoid accidental infection of networks or systems. In that respect, the practices at play should ensure that the analysis and investigations on the procedures undertaken by attackers never spread malware further or render extra systems compromised. This includes warning other research collaborators of these associated dangers and preparing/training on any involved precautions to avert an unwanted malicious release into the wild. Safety and respect regarding privacy may relate to privacy and analyzing application data:. Any processing or storage of PII by researchers should not violate the legal or regulatory frameworks, including the GDPR. When the data of users forms part of logs collected or metadata of apps, researchers should ensure the data anonymization technique is strong enough to remove or mask personally identifiable sensitive details. Further, strong transparency in what data is collected,

how that data has been analyzed, and for what purpose it is put to will build trust with stakeholders and ensure this research follows international privacy and best practices in the protection of data.

Responsible disclosure is another ethic that takes form in malware detection research. If during a study new attack vectors, previously unknown vulnerabilities, or zero-day exploits are found, researchers are ethically bound to immediately share these with the affected vendors or relevant authorities. By following coordinated vulnerability disclosure protocols, the researcher can help ensure that security patches get developed and deployed before malicious actors can take advantage of the newly revealed vulnerabilities. This cooperative approach not only furthers the collective security posture but also is a show of respect to the greater community with which secure Android ecosystems are entrusted.

Chapter 4

Methodology

4.1 Overview of Proposed Approach

In this work, our methodology revolves around creating a streamlined pipeline for detecting "Adware" and "Trojan" classes in a real-world Android malware dataset. Below is an overview of the major steps involved:

1. **Initial Data Ingestion and Filtering:** The dataset is first loaded from a `.parquet` file, after which the code selectively focuses on two specific malware classes: "Adware" and "Trojan." This filtering step effectively limits the scope to a binary classification problem, simplifying the subsequent feature engineering and evaluation process.

2. **Statistical Feature Screening:** Before deeper preprocessing, a one-sample t-test is applied to each numeric column to check whether its mean differs significantly from zero (`p-value` ¡ 0.05). Columns passing this criterion are retained for further analysis, functioning as an initial form of feature selection. Although somewhat unconventional, this strategy can eliminate low-variance or irrelevant features and reduce dimensionality in an efficient manner.

3. **Label Encoding and Balancing:**

   - **Encoding:** The labels are mapped to numerical values, with "Adware" represented by 0 and "Trojan" by 1. This conversion from categorical labels to integers is crucial for compatibility with popular machine learning algorithms.
   - **SMOTE for Imbalance:** To address potential class imbalance, the *Synthetic Minority Over-sampling Technique* (SMOTE) is employed. SMOTE synthesizes new, plausible data points for the under-represented category by interpolating between existing samples, thereby ensuring an equal representation of both classes. This step mitigates the risk of model bias towards the majority class.

4. **Feature Scaling:** Following the balancing step, each numeric feature is normalized into the range $[0, 1]$ using the `MinMaxScaler`. Scaling ensures that no single feature dominates the training process due to differing numerical ranges, which is particularly beneficial for distance-based and gradient-based models.

5. **Train–Test Split:** In order to evaluate model performance on unseen data, the dataset is split into training (60%) and testing (40%) subsets. This separation is performed randomly with a fixed seed (`random_state=42`) to ensure reproducibility. The model is trained on the 60% subset, while the remaining 40% is reserved strictly for performance evaluation.

6. **Model Training and Evaluation:**

   - **Algorithms:** Four classification models are explored: Logistic Regression, Random Forest, XGBoost, and CatBoost. These models were chosen to capture a range of learning paradigms (linear methods, ensemble techniques, and gradient boosting).
   - **Metric Computation:** Once trained, each model generates predictions on the test set. Key metrics include Accuracy, Precision, Recall, F1-score, and ROC AUC. Balanced Accuracy is also reported to address any lingering imbalance issues.
   - **Visualization and Interpretability:** Confusion matrices illustrate correct versus incorrect predictions, while ROC Curves highlight the trade-off between True Positive Rate and False Positive Rate. For deeper insight into model decisions, SHAP (SHapley Additive exPlanations) is employed on tree-based models, thereby revealing the relative importance of each feature to a given prediction.

7. **Comparison of Results:** Finally, the results from all four models are consolidated into bar plots that compare their performance across the chosen metrics. This comprehensive view helps identify the best-performing approach and highlights how different algorithms respond to the data preprocessing pipeline.

It combines statistical feature screening with SMOTE-based rebalancing and Min-Max normalization. The proposed pipeline ensures the ground for model development and its further evaluation effectively. Integration of multiple classifiers along with robust interpretability techniques further supports both the identification of Trojan malware and the broader goal of improving the Android malware detection frameworks.

4.2 Workplan

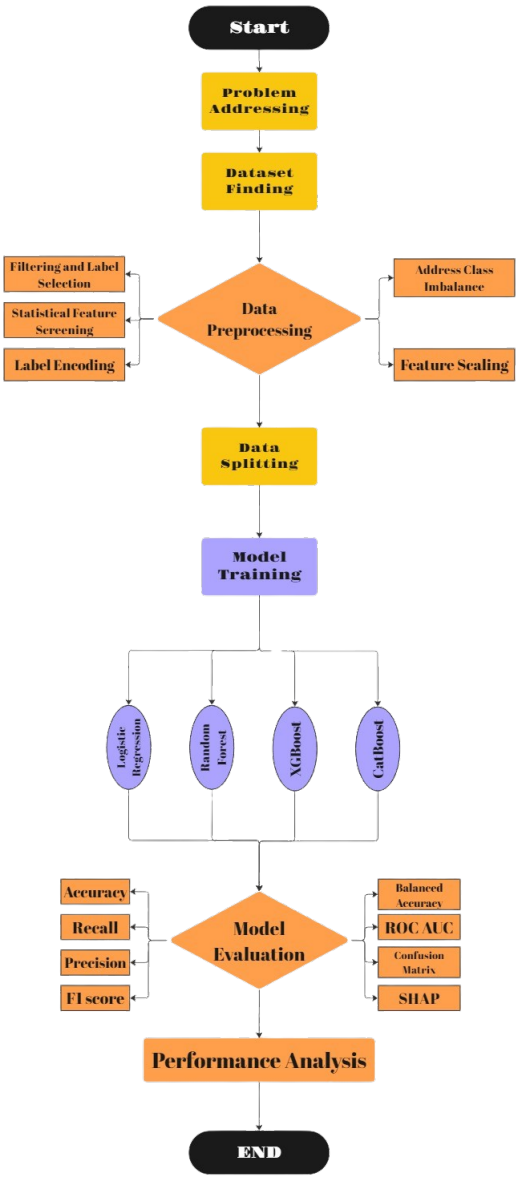


Figure 4.1: Workplan

4.3 Experimental Setup

- **Hardware/Software Environment:** All experiments are conducted within a Python environment where libraries such as `scikit-learn`, `XGBoost`, `CatBoost`, and `imblearn` are installed. The setup also makes use of `gdown` for data retrieval from Google Drive and `pyarrow` for efficient file I/O operations with `.parquet` files. The specific choice of hardware (GPU or CPU) can impact training speed; however, the primary workflow remains consistent regardless of the underlying compute resources.

- **Data Loading and Handling:** The dataset, available in `.parquet` format, is downloaded using `gdown` and loaded into a `pandas` DataFrame via `pyarrow`. This format facilitates parallel reading (via `use_threads=True`), which can speed up data ingestion for large files. Once loaded, the DataFrame undergoes routine inspections (e.g., `.head()`, `.shape()`, and `.describe()`) to confirm validity.

- **Filtering and Column Selection:** As part of the code, rows that do not correspond to either "Adware" or "Trojan" labels are removed. Additionally, numeric columns are extracted for preprocessing. A one-sample t-test, applied to each numeric column, ensures that only columns with statistically significant deviation from zero (`p-value` < 0.05) are retained. This step reduces feature dimensionality and is thus beneficial for model training efficiency.

- **Imbalanced Dataset Resolution:** The Synthetic Minority Over-sampling Technique (SMOTE) addresses any class imbalance by creating new minority-class samples through interpolation. This is applied after filtering and before the final data split to promote balanced training. Such balancing is critical when one class (often Trojan) is heavily outnumbered and might otherwise lead to biased model predictions.

- **Scaling and Partitioning:** Prior to training, the selected numeric columns are scaled using `MinMaxScaler` to map feature values into the $[0, 1]$ range. Afterwards, the dataset is divided into training (60%) and testing (40%) sets. A random seed (`random_state=42`) is fixed to facilitate reproducible splits and consistent performance comparisons across different runs. This ensures that all reported metrics reflect a uniform dataset partitioning scheme.

- **Model Development and Libraries:** Several classification algorithms are trained and compared, each leveraging a different library:

  - **Logistic Regression** via `scikit-learn`
  - **Random Forest** via `scikit-learn`
  - **XGBoost** via the `xgboost` library
  - **CatBoost** via the `catboost` library

  The mix of linear, ensemble, and gradient-boosted models showcases a broad spectrum of machine learning methodologies, providing a robust experimental comparison.

- **Performance Metrics and Visualizations:** Each trained model is evaluated on the test set using Accuracy, Precision, Recall, F1-score, ROC AUC, and Balanced Accuracy. Visual outputs include confusion matrices, ROC curves, and bar charts summarizing model results. Additionally, SHAP (SHapley Additive exPlanations) plots are generated for tree-based models to offer interpretability regarding feature impact on classification decisions.

4.4 Feature Engineering

Feature engineering in this study involves a concise yet impactful set of techniques to refine the dataset for improved classification performance:

1. **Statistical Column Filtering:** After isolating "Adware" and "Trojan" samples, a one-sample t-test is applied to each numeric column. This step preserves only those features whose mean significantly differs from zero (i.e., `p-value` < 0.05). While atypical compared to traditional feature-selection methods, it helps eliminate columns that offer minimal discriminatory power.

2. **Handling Class Imbalance (SMOTE):** In light of uneven class distributions, the Synthetic Minority Over-sampling Technique (SMOTE) is employed to generate new minority-class samples. This prevents models from becoming biased towards the majority class and enhances the overall robustness of the classification approach.

3. **Min–Max Scaling:** Finally, the retained numeric features are normalized to the $[0, 1]$ range using `MinMaxScaler`. This uniform scaling ensures that features with naturally larger numeric ranges do not overshadow those with smaller ones, facilitating more stable training for both linear and tree-based algorithms.

Chapter 5

Result

5.1 Performance Evaluation Metrics

Performance evaluation in classification consists of comparing the model's predicted labels against actual or ground-truth labels. While each metric ultimately rests on these comparisons, they indeed offer different perspectives on the performance of a model. Before interpreting the results from any of these metrics, it is relevant to understand how class imbalance, misclassification costs, and model calibration may influence the choice of metric. Once a model has been trained and tuned-e.g., via cross-validation-it is usually evaluated on a held-out test set in order to estimate real-world predictive performance. This section describes several metrics in common use for binary classification.

