A Hybrid Analysis and Detection of Android Malware Using Machine Learning and Blockchain Technology

By: Habtamu Alemayehu

A Thesis Submitted to the

Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in partial fulfillment of the requirement for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2 0 1 9

Adama , Et h i o p i a

A Hybrid Analysis and Detection of Android Malware Using Machine Learning and Blockchain Technology

By: Habtamu Alemayehu

Name of Advisor: Mesfin Abebe (PhD)

A Thesis Submitted to the

Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in partial fulfillment of the requirement for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2 0 1 9

Adama , Et h i o p i a

Declaration

I hereby declare that this MSc Thesis is my original work and has not been presented for a degree in any other university, and all sources of material used for this thesis have been duly acknowledged.

Name: _____

Signature: _____

This MSc Thesis has been submitted for defense with my approval as a thesis advisor.

Name: _____

Signature: _____

Date of Submission: _____

Approval of Board of Examiners

We, the members of the board of examiners of the final open defense by *Habtamu Alemayehu* have read and evaluated his/her thesis entitled "*A Hybrid Analysis and Detection of Android Malware Using Machine Learning and Blockchain Technology*" and examined the candidate. This is, therefore, to certify that the thesis has been accepted in partial fulfillment of the requirement of the Degree of Computer Science and Engineering.

_____	_____	_____
Supervisor/Advisor	Signature	Date
_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

Acknowledgment

Before all else, I would like to praise Almighty God and His Mother St. Merry for everything that happened, for all his blessings, and to help me pass all the challenges and those hard times that I could not pass on my own, thank you so much Dear Lord.

I owe my most profound thankfulness to my advisor Dr. Mesfin Abebe, whose guidance sincerity, constructive comments, remarks, and commitment through the whole process of this study came to the realization. He was the one who guided me all the way and kept turn-on the light over my obstacles with no time, during my study periods, as an advisor, instructor, and postgraduate coordinator. I am incredibly thankful to him for providing such helpful support.

Next, I would like to express my truthful thanks to my Family and Betelhem Teshome; I never forget the help of your assistance, God always watches over you. I hope you will always get close to my movement.

Finally, it is my radiant sentiment to place on record my best regards, most profound sense of gratitude to my friends and classmates for their timely help and wholehearted support rendered to me in the completion of my thesis. Additionally, I also would like to thank my fellow smart software systems SIG mates: Amanuel Negash, Daniel Aschalew, Dereje Seyfe, Dureti Genemo Elias Mohamed, Moti Bacha and Yonas Kenenisa for their appreciation to finish this work. My special thanks go to Asnake Ayele for providing required real devices for the study.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Algorithms

List of Abbreviations and Acronyms

ABI	Application Binary Interface
AUC	Area Under Curve
AOSP	Android Open Source Project
API	Application Programming Interface
ARM	Advanced RISC Machine
ART	Android Runtime
Dex	Dalvik Executable Format
ES	Elasticsearch
GPL3	General Public License
HAL	Hardware Abstraction Layer
IEEE	Institute of Electrical and Electronics Engineers
JDK	Java Development Kit
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbor
MIME	Multipurpose Internet Mail Extensions
ML	Machine Learning
MobSF	Mobile Security Framework
OHA	Open Handset Alliance
OS	Operating System
P2P	Peer-to-Peer
PDA	Personal Digital Assistant
RISC	Reduced Instruction Set Computing
ROC	Receiver Operating Characteristic Curve
SHA256	Secure Hash Algorithms
SLR	Systematic Literature Review
UI	User Interface
UID	Unique user ID
VM	Virtual Machine
XML	Extensible Markup Language

Abstract

Through an increasing number of mobile devices running the Android mobile operating system, their extensive usage, and various application possibilities, those devices have become valuable targets for malicious applications. The goal of this study is to design a hybrid analysis framework for android Malware Detection through integrating Machine learning and Blockchain technology. With static and dynamic analyses, researchers gain valuable insights into the technique of malware, where machine learning is often used to detect new android maliciously. Android malware is continuously developing, so training a machine learning model using out-of-date malware could negatively affect the performance of the predictive identification of more recent malware. Several existing solutions used outdated malware collections. In this study, recent malicious and benign Android apps from suitable repositories were collected. Component and interactive hybrid analyses were implemented to extract dynamic and static data from Android apps. Both methods attempt to increase the code coverage of the dynamic analysis, which was executed on real devices. The analysis result of the hybrid analysis is stored permanently in Elasticsearch in JSON format. Random forest and Extra trees classifier used to perform binary classification on malicious and benign Android apps using a total of 49554 and 46625 features of the component and interactive, respectively. A Virustotal service is applied to reduce the training noise. The classification result, Android malicious codes sent to the blockchain for automatically generating new blocks. K-fold cross-validation was used to evaluate test error, measured using well-known metrics: Precision, Recall, and F1-score. Finally, the prediction of unknown application results in an accuracy of approximately 93%. From this result, it is possible to conclude that the hybrid analysis is preferred since it has a better performance and is less error-prone compared to the individual analysis approach.

***Keywords**: Android Application, Blockchain, Hybrid Analysis, Machine Learning, Malware*

CHAPTER ONE

1. Introduction

This chapter introduced the background, motivation, statement of the problem, contribution, objective, methodology, scope, and application result of the study, followed by the organization of the thesis.

1.1. Background of the Study

An android-based system, as one of the most popular mobile operating systems, it is an obvious target of malicious developers. In the 2018 Mobile Malware Evolution Report [1], "Kaspersky lab detected 94,368 mobile banking Trojans, 5,730,916 malicious installation packages, and 544,107 mobile ransomware Trojans, which have been significant increases compared with those of the previous year". Although Google has created the Bouncer tool to perform malice detection on applications of Google Play, attackers could still bypass Bouncer to drive malware into Google Play [2].

The existing malware detecting technologies for Android devices classified into two categories: *static-based* and *dynamics-based analysis* [3]. The static-based analysis method used the decompiler technology or performance analysis of the control flow and data flow in the small intermediate code, which was applicable for automatic analysis through a large number of software samples. However, it could not solve the problems of code obfuscation, encryption, and other issues, which performed in dynamic execution [3]. The dynamic-based analysis method stimulated the execution of software to avoid code obfuscation and encryption problems. However, the coverage of its dynamic testing code always is not enough. Besides, some malicious programs might camouflage themselves when running under simulators, [4].

