People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

Kasdi Merbah Ouargla University

Faculty of new information and telecommunication Technologies

Department Of:

Informatique

2nd year MASTER

Speciality: Computer Sciences: Network Administration and Security

Submitted by:

- BOUKHECHEBA FARES

- BENMOUSSA AHMED REDOUANE

DETECTION AND CLASSIFICATION ANDROID MALWARE BY XGBOOST MODEL

Presented before the jury composed of:

Supervisor: Mr BOUKHAMLA Akram

President of the Jury: Mr.KAHLSNANE Fares

Examiner: Mr.BENKADOUR Mohamed Kamel

2024/2025


بسم الله الرحمن الرحيم

Acknowledgements

We would like to express our deep gratitude and thank you very much for your support and valuable guidance throughout this period as our Superintendent. I have been a strong pillar and source of inspiration for us in writing this memo, and we wish to extend our warm thanks for all your efforts. You have guided and motivated us in an exceptional way, and have not hesitated to share your experience and deep knowledge of the subject. We had full confidence in your ability to guide us and help us overcome the various challenges we faced. We have always been dedicated to your task as a supervisor, and you have always been available to listen and answer our questions and queries with patience and interest. It has provided us with the necessary guidance and the opportunity to grow and develop in the area we are studying.

Abstract

The Android operating system is one of the most widely used platforms on smart devices worldwide, making it a prime target for malware attacks. With the increasing complexity and frequency of security threats, there is a pressing need to adopt advanced analysis techniques to enhance the security of this system. In this context, the importance of malware analysis through static, dynamic, and hybrid methods becomes evident, along with the integration of artificial intelligence techniques particularly machine learning for early and accurate detection.

The Kronodroid dataset, one of the largest specialized databases in this field, was used to train and test a machine learning model based on the XGBoost algorithm. The adopted methodology includes steps such as data acquisition, preprocessing, and feature selection before applying the model.

The study concluded that the use of machine learning especially when combined with hybrid analysis of software effectively enhances the ability of cybersecurity systems to detect Malware software and accurately understand its behavior, thereby supporting the development of more efficient preventive solutions for the Android platform.

Keywords: Android, Malware, static analysis, dynamic analysis, hybrid analysis, machine learning, Kronodroid, XGBoost.

Resumé

Le système d'exploitation Android est l'une des plateformes les plus utilisées sur les appareils intelligents dans le monde, ce qui en fait une cible de choix pour les attaques de logiciels malveillants. Avec la complexité croissante et la fréquence des menaces de sécurité, il devient impératif d'adopter des techniques d'analyse avancées pour renforcer la sécurité de ce système. Dans ce contexte, l'importance de l'analyse des logiciels malveillants à travers des méthodes statiques, dynamiques et hybrides devient évidente, en particulier lorsqu'elle est combinée avec des techniques d'intelligence artificielle, notamment l'apprentissage automatique, pour une détection précoce et précise.

Le jeu de données Kronodroid, l'une des plus grandes bases de données spécialisées dans ce domaine, a été utilisé pour entraîner et tester un modèle d'apprentissage automatique basé sur l'algorithme XGBoost. La méthodologie adoptée comprend des étapes telles que l'acquisition de données, le prétraitement et la sélection des caractéristiques avant l'application du modèle.

L'étude a conclu que l'utilisation de l'apprentissage automatique, notamment lorsqu'elle est combinée avec une analyse hybride des logiciels, améliore efficacement la capacité des systèmes de cybersécurité à détecter les logiciels malveillants et à comprendre avec précision leur comportement, contribuant ainsi au développement de solutions préventives plus efficaces pour la plateforme Android.

Mots-clés : Android, logiciels malveillants, analyse statique, analyse dynamique, analyse hybride, apprentissage automatique, Kronodroid, XGBoost.

ملخص

يُعد نظام التشغيل أندرويد من أكثر المنصات استخدامًا على الأجهزة الذكية في العالم، مما يجعله هدفًا مفضلاً لهجمات البرمجيات الخبيثة. ومع تزايد تعقيد وتكرار التهديدات الأمنية، يصبح من الضروري اعتماد تقنيات تحليل متقدمة لتعزيز أمان هذا النظام. وفي هذا السياق، تبرز أهمية تحليل البرمجيات الخبيثة من خلال الطرق الساكنة والديناميكية والهجينة، خاصةً عند دمجها مع تقنيات الذكاء الاصطناعي، وعلى وجه الخصوص التعلم الآلي، من أجل تحقيق كشف مبكر ودقيق.

تم استخدام مجموعة بياناتKronodroid ، وهي من أكبر قواعد البيانات المتخصصة في هذا المجال، لتدريب واختبار نموذج تعلم آلي يعتمد على خوارزمية .XGBoost وتشمل المنهجية المتبعة مراحل مثل الحصول على البيانات، والمعالجة المسبقة، واختيار الخصائص قبل تطبيق النموذج.

وخلصت الدراسة إلى أن استخدام التعلم الآلي، لا سيما عند دمجه مع التحليل الهجين للبرمجيات، يعزز بشكل فعال قدرة أنظمة الأمن السيبراني على اكتشاف البرمجيات الخبيثة وفهم سلوكها بدقة، مما يساهم في تطوير حلول وقائية أكثر كفاءة لمنصة أندرويد.

الكلمات المفتاحية: أندرويد، البرمجيات الخبيثة، التحليل الساكن، التحليل الديناميكي، التحليل الهجين، التعلم الآلي، Kronodroid، XGBoost.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

OS = Operating System

APK = Android Package Kit

API = Application Programming Interface

APT = Advanced Persistent Threat

Malware = Malicious Software

App = Application

Apps = Applications

ML = Machine Learning

AOSP = Android Open Source Project

ART = Android Runtime

AOT = Ahead-Of-Time Compilation

HAL = Hardware Abstraction Layer

SSL = Secure Sockets Layer

TLS = Transport Layer Security

GPS = Global Positioning System

VM = Virtual Machine

UI = User Interface

OpenGL ES = Open Graphics Library for Embedded Systems

Libc = C Standard Library

AV = Antivirus

DEX = Dalvik Executable

XML = eXtensible Markup Language

3D = Three-Dimensional

ID = Identifier

SMS = Short Message Service

HTTP = HyperText Transfer Protocol

HTTPS = HyperText Transfer Protocol Secure

PCAP = Packet Capture

CPU = Central Processing Unit

JSON = JavaScript Object Notation

APKTool = Android Package Tool

JADX = Java Decompiler for Android

GDB = GNU Debugger

AI = Artificial Intelligence

SVM = Support Vector Machines

K-NN = K-Nearest Neighbors

PCA = Principal Component Analysis

t-SNE = t-Distributed Stochastic Neighbor Embedding

CSV = Comma-Separated Values

XGBoost = Extreme Gradient Boosting

ExtraTree = Extremely Randomized Trees

General Introduction

1) Context:

Android applications are among the most widely used and widespread applications around the world, due to the open-source nature of the operating system [1] and the ease of developing applications on it and distributing them through the Google Play Store or other platforms [2]. Statistics show that Android powers more than 70% of smartphones globally [3], making it an attractive environment not only for developers but also for attackers. This massive adoption has encouraged innovation and accessibility in mobile application development, but it has also brought along significant security concerns [4].

2) Problem Statement:

With the increasing reliance on Android devices for daily activities and personal data storage, the platform has become a primary target for cyberattacks [5]. Among the most dangerous threats are malware attacks, which aim to exploit vulnerabilities in applications or the operating system to access, steal, or manipulate user data [6]. These malicious programs include spyware, Trojans, ransomware, and other sophisticated forms of attacks that have grown in complexity and stealth in recent years [7].

Studies reveal that many users are tricked into downloading seemingly legitimate applications that contain hidden malicious code, often bypassing initial security checks on official app stores [8]. As a result, the early detection of such threats has become a critical challenge in cybersecurity, particularly within the Android ecosystem [9].

3) Objectives:

This study aims to analyze Android malware using two fundamental approaches:

Static analysis, which involves examining the application’s code and structure without executing it.

Dynamic analysis, which observes the behavior of the application during runtime in a controlled environment.

By combining both approaches, we aim to provide a comprehensive understanding of how malware operates, identify potential indicators of compromise, and contribute to building more robust and proactive security solutions.

General Introduction

This research seeks to highlight key security challenges within the Android ecosystem, analyze real malware samples using advanced techniques (including machine learning), and propose practical solutions to help strengthen defense mechanisms and improve early threat detection capabilities.

Chapter I: Android Security Overview

I.1 Introduction to Android System

Android is an open-source operating system developed by Google, based primarily on the Linux Kernel [10]. The first version of Android was launched in 2008, and since then it has become the most widespread system among smart devices, powering more than 70% of mobile phones around the world [11].

The Android system was distinguished by several characteristics that made it popular globally:

-Flexibility and customization: developers and manufacturers can modify the system according to their needs [12].
-Supporting an open source community: provides the opportunity to contribute to the development of the system [13].
-Google services integration: such as Google Play, Google Maps and Google Assistant [14].

The openness of the system has also led to the emergence of increasing security challenges, which requires an in-depth study of the Android system infrastructure and an understanding of its protection mechanisms [15].

Figure 1. Android Logo

I.2 Android System Architecture

The Android system consists of four to five interconnected layers that make up its basic structure [16], [17]:

1) Linux Kernel

The kernel is the basic layer in the Android system, and acts as an intermediary between the hardware and the rest of the software components of the system [18]. The kernel includes device drivers such as keyboard, monitor, Bluetooth, camera, and storage. The kernel also manages basic processes such as memory management [19], process scheduling, security, and access control mechanisms. Android relies on the Linux kernel to ensure stability and high performance, with specific customizations to meet mobile device requirements.

2) Hardware Abstraction Layer - HAL

This layer acts as an intermediary between the kernel and hardware devices, unifying interaction with device components such as the camera, Bluetooth, and sensors. It provides standard interfaces that allow the system to communicate with various devices without having to modify the kernel directly [20].

3) Android operating environment (Android Runtime - ART)

The operating environment is the layer responsible for executing applications. Starting with the release of Android 5.0 (Lollipop), the ART environment replaced the Dalvik Virtual Machine. ART uses Ahead-of-Time Compilation (AOT) technology to convert Bytecode into machine code during installation, improving performance and reducing power consumption [21]. ART also supports improvements such as efficient memory management and garbage collection to ensure smooth operation of applications.

4) Native Libraries

This layer contains a set of libraries written in languages such as C and C++ to achieve high performance and efficiency [22]. These libraries include:

-**SQLite:** for local database management.
-**OpenGL ES:** to support 3D graphics.
-**WebKit:** To interpret and display web pages in browsers.
-**Media Framework:** To play and record audio and videos.
-**Libc:** Standard C library to support basic operations.
-**SSL/TLS**: To provide secure connections [23].

These libraries support applications and the framework, allowing fast and straightforward performance with hardware.

5) Java API Framework

This layer provides application programming interfaces (APIs) that allow applications to interact with system components [24]. Main services include:

-**Location Manager:** To determine geographical locations using GPS or networks.
-**Camera API:** To control the camera and take photos or videos.
-**Notification Manager:** To manage notifications sent to the user.
-**Activity Manager:** To manage the lifecycle of applications and windows.
-**Content Providers:** To share data between applications.
-**Telephony Manager:** To manage communication functions such as calls and sending messages.

The framework acts as a bridge between applications and the rest of the system, enabling the development of feature-rich applications.

These libraries support applications and the framework, allowing fast and straightforward performance with hardware.

6) Applications Layer

This is the top layer that interacts with the user directly. Include:

-**Pre-installed apps:** such as Google apps (Gmail, YouTube, Maps) or device manufacturer apps (such as Samsung or Xiaomi apps).

-**User-installed apps:** Downloaded from the Google Play Store or third-party sources such as alternative app stores (such as APKMirror) or manual installation (Sideloading) [25].

These applications rely on the software framework and native libraries to access system functionality and deliver an integrated user experience.