5.1.1 Accuracy

Accuracy is one of the most straightforward metrics, reflecting the proportion of correctly classified samples out of the total samples. Formally:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}.$$

Although easy to interpret, accuracy can be misleading in scenarios where class distributions are heavily imbalanced. In such cases, a model that predominantly predicts the majority class may still achieve high accuracy but perform poorly on the minority class.

5.1.2 Precision

Precision quantifies how many of the instances predicted as positive are truly positive:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}.$$

A high precision means the model commits fewer false alarms. In contexts where false positives carry a high cost (e.g., flagging benign applications as malicious or mislabeling healthy patients as sick), precision becomes an especially critical metric.

5.1.3 Recall

Also known as *Sensitivity*, recall measures the proportion of actual positive instances that the model correctly identifies:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}.$$

A high recall indicates that the model effectively captures most or all of the positive cases. This metric is essential when missing a positive case (e.g., failing to detect malware or a particular illness) is particularly costly.

5.1.4 F1 Score

The F1 score is the harmonic mean of precision and recall:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

It provides a single metric that balances both false positives and false negatives, making it especially useful when class distribution is skewed and one seeks a trade-off between precision and recall.

5.1.5 Balanced Accuracy

Balanced Accuracy addresses imbalanced datasets by taking the average of recall measured on each class:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}}\right).$$

This metric ensures that performance on the minority class is given equal weight, even when one class substantially outnumbers the other. A model that simply predicts the majority class will yield a low balanced accuracy despite potentially high raw accuracy.

5.1.6 ROC AUC

Receiver Operating Characteristic (ROC) curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The area under this curve (AUC) summarizes the overall ability of the model to discriminate between positive and negative classes:

$$\text{ROC AUC} = \int_0^1 \text{TPR}\, d(\text{FPR}).$$

A higher ROC AUC indicates better classification performance across different decision thresholds. This metric is particularly informative for comparing multiple models or varying thresholds.

5.1.7 Confusion Matrix

A confusion matrix displays the number of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). It enables a granular view of how the model performs on each class:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Table 5.1: Structure of a Confusion Matrix for Binary Classification

5.2 Performance Analysis of static features

To evaluate the effectiveness of each classifier, multiple metrics—Accuracy, Precision, Recall, F1 score, ROC AUC, and Balanced Accuracy—were computed on the test set. This section provides a concise overview of each model's performance, referencing three visual aids per model: a confusion matrix, a ROC curve, and a feature explanation plot. At the end of this section, a comparative table summarizes the numerical results for a quick cross-model comparison.

5.2.1 Logistic Regression

- **Metrics:**
  - Accuracy: 0.9663
  - Precision: 0.9623
  - Recall: 0.9704
  - F1: 0.9663
  - ROC AUC: 0.9664
  - Balanced Accuracy: 0.9664

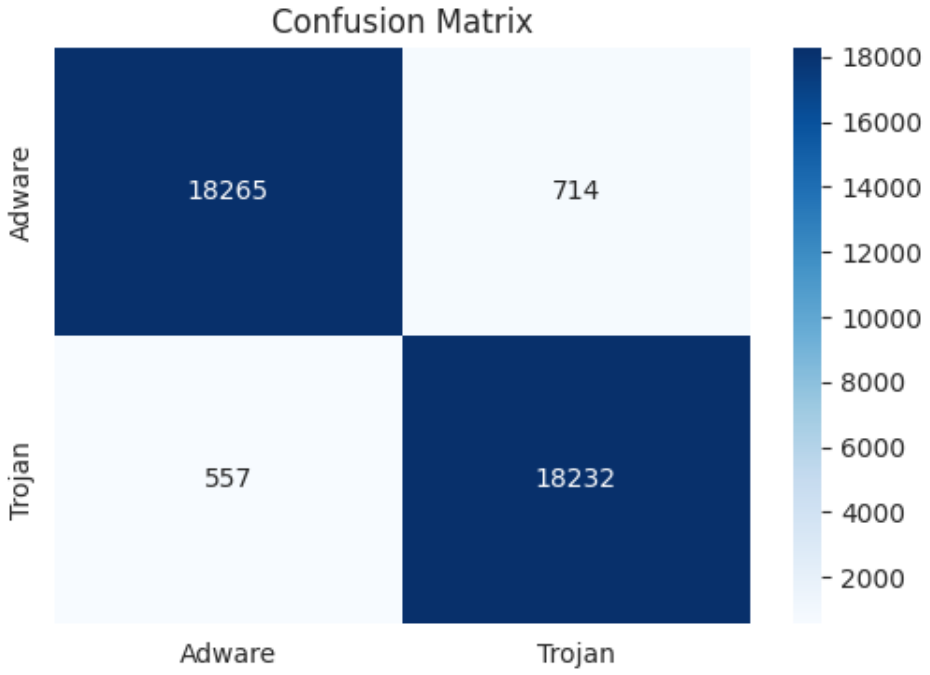


Figure 5.1: Logistic Regression Confusion Matrix

The confusion matrix shows that the logistic regression model can accurately identify 18265 samples as Adware and 18232 samples as Trojan, though it misclassified 714 Adwares and 557 Trojans making it in total of 1271 misclassifications.

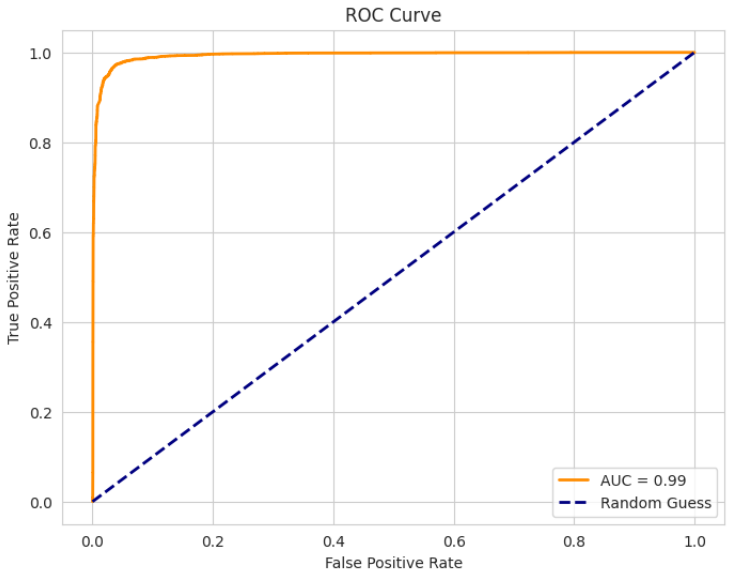


Figure 5.2: Logistic Regression ROC Curve

With false positive rate in the x axis and true positive rate in the y axis, the ROC curve shows the AUC value of 0.99. It means that the Logistic Regression model can differentiate the classes accurately in 99

Figure 5.3: Logistic Regression Feature Explanation

The top twenty most crucial features of the Android malware detection model appear in bar charts in Figure 5.3. Log-odds values of features that affect categorization appear on the x-axis in this illustration. Each feature name exists on the y-axis as its associated identifier shows. The probability that a

sample will be classified as malware increases when the characteristics possess higher values due to the positive coefficients (green bars) which enhance malware detection. Negative coefficients (red bars) indicate characteristics that contribute to reducing the potential for malware classification. The investigation reveals that malware detection benefits the most from feature F8. The feature F5199 has the largest negative influence since it causes a sample to become less likely to be identified as malware.

5.2.2 Random Forest

- **Metrics:**
  - Accuracy: 0.9876
  - Precision: 0.9865
  - Recall: 0.9887
  - F1: 0.9876
  - ROC AUC: 0.9876
  - Balanced Accuracy: 0.9876

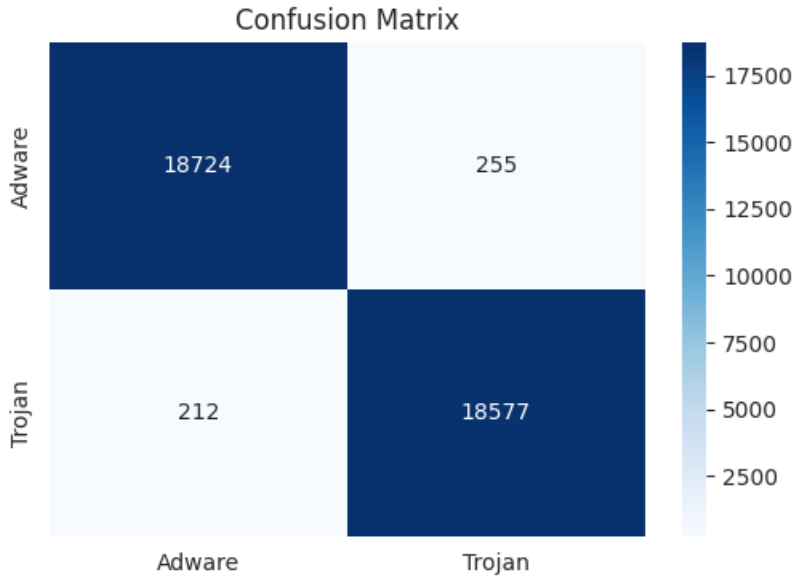


Figure 5.4: Random Forest Confusion Matrix

Random Forest Model has identified the classes most precisely compared to the other three models (Logistic Regression, XGBoost & CatBoost). It misclassified only 255 samples as Trojan and 212 samples as Adware which also signifies that the values of the evaluation matrices will also be high for this particular model.

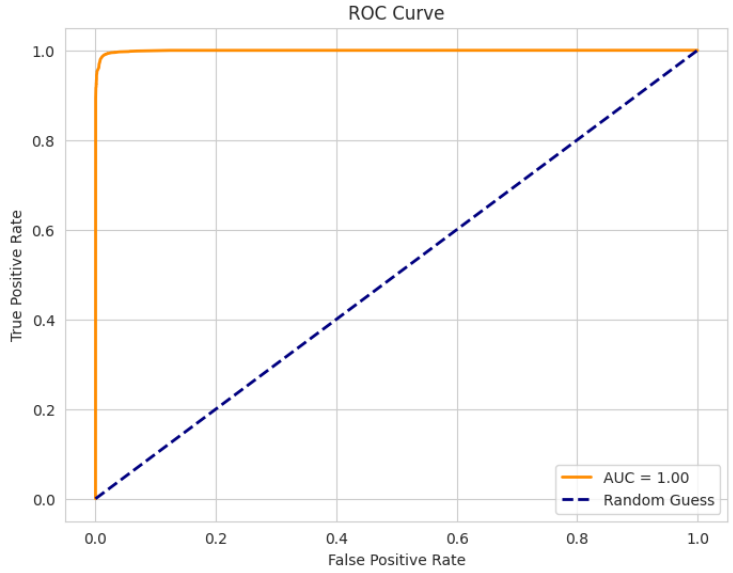


Figure 5.5: Random Forest ROC Curve

Since the value of AUC is 1 for Random Forest Model, it denotes that the model is a perfect classifier for distinguishing between the classes - Adware and Trojan.