In 2016, Jacob Poushter reported that in Ethiopia 4 % of the population owning a smartphone [5]. With the proliferation of smartphones, the challenges for smartphone security are becoming very similar to that personal computer encounters. Malware on Android mobile can make the phone partially or entirely unusable; cause unknown billing; steal private information; send unnecessary SMS messages, or infect the contact of the users in the phone-book.

On the other hand, there is a variety of malware in different families with various features, which also increased the difficulty of detection. At the same time, the extraction of some features in

the existing methods required high time and cost. Therefore, we need to continue to delve into the research of malicious code or malware detecting technology in Android-based devices. The integration of blockchain technology with the new Android malware detection engine provides full security. *Figure 1. 1*: G DATA analysis statics result of new Android malware samples.

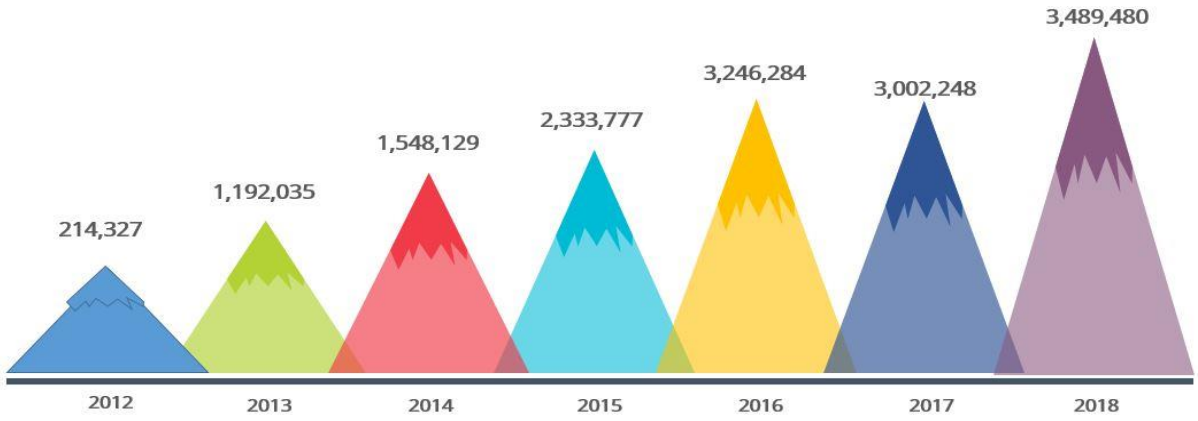


Figure 1. 1: G DATA analysis statics result of new Android malware samples.

Recently, the blockchain technology, as a new type of distributed computing paradigm, has gained much importance due to its high efficiency, high data security, high credibility, and low cost. The research of the blockchain technology divided into three categories: the public blockchain, and the consortium blockchain, the private blockchain [5]. The consortium blockchain may allow each member to read and limited to participants.

In light of this, this study proposed a framework for malware detection in android based mobile devices using machine learning and blockchain technology. The proposed framework can contribute to enabling users to obtain the latest security information of an application timely. The peer-to-peer network is far less susceptible to targeted denial of service attacks.

Strengths, Weaknesses, Opportunities, Threats (SWOT) analysis can be used to study relationships between the planning aspects in a country and internal and external environmental factors. Through SWOT analysis, we can assess the value of mobile phones in the context of existing policy, regulatory and legal framework, particularly in the areas of information sharing and work towards strengthening the capacity to regulate the sector. Mobile technology can increase connectivity and provide access to communications throughout a country. International experience suggests that the expansion of communications (through mobile phones) networks is strongly correlated with economic growth, and the access in rural areas diminishes the

urban/rural economic disparities, provide access to market information, and facilitates service delivery in health, education, and others, reinforcing poverty reduction

In terms of strengths, Ethiopia is eager to absorb new technology, and the government is making aggressive plans. The Ethiopian Telecommunications Corporation (ETC) in charge of ICT industry is providing incentives to push the mobile industry. As a result, there is significant growth in both government and non-government entrepreneurship for the development of mobile applications. Specifically, there is strong development in the capital of Addis Ababa as well as the nearby towns Bishoftu (Debre Zeit) and Adama (Nazreth). Promoting e-commerce will have a positive impact on the mobile industry [6].

In terms of weaknesses, there is a significant absence of resources for development. Customs regulations are too rigid. Hackers and contraband goods are not addressed. There is a lack of training for technical staff. Finally, most of the focus is on Addis Ababa and the surrounding towns, which ignores other areas[6].

In terms of opportunities, Ethiopia can become a big player in the Sub-Saharan belt. Collaborating with external vendors would help the mobile industry grow. Inviting investors from abroad would enhance resources. There is also the potential for capacity building through training of local staff. Investment in mobile industry might lead to increased social and economic returns. Finally, there is the chance to expand mobile services to other areas, especially rural areas, to cover the entire nation.[6]

In terms of threats, one important consideration is political instability. Further, another important threat to consider is friction between the government sector and the private sector.

1.2. The Motivation of the Study

Google applies the permission system as a measure to restrict access to free resources of the system. Android granting policy is all-or-nothing permission, and this means: When a user is installing an application, he/she must either refrain from installing the application or grant all permissions to the application that it requests. The lack of understanding of permissions by users [7], who authorize malware unconsciously to infiltrate, is a problem.

With decreasing overall trading prices, the amount of mobile devices worldwide is still increasing. Particularly the portability of mobile devices makes them attractive to users.

Nowadays, Android is the most popular mobile operating system and is dominating the market. According to the *International Data Cooperation (IDC)*, it had an overall market share of approximately 85.1 % worldwide at the beginning of 2018.

Since the beginning, Android was always a target of attacks because users store valuable information about their identity, user credentials, and payment information on their devices [8]. Furthermore, the number of mobile apps increases. Especially malicious apps are in the focus of security researchers all over the globe. Many publications regarding mobile malware detection address the Android platform.