What was mentioned previously is shown in the following ***Figure 2***:

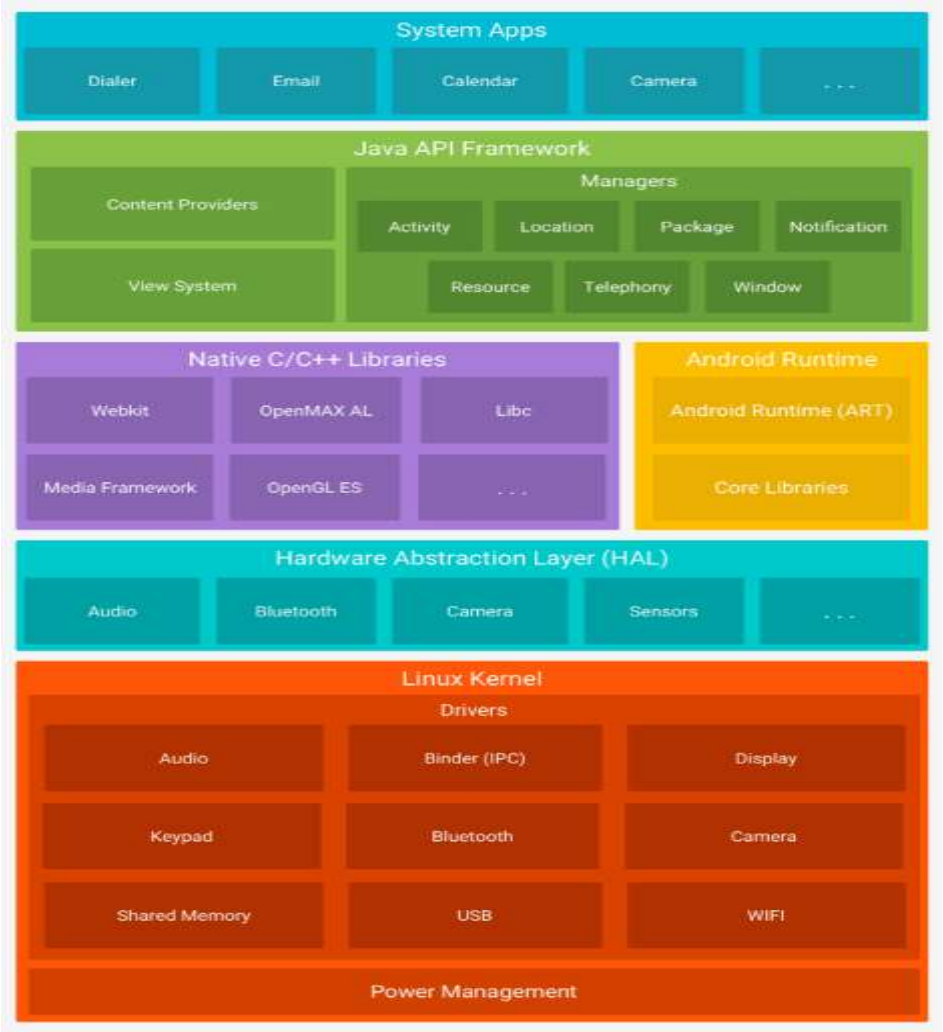

Figure 2. Android platform architecture

I.3 Android Application Mechanism (APK)

Android applications are developed using languages such as Java and Kotlin, and the application components are compiled into a file with the .apk extension (Android Package) [26], [27]. This package includes:

1) Compiled code (classes.dex files):

The code (written in languages like Java or Kotlin) is translated into a special syntax called DEX (Dalvik Executable), which is a syntax that the Android system understands to run the application [28].

2) Androidmanifest.xml file:

This file is similar to an application's "ID card", specifying:

-Application information (such as name and version).

-Required permissions (such as access to the camera, microphone, or messages).

-Application components (such as screens or services).

It is essential for the system to understand how the application should interact with the device and user [29], [30].

3) Resource files:

Includes files used by the application such as:

-Images (such as icons or backgrounds).

-Language files (to support multiple languages).

-User interface layouts (UI Layouts) that define the look of screens.

They are bundled within the APK and accessed by the application through the R class [31].

4) Digital Signature:

The APK file is signed with a digital certificate to ensure its security. This signature ensures that the file has not been modified after being signed by the developer [32], protecting the application from tampering.

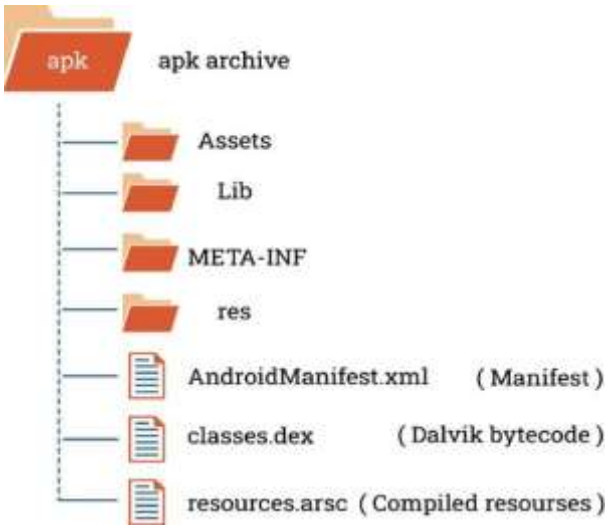

Figure 3. APK File Structure

How does the application work after installation?

When you install the app on an Android device, it runs inside an isolated environment called Sandbox.

A sandbox is a security environment that limits an application's interaction with other applications or sensitive system components (such as internal system files).

The app can only access anything outside its scope through specific interfaces (APIs) and with the user's explicit permission (such as requesting permission to access the camera or contacts).

This design protects user privacy and prevents applications from causing damage to the system or other applications.

I.4 Security Threats on Android

I.4.1 Types of Malware

The Android environment suffers from a great diversity of types of security threats, most notably:

-**Viruses:** They self-replicate and spread themselves among other files or applications [33].
-**Trojans:** Applications that appear normal but contain Malware code intended to steal user information [34].
-**Spyware:** It monitors user activity such as calls and geographical locations without his knowledge [35].
-**Ransomware:** encrypts files and asks for a sum of money to re-decrypt [36].

Example: The "Joker" software, which was discovered in several applications on the Google Play Store, was registering users for paid services without their knowledge [37].

I.4.2 Malware Diffusion Methods

Malware spreads on Android in a number of ways, most notably:

-**Fake Apps:** Popular apps are imitated and attached with malware [38].

-**Phishing links:** Phishing links are sent via messages or email [39].

-**Malvertising:** Ads are used as a means of spreading Malware code [40].

-**Sideloading:** Installing applications from unofficial sources is one of the biggest sources of threats [41].

I.5 Importance of Malware Analysis

I.5.1 Early Detection of Threats

Malware Analysis allows us to understand the abnormal behavior that may be caused by a particular application. Using tools like static analysis and dynamic analysis, professionals can detect malware before it actually harms the system or user [42].

I.5.2 Understanding Malware Behavior

Understanding the mechanisms of malware helps:

-Identify the security vulnerabilities you exploit.
-Develop detection algorithms based on behavior, not just signature.
-Improved built-in protection technologies such as Google Play Protect [43].
-Constantly updating protection systems to comply with new threats.

Example: Parsing the onCreate () function in a Malware application could detect an attempt to access text messages or send location data to an external server.

I.6 Conclusion

With its widespread adoption and open nature, the Android system offers a rich environment for application development, but it also faces increasing security challenges. In this chapter, we explored the fundamental structure of the Android system and how applications operate within it, as well as the security threats it faces particularly from malware and its methods of propagation. We also addressed the importance of malware analysis as a crucial tool for early threat detection and understanding malicious behavior.

Understanding these core concepts is a necessary step for any researcher or developer aiming to enhance the security of Android applications or to counter cyberattacks targeting this system serving as a foundation for the upcoming chapters of this work.

Chapter II : Malware Analysis Techniques

II.1 Introduction

Malware analysis is an essential step in the field of information security, as it enables us to understand how this software works and detect its Malware behavior. Analysis techniques vary into three main types: static analysis, dynamic analysis, and hybrid analysis, where each type is characterized by its own methodology and tools that complement each other to provide comprehensive and accurate analysis. Due to the increasing spread of malware on the Android system, there has been a need to develop multiple effective techniques to detect these threats, which has made it necessary to provide a detailed explanation of each technique [44], [45].

II.2 Static Analysis

II.2.1 Definition

Static analysis is the process of analyzing Android applications without actually running them. This is done by examining the source code, bytecode, and application package (APK) files, with the aim of identifying indicators that may indicate the presence of Malware activity [46].

This type of analysis is useful because it is relatively fast, does not require a special operating environment, and allows the entire structure of the application to be understood, but it may not detect Malware behaviors that are only activated during operation [47].

Static analysis is used because:

-Does not require the application to run, making it safe when dealing with dangerous software.
-Provides a quick and comprehensive overview of the internal structure of the application, including source code, permissions, and helper files.
-Helps in early detection of suspicious codes or Malware commands buried within the application.
-Effective in analyzing a large number of applications in a short time using automated tools [48].

Example: Through static analysis, unusual permissions (such as sending SMS messages or accessing the camera) can be detected in a simple application, raising suspicion [49].

II.2.2 Static analysis components

Static analysis includes several basic elements [50]:

APK file: It is the package that includes all application files (code, images, data).

AndroidManifest.xml file: Contains the definition of application components (activities, services, permissions).
DEX (Dalvik Executable) files: contain the code that executes on an Android device.
Resource files: images, languages, XML formats used within the application.
External libraries: such as Java or C/C++ libraries associated with the application.

II.2.3 Static Analysis Steps

Decompress APK file:
Use tools like Apktool [51] to decompress the APK file and extract internal components.
Read AndroidManifest.xml file:
To specify permissions, operating methods, and access granted to the application.
Source code analysis or DEX:
Suspicious functions (Methods), commands (Instructions), and API calls are scanned [52].
Search for Indicators of Compromise (IOCs):
Such as suspicious addresses, stored passwords, or encryption/decryption codes.
Obfuscation determination:
These are code hiding techniques to mislead parsing tools, such as name obfuscation or code reverse merge.

II.2.4 AndroidManifest.xml file

This file is present in every Android app and acts as an "ID" for the app.

Contains:

-**Permissions:** such as accessing your camera, location, or messages.

-**Activities and Services:** Interface components or back-end processes.

-**Intents and Receivers:** Mechanisms for communicating with the system or other applications.

-**Features:** Such as checking the application's compatibility with the Android version or device type.

Analyzing this file is necessary because it reveals the capabilities of the application, and sometimes bad intentions, such as requesting permissions that are not in line with its functionality.

How to analyze AndroidManifest.xml file?

1) Open File:
You can open AndroidManifest.xml file using any text editor or specialized tools such as:
-**APKTool:** To disassemble the application package and view the file.
-**Jadx**: To convert APK file to source code and view file.

2) Search for suspicious permissions:

Permissions are requests made by the application to access the device's resources. Some permissions may be suspicious if they do not fit the application function. Here are examples of suspicious permissions:

-**READ_SMS:**

It allows the app to read text messages.

Why would you be suspicious? If the app is not a messaging app or linked to messages.

-**SEND_SMS:**

It allows the app to send text messages.

Why would you be suspicious? The Malware app may use it to send text messages to premium numbers (costing you money).

-**ACCESS_FINE_LOCATION:**

The app allows access to your exact location via GPS.

Why would you be suspicious? If the app does not need to know your location (such as flash app or calculator).

-**INTERNET:**

The app allows access to the Internet.

Why would you be suspicious? If the app does not need an internet connection (such as a simple feedback application)

3) Search for suspicious ingredients:

-**Activities:**

Find unknown or suspicious screens.

-**Services:**

Find services that work in the background for no obvious reason.

-**Broadcast Receivers:**

Find components that receive suspicious signals (such as receiving text messages).

II.2.5 Source code analysis or DEX

Source code analysis or DEX, is the process of examining the code or file of a Dalvik Executable application with the aim of:

-Detecting harmful behavior (Malware Behavior).
-Understand how the application works.
-Extract important or dangerous information.
-Detect modifications or forgery in the application
It is often used in malware analysis, application security, or reverse engineering.

When analyzing a DEX, several features are extracted that are useful for determining whether an application is Malware or suspicious. These features can be classified into several types:

1- Features of suspicious methods:
Example:
-Runtime.exec() → Execute system commands.
-DexClassLoader → Dynamic external code loading.
-getDeviceId() → Collect device ID.
-sendTextMessage() → Send SMS messages.
-MediaRecorder.start() → Record audio/video.

2- Code features (Instructions):
Example: invoke-static, invoke-direct, invoke-virtual.
It is used to call various functions and may indicate the use of dangerous or hidden functions. Instructions related to abnormal flow control (eg goto, if-nez) could be an attempt to mislead or encrypt.