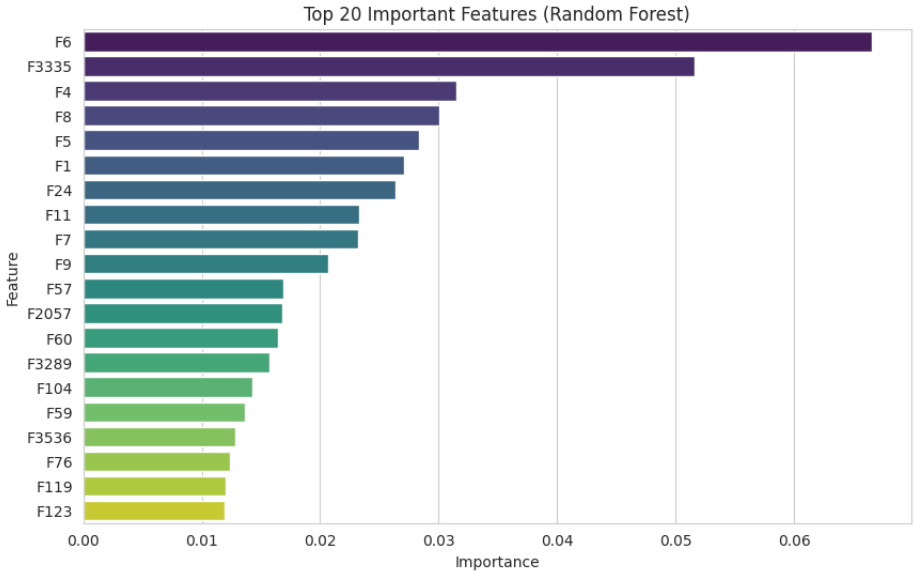


Figure 5.6: Random Forest Feature Explanation

This illustration presents the top 20 crucial features which belong to a Random Forest Model utilizing mean decrease in impurity as the metric for measuring significance. The horizontal scale depicts feature importance which measures from 0 to 0.06 for all features. The graph shows features with labels of F and a sequential number which appear on the vertical scale. Theme bar proportions reflect the significance levels of separate categories. Higher-importance features receive darker shades of purple, blue, teal, green, and yellow through color gradient features. The most crucial feature is F6 and after it comes F3335 but features F4 and F8 show a moderate importance level. Features

F123 together with F119 and F76 demonstrate the lowest importance from the top 20 features.

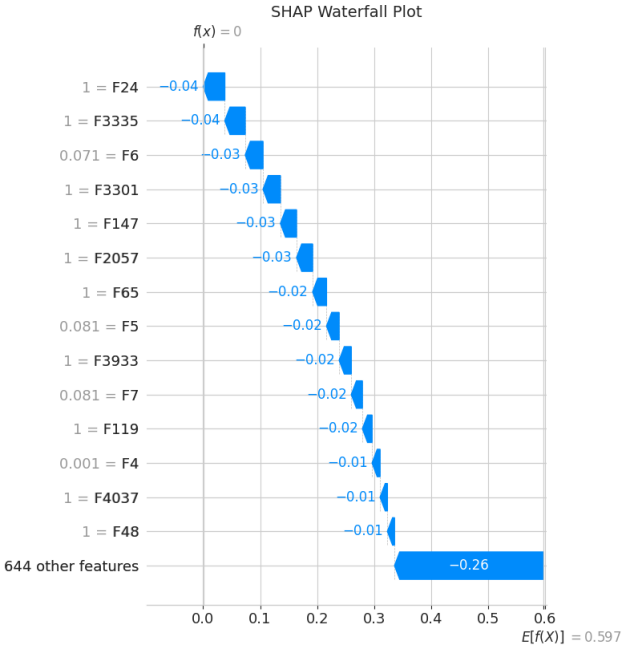


Figure 5.7: Random Forest Shap Analysis

The given figure represents a SHAP Waterfall Plot for a Random Forest Model. The graph shows how individual features contribute to a specific prediction in a Random Forest model. The x-axis denotes the cumulative SHAP values which is the expected value (E[f(x)] = 0.597) of Baseline model prediction. The final prediction of the instance is approximately 0.26 which is lower than the expected value due to feature contributions. This SHAP waterfall plot indicates an individual feature's SHAP value, whether the value increases or decreases based on the model's prediction. The positive SHAP value of a feature means the prediction is higher, similarly, The negative SHAP value of a feature means the prediction is lower. Here, the most influential features in this instance are F24 and F3335, but the value of both instances is negative (-0.04). Rest of the features have insignificant contributions. Through this analysis , the model will be able to do the decision-making process smoothly by Understanding such contributions.

5.2.3 XGBoost

- **Metrics:**
  - Accuracy: 0.9859
  - Precision: 0.9840
  - Recall: 0.9878
  - F1: 0.9859
  - ROC AUC: 0.9859
  - Balanced Accuracy: 0.9859

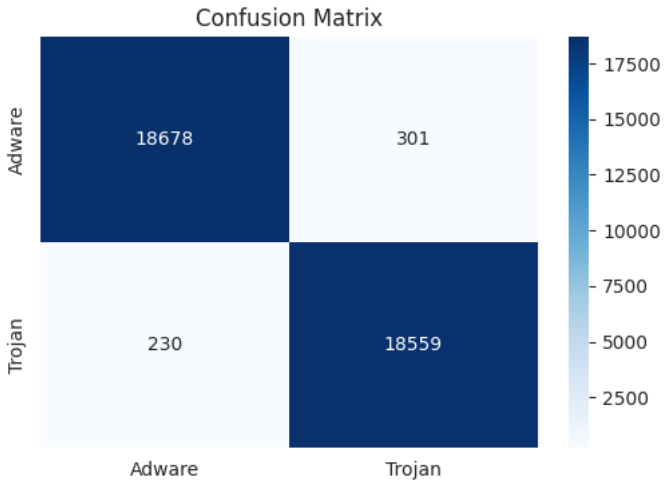


Figure 5.8: XGBoost Confusion Matrix

Though XGBoost outshines Logistic regression in terms of classification as 18678 samples were correctly recognized as Adware and 18559 samples as Trojan,it is still lagging behind. The model failed to pick out 531 samples accurately.

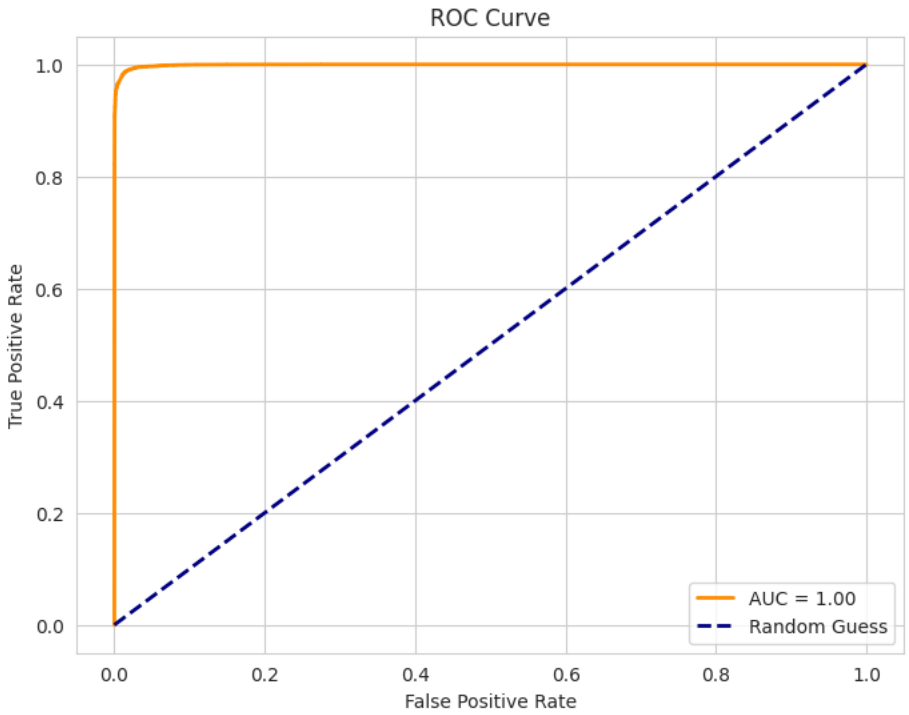


Figure 5.9: XGBoost ROC Curve

The AUC value of ROC curve for XGBoost Model is 1 implying it to be a perfect classifier for differentiating all the positive and negative instances.

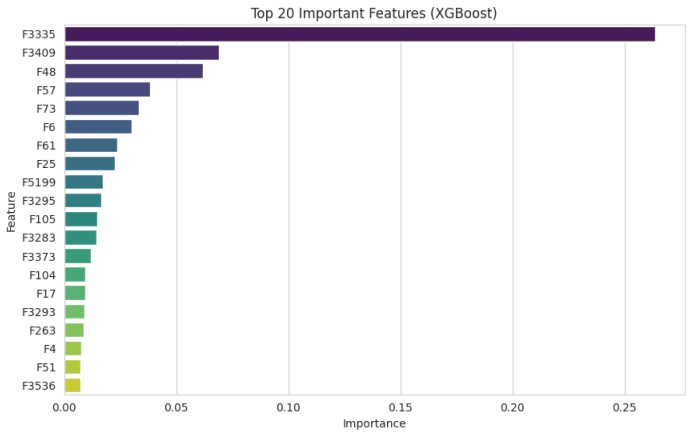


Figure 5.10: XGBoost Feature Explanation

XGBoost Model recognizes the 20 most important features which can be viewed in Figure 5.10. Functions of importance within the horizontal bar chart receive presentation by rating their significance on the x-axis as scores while displaying the feature names on the y-axis. The model classifies F3335 as its most influential feature and places it ahead of F3409 and F48 respectively in terms of significance. Each feature has a distinct value in the model predictions based on its importance score.