Training a machine learning model using out-of-date malware can affect the prediction accuracy for more recent malware. New malware approaches and their specific characteristics never trained, so the specific features remain unknown to the machine learning algorithm. The central problem is that many scientific publications, as well as more recent ones, utilize outdated malware in supervised machine learning approaches to predict the detection of new Android malware. Often a dedicated static or dynamic analysis approach is used that does not combine the results to mitigate the drawbacks of each analysis. This study focused on hybrid analysis, which can mitigate the drawbacks of each analysis and increase detection accuracy. Hybrid analysis integrate run-time data mined from dynamic analysis into a static analysis algorithm to detect malicious functionality in the applications.

1.3. Statement of the Problem

Due to Android's top popularity and open-source nature, it has also been the priority target of malicious Apps over other mobile operating systems, so the study focuses on Android malicious app prediction. There are security harms met by the android; some of them mentioned below[8].

- Without permission requests, some apps can exploit the features of another app.
- When apps being upload on the market, the security scan is not robust.
- Android's permission security model grants privilege to the user to decide whether an app should be trusted or not. The user privilege leads to many risks in the Android system.
- Open Source is available to hackers as well as genuine developers too. Thus the Android framework cannot be trusted to develop critical systems.

- Any application on the android stage gets to information like the GSM marketer Ids and SIM while not the permission of the user.
- The android platform gives all security features, yet there dependably be a risk if the client introduces suspicious applications or enable authorization to an application without focusing.

The surveyed techniques can detect (and only a few can prevent) inter-application privacy leaks. The majority of these techniques, however, do not consider whether the leakage of a sensitive information is user intended or not. Likewise, most of these approaches detect privilege escalation attacks, but they limit themselves to Intent-based and kernel IPC-based cross-process attacks, and thus do not consider the trickiest cross-process attacks involving network channel and user manipulations.

Most existing approaches are static, and thus only able to detect a potential vulnerability or attack. Only a limited number of techniques uses dynamic analysis to actually prevent an attack at runtime. In addition, all approaches surveyed incur drawbacks that limit their precision, e.g., FlowDroid [39] is oblivious to multithreading and thus cannot properly resolve reflective calls.

The above defined and explained problems summarized in the following research questions:

RQ1: ***What are the better approaches for detecting Android malware by integrating machine learning and blockchain?*** With this research question, we survey the issues that have not yet addressed by other researchers.

RQ2: ***How to improve the performance of the Android malware detection system on a peer-to-peer node by the decentralized solution?*** With this research question, we survey the various issues targeted by the decentralized solution.

RQ3: ***How precise is the malware detection of recent Android malware by comparing components and interactive analysis approaches?*** We study the depth of the analysis procedure in detail. To that end, we examine the following sub-questions: RQ 3.1: What is the code used to represent the hybrid analysis process? RQ 3.2: What are the main techniques used in the hybrid analysis of Android apps?

1.4. Objectives of the Study

1.4.1. General Objective

The general objective is to develop a hybrid analysis and detection framework for android malware using machine learning and blockchain technology.

1.4.2. Specific Objective

To achieve the general objective, the following mentioned specific objectives have to address:

- Conduct a literature review on android malware detection frameworks.
- Identify security methods of Android apps as well as their vulnerabilities and limitations
- Implement hybrid analysis and extract hybrid features of the Android application.
- Study machine learning models to apply for the Android malware detection
- Identify the features creates by the Smartphone process are malicious or benign behavior.
- Implement a suited machine learning approach with hybrid analysis
- Determine the best analysis technique for malware behavior detection.
- Build a server-less decentralized access control solution based on a peer-to-peer network.
- Establish a fact-base of distributed Android malicious codes using blockchain technology.
- Evaluate the prediction performance of both hybrid approach

1.5. Methodology

To attain the above objectives and to answer the research questions, the following methods and techniques followed. *Figure 1. 2* depicts the research process of this study. We mentioned the following steps as a research methodology.

- First, formulating the problem by reviewing different literature and documents.
- Secondly, collecting data from different guidelines, documents, and web sites related to Android malware detection.
- Thirdly, mention the research methodologies such as for systematic literature review, data collection method, Android malware detection methods, proposed framework design method, the framework implementation method, the prototype evaluation method, and the design and implementation tools used for the research work.

- Then, designing a malware detection framework for the Android-based smartphone application.
- Then after implementing the proposed framework by using machine learning and blockchain algorithms
- Finally, evaluating and testing the functionality and usability of the proposed framework.

Figure 1. 2: Research process

1.5.1. Literature Review

To attain the objectives of the research, an in-depth literature review on Android malware detection and related works done. Books, articles, thesis papers, and electronic materials used to collect the necessary information for this study. Based on the information obtained from the reviewed literature, the input analysis, underlying representation, and output representation techniques and tools are selected. The techniques and tools are chosen by considering various factors like their effectiveness in similar previous works. In recent years, many works have conducted; Even if various researches on Android malware detection using machine learning conducted, no researchers have done on Android malware detection by incorporating machine learning and blockchain technology.

1.5.2. Data Collection

Data Collection is a fundamental aspect of any analysis study. The data collection starts with determining what kind of data are needed to be followed by techniques of information collection. In this research, the following data categories collected.

- Benign applications and
- Malicious applications

1.5.2.1. Benign applications

Some top free applications selected from the Google Play market. The selected applications are clean from any malware, the application developers' reputation was verified, and they looked to be genuine application distributors.

We selected free applications in Google Play based on the descriptions, the number of downloads and the ratings given by users: only the top ones are picked. There is no mean to prove the goodness of applications; even the official market can contain malware. Applications from Google Play offer, however, a high legitimacy compared to those from alternative sources. Each application taken from Google Play has been scanned by fifty-seven engines from renowned anti-viruses on VirusTotal [46] and only the ones that succeed all virus tests are considered as benign and kept inside the dataset of normal applications.

1.5.2.2. Malicious applications

The study incorporates public and private data collection to train and test the selected classifier. Virusshare is a public dataset, which is a collection of malware categorized into different families. Surprisingly, some of them in this set collected from Google Play. A private dataset is a collection of the latest mobile malware.

Public access to known Android malware samples is mainly provided via the Android Malware Genome Project and Contagio Mobile. The malgenome-project was a result of the work done by Zhou and Jiang [45] and contains over 1200 Android malware samples, classified in 49 malware families and were collected in the period of August 2010 to October 2011. Contagiodump offers an upload dropbox to share mobile malware samples among security researchers and currently hosts 114 items.