3- Features of API calls (API Calls):
All API calls are extracted and checked against a list of dangerous or system-specific APIs.
Example:
-Network: HttpURLConnection, OkHttp.
-Privacy getAccounts(), getSimSerialNumber().

II.3 Dynamic Analysis

II.3.1 Definition

Dynamic analysis is a technique in which an application is run in a monitored environment (sandbox) and its behavior is followed during operation. The goal is to catch any Malware interaction like sending data, installing files, accessing camera or location, etc.

It shows the real behavior of the application after booting, but it requires a safe and isolated environment to avoid any possible damage.

Dynamic analysis is used because:

-It monitors the behavior of the application during actual operation, allowing Malware activity that does not appear in the code to be detected directly.
-Useful for detecting malware that uses camouflage or activates Malware behavior after certain conditions are met.
-It allows examining interaction with the system, such as access to files, networks, user interfaces, etc.

Example: Some applications may not show any Malware behavior in the code, but during operation they connect to external servers or send sensitive data.

II.3.2 Dynamic analysis components

-**Logcat Output (logcat.txt):** System log file in Android.
-**System Calls Trace (strace.txt):** A file containing all system calls made by the application.
-**Network Traffic (pcap file - network. pcap):** A file containing all network traffic.
-**File System Changes (fs_diff.txt):** A log showing changes in files.
-**Monitoring Logs (api_monitor. json / .txt):** Files tracking important API calls.
-**Battery/CPU/Memory Usage (performance.log):** A file that records resource consumption.
-**Screenshots/Record Videos:** Documentation of the user interface during use.
-**Intents Log:** A file that records messages between activities, services and receivers.

II.3.3 Dynamic Analysis Tools

***Android Emulators:**
Android Studio Emulator or Genymotion to run the app in a simulated environment.
***Monitoring Tools:**
-**Wireshark:** to monitor network traffic.
-**Frida:** To perform dynamic injection and monitor the recalls of the function.
-**Drozer:** To test the security of Android applications.
***Memory Analysis Tools:**
GNU Debugger or Valgrind to analyze memory use.
***Mobile Security Framework:**
Supports dynamic analysis as well as static analysis.

II.3.4 Dynamic Analysis Steps

-Running the application:
Use a simulator or actual device to run the application.
-Behaviour Control:
Use tools like Wireshark to monitor network traffic.
Use Frida to monitor the recalls of varieties.
-Simulation of user interactions:
Use tools like Monkey or Appium to simulate clicks and entries.
-Results analysis:
Analyze the data collected to identify suspicious behaviors.
-Threat assessment:
Determine whether behaviors indicate the presence of malware.

II.4 Comparison of static and dynamic analysis

This table represents the difference between static and dynamic analysis.

Components	Statics Analysis	Dynamic Analysis
Target code execution	Not possible	Possible
Type of execution	Code investigation – No runtime operations	Runtime investigation and operations
Time required	More time required to go through the lines of codes	Less time since using an automation method
Input Type	Binary files, scripting language files, etc	Memory snapshots, runtime API data
Advantage	Fewer resources and time Exam all possible execution tracks through one file: Android manifest.xml. Can help to detect many vulnerabilities i.e. private data leaks, unauthorized access to protected or private resources, and intent injection	Deep examination and higher discovery rate with obscure malware spot More precise due to the current execution of the program to reach appropriate code coverage
Disadvantage	Constrained signature database and can identify only within the scope of the known malware types	Power consumption and more time to perform the process
Accuracy of the results	Low rate of accuracy compared with dynamic analysis	Improved than static analysis

Table 1. Comparison between static and dynamic analysis [66]

II.5 Hybrid Analysis

II.5.1 Definition

Hybrid analysis is an approach that combines the features of static analysis and dynamic analysis in order to achieve a comprehensive understanding of malware behavior. This type of analysis relies on taking advantage of the accuracy and speed of static analysis, in addition to the effectiveness of dynamic analysis in detecting Malware activities that only appear while the application is running.

This method is one of the most effective methods for analyzing modern malware applications, especially those that use camouflage techniques or conditional activation of malware.

II.5.2 Advantages of hybrid analysis

-Comprehensiveness of analysis: Provides a more complete picture of application behavior.
-Detect hidden threats: Thanks to the ability to monitor the code and analyze its implementation in real time.
-Improving detection accuracy: by combining behavior indicators and structure indicators.
-Enhancing automated classification results: Hybrid analysis results can be used as a rich source of input properties in machine learning algorithms.

II.5.3 Hybrid analysis tools

-Mobile Sandbox: Analyzes applications using both static and dynamic methods.
-VirusTotal: combines static analysis through multiple detection engines, and dynamic analysis using a simulated environment.

II.5.4 How to perform hybrid analysis

-**Static analysis:**

*Disassemble the APK application.
*Androidmanifest.xml Inspection.
*Recognize suspicious API calls or excess permissions.

-**Dynamic analysis:**

*Run the application within a virtual environment (emulator or sandbox).
*Record all activities performed by the application during operation (such as accessing files or sending data over the Internet).

-**Consolidation and analysis:**

*Comparison of actual application behaviour with that detected in static analysis.
*Look for inconsistencies or modal activities that were not evident in the code.

II.5.5 Effectiveness of hybrid analysis

Some Malware applications do not show their true behavior until after a certain number of interactions or entering certain data. Static analysis cannot detect these cases, but when using hybrid analysis, the analyst can see the suspicious code as well as detect it during its actual execution.

Some software may use obfuscation techniques to hide its intentions, making it difficult to detect them through static analysis alone, while its actual activity can be detected upon launch.

II.6 Conclusion

Malware analysis is a fundamental component in the field of information security, especially within the Android environment, which is constantly exposed to threats. In this chapter, we reviewed the three main analysis techniques: static analysis, which allows studying the application without executing it; dynamic analysis, which provides deeper insight by monitoring the app's behavior during execution; and finally, hybrid analysis, which combines the advantages of both methods to achieve more accurate and comprehensive results.

Each of these techniques has its strengths and limitations, but their integrated use enhances the effectiveness of malware detection and analysis. This chapter provides both the theoretical and practical foundation for understanding these methods, paving the way for practical applications and advanced analysis in the upcoming chapters, ultimately contributing to improved defense and protection tools in the Android system.

Chapter III: Machine Learning in Android Malware Detection

III.1 Introduction to Artificial Intelligence in Cybersecurity

In recent years, artificial intelligence (AI) has become a mainstay in cybersecurity. As security threats evolve and become more complex, traditional methods based on malware signatures are no longer sufficient. Here comes the role of artificial intelligence, especially machine learning, which enables systems to learn and adapt to new threats by analyzing huge amounts of data and detecting suspicious patterns [52] [53].

In the field of Android applications, where thousands of new applications are developed daily, the use of artificial intelligence is essential to detect malware quickly and efficiently [54].

III.2 Machine Learning in malware detection

III.2.1 Overview

Machine learning is a branch of artificial intelligence that aims to enable systems to learn from data and make decisions without having to program them directly [52]. In the field of malware detection, machine learning models are trained using databases containing applications known to be healthy or Malware. After training, the model can predict whether a new application is harmful or not [55].

III.2.2 Types of machine learning

Machine learning is divided into four main types, and each type is used depending on the nature of the problem and the type of data available [56]:

1) Supervised Learning:

Supervised Learning is based on training data that contains previously known inputs (Features) and outputs (Labels). The goal is for the model to learn the relationship between these inputs and outputs, to later be able to make a correct prediction when new data is introduced [57].

This type is widely used in malware detection, where the model is trained on data containing examples of "Malware" and "Bening" software, to later classify new applications [55].

The model is trained using data containing specific inputs and outputs. Among the most famous types:

A. Classification:

Objective: To classify data into categories or groups.

Examples: Classifying messages as "spam" or "plain mail", or classifying applications as "Malware" and "bening" [55] [58].

Classification types:

-**Binary:** Only two options (such as Yes/No, Malware/Proper).

-**Multi-class:** More than one category (such as classifying types of malware).

B. Regression:

Objective: To predict continuous numerical values based on certain inputs.

Examples: Predicting the rate of infection with a virus after analyzing file properties, or predicting the risk of application behavior.

Among the machine learning algorithms used in the classification are:

1/ Decision Trees:

This algorithm constructs a tree-like structure in which internal nodes represent tests on features, branches represent the outcomes of these tests, and leaf nodes represent class labels. It works by recursively partitioning the data space based on the most discriminative features, allowing for intuitive interpretation and fast inference.

Algorithms based on Decision Trees include the following:

1-Boosting: Boosting is an ensemble technique that improves performance by combining multiple weak learners in a sequential manner. Each new tree focuses on correcting the mistakes of the previous ones. Popular boosting methods include AdaBoost, Gradient Boosting, XGBoost, and LightGBM, all of which can significantly enhance the predictive accuracy of decision tree-based models [59].

2-Binning: is the process of converting continuous numerical data into classes (Intervals or Bins). The goal of binning is to simplify data and improve the performance of some machine learning models. Popular Binning methods include Random Forest.

2/ Support Vector Machines (SVM):

SVM aims to find the optimal hyperplane that best separates the data into distinct classes. It is particularly effective in high-dimensional spaces and is known for its robustness in handling both linear and non-linear classification problems through the use of kernel functions.

3/ K-Nearest Neighbors (K-NN):
A non-parametric algorithm that classifies a sample based on the majority class among its k-nearest neighbors in the feature space. It assumes that similar instances exist in close proximity and relies heavily on distance metrics to make predictions.

4/ Neural Networks:
A parametric, supervised learning algorithm inspired by the structure of the human brain. It consists of layers of interconnected nodes (neurons), including input, hidden, and output layers. Neural networks learn complex patterns in data by adjusting weights through backpropagation and gradient descent. They are especially effective in tasks like image recognition, natural language processing, and time-series prediction.

2) Unsupervised learning:

The data has no known labels or outputs. It is used to discover hidden patterns or groups within data. It is useful for analyzing new applications that have not yet been classified, and is also used to detect abnormal behaviors.

It can be used to group apps according to their behavior, which helps identify suspicious apps that have not yet been classified [58].

In this type, the data does not have a labeled output, and the model aims to detect underlying structure or hidden patterns:

A. Clustering:

Objective: To divide data into groups that share certain characteristics.
Examples: grouping applications by behavior, or grouping users based on their usage patterns.

Popular algorithms:
-K-Means.
-DBSCAN.
-Agglomerative Hierarchical Clustering.

B. Dimensionality Reduction:

Objective: Reduce the number of properties in the data while retaining important information.
Examples: simplifying network data to detect attacks, or compressing large images/data [62].

Popular algorithms:
-PCA (Principal Component Analysis).
-t-SNE (t-Distributed Stochastic Neighbor Embedding).
-Autoencoders (unsupervised neural networks).

3) Semi-Supervised Learning:

Combines the previous two types, where one set of data is tagged and the other is not tagged. This approach is used when the data classification process is costly or time-consuming. It is useful when analyzing a large number of applications, but not all of them have labels.

4) Reinforcement learning:

It is a different type of learning that depends on interaction with a specific environment through reward and punishment. It is not typically used to detect malware, but it is important in cybersecurity in situations such as adaptive defenses or detecting repeated attacks in real time.

Supervised learning is most commonly used in the field of Android malware analysis, due to the availability of tagged datasets such as Kronodroid, which helps train models and achieve accurate results.

III.2.4 Feature extraction (Feature Engineering)

Property extraction means converting application data into a form that can be used in machine learning models [60].

Among the important characteristics used:

-API calls (API Calls).
-Permissions required from the application.
-Network Communications.
-Activities and services declared in the AndroidManifest file.
-Interaction with system files.

Good selection of properties helps improve model performance and accuracy [61].

III.3 Kronodroid Dataset

III.3.1 Definition

The Kronodroid dataset is a dataset intended for cybersecurity research purposes, specifically for analyzing and detecting malware on Android operating systems. It was created to support studies and develop machine learning models aimed at distinguishing between Benign and Malware applications on Android devices [63].

The Dataset includes:
-It contains samples of Android apps, divided into two categories: Malware apps and secure apps.

-Includes data about application behavior, such as required permissions, application programming interfaces (API Calls) used, and patterns of activity within the application.
-It also contains static features such as code analysis, and dynamic features such as application behavior during operation.

Dataset used for:

-They are used to train and test machine learning algorithms to detect malware.

-It helps researchers understand behavior patterns associated with Malware applications compared to secure applications.

-Supports the development of real-time threat detection systems.