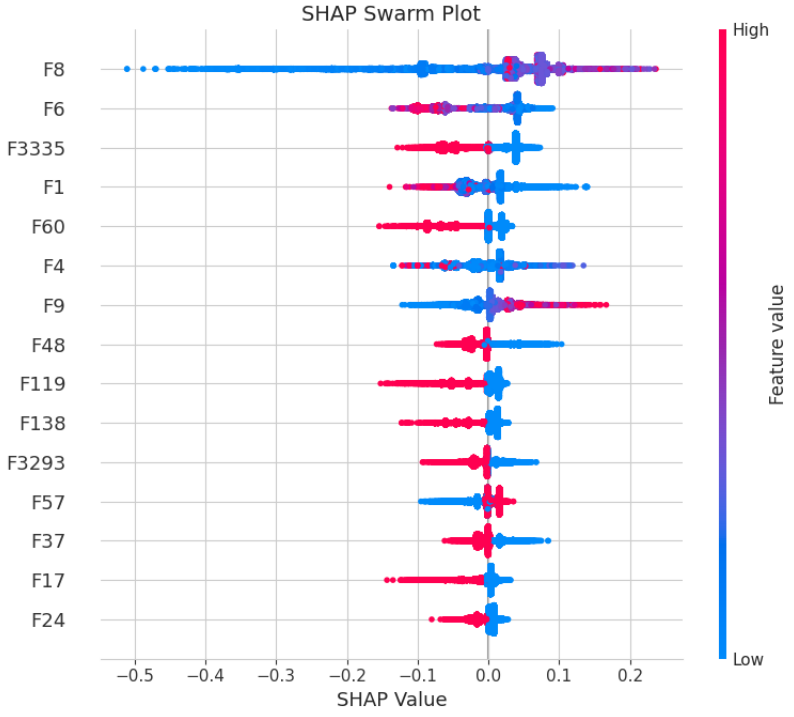


Figure 5.11: XGBoost Shap Analysis

The given figure represents a SHAP Swarm Plot for a XGBoost model. The graph shows how individual features contribute to a specific prediction in aXGBoost model. Each point of the plot indicates a feature value of a specific instance of the dataset. The x-axis denotes that the contribution of the feature in the prediction is higher or lower. The colour gradient represents the

feature values. Here, the high feature values are indicated by red colour and the low feature values are indicated by blue. In this graph, F8, F6, F3335, and F1 have strong impact on the model prediction with both positive and negative SHAP values. Based on the SHAP values, have a major impact on the model's predictions. F9 and F6 reflect an imbalanced distribution and a large distribution, respectively. This is crucial in improving profiles considering efficiency, effectiveness, feature selection, classification accuracy, and transparency because this helps in optimizing the XGBoost model for Android malware detection.

5.2.4 CatBoost

- **Metrics:**
  - Accuracy: 0.9852
  - Precision: 0.9847
  - Recall: 0.9855
  - F1: 0.9851
  - ROC AUC: 0.9852
  - Balanced Accuracy: 0.9852

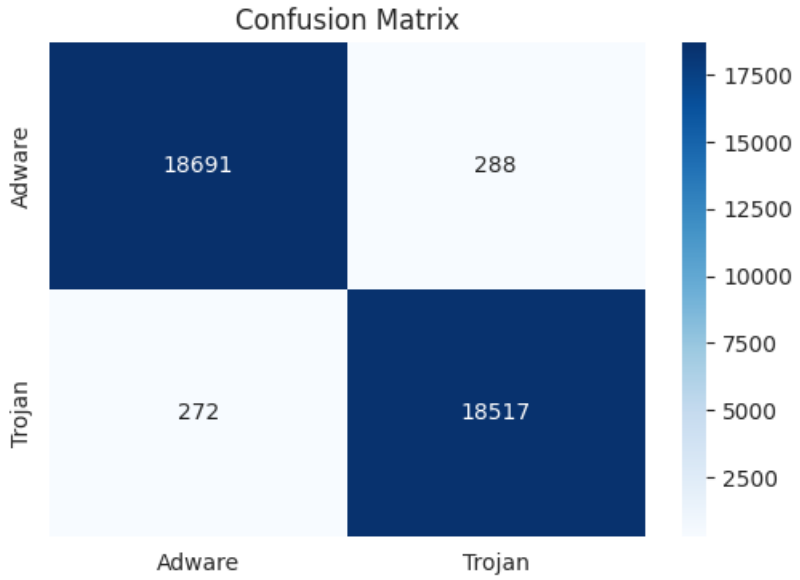


Figure 5.12: CatBoost Confusion Matrix

CatBoost has perfectly distinguished 18691 samples as Adwares and 18517 samples as Trojans. The number of misclassified samples is 560 which is slightly higher than the XGBoost model.

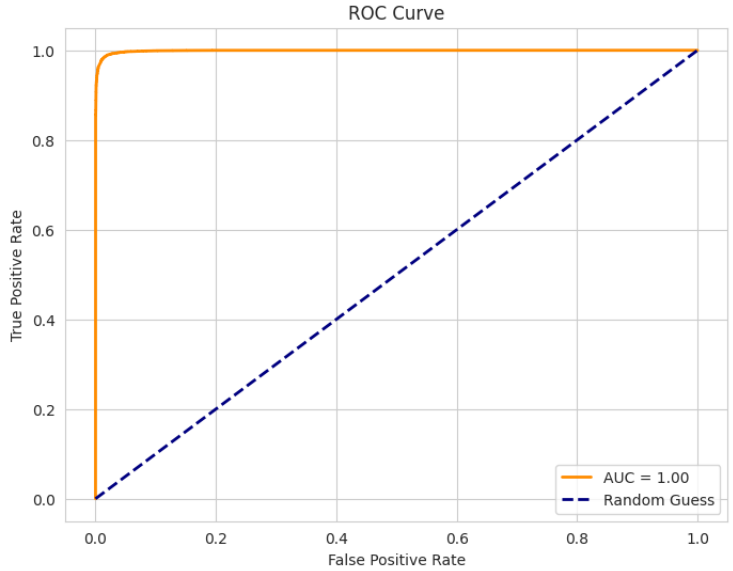


Figure 5.13: CatBoost ROC Curve

The ROC curve of the CatBoost Model has touched the top left corner where the true positive rate is 1 making the AUC value 1 as well .

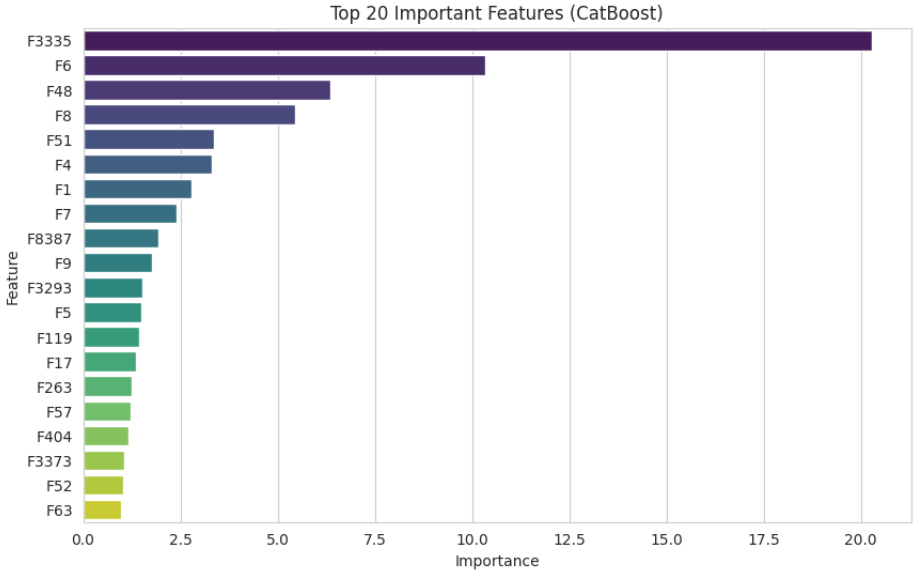


Figure 5.14: CatBoost Feature Explanation

The 20 most weighty features selected by the CatBoost Model are illustrated in Figure 5.14. The horizontal bar chart positions features by the degree they influence target variable predictions. The bar chart uses the importance score on the x-axis together with feature names displayed on the y-axis. Feature F3335 stands as the vital prediction factor followed by F6 then F48 and F8 within the selected other features. Figure 5.14 displays three bar colors to differentiate between significant and less significant features out of the top 20. This analysis reveals important features used in making predictions which helps select vital features for optimizing the model performance.

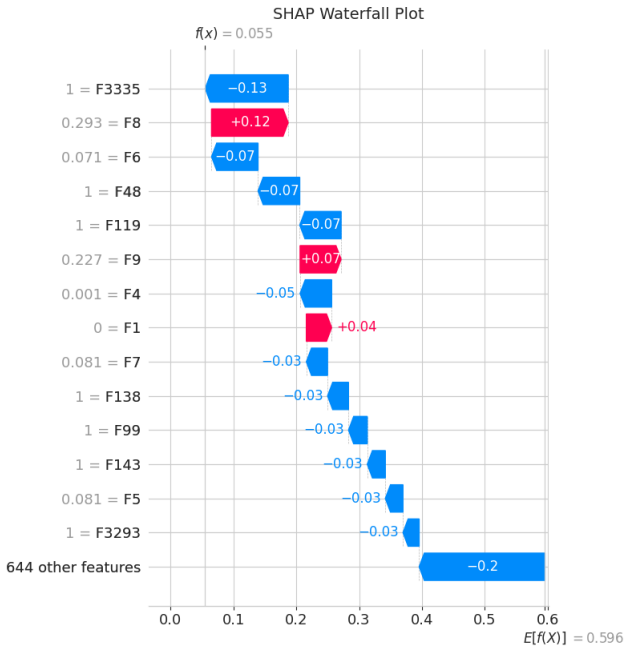


Figure 5.15: CatBoost Shap Analysis

The given figure represents a SHAP for a CatBoost model analysis which shows how each feature contributes to a particular prediction. Each feature's effect on the model's output is indicated by its SHAP value. Here, positive values increase predictions and negative values decrease them. F3335 has a significant negative impact which is the most impactful characteristic. Moreover, the other most important is also additionally impacted to various degrees by other significant variables, including F6, F48, and F119. The final predicted value is the consequence of these characteristics' which have combined impact. Model evaluation and feature selection are made easier by this SHAP analysis.

5.2.5 Comparative Results

Table 5.3 summarizes the key performance metrics for all four classifiers.

Model	Accuracy	Precision	Recall	F1	ROC AUC	Balanced Acc.
Logistic Regression	0.9663	0.9623	0.9704	0.9663	0.9664	0.9664
Random Forest	0.9876	0.9865	0.9887	0.9876	0.9876	0.9876
XGBoost	0.9859	0.9840	0.9878	0.9859	0.9859	0.9859
CatBoost	0.9852	0.9847	0.9855	0.9851	0.9852	0.9852

Table 5.2: Comparison of Performance Metrics Across Models

As observed, the ensemble methods (Random Forest, XGBoost, and CatBoost) outshine the linear Logistic Regression with 96% accuracy rate in nearly all metrics, with Random Forest claiming the highest scores. Nevertheless, Logistic Regression remains a simpler and more interpretable option. Choosing the optimal classifier may depend on the trade-offs between accuracy, interpretability.