1.6. Scope and Limitations

1.6.1. The Scope of the Study

The study examines its principles of work, technical specifications and features of blockchain, and also it identifies how to use the hybrid analysis in malware detection. This study explores the recent level of Blockchain in Android-based malware detection applications. Further, as

machine Learning loves to work with many data, it creates an opportunity to build better models by taking advantage of the hybrid analysis and decentralized nature of blockchains. By evaluating the classification accuracy of the classifiers, the classifier algorithm with the highest classification accuracy is used in modeling the classifier.

1.6.2. The Limitation of the Study

Implementing a hybrid analysis is more challenging and complex than a single static approach. Due to this fact using graph results from flow analysis is beyond the scope of this study. Beyond this study, do not address the very complex behavior of the Android applications due to different constraints like the schedule and resource limitations such as standard datasets. Even though the blockchain technology has been around for a while now, there are limitations when it comes to the literature in hand, and there are very few academic studies available regarding blockchain and its representation. Because some information is missing or incomplete, so the review of the technology is based on the literature available.

1.7. Contribution of the Study

The study extends the effort exerted in solving the security problem of the android operating systems. The research contributes to its part in the detection of android malware. The benefits of this research are;

- Detect Android malware using machine learning and blockchain technology.
- Establish a fact-base of the malicious Android codes using the blockchain technology to detect malware information.
- Generate new blocks of the hybrid analysis result.
- Enables users to obtain the latest security information of the android application timely.
- Users can also obtain historical analysis results.
- Provides to users with more information to evaluate application security risks.
- Less susceptible to targeted denial of service (dos) attacks.
- Increase the availability of detection framework.
- Neutralize the disadvantages of dynamic and static analysis.
- It provides information to evaluate application security risks.

1.8. Organization of the Thesis

This section outlines the organization of this study and its sections:

- *Chapter one* discussed the introduction of the study, the motivation, and the statement of problems, the research questions that answered in the proposed solutions, the scope and limitation, the objective of the study, the methodology, and the beneficiaries of application results.
- *Chapter two* presents about literature review. It includes systematic literature review ,the overview of the mobile application, types of smartphone operating systems, mobile OS security, Basic concept of Machine learning, Blockchain technology, and related works.
- *Chapter three* discusses the research methodology, which includes a methodology for, data collection method, Android malware detection methods, design proposed framework method, the framework implementation method, and the prototype evaluation method. It also describes the design and implementation tools used for the research work.
- *Chapter four* discusses the proposed framework design and describes each proposed framework components.
- *Chapter five* describes working Environments, programming language and implementation of the proposed framework
- *Chapter six presents* result in evaluation and discussion about the framework based on sample prototype implementations.
- *Chapter seven* consists of a conclusion, recommendation, and identifies future works for further research direction on this research topic

CHAPTER TWO

2. Literature Review and Related Works

2.1. Systematic Literature Review

The approach that we followed for the systematic literature review is based on the rules provided by Kitchenham [9] and which have already been used by other SLRs.

The literature review in this study has gone through the following steps:

- In the first step, we list the research questions which are used to identify and collect the vital information the publications.
- Then, we enumerate a list of search keywords, and state the two scenarios used to conduct the search process: the first one identifies the public publication repositories, while the second one emphases the lists of publications from top venues.
- Exclusion criteria are applied to the search results, which remove irrelevant papers,
- We unify the sets of results from each search scenarios.
- Finally, a simple filtration of the selected publications is performed.

2.1.1. Search Strategy

The searching strategy is performed using the following aspects:

- Search for synonyms and other words.
- Boolean OR used to associate synonyms.
- Boolean AND used to combine significant terms.

2.1.1.1. Search Strings

The leading search string is done by connecting different keywords through logical connective. *Table 2. 1* shows the actual keywords used for retrieving some essential publications. Related strings are on the same line.

Table 2. 1: Search string items

Search strings
ALL(‘Static Analysis’ OR ‘Dynamic Analysis’ OR ‘Hybrid Analysis’
Alternatively, ‘Android Malware Detection’ OR ‘Intrusion detection’ OR ‘Malware’ OR ‘Malicious Android Apps.’

Alternatively,' Android' OR 'Smartphone*' OR 'Mobile.'
Alternatively, 'Analysis' OR 'Analyze*' OR 'Analys*.'
Alternatively, ('Consortium Blockchain' 'Blockchain Application' blockchain' OR 'block-chain' OR 'distributed ledger') AND ('cybersecurity')
Alternatively, 'Machine learning' OR 'Random Forest' Or 'Decision tree' OR 'Ensemble. ')

The platforms searched were:

- IEEE Xplore Digital Library
- ScienceDirect
- SpringerLink
- ACM Digital Library
- Google Scholar

The searches conducted on 23rd April 2019, and we used studies that had been published up to this date. *Table D. 1* lists these publication venues.

2.1.2. Inclusion and Exclusion Criteria

Papers are filtered using inclusion and exclusion criteria. Whenever paper meets all inclusion criteria, it kept for further considerations, and when the paper meets exclusion criteria, it removed from the study. For this SLR, we use the following Inclusion / Exclusion criteria:

- First, we filter out non-English written publications.
- Then, we attempt to identify duplicate papers for exclusion.
- Finally, exclude such on-Android-related papers.

2.1.3. Search Result

As illustrated in *Figure 2. 1*, we adopted a search process on a five-reference repository, and we got a different number of papers for each venue. JabRef is used to manage references and thereby remove duplicates. From a total of 24,449 references obtained with an automatic searching, 7,206 of them were removed as a result of duplication.

Second, title based selection performed and checked against inclusion and exclusion criteria, thereby maintain studies that were related to this work and include those articles into the next phase. The step differed with 154 papers for the next phase. Then, we filtered research papers by reading abstraction and the introduction section of each paper. In this step, we obtained 95 research papers for further study. Finally, the full content of each paper was examined and made

a final decision about each paper. This step resulted in 30 references: *Table D. 2 lists* these papers, and *Table D. 3* detail the number of papers taken from each repository.