III.3.2 Description and Source

The Kronodroid dataset is a publicly available benchmark dataset specifically designed for Android malware detection using machine learning techniques. It was introduced to provide researchers with a large and well-structured dataset that captures a wide variety of real-world Android applications, both benign and Malware. The dataset was collected from reliable and traceable sources such as AndroZoo and VirusTotal, ensuring authenticity and proper labeling [63].

III.3.3 Size and Diversity

Kronodroid contains over 60,000 Android APK files, including both benign and Malware samples. The malware samples span multiple families and categories, such as Trojans, spyware, ransomware, adware, and more. This diversity ensures that machine learning models trained on this dataset can generalize well and detect a wide spectrum of malware behaviors and techniques [63].

III.3.4 Format

The dataset includes raw APK files and pre-extracted features in various formats: CSV, JSON. These features can include permission usage, API calls, intent filters, and other static and dynamic properties. Some versions of the dataset also provide metadata, such as the hash of the APK file, the date it was seen, and labeling from antivirus engines [63].

III.3.5 Role and Relevance

The Kronodroid dataset plays a crucial role in the evaluation of malware detection systems. It provides a standard foundation for training, testing, and comparing machine learning models. Its relevance lies in its size, diversity, and the up-to-date nature of its samples, making it suitable for modern malware analysis. Additionally, its open accessibility encourages reproducibility and transparency in academic research [63].

III.3.6 Processing and Challenges

Before using the Dataset for machine learning, several preprocessing steps are required. These include feature extraction, label normalization, removal of duplicates, and balancing the dataset to address class imbalance.

Researchers may encounter certain challenges such as:

-Obfuscated code in APKs, which makes feature extraction difficult.

-Imbalanced classes, where benign samples vastly outnumber Malware ones.

-Dynamic behaviors that cannot be fully captured by static analysis alone.

-Label inconsistencies due to differences in antivirus engine classifications.

Addressing these challenges is essential to ensure reliable performance of detection models trained on Kronodroid.

III.4 Previous Work

Many recent studies have addressed the topic of malware detection on Android using traditional machine learning techniques. One of the most prominent of these studies is what was presented in the scientific paper called: "Classification Using the Enhanced KronoDroid DatasetEffective and Efficient Android Malware Detection and Category" Submitted by Mudassir Waheed and Sanaa Qadir, where a malware classification and detection model was developed based on an enhanced KronoDroid dataset, which has hybrid (static and dynamic) properties collected from software execution on real hardware, making it Effective in detecting advanced and anti-dynamic malware. In this research, several machine learning algorithms were used, including Random Forest, Decision Tree, and SVM.The ExtraTree algorithm and methods such as mutual information were also used to choose the features that best affect the performance of models.MinMax Scaling technology was applied to improve the distribution of data and reduce the discrepancy between values, which helped reduce the cost.Calculation and performance improvement. This study showed the superiority of the "Random Forest+Minmax+ExtraTree 50" model, as it achieved, using a hyperparamater, accuracy:

-/98.03% for malware detection.

-/87.56% for being classified in 16 categories.

Although the results in binary classification are strong, the model's performance in classifying multiple categories is relatively poor. This may be due to: The imbalance in the categories is clear, as some categories contained only a few dozen samples (such as Trojan-dropper), while others contained thousands of samples (such as Adware).

Research objectives:

This research aims to develop an effective model for the detection and classification of malware on Android using the XGBoost algorithm, taking advantage of the features of this algorithm in dealing with high-dimensional and unbalanced data, focusing on the following

-Improved multi-detection and classification accuracy by combining static and dynamic characteristics in the KronoDroid array, while addressing imbalance using XGBoost's adaptive weights technology.

A comprehensive comparison with the results of the previous study in terms of accuracy, recall, and F1-score, with a focus on underperforming groups to evaluate the model's strength in generalization.

Note: The classification of samples was updated on 10/01/2025

III.5 Conclusion

The field of information security has witnessed significant advancements through the integration of artificial intelligence techniques particularly machine learning in addressing complex cyber threats. In this chapter, we highlighted the role of machine learning in detecting malware on the Android platform, starting from the basic concepts and types of learning, through the feature extraction phase, and finally examining the Kronodroid dataset, which serves as an important reference in this field.

This chapter provides a comprehensive overview of how data can be utilized and analyzed to build intelligent models capable of accurately distinguishing between benign and malicious applications. It also addressed the challenges researchers may face when processing data and applying it in intelligent models. This discussion paves the way for a deeper understanding of classification and prediction techniques, which will be explored in the upcoming chapters and are essential for building effective and practical malware detection systems.

Chapter IV: Implementation and Results

IV.1 Introduction

This chapter aims to clarify the methodology, which was adopted in order to build an effective model for detecting malware in the Android system using machine learning techniques. The steps of Dataset analysis, pre-processing, extraction of properties, selection of algorithms and, finally, model evaluation will be explained.

IV.2 Steps in Implementation

IV.2.1 Data Acquisition:

The "kronodroid_improved_hybrid" dataset was used, which is an enhanced dataset from the original krondroid data that includes dynamic features extracted by running applications on real devices and not a simulated environment, ensuring that the behavior of anti-dynamic malware is captured that avoids implementing its behavior. Malware in simulated environments [64].

The "kronodroid_improved_hybrid" dataset also contains two categories:

1) Whole_BD_leg_mal_clean_v3: It is data intended for binary classification, and determining whether an application is malware or benign (Malware application detection) [63], as it contains:

41,382 applications are malware and 36,755 are benign.

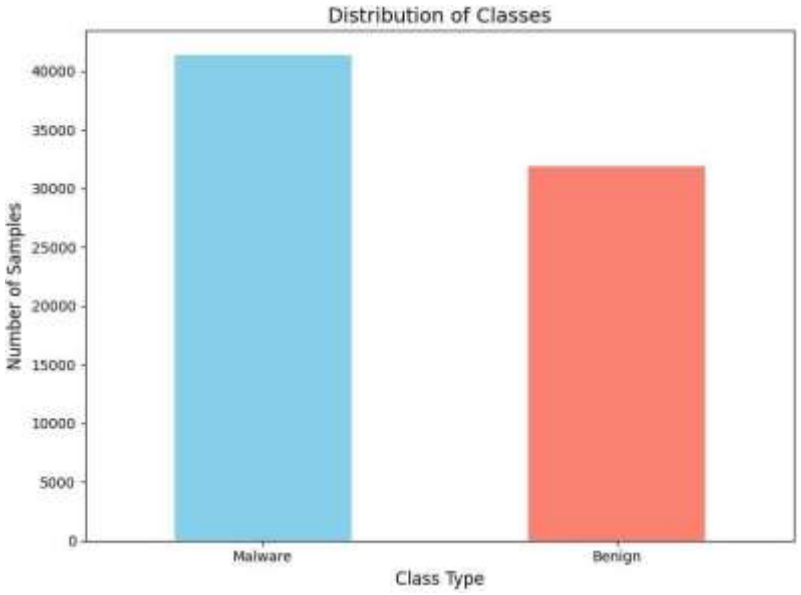

Figure 4. Malware Application Detection

2) Whole_CC_leg_mal_final_v2: This is data intended to classify and identify the malware family (multi-class classification task).The class labels were obtained from online antivirus repositories such as:

-VirusTotal.

-F-Secure.

-FortiGuard.

*/If there were inconsistencies in nomenclature (such as naming a sample as "Riskware" by one repository and "Adware" by another), they were resolved by:

*/Cross-validation with sample labels in other datasets.

*/If the sample did not exist in other datasets, the label with the highest number of votes from the repositories was adopted.

Note: The classification of samples was updated on 10/01/2025.

14 categories of malware families and Blank-Cat have been identified, we mention them as follows:

The classes	Number of Sample
Adware	11183
Trojan	7541
Trojan-SMS	6347
Riskware	5358
Trojan-Spy	4336
Ransomware	2029
Trojan-Banker	1681
Trojan/Riskware	1048
PUA	881
File-Infector	442
Spyware	260
Blank-Cat	166
Trojan-Dropper	46
Scareware	42
Trojan-Backdoor	22

Table 1. Classify and identify the malware family in last update

In addition to the class of benign applications, which contains: **36,755**.

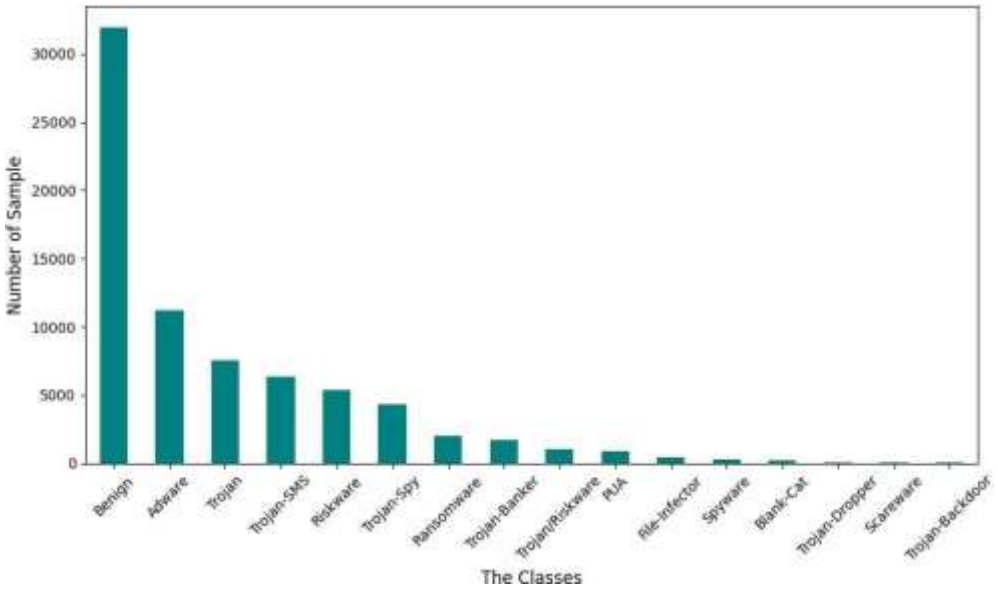

Figure 5. Classify and identify the malware family

IV.2.2 Data Preprocessing

1) Exploratory Data Analysis - EDA: We scanned the data using the Pandas library from Python to understand the structure of the data, check for missing or anomalous values and analyze the distribution of categories.

It was determined that non-numerical features (such as sha256 and EarliestModDate, HighestModDate) have only informative value and do not affect the results.

2) Data Cleaning and Integration: The Pandas framework was used to remove the non-numerical features mentioned previously, and the non-effective features (which always contain similar values in all samples) were also deleted.

At this stage, the number of features was reduced from 489 to 299 features.

IV.2.3 Feature Selection

Feature selection techniques can be divided into the following three main types:

1) Filtering methods: These are methods that are based on evaluating the importance of features based on their statistical properties or their association with the target variable (such as classification type), without using a machine learning model. The algorithm used for training, such as the Mutual Information Algorithms and the ExtraTree Workbook, is not relevant.

2) Encapsulation methods: These are methods that rely on training a machine learning model to evaluate the performance of different subsets of features, and choose the group that achieves the best performance, such as the Redundant Feature Removal (RFE) algorithm.

3) Embedded methods: These are methods that integrate the feature selection process into the machine learning model training process. The importance of features (feature significance) is evaluated during tree construction, and this evaluation is used to select the most influential features.

In this context, we employed two methods to select features, which are listed below:

A- ExtraTree:

Among the different filtern methods available, we chose the ExtraTree algorithm. This is an algorithm based on an ensemble learning method in which multiple decision trees are built in parallel, and a single tree is created using a random subset of features and data points. During the tree-building process, you randomly choose feature splits, making them less likely to be overridden. Evaluating the significance of features across multiple trees.

ExtraTrees implicitly classifies features based on their contribution to impurity reduction (genetic impurity for classification) and makes accurate predictions. Mutual information is used to measure the relationship between individual features (independent variables) and the target variable (dependent variable). It helps to assess the relevance of each feature in predicting the target variable.

B-The importance of features based on the Gain scale:

A method included in the XGBOOST model, Gain is calculated based on the improvement in a performance measure (such as error reduction or Log Loss optimization) achieved by a given partition using a particular feature in the decision tree. The Gain values for each attribute are aggregated across all partitions that include that attribute in all trees in the model. Choose the features that achieved the highest Gain values.