5.3 Performance Analysis of dynamic features

To evaluate the effectiveness of each classifier, multiple metrics—Accuracy, Precision, Recall, F1 score, ROC AUC, and Balanced Accuracy—were computed on the test set. This section provides a concise overview of each model's performance, referencing three visual aids per model: a confusion matrix, a ROC curve, and a feature explanation plot. At the end of this section, a comparative table summarizes the numerical results for a quick cross-model comparison.

5.3.1 Logistic Regression

- **Metrics:**
  - Accuracy: 0.8199
  - Precision: 0.8192
  - Recall: 0.8229
  - F1: 0.8211
  - ROC AUC: 0.8199
  - Balanced Accuracy: 0.8199

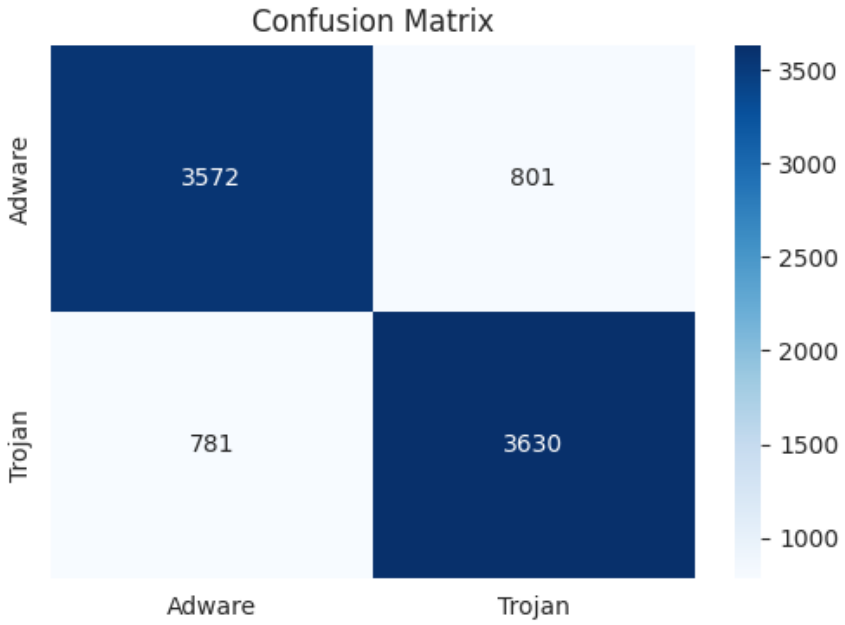


Figure 5.16: Logistic Regression Confusion Matrix

This matrix represents the Logistic Regression model's predictions, highlighting noticeable misclassifications between "Adware" and "Trojan" with larger off-diagonal values compared to other model.

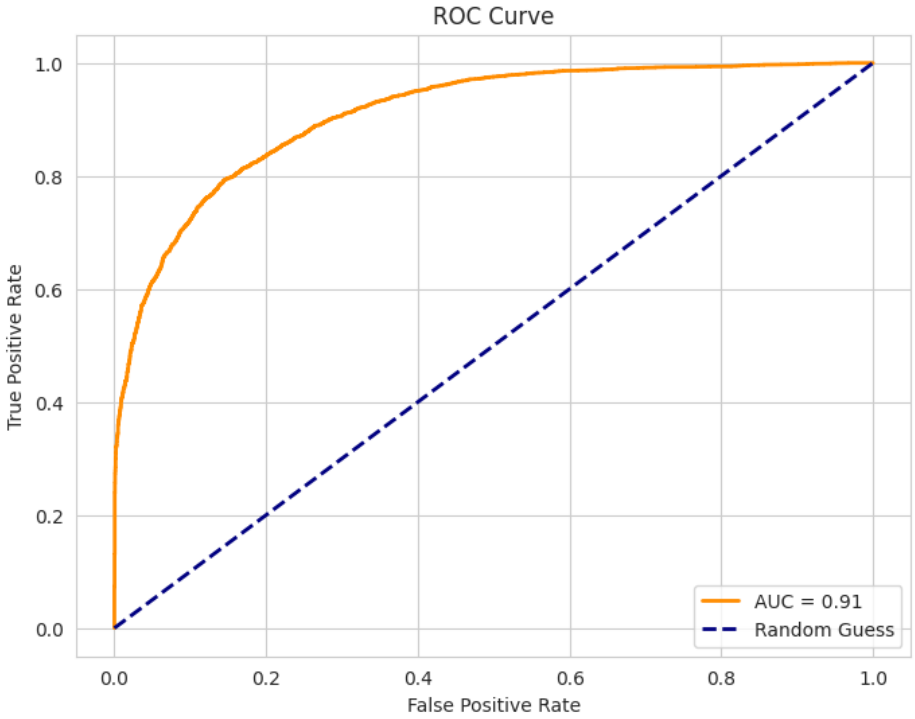


Figure 5.17: Logistic Regression ROC Curve

This ROC curve has an AUC of 0.91, meaning the model performs well but is slightly less effective than the first. The curve remains above the random guess line, demonstrating storing predictive power.

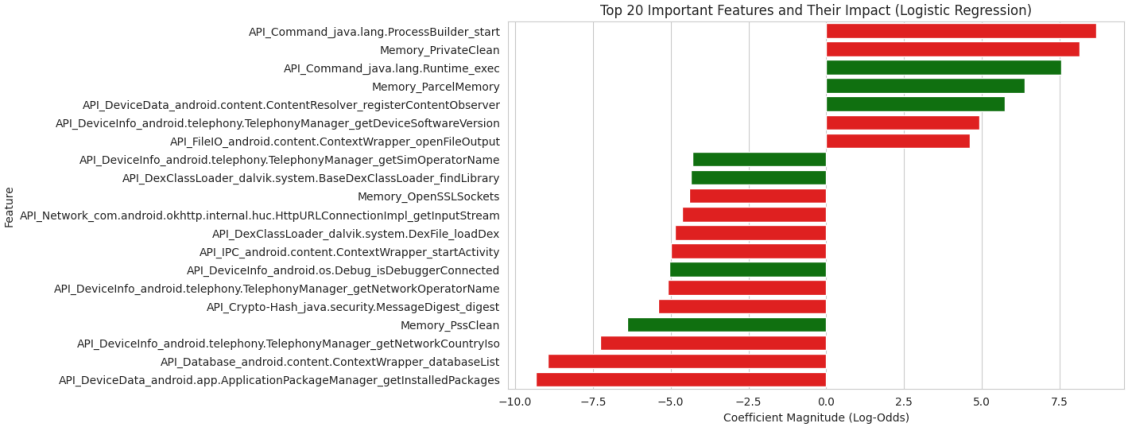


Figure 5.18: Logistic Regression Feature Explanation

This visualization presents the top 20 features with the highest impact in a Logistic Regression model, based on co-efficient magnitudes. The red bars indicate positive correlations, while green bars show negative correlations with the target variable. Key features include process executions, memory operations and telephony-related API calls. The length of each bar reflects the strengths of influence on classification decisions.

5.3.2 Random Forest

- **Metrics:**
  - Accuracy: 0.9689
  - Precision: 0.9823
  - Recall: 0.9553
  - F1: 0.9686
  - ROC AUC: 0.9690
  - Balanced Accuracy: 0.9690

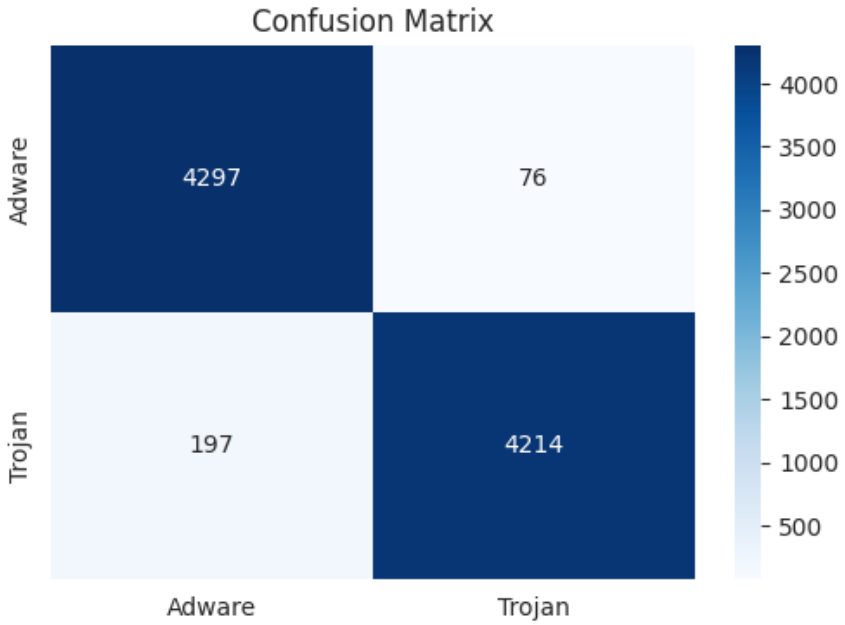


Figure 5.19: Random Forest Confusion Matrix

This Random Forest Model's confusion matrix demonstrates good predictive accuracy, with slightly higher misclassifications than Catboost but less than logistic Regression.

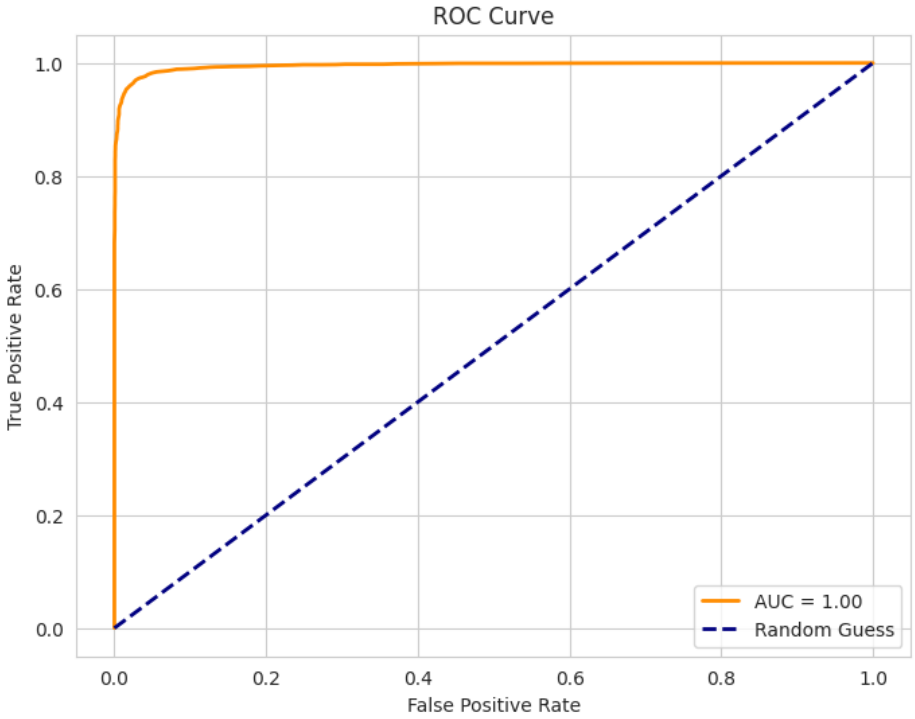


Figure 5.20: Random Forest ROC Curve

The ROC curve achieves a perfect AUC of 1,00, suggesting a flawless model with ideal classification performance. The curve reaches (0.1) almost immediately, indicating no false positive.