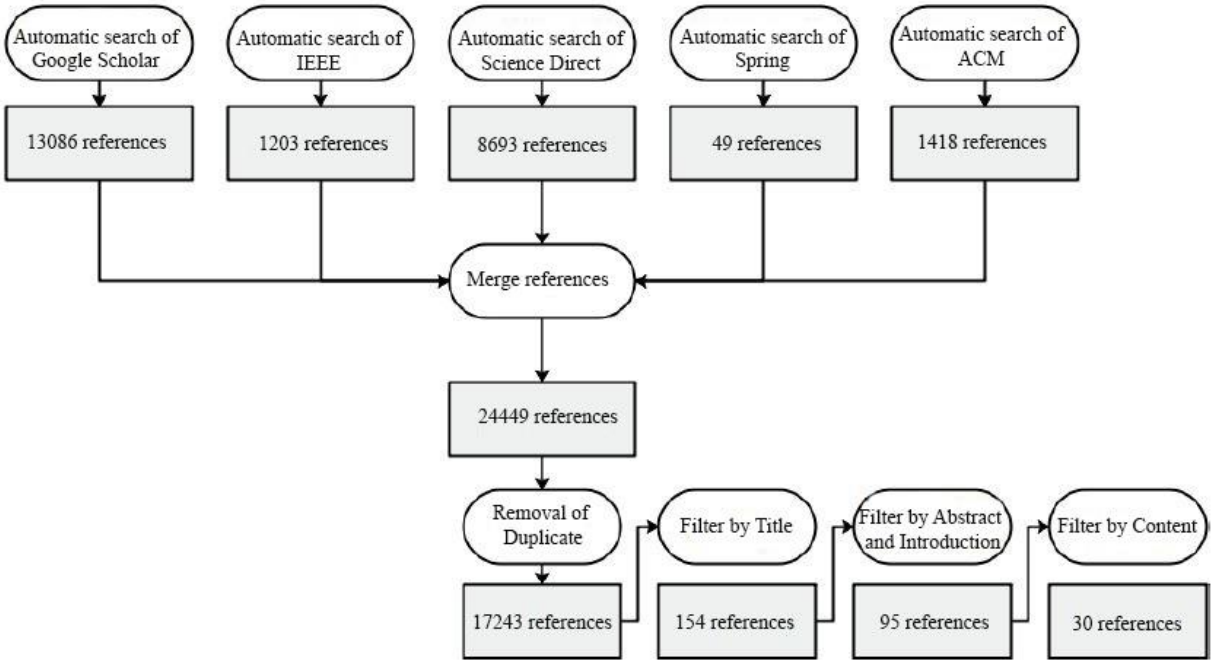


Figure 2. 1: SLR process

2.2. Mobile Phone Device

A mobile phone is a portable, handheld communications device connected to a wireless network that permits users to launch applications, make voice calls, and send text messages. There are many different styles, types, and models of mobile phones available on the market nowadays. *Figure 2. 2*[10] depicts the digital market overview of 2018, report.

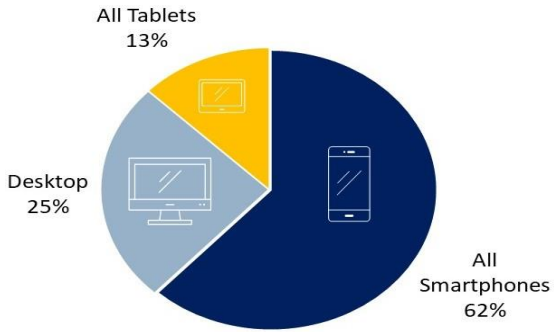


Figure 2. 2: Digital market overview report

2.3. Popular Mobile Operating System

A mobile operating system is an OS that is specifically designed to run on mobile devices such as smartphones, mobile phones, tablet computers, PDAs, and other devices. A mobile OS is the software platform on top of which other programs can launch on mobile devices.

The OS is accountable for determining features available and the functions on the mobile device, such as keyboards, thumb wheel, synchronization with applications, wireless application protocol, text messaging, email, and more [33]. The next sub-sections review six of the most popular handheld device OS.

2.3.1. BlackBerry OS

Blackberry is a proprietary mobile OS that was developed by the Canadian company RIM. This OS was applicable only for their devices, like Blackberry mobile and Tablet devices [11]. The Blackberry OS differs from other types of smartphone OS because Blackberry developed for business usage. One of the main advantages of the Blackberry device is the capability to store large numbers of emails.

2.3.2. iPhone OS

iPhone OS(iOS) is a widely used type of OS for Apple's smartphone devices, which is developed by Apple Inc. Currently, this OS is applicable for different types of Apple products such as iPhone, iPod Touch, iPad, and Apple TV[11].

2.3.3. Symbian OS

Symbian is a smartphone OS, which was launched by Nokia [11]. Previously, there were different hardware vendors, including Motorola, PSION, Ericsson, and Nokia were formed Symbian OS. Today, the Symbian foundations are run on Nokia only.

2.3.4. WebOS

Web OS is a proprietary and Linux kernel-based OS for smartphones and smart TV devices, which was developed by Palm. This OS is called HP web OS and Palm web OS. Both native and third-party apps implemented by HTML, CSS, and JavaScript code.

2.3.5. Android OS

The Android mobile OS is Google's open and free software stack that includes an OS, middleware, and application layers for use on mobile devices. Android developed by an OHA, which is a group of over 30 companies and that are governed by Google [12].

2.3.6. Windows Mobile

Microsoft Corporation launched this OS in the name of Windows Phone 7 in 2010 [11]. Today, various mobile manufacturing companies are developing Windows Phone devices, including HTC, Nokia, Samsung, and LG.

2.4. Android

In 2005 Google acquired the startup *Android Inc.* based in Palo Alto, California, which at this time was developing an early version of the mobile OS named Android. In the last decade, the Android OS developed by Google Inc. and a group of companies known as the *OHA* [13]. Nowadays, a lot of other corporations have invested in Android [13]. The system continuously developed and released under the *Android Open Source Project (AOSP)* using open source licenses. The majority of Android software is licensed under the Apache Software License version 2.0 with exceptions like the Linux kernel, which uses the *GNU General Public License (GPL)* [14].

In most cases, *Google Mobile Services (GMS)* - a collection of Google applications - is added to the system. This collection uses a different licensing approach called *Google licensing model* and contains apps like Google Maps, YouTube, Play Store, and Google Play Services. As the Android OS evolves, different versions like Android 4.4 or Android 8.0 exist. In general, new versions introduce new features, changes in the API, and security patches. Android's major API versions are named by a candy-word, whereas developers more often use the precise API-level numbers. *Table 2. 2*[23] provides data about the relative number of devices running a given version of the Android platform.