We created 5 different subsets of features extracted from the dataset using the two filtering methods mentioned above, explained in the table below:

	Number of Features	Feature Selection Algorithm
1	Top50	ExtraTree classifer
2	Top50	Importance features "gain"
3	Top100	ExtraTree classifer
4	Top100	Importance features "gain"
7	All features	

Table 2. Features extracted from the dataset

IV.2.4 Model Training

After completing the preprocessing phase and selecting the most important features, several machine learning models were trained in order to compare their performance and determine the best model for both binary and multi-class classification tasks.

The following machine learning and statistical model were used:

XGBoost (Extreme Gradient Boosting): is an advanced machine learning technique used for classification and prediction tasks (classifying data). It relies on the concept of Boosting, where sequential models (decision trees) are built, with each tree correcting the errors of the previous one to improve overall accuracy [59].

Components of XGBoost:

-Decision Trees: XGBoost relies on decision trees as its building blocks. Each tree splits the data based on features to make predictions.
Trees are not independent; they are built sequentially, with each tree focusing on the errors of the previous ones.

-Gradient Boosting: The idea is to improve the model gradually using a technique called "gradient" optimization.
It calculates the "loss" (difference between predictions and actual values) and uses it to guide the next tree to correct errors.

-Regularization: XGBoost uses regularization techniques (L1 and L2) to prevent overfitting, where the model becomes too complex and performs well only on training data but fails on new data.
This makes XGBoost more stable compared to other boosting methods.

-Learning Rate: Controls how much the model adjusts at each step.
A small learning rate means slow and precise improvements but may require more trees. A large learning rate speeds up training but may lead to errors.

-Handling Missing Data: XGBoost automatically handles missing values. When splitting data in a tree, it determines the best way to deal with missing values based on improving predictions.

Advantages:

-High Speed: Uses techniques like parallel processing and memory optimization to speed up training.

-Excellent Accuracy: Effectively reduces errors by focusing on difficult cases.

-Flexibility: Supports various data types and learning tasks (classification, prediction, ranking).

-Handling Complex Data: Manages missing values and reduces overfitting using techniques like regularization.

Class weight:
1) Binary Classification:

A-Formula for scale_pos_weight in XGBoost:

$$scale\_pos\_weight = \frac{n\ negative}{n\ positve}$$

Where:

-n negative: number of negative class samples

-n positive: number of positive class samples

B-Formula for per-class sample weight:

$$weight\ i = \frac{N}{K \times n\,i}$$

Where:

-weight i: weight for class i

-**N:** sum of samples

-**K=2:** Number of varieties

-**n i:** Number of samples in class i (either 0 or 1)

2) Multi-class Classification:

Where K > 2 classes:

$$weight\ i = \frac{N}{K \times n\,i}$$

Where:

-weight i: weight for class i

-**N:** sum of samples

-**K:** Number of varieties

-**n i:** Number of samples in class i (either 0 or 1)

The dataset was split using the Train-Test Split method, with 70% allocated for training and 30% for testing. Stratified Sampling was applied to ensure a balanced class distribution across the split.

To ensure reproducibility of the results, a fixed value of the random_state parameter was used during the partitioning process, so that the same partition is obtained every time the experiment is performed.

IV.2.5 Model Evaluation and Tuning

A- Confusion Matrix in Malware Classification:

The Confusion Matrix is an evaluation tool used to measure the performance of a model in classifying applications as either Malware or Benign.

It is represented as:

$$\text{Confusion Matrix} = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

Where:

-**True Positives (TP):** Number of applications correctly classified as malware.

-**True Negatives (TN):** Number of applications correctly classified as benign.

-**False Positives (FP):** Number of benign applications incorrectly classified as malware (false alarms).

-**False Negatives (FN):** Number of Malware applications incorrectly classified as benign

(missed detections).

***/Metrics Derived from the Confusion Matrix:**

1. Accuracy

Measures the overall proportion of correct predictions among all applications:

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

2. Recall (Sensitivity)

Measures the model's ability to identify all actual Malware applications:

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

3. Precision

Measures how accurate the model's Malware predictions are:

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

4. F1-Score

A balanced metric that combines Precision and Recall, especially useful when dealing with imbalanced datasets (where Malware apps are fewer than benign ones):

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

B-Confusion Matrix in Multiclass Classification:

In multiclass classification, where the model predicts one label out of multiple classes, the confusion matrix is extended to an n × n matrix, where n is the number of classes.

The confusion matrix is structured as follows:

$$Confusion\ Matrix = \begin{bmatrix} C11 & \cdots & C1n \\ \vdots & \ddots & \vdots \\ Cn1 & \cdots & Cnn \end{bmatrix}$$

Where:

-Cij represents the number of samples whose true class is class i but predicted as class j.

-The diagonal entries Cii represent the number of correctly classified samples for class i

(True Positives for class i).

-Off-diagonal entries represent misclassifications.

***/Evaluation Metrics for Multiclass Classification:**

For each class i, we can compute:

$$Precision\ i = \frac{C\ ii}{\sum_{k=1}^{n} C\ ki}\ , \quad Recall\ i = \frac{C\ ii}{\sum_{k=1}^{n} C\ ik}\ ,$$

Where:

-$\sum_{k=1}^{n} \boldsymbol{C\ ki}$: is the total predicted as class i.

-$\sum_{k=1}^{n} \boldsymbol{C\ ik}$: is the total actual samples of class i.

The **F1-score** for class i is:

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

***/The steps in Implementation are presented in Figure:**

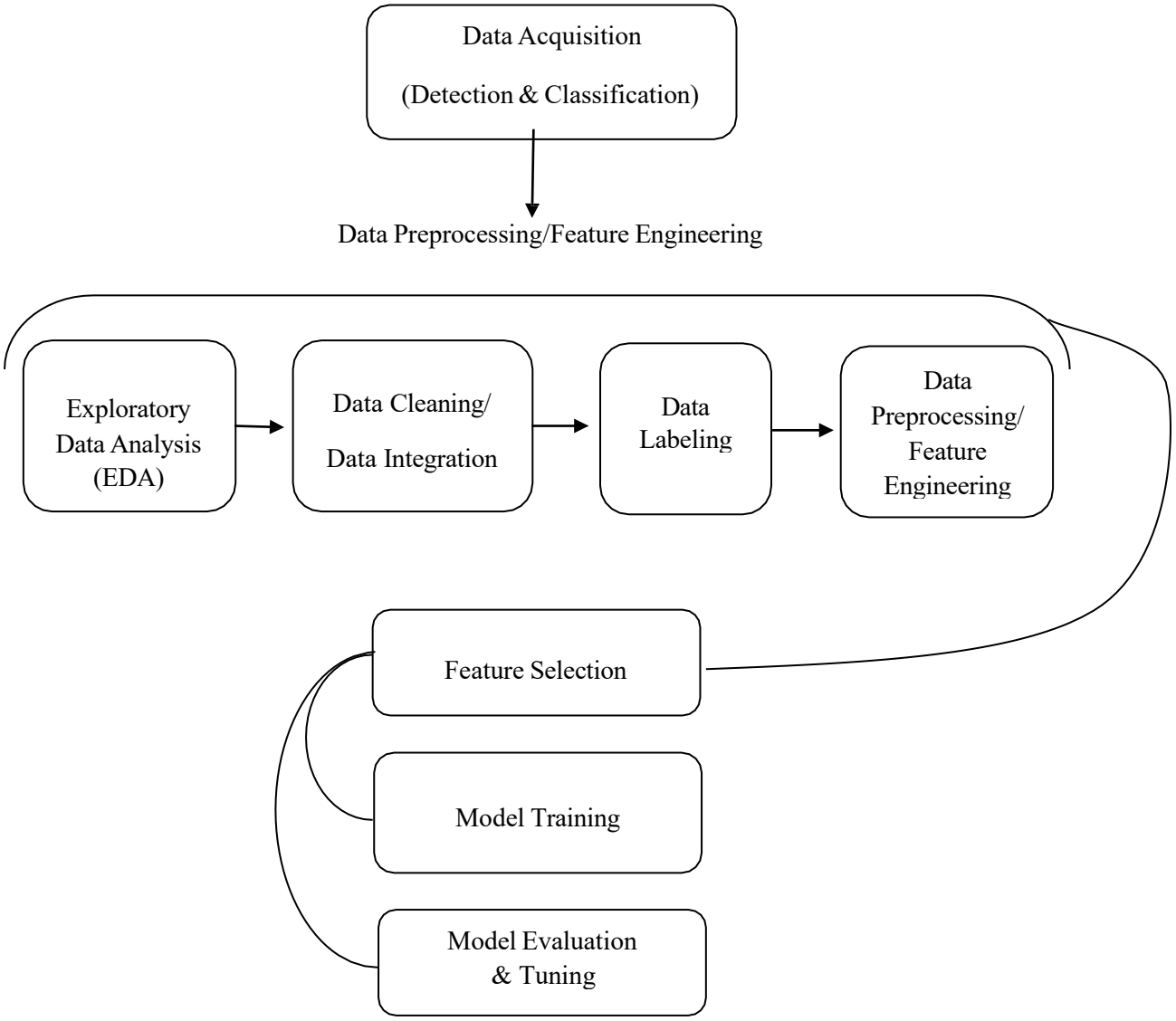

Figure 6. Implementation used to develop the malware detection model

IV.3 Results of applying the model to the selected features

IV.3.1 Detection Malware

Features	XGBoost (%)
Top50 ExtraTreeclassifer	98,64%
Top50 Importence feature («Gain»)	98,50%
Top100 ExtraTreeclassifer	**98,76%**
Top100 Importence feature («Gain»)	98,74%
All The Feature	98,74%

Table 3. Accuracy of Malware Detection.

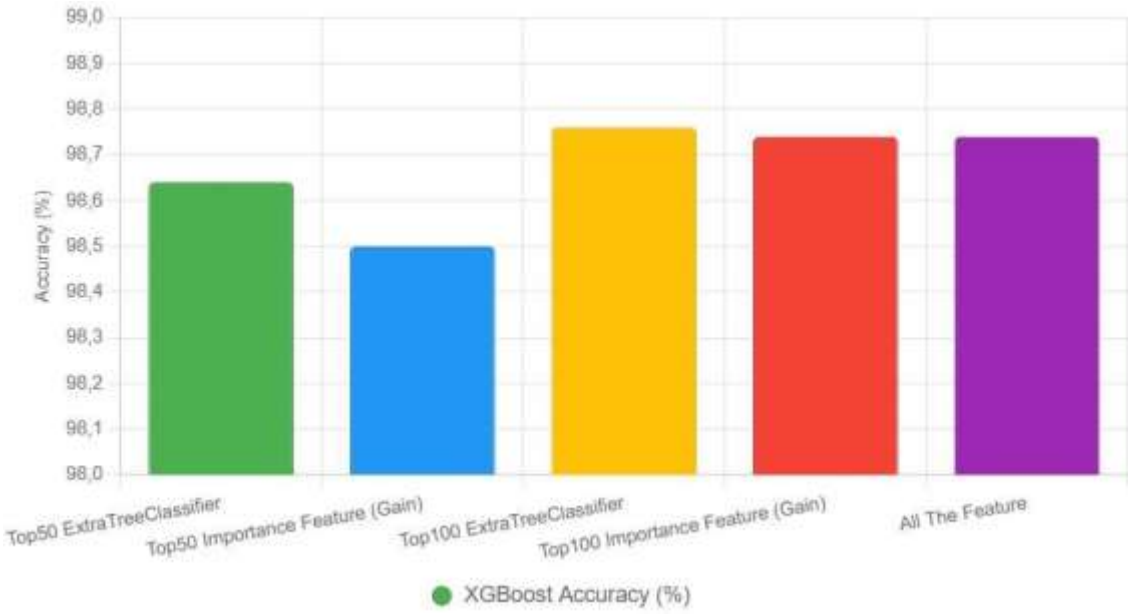

Figure 7. XGBoost Accuracy

Analysis and discuss results:

The results of the **XGBoost** model experiment in malware classification showed outstanding performance when using different feature selection strategies. It achieved the highest classification accuracy of 98.76% when using the top 100 features extracted by the **ExtraTreesClassifier** algorithm, outperforming the use of all features, which had a performance of 98.74%. Features selected via **ExtraTreesClassifier** also outperformed their counterpart based on **XGBoost**'s "Gain" metric, doing so in both the Top 50 and Top 100 groups.