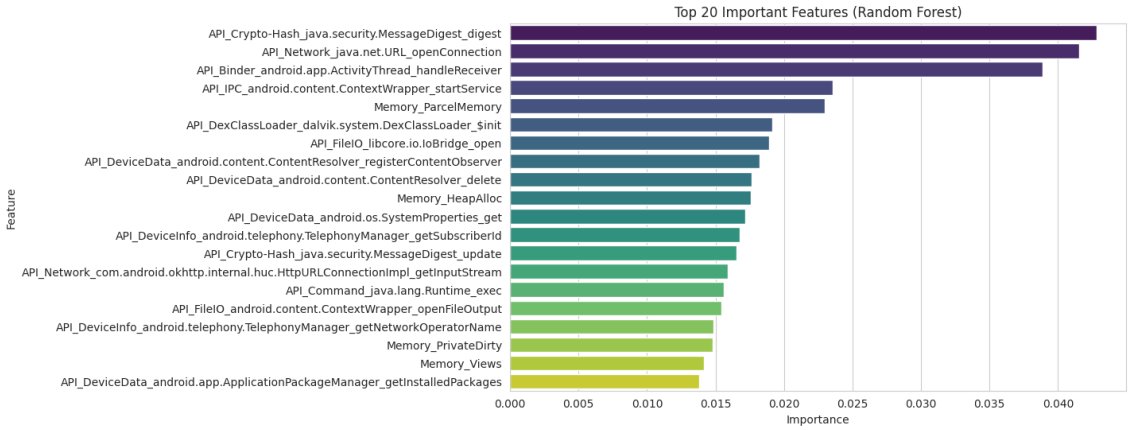


Figure 5.21: Random Forest Feature Explanation

This chart ranks the top 20 features selected by the Random Forest Model based on their contribution to decision-making. Cryptographic hashing, network request and file operations appear as dominant factors. The API_Crypti-Hash_java.security.MessageDigest_digest is identified as the most important feature. The color gradient represents the relative importance of each feature.

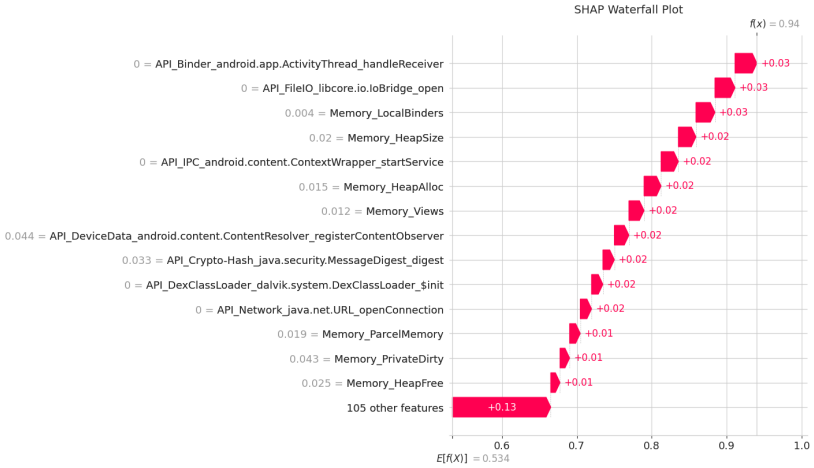


Figure 5.22: Random Forest Shap Analysis

The SHAP waterfall plot demonstrates the influence of specific features across a prediction, illustrating key API and memory-related feature contributions that led to the model's outcome.

5.3.3 XGBoost

- **Metrics:**
  - Accuracy: 0.9674
  - Precision: 0.9684
  - Recall: 0.9667
  - F1: 0.9676
  - ROC AUC: 0.9674
  - Balanced Accuracy: 0.9674

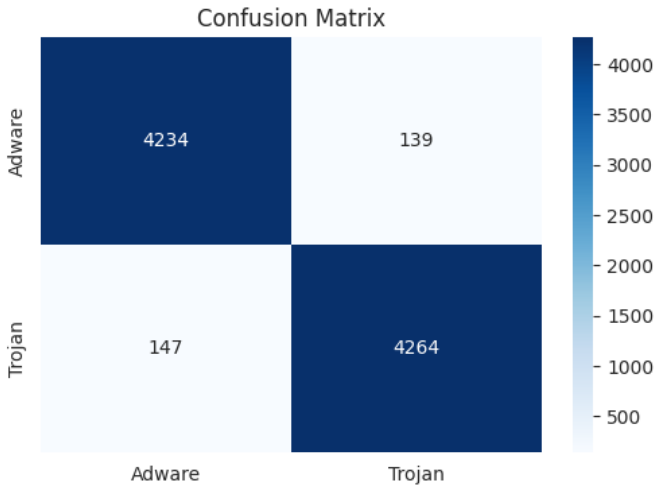


Figure 5.23: XGBoost Confusion Matrix

This matrix illustrated the XGBoost Model's performance, showing a balance between true positives and negatives with minimal classifications.

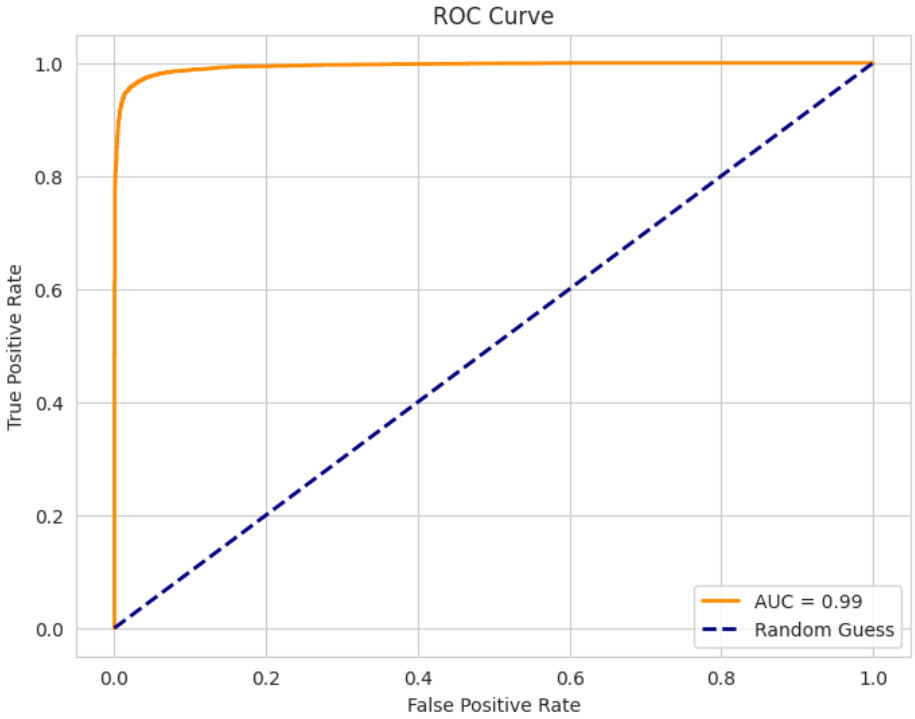


Figure 5.24: XGBoost ROC Curve

The ROC curve showcases the XGBoost Model's high AUC (0.99), indicating excellent discrimination between the "Adware" and "Trojan" categories.

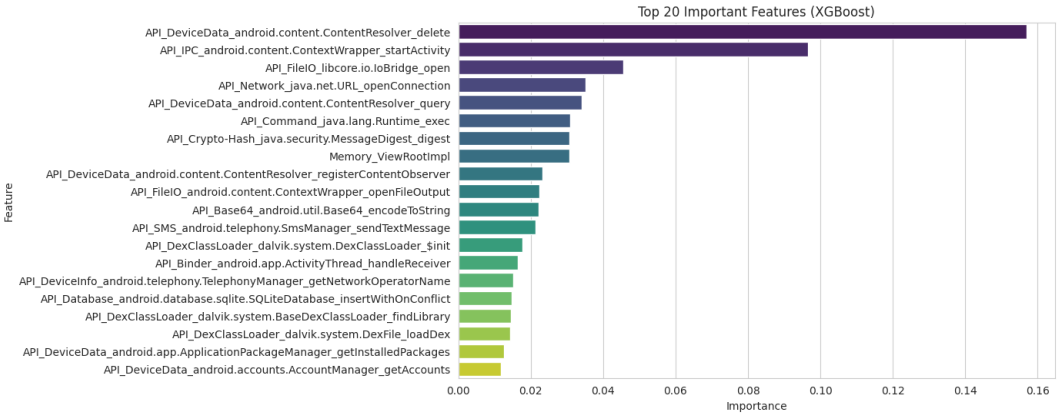


Figure 5.25: XGBoost Feature Explanation

The XGBoost Model's top 20 important features are displayed in this bar chart, focusing on system interactions such as content deletion, inter-process communication, and cryptographic functions. API_Devicedata_android.content. ContentResolver_delete emerges as the most critical feature. The varying bar length indicates the feature's relative influence in classification tasks.

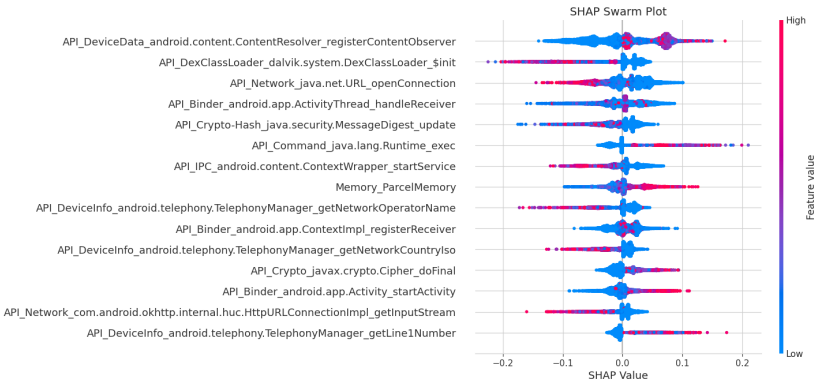


Figure 5.26: XGBoost Shap Analysis

This swarm plot visualizes feature importance across multiple predictions, where the impact of individual feature like API-related calls and memory usage is distributed across high and low SHAP values, indicating varying influence on predictions.