The distribution of platform versions of the Android ecosystem is significant for the development of apps. As shown, the older Android versions *Gingerbread* (API level 9 and 10) and *Ice Cream Sandwich* (API level 14 and 15) represent a tiny fraction of the ecosystem.

Table 2. 2: Platform version distribution of Android

No.	*Version*	*Codename*	*API*	*Distribution*
1	2.3.3 -2.3.7	Gingerbread	10	0.2%
2	4.0.3 -4.0.4	Ice Cream Sandwich	15	0.3%
3	4.1.x-4.3	Jelly Bean	16	1.1%
	4.2.x		17	1.5%
	4.3		18	0.4%
4	4.4	KitKat	19	7.6%
5	5.0	Lollipop	21	3.5%
	5.1		22	14.4%
6	6.0	Marshmallow	23	21.3%
7	7.0	Nougat	24	18.1%
	7.1		25	10.1%
8	8.0	Oreo	26	14.0%
	8.1		27	7.5%

The same applies to the newest Android release, *Oreo*. If an Android app distributed for *Lollipop*, *Marshmallow* (API level 23), and *Nougat* (API level 24 and 25), it is compatible with nearly 75% of the devices. *Figure 2. 3 [13]* Illustrate the worldwide Market share of the mobile operating system.

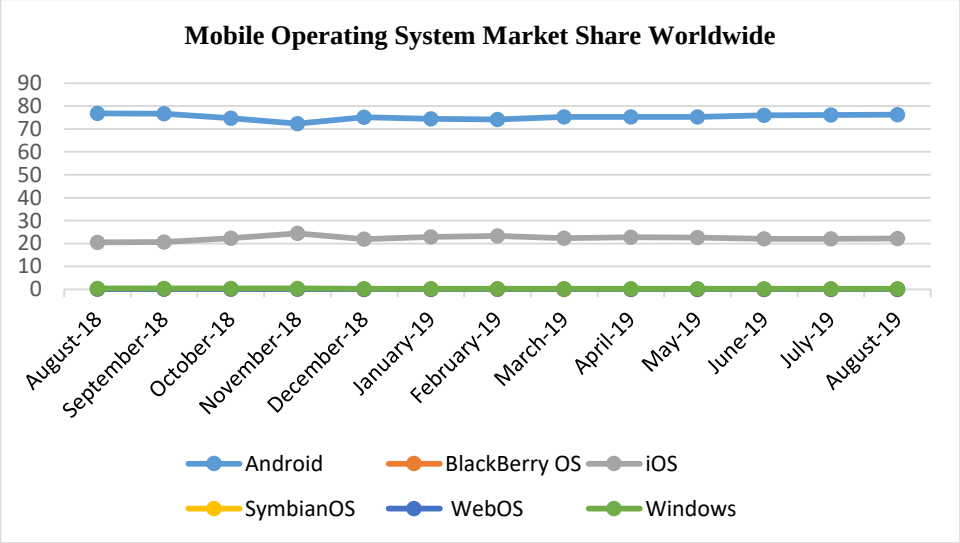


Figure 2. 3: Worldwide market share of mobile operating system

2.4.1. Android Operating System

In general, android is a software stack, which illustrated in *Figure 2. 4*[13]. At the core, it contains a Linux kernel in a modified version. This kernel is known for well-designed security features and enables device manufacturers to develop hardware drivers for a well-known kernel. The overlying *Hardware Abstraction Layer (HAL)* mainly contains APIs for the upper layers in order to use hardware in a unified and straightforward way. The upcoming layer is composed of two modules. The left one contains native libraries like OpenGL, Webkit, or SSL/TLS (Secure Sockets Layer/ Transport Layer Security), which provide essential features for applications. The right one contains *ART*, a modified *Java Virtual Machine (VM)*, in order to run user applications not implemented in native code. ART executes *Dalvik Executable Format (Dex)* and uses the Dex bytecode specification [15]. The predecessor is called Dalvik. Apps constructed for Dalvik will also run on ART. Eventually, some techniques will not be available. ART comes with *ahead-of-time compiling (AOT)*, which will perform complete bytecode translation after installing but before running the application. ART also provides improved garbage collection and new debugging features [15]. The next layer, *Java API Framework* contains the entire feature set of the Android OS accessible through APIs. These APIs are fundamental components for building Android applications. One example is the *View System* to build the app's *user interface (UI)*. Another one is the *activity manager* that manages the life cycle of apps. A *content provider* allows access to data from other apps.

Each of the described components relies on the security of its underlying components. Almost every system component located above the kernel is restricted by an application sandbox technology. Only a small amount of the Android OS is running with root privileges by default [16].

Sandbox technology means that for every application, Android assigns a *unique user ID (UID)* and runs it within its process. With this approach, memory, and file system, isolation can be enforced. The kernel application sandbox prevents uncontrolled mutual interaction between applications. Access to other apps or resources made available solely via *Inter-process communication* through the kernel. There is no distinction between native code and byte code applications. All user apps are restricted by an enforced Sandbox. Without access to specific features, these apps would have very restricted functionality. Therefore a permission system is established, which grants apps access to specific features.