These results highlight the pivotal role of feature selection in improving model performance, as experience showed that reducing the number of features to only 50 did not negatively affect accuracy, but rather the model maintained a high performance of 98.64%. Therefore, it can be concluded that the systematic selection of features contributes to enhancing the accuracy of the model and reducing its computational complexity, leading to more efficient and stable models in practical application.

IV.3.2 Classification Malware

Features	XGBoost (%)	
	Accurcy	F1 score
Top50 ExtraTreeclassifer	91,43%	74,21%
Top50 Importence feature(« Gain»)	89,61%	71,65%
Top100 ExtraTreeclassifer	91,81%	**76,99%**
Top100 Importence feature(« Gain»)	91,67%	76,19%
All The Feature	**91,91%**	75,01%

Table 3. XGBoost Classification

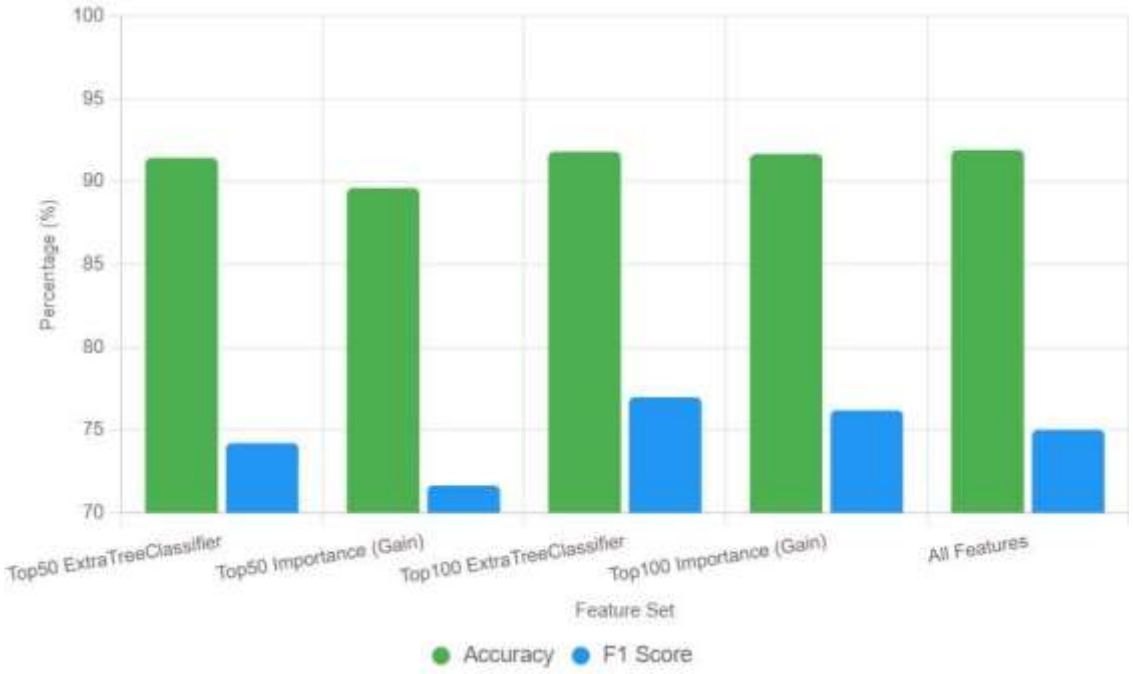

Figure 8. XGBoost Classification

Analysis and discuss results:

Because there is a large discrepancy in the distribution of samples between the different sampled classes (Class Imbalance) in the data set, the F1-score scale was adopted as the main indicator to evaluate the performance of the models, as it better reflects the ability to detect the Malware class by achieving a balance between accuracy and recall [65]. The results showed that the best performance was achieved using the **XGBoost** model with the top 100 properties selected using **ExtraTreesClassifier,** scoring the highest F1-score at 76.99%, which underscores the importance of choosing properties carefully when dealing with unbalanced data.

The following list shows some of the attributes that were chosen by applying the Extra Trees Classifier algorithm, and represent a combination of static and dynamic attributes:

Static attributes include permissions such as: SEND_SMS, READ_PHONE_STATE, RECEIVE_SMS, and ACCESS_FINE_LOCATION, in addition to structural properties such as the number of permissions or their type (dangerous, normal), which are extracted from APK files without executing the application.

As for dynamic attributes, they result from monitoring the behavior of the application during execution, and include system calls such as SYS_333, getpriority, setrlimit, and CALL_PHONE, which provide accurate indications of direct interaction with system resources.

IV.3.3 Hyperparameter Optimize

In this study, one of the most effective and widely used methods for obtaining the optimal hyperparameter values of the model was applied, namely the Grid Search with Cross-Validation (GridSearchCV) technique. This technique systematically explores a predefined set of possible hyperparameter combinations through cross-validation, aiming to accurately assess the model's performance for each combination. Upon completion of the search process, the combination of hyperparameters that yields the highest performance according to the chosen evaluation metric (such as accuracy) is selected.

The following section presents the optimal hyperparameter values obtained by GridSearchCV for the model under investigation.

1-In Detection Malware:

-XGBoost (TOP 100) : n_estimators=600, max_depth=5, learning_rate=0.2 , subsample=0.9 , colsample_bytree=1,

-XGBoost (TOP 50) : n_estimators=500, max_depth=9, learning_rate=0.22, subsample=0.8 , colsample_bytree=0.5,

2- In Classification Malware:

-XGBOOST (TOP 100) : n_estimators=300, max_depth=10, learning_rate=0.04 , subsample=0.99, colsample_bytree=54,

-XGBoost (TOP 50) : n_estimators=700, max_depth=12, learning_rate=0.16, subsample=0.80 , colsample_bytree=0.70,

IV.3.4 Model Evaluation

A) Confusion Matrix:

1- In Detection Malware:

XGBoost (Top 50):

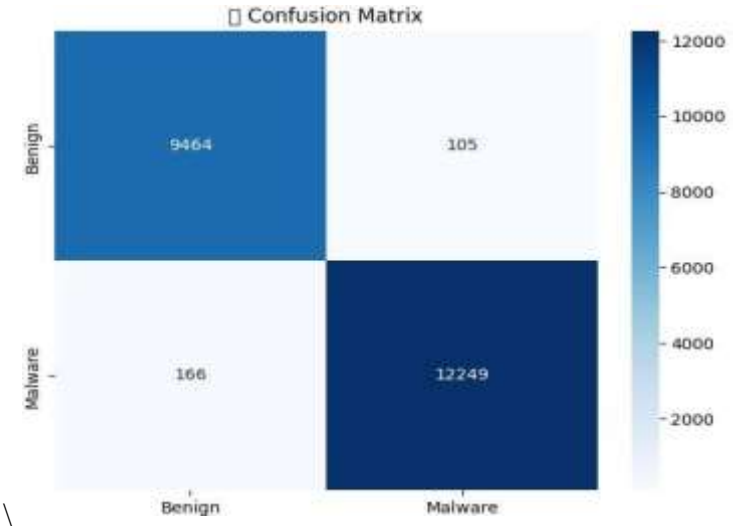

Figure 9. Confusion Matrix in XGBoost (Top 50)

-TP=9464 -FP=105
-FN=166 -TN=1224

XGBoost (Top 100):

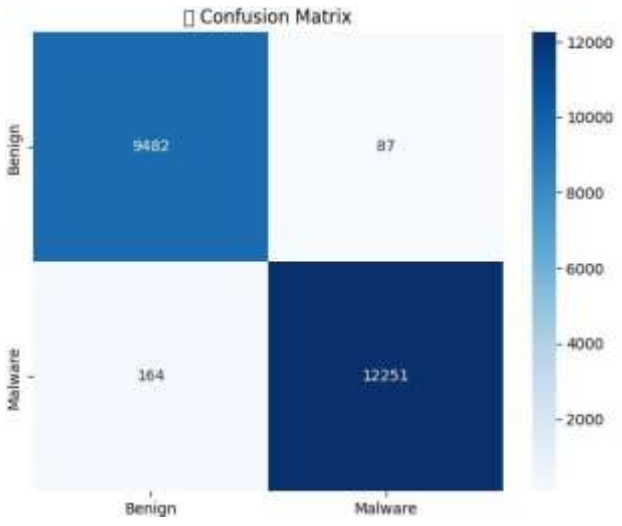

Figure 10. Confusion Matrix in XGBoost (Top 100)

-TP=9482 -FP=87
-FN=164 -TN=12251

In Classification Malware:

XGBoost (Top 50):

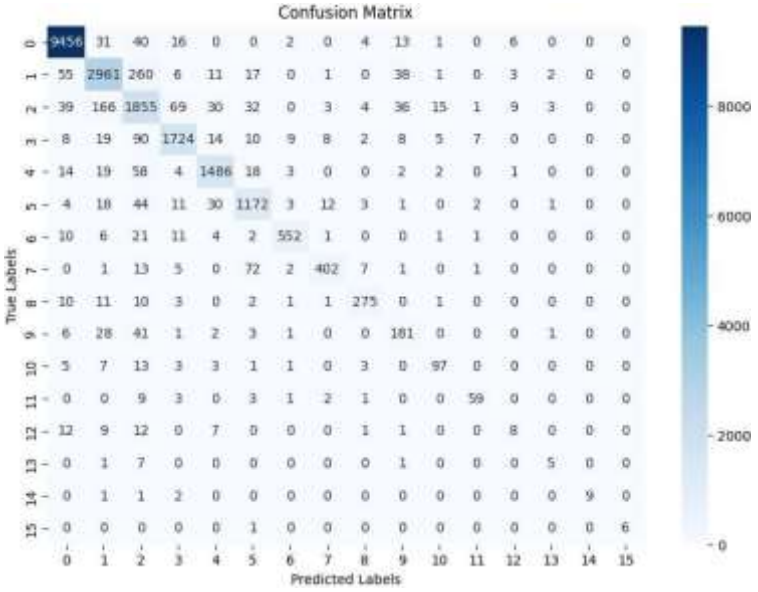

Figure 11. Confusion Matrix in XGBoost (Top 50)

XGBoost (Top 100):

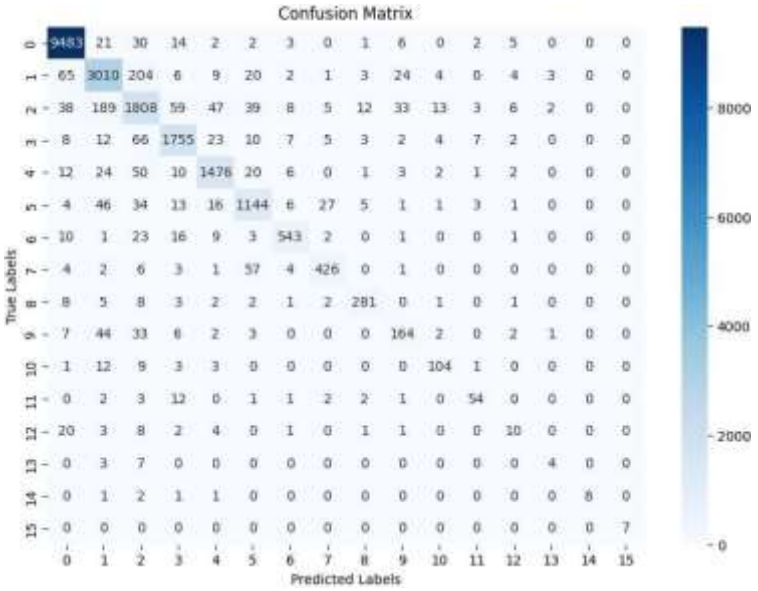

Figure 12. Confusion Matrix in XGBoost (Top 100)

b) Evaluation metrics:

Task	**Metric**	Xgboost (Top 50)	Xgboost (Top 100)
Malware Detection	Accuracy	98,77%	**98,86%**
	Precision	98,71%	98,80%
	Recall	98,78%	98,88%
	F1 Score	98,75%	**98,84%**
Malware Classification	Accuracy	92,21%	92,1%
	Precision	86,11%	82,28%
	Recall	71,72%	76,50%
	F1 Score	78.67%	78,96%

Table 4. Detection and classification Malware

Figure 13. Accuracy and F1 Score

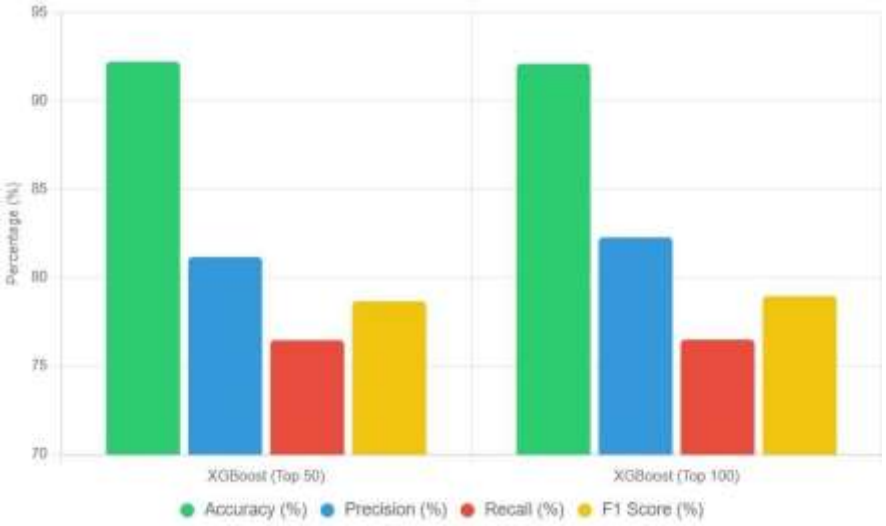

Figure 14. Classification Malware

Analyze and discuss results:

1-Malware Detection

The experimental results demonstrate consistently high performance of the XGBoost model across all key evaluation metrics: precision, accuracy, recall, and F1 score. The configuration using the top 100 selected features combined with extensive hyperparameter optimization achieved the best results, with an accuracy of 98.86% and an F1 score of 98.84%. This performance reflects the model's strong ability to distinguish between benign and malicious applications.