5.3.4 CatBoost

- **Metrics:**
  - Accuracy: 0.9678
  - Precision: 0.9695
  - Recall: 0.9662
  - F1: 0.9679
  - ROC AUC: 0.9678
  - Balanced Accuracy: 0.9678

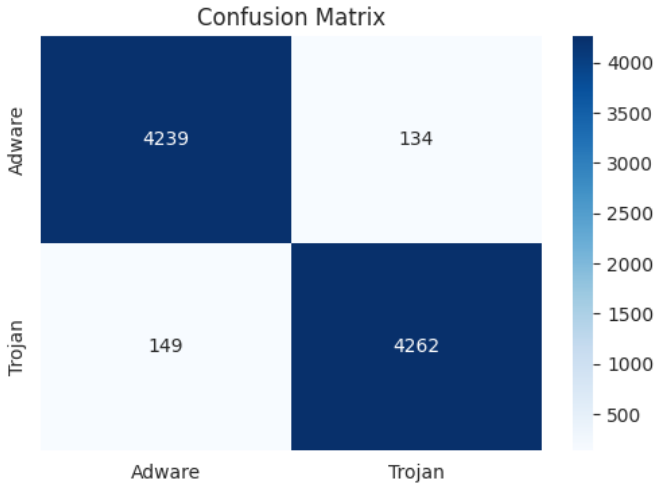


Figure 5.27: CatBoost Confusion Matrix

This Matrix visualizes the performance of a CatBoost model, showing strong accuracy with minimal misclassification between “Adware” and “Trojan” categories, as indicated by high diagonal values.

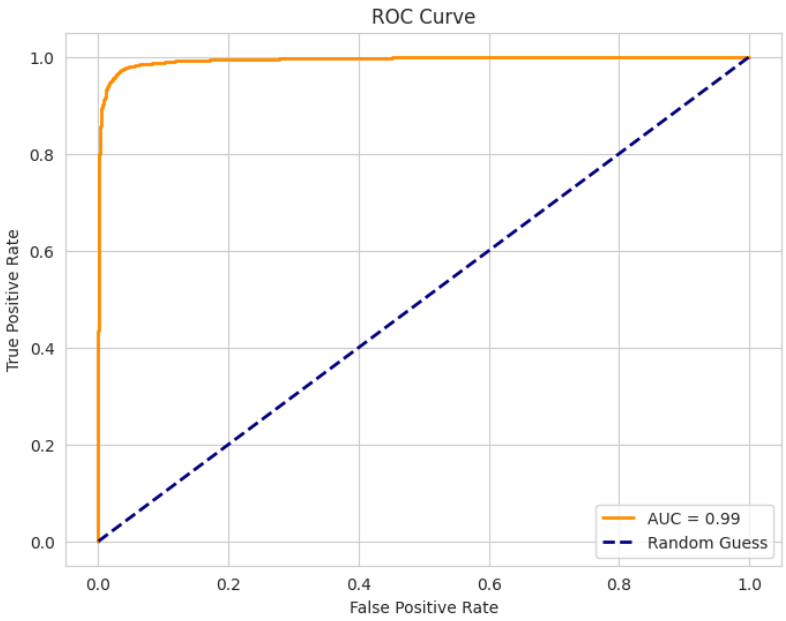


Figure 5.28: CatBoost ROC Curve

The ROC curve shows a high-performing model with an AUC of 0.99. The curve stays close to the top-left corner, indicating excellent classification ability. The dashed blue line represent a random guess.

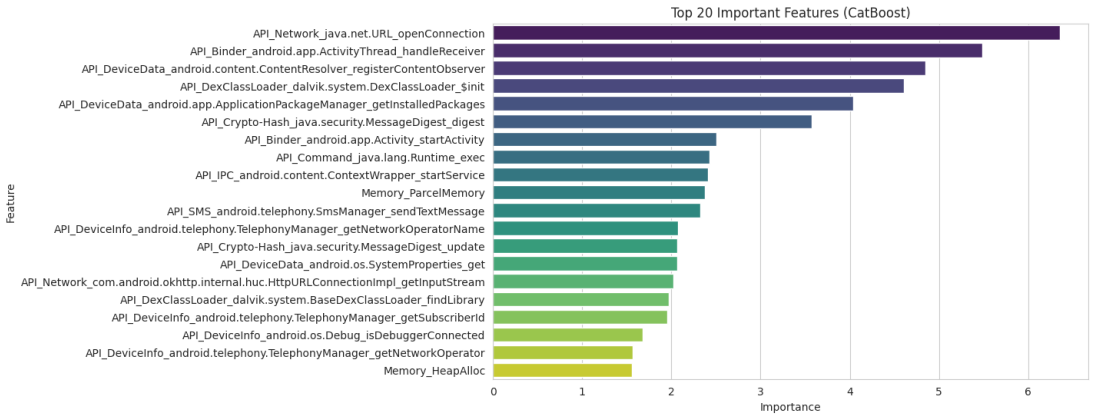


Figure 5.29: CatBoost Feature Explanation

This bar chat displays the top 20 most influential features selected by the CatBoost model. It highlights API calls related to networking, cryptographic hashing, and system data access. The most important feature is API_Netwok_java.net.URL_openConnection, indicating the significance of network-related activities. The varying bar lengths represent the relative importance of each feature in model predictions.

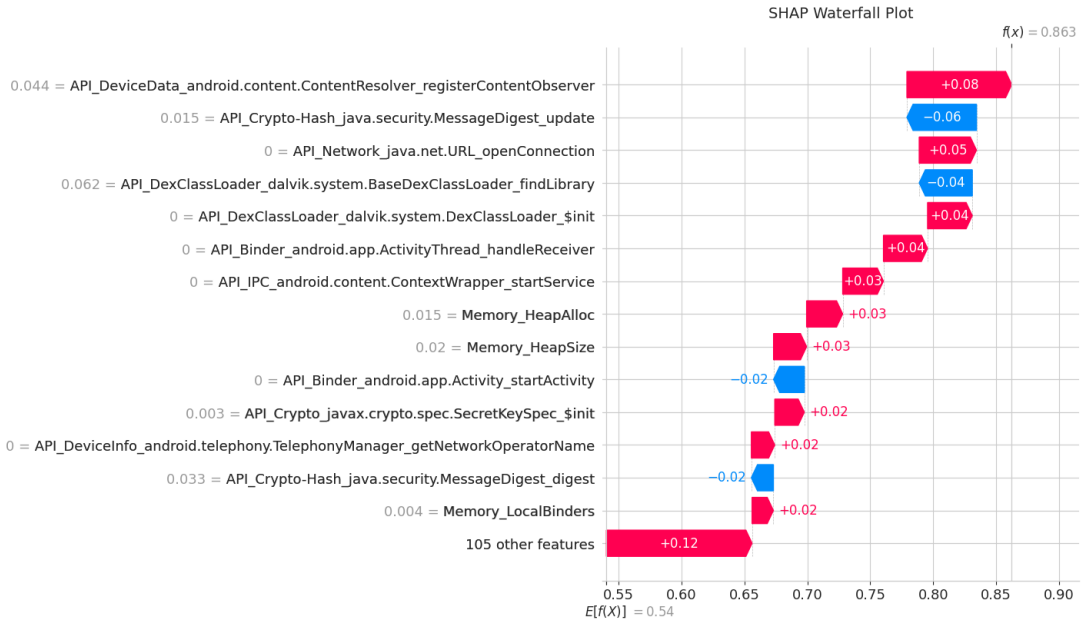


Figure 5.30: CatBoost Shap Analysis

This plot explains the prediction for a single instance made by the CatBoost Model, revealing the contribution of individual features like API-related calls and memory metrics to the final prediction.

5.3.5 Comparative Results

Table 5.3 summarizes the key performance metrics for all four classifiers.

Model	Accuracy	Precision	Recall	F1	ROC AUC	Balanced Acc.
Logistic Regression	0.8199	0.8192	0.8229	0.8211	0.8199	0.8199
Random Forest	0.9689	0.9823	0.9553	0.9686	0.9690	0.9690
XGBoost	0.9674	0.9684	0.9667	0.9676	0.9674	0.9674
CatBoost	0.9678	0.9695	0.9662	0.9679	0.9678	0.9678

Table 5.3: Comparison of Performance Metrics Across Models

As observed, the ensemble methods (Random Forest, XGBoost, and CatBoost) outshine the linear Logistic Regression with 81% accuracy rate in nearly all metrics, with Random Forest claiming the highest scores. Nevertheless, Logistic Regression remains a simpler and more interpretable option. Choosing the optimal classifier may depend on the trade-offs between accuracy, interpretability.

5.4 Discussion on Result

The results have proven that the two categories of malware chosen, Adware and Trojan, can easily be differentiated by machine learning-based detection methods. Even a relatively simple model like Logistic Regression reaches accuracy greater than 81% for dynamic features and 96% for static features , indicating that linear

classifiers can learn meaningful patterns in this dataset. On the other hand, all three ensemble and gradient boosting methods, Random Forest, XGBoost, and CatBoost, discussed further improve almost all metrics from the Logistic Regression class, highlighting better suitability for finding complex high-dimensional relations among features.

The Random Forest model achieved the best scores among the ensemble models for all metrics: Accuracy, Precision, Recall, and F1. Thus, besides performing very well for general predictions, it also keeps a good sensitivity for both classes. Next come CatBoost and XGBoost, each achieving results that are very close to those of the leader. Overall, small differences among these three methods indicate their comparable ability to handle sophisticated decision boundaries and robust handling in case of features that can be potentially imbalanced.

Interpretation of feature importances and SHAP plots provides insight into the underlying attributes most decisive in classifying samples as Adware or Trojan. This may inform future refinements in feature engineering or dataset curation, such as incorporating dynamic behaviors or extended metadata. On the whole, results strongly indicate ensemble models, particularly Random Forest, as powerful tools in Android malware detection, while holding great promise for real-world security applications where there is a pressing need for both accurate and explainable classification.

5.5 Comparison with previous works

The hybrid malware detection method presented in this study takes things a step further compared to previous research by blending static and dynamic analysis with ensemble learning. This combination helps achieve better accuracy, scalability, and a clearer understanding of how the system makes decisions. For example, M and Chandran (2024)[20] AndroPack framework achieved 95% accuracy using a hybrid strategy, but their methodology had some limitations by reliance on a particular dataset (KronoDroid), which may not generalize well across various malware families. In contrast, our work used a large and balanced dataset that includes both old and new risks, resulting in increased robustness. Similarly, Alamro et al. (2023)[15] used ensemble approaches like LS-SVM and KELM in their AAMD-OELAC model, which demonstrated great accuracy but they didn't work with dynamic behavioral aspects, limiting the framework's response to runtime threats. This made their framework less effective when it came to detecting threats that only reveal themselves at runtime. Our approach improves on this by combining those dynamic behaviors with static features, allowing the system to better catch hidden or obfuscated malware.