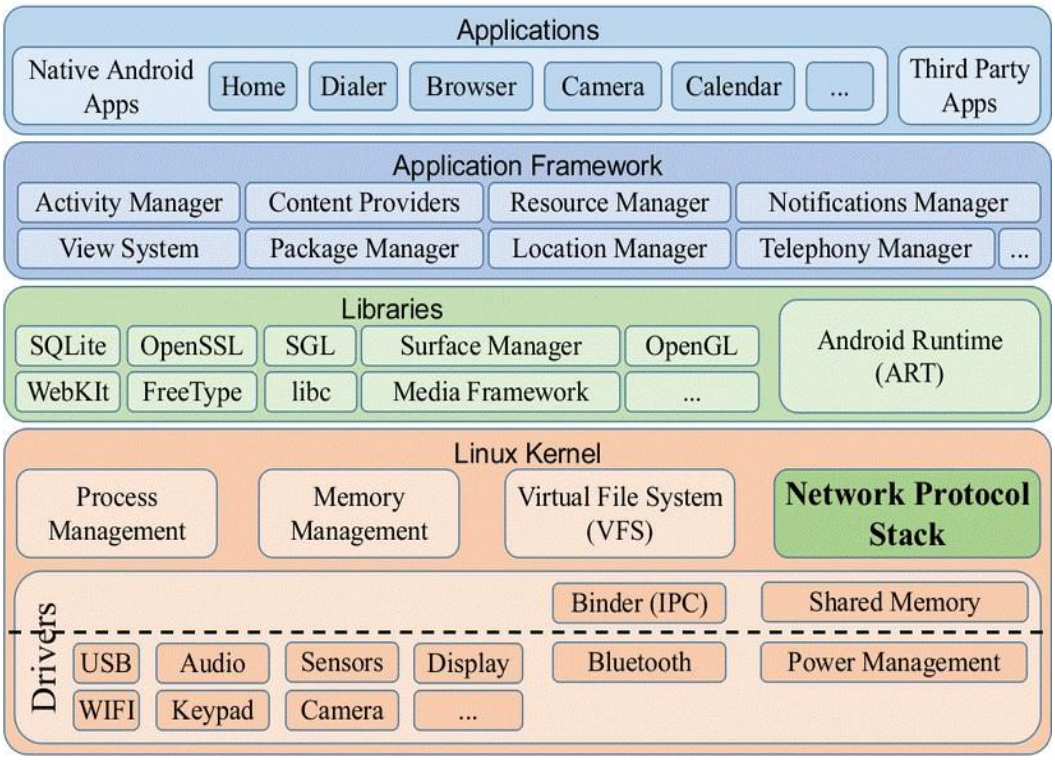


Figure 2. 4: The Android stack

The only way to bypass this system is to compromise the Linux Kernel by itself [17]. The Android OS runs on various devices or emulators, where the target architecture for the OS emulated on the host system.

The Android OS booted by using a bootloader, this can vary on devices manufactured by different vendors. The bootloader is capable of booting into a recovery image, which will provide a debug mode for the device. Vendors often encrypt and lock their bootloaders to prevent modifications of the booted Android operating system. Unlocking a bootloader will enable altering or replacing the OS or recovery image. One way of rooting a device will use an unlocked bootloader to install custom recovery images, which can modify the system. Rooting, in general, is used to gain superuser privileges on a system, which is usually prevented by default. This approach will allow the user to change the system.

2.4.2. Android Apps and its Structure

The market share and customization abilities of the Android ecosystem are mainly the results of the ability to install applications of 3rd-party providers. A developer can develop an Android application and publish it via the Google Play Store. Due to the openness of Android, it is

possible to install apps via alternative market places or through a directly connected device. The *Apk (Android Package)* file format used to transfer and install mobile applications on an Android device. This Zip file containing several different files and folders.

Every application runs in an own virtual machine. In order to enable the specific functionality of the application, a permission system is used. Every app has to declare a permission list to access specific features like Bluetooth, GPS, or the contact list. Before Android 6.0 (API level 23), permissions need to be granted by the user before installing. In order to install an app, the user had to grant all permissions. Only a hidden feature was able to allow and reject individual permissions in Android 4.3. The so-called *App Ops* feature was later removed in Android 4.4. Since API level 23, the permission system was changed by introducing *Runtime permissions*. Permissions split into two groups: *healthy* and *dangerous permissions*. Standard permissions are automatically granted by the system, whereas dangerous permissions must be approved by the user explicitly [18] instead of requesting upfront, the dialog happens during runtime. *Figure 2. 5* contains the four core components of an Android application.

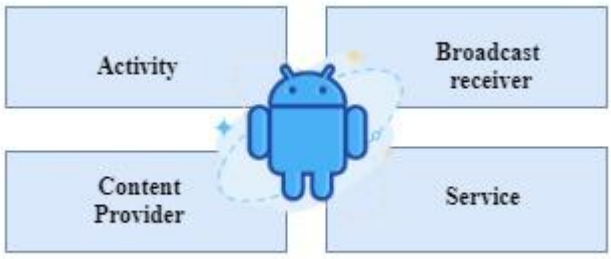


Figure 2. 5: Android core components

An activity interacts with the user and symbolizes a single screen with a user interface. To use an activity, developers can subclass the activity class and implement the app-specific program code in the appropriate life cycle methods. Activities are used as the main entry point for applications.

A service is a component to process a specific task in the background. It can run for a long time and does not require a user interface. To process data or wait for network data, developers are encouraged to start a service component to prevent the UI thread from blocking.

A *broadcast receiver* is a component that is responsible for transmitting messages between Android applications. The OS uses a broadcast receiver to signal status messages like screen status.

Content providers are used to managing data in Android apps. The component allows an app to read or write data like contact information through an API. In contrast to other programming languages, Android has no single main entry point like the main-method. Communication between components or apps is established through asynchronous messaging called *intents*. The standard Android archive format Apk contains the components depicted in *Figure 2. 6.*

Figure 2. 6: Contents of an Android package

The AndroidManifest.xml contains essential information about the application. It is formatted in XML and contains necessary information like app name, app version, or target API level of the application. It also contains the minimum API, which is suitable for the app. If the host Android system is lower, the application will not run on the device. The manifest also contains the permissions required by the app. An entry in the manifest is required for services, activities, content providers, and broadcast receivers. However, a broadcast receiver can also be declared in a dynamic way [19].

The classes.dex file contains the bytecode of the Android application. The general design of the bytecode is described on the website of the Android source code. The Dex file has a reference limit of 65,536 methods. If an app exceeds this limitation, the multitudes feature can be used, which creates additional numbered Dex files. The Android UI is defined by particular resource files in the XML format. They are separated from the program code. User interface elements for an activity, for example, text fields or buttons, are defined within the XMLs. All resources are located in the folder res. The interconnection between UI elements in the program code and the definition file is realized by unique ids and the Android R class. Details regarding the R class

are stored in the resources. arsc file. The assets folder contains additional files, which can be accessed through the application. If native code is used in the application, the compiled files are stored in the lib folder. The folder META-INF contains information like hash sums for the files in the Apk and the appropriate digital signature. *Figure 2. 7* [20]depicts the build process of Android applications.