Notably, the XGBoost model using only the top 50 features achieved nearly equivalent performance (98.77% accuracy and 98.75% F1 score), indicating that a significant reduction in feature dimensionality does not substantially degrade classification quality. This efficiency is attributable to XGBoost's gradient boosting framework, which incrementally minimizes classification errors, as well as the fine-tuning of critical hyperparameters such as learning rate, number of estimators, tree depth, and regularization parameters that help prevent overfitting and promote generalization.

These findings underscore the suitability of the XGBoost model especially with the 100 feature configuration for robust and scalable malware detection in static analysis. Furthermore, the strong performance of the 50-feature model highlights its practical feasibility in resource-constrained environments where computational efficiency is essential.

2-Malware classification

For malware classification, results indicate that the XGBoost model with the top 100 features and optimized hyperparameters performed best, with an accuracy of 92.1% and an F1 score of 78.96%. The 50-feature variant followed closely, with an accuracy of 92.21% and an F1 score of 78.67%. Although the performance gap between the two configurations is small, it suggests that the additional features contribute to a marginal improvement in fine-grained classification accuracy.

The comparable performance of the XGBoost-50 and XGBoost-100 models demonstrates the model's ability to extract discriminative patterns even from a reduced feature set. However, the slight improvement seen with 100 features indicates that the extra information helps better separate the malware categories.

Crucially, XGBoost's strong performance is largely due to meticulous hyperparameter tuning specifically parameters like max_depth, eta (learning rate), subsample, colsample_bytree, and lambda which allowed the model to adapt to class variations while reducing overfitting.

3-Impact of paramater optimization

The importance of improving parameter is clearly evident in the performance achieved, as thoughtful configuration contributed to reducing errors and improving the balance between metrics.

In particular, the improvements in F1 Score were tangible when compared with the results of models with default configuration (baseline), reflecting the positive impact of the optimization strategies adopted.

IV.3.5 Validation of results:

The above models are validated using k-fold (k=5) cross-validation, formal malware detection and malware class classification. The validation process splits the dataset 5-fold. In each iteration, k−1 times the data set is used for training and 1 times the data set is used for testing. A model training technique was implemented on different parts of the dataset and the remaining part tested to validate the model's accurate performance by avoiding model overloading or underloading. The results are summarized in Table 4 and clearly demonstrate the validity of the models.

Task	Matric	Xgboost (Top 50)	Xgboost (Top 100)
Malware Detection	Accuracy	98,78%	**98,92%**
	F1 Score	98,92%	**99,05%**
Malware Classification	Accuracy	92,24%	**92,57%**
	F1 score	77,68%	**79,86 %**

Table 5. Results of Cross-Validation in detection and classification malware

IV.4 Comparison with related work:

IV.4.1 Impact of Updated Class Labels on Malware Classification Results:

As mentioned previously, the classifications were updated in data classification on 10/01/2025, resulting in changes in the number of samples within several malware classes. The following table shows the differences in the number of samples before and after the update:

The classes	**Number of Sample** (New)	**Number of Sample** (old)	**difference**
Adware	11183	11185	-2
Trojan	7541	6690	+215
Trojan-SMS	6347	6475	-128
Riskware	5358	5340	+18
Trojan-Spy	4336	4281	+55
Ransomware	2029	1964	+65
Trojan-Banker	1681	1681	0
Trojan/Riskware	1048	54	+994
PUA	881	657	+224
File-Infector	442	436	+6
Spyware	260	260	0
Blank-Cat	166	166	0
Trojan-Dropper	46	62	-16
Scareware	42	1036	-994

Table 6. Comparison of Malware Samples (New VS Old)

As the categories and distribution in the dataset changed, it became necessary to re-evaluate the performance of the Random Forest model as used in the reference study.

Additional note:

The replacement category Backdoor (1,095 samples) became Trojan-Backdoor (22 samples) in the updated data, indicating a reclassification with a decrease in samples in this category.

IV.4.2 Reimplementation and Evaluation of Random Forest on the Updated Dataset:

In this section, we reapply the Random Forest model and evaluate its performance on the updated dataset, just as was done in the previous study entitled ("Classification Using the Enhanced KronoDroid Dataset: Effective and Efficient Android Malware Detection and Classification")

In training and testing the model, we relied on two different sets of features:

-Top 100 features selected using the advanced random tree classifier algorithm (ExtraTrees classifier).

-Top 50 features selected by the same algorithm.

The results are summarized in the following table:

Features	RF (%)	
	Accurcy	F1 score
Top50 ExtraTreesClassifier	**91,94%**	**75,79%**
Top100 ExtraTreesClassifier	91,58%	74,73%

Table 7. Features Selected by ExtraTreesClassifier

Note: Minmax Normalization was used as applied in Rrevious Work.

Analysis and discuss results:

The results indicate that using the Random Forest model with the top 50 properties performs better than using it with 100 properties, both in terms of accuracy and F1 Score. This slight decrease in performance when increasing the number of characteristics is attributed to the introduction of excess characteristics that may be unhelpful or noisy, leading to unnecessary complexity without actual improvement in classification. This underscores the importance of choosing characteristics carefully, as increasing their number does not always guarantee better performance, but may even negatively affect the efficiency of the model.

IV.4.3 Hyperparameter Tuning and Model Optimization:

The best performing model was trained using the following hyperparameters (as applied in the related work):

-max_depth = 300.

-max_features = 'log2'.

-bootstrap = True.

These settings were selected to optimize the Random Forest model's performance and ensure consistency with prior studies.

We obtained the results summarized in the following table:

Metric	**RF** (Top 50)
Accuracy	91,95%
Precision	90.95%
Recall	73,64%
F1 Score	78,54%

Table 8. Results of Matric by RF

Analysis and discuss results:
After applying hyperparameter tuning, the random forest model showed strong performance, achieving high accuracy of 91.95% and high accuracy of 90.95%, reflecting its effectiveness in reducing false positives for benign applications. Although the recall is relatively low at 73.64%, the F1 score improved significantly to 78.54%. This improvement is particularly important given the unbalanced nature of the data set, where balancing accuracy and recall is usually difficult. These results suggest that hyperparameter optimization contributed to enhancing the model's overall ability to detect malware applications, yet more efforts are needed to enhance recall and reduce false negatives in security-critical environments.

IV.4.4 Comparative Performance Analysis of XGBoost and Random Forest:

It is worth noting that the XGBoost algorithm achieves high performance when using the top 50 and top 100 features. In contrast, Random Forest showed better results when using the top 50 features than the top 100 features. To ensure a fair comparison between the two models under equivalent conditions, the number of features was set at the top 50 features for each algorithm. Subsequently, their performance was evaluated and compared to malware detection and classification tasks.

The table below summarizes the performance metrics of XGBoost and Random Forest, both of which use the top 50 features, across these missions:

Task	**Metric**	Xgboost (Top 50)	RF (Top 50)
Malware Detection	Accuracy	98,77%	98.03%
	Precision	98,71%	98.51%
	Recall	98,78%	97.73%
	F1 Score	98,75%	98.12%
Malware Classification	Accuracy	**92.21%**	91,95%
	Precision	81,17%	**90.95%**
	Recall	76,47%	73,64%
	F1 Score	78.67%	78,54%

Table 9. The performance metrics of XGBoost and Random Forest

Analysis and discuss results:

1-Malware Detection:

Accuracy:
The XGBoost model achieved an accuracy of 98.77%, outperforming Random Forest which recorded 98.03%, with a difference of 0.74%. This indicates that XGBoost is more effective at correctly classifying samples as either malicious or benign.

Precision:
XGBoost attained a precision of **98.71%** compared to **98.51%** for Random Forest, a slight improvement of **0.2%**, indicating fewer false positive alarms.

Recall:
XGBoost achieved a recall of **98.78%**, surpassing Random Forest's **97.73%** by **1.05%**, demonstrating greater effectiveness in identifying true malicious cases.

F1 Score:

The F1 score for XGBoost was **98.75%**, compared to **98.12%** for Random Forest, a difference of **0.63%**, reflecting a strong balance between precision and recall.

Detection Task:
Given that accuracy is the primary metric for evaluating malware detection performance, XGBoost clearly outperforms Random Forest, making it the superior model for this task.

2-Malware Classification:
Accuracy:
XGBoost achieved an accuracy of **92.21%**, slightly higher than Random Forest's **91.95%**, with a marginal difference of **0.26%**.

Precision:
Random Forest outperformed XGBoost with a precision of **90.95%** compared to **81.17%**, indicating better ability to reduce false positive classifications.

Recall:
XGBoost recorded a recall of **76.47%** versus **73.64%** for Random Forest, a difference of **2.83%**.

F1 Score – The Primary Evaluation Metric for Classification:

Since F1 score is the key metric for assessing classification performance, XGBoost's score of **78.67%** compared to **78.54%** for Random Forest demonstrates a slight yet meaningful advantage in achieving a better balance between precision and recall.

Classification Task:

Considering that F1 score is the primary evaluation criterion in malware classification, XGBoost is the preferred choice due to its better overall balance between reducing false alarms and correctly detecting malicious samples.

IV.4.5 Result of the comparison

The evaluation shows that XGBoost consistently outperforms Random Forest in malware detection, in all metrics providing higher accuracy, recall and F1 scores. This makes XGBoost more reliable for distinguishing between harmful and benign samples.
In malware classification, where an F1 score is the primary metric, XGBoost also achieves a better balance between accuracy and recall, although Random Forest has a higher accuracy. This balance makes XGBoost generally more effective for multi-class malware classification. While both models perform well, XGBoost's consistent advantage in critical metrics positions it as the preferred choice for comprehensive malware detection and classification tasks. Random Forest remains valuable when reducing false positives is especially important.

IV.5 Conclusion:

The results demonstrated the superiority of XGBoost models whether using the top 50 features or the top 100 features over the Random Forest model in malware detection and classification tasks. Notably, the XGBoost model with the top 100 features achieved the highest values in accuracy, recall, and F1-score, making it the most effective and reliable model for distinguishing between malicious and benign applications.

The study highlights the importance of careful feature selection. Features extracted using the ExtraTreesClassifier proved to be more effective than those selected based on the Gain metric of XGBoost, in both classification and detection tasks. Furthermore, reducing the feature set to 50 features did not significantly degrade performance, which supports the feasibility of deploying the model in resource-constrained environments.

XGBoost demonstrated more balanced and efficient performance than Random Forest, especially in multi-class malware classification tasks.

General Conclusion

This study addressed the growing challenge of Android system security, a concern that has gained significant attention due to the widespread use of Android devices and their vulnerability to malware attacks. The research began with a comprehensive review of Android security concepts and the various types of threats targeting the system, emphasizing the critical role of both static and dynamic malware analysis.

A practical methodology was proposed, combining static and dynamic analysis techniques with machine learning algorithms to build an intelligent model capable of accurately classifying malware applications and distinguishing them from benign ones. The rich and diverse Kronodroid dataset was utilized for training, applying several widely-used machine learning algorithms, notably XGBoost.