Moreover, we used advanced data preprocessing steps, such as t-test-based feature filtering and SMOTE-based oversampling, which are not used in other recent works, such as PermDroid (Mahindru et al., 2024)[21], which focuses on permission and user metadata features but lacks deep dynamic behavioral context. On top of that, we used SHAP values to make the model's decisions more explainable, which gives cybersecurity analysts the insights they need to trust and fine-tune the detection process. Altogether, this work fills important gaps in recent research and offers powerful solution for detecting Android malware in today's evolving threat landscape.

Chapter 6

Conclusion

6.1 Conclusion of the study

This research showed that machine learning models can detect and classify malware in the Android ecosystem, before honing in on two of the most frequent classes of malware: Adware and Trojans. In this paper, a statistical t-test was used for filtering features. Class balancing has been done using SMOTE and feature scaling was done using MinMaxScaler. It also formed the basis for the careful curation that will later ensue, important for enhancing both the generalizability and stability of resulting classifiers, hence better representation for minority and majority classes alike.

It is evident from the experimental results that advanced models such as Random Forest, XGBoost, and CatBoost outperform traditional linear methods representative, such as Logistic Regression, on nearly all key performance metrics: Accuracy, Precision, Recall, F1, Balanced Accuracy, and ROC AUC. Of all, Random Forest attained the highest performance scores, therefore being robust in terms of modeling complex relationships in the data. At the same time, the small margin between both the gradient boosting models, XGBoost and CatBoost, underlines the general efficacy of the ensemble strategy on dealing with both diverse and high-dimensional feature spaces.

Perhaps one of the most striking things is the importance of interpretability. Although most of these ensemble methods really work like black boxes, SHAP-based analyses underlined the relative contributions of every feature, giving insight into how the classifier was making decisions. Such explanations build up the trust and transparency but at the same time drive future versions of feature engineering, where practitioners can further refine models by putting more attention on the attributes most indicative of malicious or benign behavior. This is an iterative, feedback-driven approach toward feature selection and model tuning, and it is very likely to gain even more importance with malware evolution in terms of complexity and diversity.

This study confirms that machine learning is a very strong backbone for Android malware detection, and thus directly finds its applications in cybersecurity solutions. It seems therefore that these ensembles might act as powerful real-time classification engines, flagging most threats correctly without a high number of false positives. Since the mobile landscape keeps growing in size, possible extensions of the present work might be carried out in three ways: adding other malware families, incorporating dynamic analysis features, and adopting either a federated or transfer learning

paradigm to cope with the threat environment updated in a continuous manner.

6.2 Significance of the Research

Recently, especially variants like Adware and Trojan, the ever-increasing evolution of Android malware has posed a significant threat to the privacy, security, and trustworthiness of mobile systems. Smartphones carry sensitive personal information and serve as a gateway to important business processes; thus, effective mechanisms for the detection and mitigation of these threats are highly desirable. This research, which leverages machine learning algorithms for malware identification, serves to be one of the definitive finding steps in safeguarding mobile users against rapidly changing malicious actors. By systematically applying several classifiers and measuring their relative performance under strict experimental conditions, the study forms a good backbone on which more advanced, large-scale detection systems can effectively be carried out.

The most important point of significance lies in the design of the data preprocessing strategy, combining statistical feature screening, class balancing through SMOTE, and feature scaling. This methodology directly addresses some of the pressing challenges in malware detection, ensuring high-quality data and reducing the impact of a skewed class distribution. It contributes, in turn, towards actionable insights by practitioners and academics on how to optimize the pre-processing steps, increasing the chances of coming up with more generalizable models with better performance. The completeness of the pipeline underlines the importance of transparent and reproducible experimentation critical for the advances of the wider cybersecurity field.

The study also shows the value of explainable machine learning in cybersecurity. Techniques such as SHAP (SHapley Additive exPlanations) do more than enhance academic rigor: they can guide practitioners in identifying which features or behaviors most strongly predict malicious activity. This interpretability is indispensable in regulated environments where auditors or stakeholders demand clear justifications for automated decisions. Moreover, the key indicators of malicious behavior pinpointed by the study allow the development of more focused defensive measures, the patching of vulnerabilities, and adaptation of protection against freshly emerging threats.

The implications of this research go beyond the topic of detection in Android malware. The findings illustrate how a judicious integration of ensemble methods with well-curated training data can yield highly accurate, robust, and interpretable classification models. Hence, such insights can be transferred to a wide variety of areas in cybersecurity, from network intrusion detection to phishing email filtering. As a result, it improves immediate defenses against Android malware but also spurs innovation and cross-pollination of techniques across the entire domain of modern threat detection.

6.3 Future Works

The following points outline potential directions for future research based on the findings:

- **Extension to Multiple Malware Families:** While this study focused on distinguishing Adware and Trojan samples, future work could involve expanding the classification to include additional malware types or the full spectrum of Android threats. Such broader coverage would improve the generalizability of the detection pipeline.
- **Integration of Dynamic Analysis:** Incorporating dynamic features (e.g., system call traces, network traffic patterns) may capture behaviors undetectable through static features alone. Research on hybrid static–dynamic detection methods could reveal more complex malware tactics.
- **Federated Learning Approaches:** To address privacy and scalability concerns, future studies might consider federated learning frameworks that train models directly on distributed devices or servers, minimizing the transfer of sensitive data while still improving overall detection accuracy.
- **Explainable AI Advancements:** Building upon the SHAP analysis, future work could explore other explainable AI methods or refine attribution techniques for better clarity and trust in high-stakes security environments.
- **Real-Time Deployment and Efficiency:** Evaluating model inference times and resource consumption under real-world constraints would provide insights into deploying lightweight yet effective models on mobile devices. Optimizations such as quantization or pruning could be investigated.
- **Continuous Model Updates:** Given the rapid evolution of Android malware, a continuous retraining pipeline or online learning approach could help models adapt to newly emerging threats, ensuring sustained protection over time.

Bibliography

[1] D. Barrera, H. G. Kayacik, P. C. Van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," 2010, pp. 73–84. DOI: 10.1145/1866307.1866317.

[2] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," 2011, pp. 627–638. DOI: 10.1145/2046707.2046779.

[3] S. Smalley and R. Craig, "Security enhanced (se) android: Bringing flexible mac to android," in *NDSS Symposium*, San Diego, CA, 2013.

[4] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of android malware through static analysis," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*, 2014. DOI: 10.1145/2635868.2635869.

[5] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in android," *Journal of Parallel and Distributed Computing*, vol. 103, pp. 22–31, 2016. DOI: 10.1016/j.jpdc.2016.10.012.

[6] D. Dhanasekar, F. Di Troia, K. Potika, and M. Stamp, "Detecting encrypted and polymorphic malware using hidden markov models," in *Computer Communications and Networks*, 2018, pp. 281–299. DOI: 10.1007/978-3-319-92624-7_12.

[7] E. C. Bayazit, O. K. Sahingoz, and B. Dogan, "Neural network based android malware detection with different ip coding methods," in *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2021, pp. 1–6. DOI: 10.1109/HORA52670.2021.9461302.

[8] M. Dhalaria and E. Gandotra, "A hybrid approach for android malware detection and family classification," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 6, no. 6, p. 174, 2021. DOI: 10.9781/ijimai.2020.09.001.

[9] R. Feng, S. Chen, X. Xie, G. Meng, S. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1563–1578, 2021. DOI: 10.1109/tifs.2020.3025436.

[10] S. J. Hamdi, I. M. Ibrahim, N. Omar, *et al.*, "A comprehensive study of malware detection in android operating systems," *Asian Journal of Research in Computer Science*, pp. 30–46, 2021. DOI: 10.9734/ajrcos/2021/v10i430248.

[11] J. Singh, T. Gera, F. Ali, D. Thakur, K. Singh, and K. Kwak, "Understanding research trends in android malware research using information modeling techniques," *Computers, Materials & Continua*, vol. 66, no. 3, pp. 2655–2670, 2021. DOI: 10.32604/cmc.2021.014504.

[12] A. Sharma and N. Kapoor, "Approach for predicting mobile malware," in *4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, Greater Noida, India, 2022. DOI: 10.1109/icac3n56670.2022.10074091.

[13] F. Ullah, G. Srivastava, and S. Ullah, "A malware detection system using a hybrid approach of multi-heads attention-based control flow traces and image visualization," *Journal of Cloud Computing: Advances Systems and Applications*, vol. 11, no. 1, 2022. DOI: 10.1186/s13677-022-00349-8.

[14] S. Abhishek, A. Rajendran, A. Sha, T. Anjali, and A. K. Nair, "Enhancing android security: Dynamic analysis for resilient defenses using machine learning," in *7th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, India, 2023. DOI: 10.1109/iceca58529.2023.10395050.

[15] H. Alamro, W. Mtouaa, S. Aljameel, A. S. Salama, M. A. Hamza, and A. Y. Othman, "Automated android malware detection using optimal ensemble learning approach for cybersecurity," *IEEE Access*, vol. 11, pp. 72 509–72 517, 2023. DOI: 10.1109/ACCESS.2023.3294263.

[16] N. Z. Gorment, A. Selamat, and O. Krejcar, "Obfuscated malware detection: Impacts on detection methods," in *Communications in Computer and Information Science*, 2023, pp. 55–66. DOI: 10.1007/978-3-031-42430-4_5.

[17] A. Khan and I. Sharma, "Trustdroid: Enhancing trust and privacy for android mobile phones by preventing riskware and sms malware," in *2nd International Conference on Futuristic Technologies (INCOFT)*, Belagavi, Karnataka, India, 2023. DOI: 10.1109/incoft60753.2023.10425220.

[18] A. Sabbah, A. Taweel, and S. Zein, "Android malware detection: A literature review," in *Communications in Computer and Information Science*, 2023, pp. 263–278. DOI: 10.1007/978-981-99-0272-9_18.

[19] W. Huang, W. Tang, H. Chen, H. Jiang, and Y. Zhang, "Unauthorized microphone access restraint based on user behavior perception in mobile devices," *IEEE Transactions on Mobile Computing*, vol. 23, no. 1, pp. 955–970, 2024. DOI: 10.1109/tmc.2022.3221463.

[20] M. M and S. Chandran, "Andropack: A hybrid method to detect packed android malware with ensemble learning," in *12th International Symposium on Digital Forensics and Security (ISDFS)*, San Antonio, TX, USA, 2024. DOI: 10.1109/isdfs60797.2024.10527328.

[21] A. Mahindru, H. Arora, A. Kumar, *et al.*, "Permdroid: A framework developed using proposed feature selection approach and machine learning techniques for android malware detection," *Scientific Reports*, vol. 14, no. 1, 2024. DOI: 10.1038/s41598-024-60982-y.