The results demonstrated that machine learning is a powerful tool for enhancing Android security, particularly when relevant features are carefully selected and behavioral patterns thoroughly analyzed. Model effectiveness varied depending on the chosen algorithm, feature extraction methods, and data quality. This research contributes to the field of Android cybersecurity by highlighting the importance of integrating software analysis with advanced smart technologies for early threat detection.

Among the evaluated models, XGBoost consistently outperformed the Random Forest classifier in malware detection and classification tasks, whether using the top 50 or top 100 features. In particular, the XGBoost model trained on the top 100 features achieved the highest accuracy, recall, and F1-score, establishing it as the most effective and reliable model for distinguishing malicious from benign applications.

The study also underscored the significance of careful feature selection. Features extracted using the ExtraTreesClassifier proved more effective than those selected based on XGBoost's Gain metric for both classification and detection tasks. Moreover, reducing the feature set to 50 did not significantly affect performance, indicating that deploying such models in resource-constrained environments is feasible.

Overall, XGBoost demonstrated more balanced and efficient performance than Random Forest, especially in multi-class malware classification scenarios. Future work can explore deep learning approaches, network analysis techniques, and scaling up datasets to further advance Android malware detection capabilities.

Challenges related to the study

The study faced several major challenges, the most prominent of which is the extreme variation in the number of samples for each class in malicious application detection data, which represents a major obstacle in building an accurate and reliable classification model. Malicious application data is characterized by a wide variety of patterns, behaviors, and characteristics, making it difficult for models to learn a comprehensive and effective representation of all types of malware if sufficient samples are not available. This variability may lead to generalization problems, as the model can perform well on some taxa but fail to detect other species that are not adequately represented in the data.

XGBoost model setting requires special care in adjusting model parameters (hyperparameters) to achieve the best balance between accuracy and generalizability, especially under high data variability. This includes determining the number of trees, the depth of each tree, the learning rate, and other features that affect how the model is learned. Incorrect setting may lead to model overfitting on training data, or to poor performance due to underfitting.

For this reason, it is advisable to increase the sampling for rare class to ensure sufficient representation of all classes, enhancing the model's ability to recognize less prevalent malware with higher accuracy.

General Conclusion

References

[1] Android Open Source Project, “What is Android?”, Android Developers, 2025. [Online]. Available: https://source.android.com/

[2] Google, “Distribute apps to Android users with Google Play,” Google Play Console, 2025. [Online]. Available: https://play.google.com/console/about/

[3] StatCounter, “Mobile Operating System Market Share Worldwide,” StatCounter Global Stats, May 2025. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[4] A. Sharma and S. Sood, "Security issues in Android OS: A Survey," *Procedia Computer Science*, vol. 132, pp. 1058–1065, 2018. [Online]. Available: https://doi.org/10.1016/j.procs.2018.05.217

[5] Symantec, “Internet Security Threat Report,” vol. 24, Feb. 2019. [Online]. Available: https://www.broadcom.com/company/newsroom/press-releases?filtr=internet-security-threat-report

[6] S. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proc. 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2011, pp. 3–14.

[7] McAfee, “Mobile Threat Report,” McAfee Labs, 2023. [Online]. Available: https://www.mcafee.com

[8] A. Shabtai, Y. Fledel, and Y. Elovici, “Securing Android-powered mobile devices using SELinux,” *IEEE Security & Privacy*, vol. 8, no. 3, pp. 36–44, 2010.

[9] Y. Zhou and X. Jiang, “Dissecting Android malware: Characterization and evolution,” in *Proc. IEEE Symposium on Security and Privacy (SP)*, 2012, pp. 95–109.

[10] Android Open Source Project, “What is Android?”, Android Developers, 2025. [Online]. Available: https://source.android.com/

[11] StatCounter, “Mobile Operating System Market Share Worldwide,” StatCounter Global Stats, May 2025. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[12] J. Oberheide and F. Gong, “An overview of Android security,” *Xuxian Jiang Research Group*, North Carolina State University, 2012. [Online]. Available: https://www.csc.ncsu.edu/faculty/jiang/

References

[13] Google, “Build apps that integrate with Google services,” Google Developers, 2025. [Online]. Available: https://developers.google.com

[14] Y. Zhou and X. Jiang, “Dissecting Android malware: Characterization and evolution,” in *Proc. IEEE Symposium on Security and Privacy (SP)*, 2012, pp. 95–109.

[15] Android Developers, “System Architecture,” Android Developer Documentation, 2025. [Online]. Available: https://developer.android.com/guide/platform

[16] A. Srinivasan, *Android Security Internals: An In-Depth Guide to Android's Security Architecture*, No Starch Press, 2014.

[17] J. Shetty, “Linux Kernel in Android,” *International Journal of Computer Applications*, vol. 108, no. 12, pp. 15–19, Dec. 2014.

[18] Android Open Source Project, “ART and Dalvik,” 2025. [Online]. Available: https://source.android.com/devices/tech/dalvik

[19] K. Sharma and A. Kaushik, “Android Operating System: Architecture, Security and Trends,” in *Proc. International Conference on Computing, Communication and Automation (ICCCA)*, 2016, pp. 847–852.

[20] Android Developers, “APK format,” Android Developer Documentation, 2025. [Online]. Available: https://developer.android.com/studio/build

[21] R. Meier, *Professional Android*, 4th ed., Wiley, 2018.

[22] Android Open Source Project, “Dalvik Executable Format (DEX),” 2025. [Online]. Available: https://source.android.com/devices/tech/dalvik/dex-format

[23] M. Gargenta and O. Nakamura, *Learning Android Application Development*, O’Reilly Media, 2014.

[24] Android Developers, “App Signing,” Android Developer Documentation, 2025. [Online]. Available: https://developer.android.com/studio/publish/app-signing

[25] Kaspersky, “What is a computer virus?” [Online]. Available: https://www.kaspersky.com/resource-center/threats/computer-virus

[26] Symantec, “What is a Trojan Horse?” [Online]. Available: https://us.norton.com/blog/malware/what-is-a-trojan

[27] A. Shabtai et al., “Google Android: A Comprehensive Security Assessment,” IEEE Security & Privacy, vol. 8, no. 2, pp. 35–44, Mar.–Apr. 2010.

[28] Europol, “Internet Organised Crime Threat Assessment (IOCTA),” 2023. [Online]. Available: https://www.europol.europa.eu

References

[29] Zscaler ThreatLabz, “Joker malware apps found in Google Play Store,” 2022. [Online]. Available: https://www.zscaler.com/blogs/security-research/joker

[30] Check Point Research, “FakeApps: Android malware masquerades as popular apps,” 2021. [Online]. Available: https://research.checkpoint.com

[31] A. Faruki et al., “Android Security: A Survey of Issues, Malware Penetration, and Defenses,” IEEE Communications Surveys & Tutorials, vol. 17, no. 2, pp. 998–1022, 2015.

[32] Trend Micro, “Malvertising Explained,” 2024. [Online]. Available: https://www.trendmicro.com

[33] Google, “Security tips for installing apps,” 2025. [Online]. Available: https://support.google.com/android/answer/2812853

[34] M. Grace et al., “Systematic Detection of Capability Leaks in Stock Android Smartphones,” in Proc. 19th Network and Distributed System Security Symp. (NDSS), 2012.

[35] Google, “Google Play Protect,” Android Developer Documentation, 2025. [Online]. Available: https://developer.android.com/google/play-protect

[36] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici and S. Dolev, "Google Android: A Comprehensive Security Assessment," IEEE Security & Privacy, vol. 8, no. 2, pp. 35-44, Mar.-Apr. 2010.

[37] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, May 2012, pp. 95-109.

[38] M. Lindorfer, M. Neugschwandtner and C. Platzer, "MARVIN: Efficient and Comprehensive Mobile App Classification through Static and Dynamic Analysis," in Proceedings of the 2015 IEEE Annual Computer Security Applications Conference (ACSAC), Los Angeles, CA, USA, Dec. 2015, pp. 422-431.

[39] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. McDaniel and A. N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," ACM Transactions on Computer Systems (TOCS), vol. 32, no. 2, pp. 1-29, Jun. 2014.

[40] N. Peiravian and X. Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls," in Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, Nov. 2013, pp. 300-305.

[41] "Jadx - Dex to Java decompiler," [Online]. Available: https://github.com/skylot/jadx. [Accessed: May 29, 2025].

References

[42] "Apktool - A tool for reverse engineering Android APK files," [Online]. Available: https://ibotpeaches.github.io/Apktool/. [Accessed: May 29, 2025].

[43] M. Lindorfer, M. Neugschwandtner, and C. Platzer, “Marvin: Efficient and Comprehensive Mobile App Classification through Static and Dynamic Analysis,” in IEEE Annual Computer Security Applications Conference (ACSAC), 2015, pp. 270–279.

[44] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, “Dendroid: A Text Mining Approach to Analyze and Classify Code Structures in Android Malware Families,” Expert Systems with Applications, vol. 41, no. 4, pp. 1104–1117, 2014.

[45] A. Reina, A. Fattori, and L. Cavallaro, “A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors,” in European Symposium on Research in Computer Security (ESORICS), 2013, pp. 95–112.

[46] A. Desnos and G. Gueguen, “Android: From Reversing to Decompilation,” in Black Hat Abu Dhabi, 2011.

[47] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti, "Detecting Environment-Sensitive Malware," in International Workshop on Recent Advances in Intrusion Detection (RAID), 2011, pp. 338–357.

[48] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-Sandbox: Having a Deeper Look into Android Applications," in Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013, pp. 1808–1815.

[49] C. Tam, C. Y. Yeun, and S. Anbarjafari, “A Review on Hybrid Malware Analysis Techniques,” in IEEE Symposium on Computers and Communications (ISCC), 2020, pp. 1–6.

[50] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, Detection and Analysis of Malware for Smart Devices," IEEE Communications Surveys & Tutorials, vol. 16, no. 2, pp. 961–987, 2014.

[51] VirusTotal, “VirusTotal,” [Online]. Available: https://www.virustotal.com. [Accessed: May 2025].

[52] S. B. Samsudin, F. Idris, R. Salleh, and S. Anuar, “Android malware detection using machine learning algorithms,” J. Comput. Virol. Hacking Tech., vol. 17, no. 4, pp. 239–251, 2021.

[53] A. Arora and D. Garg, “A survey on Android malware detection techniques,” J. Inf. Secur. Appl., vol. 55, p. 102615, 2020.

[54] Y. Wang, J. Liu, and H. Zhang, “A survey of Android malware detection with deep learning,” ACM Comput. Surv., vol. 55, no. 3, pp. 1–36, 2023.

References

[55] H. S. Raghavan, A. Alzahrani, and N. B. Abu-Ghazaleh, “Kronodroid: A scalable and diverse dataset for Android malware detection,” in Proc. 2021 IEEE Int. Conf. Big Data (Big Data), Orlando, FL, USA, 2021, pp. 3915–3921.

[56] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, “Andromaly: A behavioral malware detection framework for Android devices,” J. Intell. Inf. Syst., vol. 38, no. 1, pp. 161–190, 2012.

[57] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, Cambridge, MA, USA: MIT Press, 2016.

[58] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed., Pearson, 2020.

[59] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., 2016, pp. 785–794.

[60] M. Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” arXiv preprint arXiv:1603.04467, 2016.

[61] A. Mosenia and N. K. Jha, “A comprehensive study of security of Internet-of-Things,” IEEE Trans. Emerg. Topics Comput., vol. 5, no. 4, pp. 586–602, Oct.–Dec. 2017.

[62] A. Nisioti, A. Mylonas, H. Gascon, and G. Kambourakis, “From Android malware detection to malware classification: A survey,” Comput. Secur., vol. 86, pp. 1–25, 2019.

[63] I. Alsmadi and A. Al-Ali, "Kronodroid: A dataset for Android malware detection research," Data in Brief, vol. 47, p. 108904, Jan. 2023. doi: 10.1016/j.dib.2022.108904.

[64] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm, "KronoDroid: time-based hybrid-featured dataset for effective Android malware detection and characterization," Computers & Security, vol. 110, Art. no. 102399, 2021. https://doi.org/10.1155/2024/7382302

[65] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

[66] N. Tarar, S. Sharma, and R. K. Challa, "Analysis and Classification of Android Malware using Machine Learning Algorithms," in Proceedings of the International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, Nov. 2018, pp. 1–6.

References