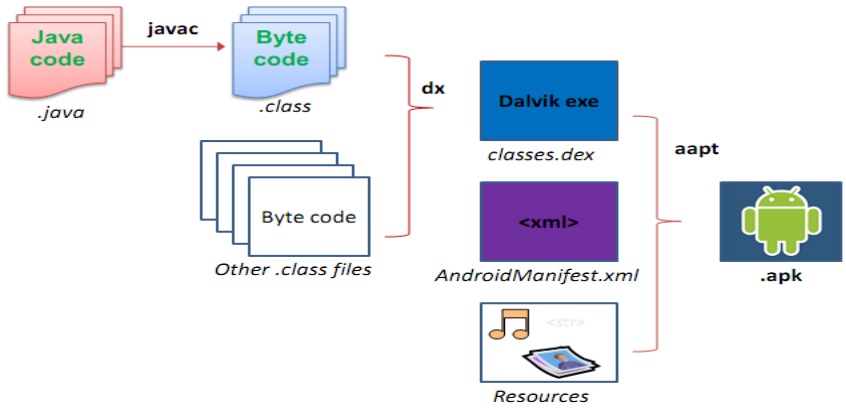


Figure 2. 7: Build process of Android applications

2.5. Android Malware Threats and their Evolution

Malware is a program that is specifically designed to perform different functions, including gaining access, encrypting, stealing, and damaging sensitive data without the knowledge and permission of a user. Smartphones have become an obvious target for malicious actors on the rise of the mobile phone's popularity. People started using their pocket devices more than PCs, carrying them all the time and using them for different purposes, starting with playing games and using multimedia programs, ending with personal conversations and confidential matters, like internet banking, financial transactions, and storing sensitive personal data. This makes smartphones more vulnerable to malware attacks and becomes the target for information and identity theft [21]. Android Malware Threats and their Evolution Types of threats found on Android mobile applications, which are summarized in this section.

2.5.1. Malware

Attackers want to gain access to a device by installing malware on it. The purpose is to steal data or damage the device. Malware is installed by tricking the user into installing a legitimately looking application or in most cases, to exploit a vulnerability in the device, e.g., a security flaw in the Web browser.

2.5.2. Spyware

One of the most common types of malicious applications for the Android platform is spyware. It is designed to get sensitive information from a victim's system and transfer this information to the attacker. Spyware can be commercial and malicious. Commercial spyware is applications installed on the user's handset manually by another person specifically to spy on the user, while malicious spyware covertly steals data and transmit it to a third party.

2.5.3. Grayware

The primary purpose of grayware is to spy on users who installed the software on their own because they thought that it is legitimate software. This is partly accurate because the authors include real functionality as advertised. However, they also collect information from the system, such as the user's address book or his browsing history. The main goal is to gather information for marketing purposes, etc.

2.5.4. Fraudware

Corresponding applications are installed by tricking the user into installing a legitimately looking application, which will gain full functionality after sending several premium-rated SMS messages. In contrast to malware, fraudware informs the user about the upcoming charges, but this information is often hidden and not minded by the user.

2.5.5. Trojan

Trojans are applications that pretend to be useful but perform malicious actions in the background, such as downloading additional malware, modifying system settings, or infecting other files on the system. Android malware is mostly Trojans.

2.5.6. Root exploits

Root exploits are possible on Android in order to gain control of the device but are considered a double-edged sword among the security community. Rooting can give the user control over a device and also gives the same amount of control to any application, which gains access to the root rights. Root privileges given to a malicious application can ultimately compromise the device, as the application can theoretically remove the root privileges from the user. This is a security flaw in this case. The malicious application pretends to be normal until it is installed on the user's device, like most Trojans. It attempts, when installed, to use one or more root exploits

to gain root access to the device. An application with root access can replace, modify, and install applications as it wishes.

2.5.7. Bot

Bots are an emerging trend in mobile malware that communicates with and receive instructions from one or more Command and Control (C&C) servers, giving the malware writer remote control over all infected devices. Malware developers obfuscate their code and use encryption to hide critical information susceptible to detect them.

2.5.8. Privilege Escalation Exploits

They are part of applications that use vulnerabilities to gain full access to a device. Every Android application runs in a security sandbox; however, if malware is successful in escalating its privileges, it can perform actions normally not allowed to applications. DroidDream contains two exploits, Exploited, and RageAgainstTheCage that used to gain root access and install a secondary application that allows the malware to install additional applications without user knowledge. Another piece of malware, jSMSHider, was signed with a compromised key that also allowed installing applications without user intervention on any mobile device firmware builds that were also signed with that key.

2.6. Malware Tricks on Android

Existing ways used by Android malware to install on user phones and generalize them into three main social engineering-based techniques: repackaging, update attack, and drive-by download. They are used during the gestation phase.

2.6.1. Repackaging

It is one of the most common techniques that malware authors use to piggyback malicious payloads into popular applications. In essence, malware authors may locate and download popular applications, disassemble them, enclose malicious payloads, and then reassemble and submit the new applications to Google Play and alternative markets. Users can be vulnerable by being enticed to download and install these infected applications.

2.6.2. Drive-by Downloads

The ability to install and to download applications outside the official marketplaces allows malware developers to mislead users to download and easily install malicious applications. It is

a class of techniques where a Web page automatically starts downloading an application when a user visits it. Drive-by downloads can be combined with social engineering tactics to appear as if they are legitimate. Because the browser does not automatically install downloaded applications on Android, a malicious Website needs to encourage users to open the downloaded file for actually infecting the device with malware.

2.6.3. Update Attack

Malware developers insert a particular upgrade component into a legitimate application allowing it to be updated to a new malicious version, unlike the first technique that typically piggybacks the entire malicious payloads into applications.

2.6.4. Remote Control

Malware authors aim to access the device during the infection phase remotely. (93.0%) turn the infected phones into bots for remote control during their analysis.

2.6.5. Phishing Scams

Malicious applications use Web pages or other user interfaces designed to lure a user into giving unconsciously his sensitive information such as account login information to a malicious party posing as a legitimate service.

2.6.6. Premium Rate SMS

Premium rate SMS is an essential way for people to charge purchases to their phone bills. SMS billing is used by many legitimate services due to its ease-of-use as a phone payment mechanism; however, malware can also use premium rate SMS messages to steal money.

2.6.7. Browser Exploits

These exploit vulnerabilities in Web browsers or applications that can be launched via Web browsers such as a Flash player, PDF reader, or image viewer in order to install malware.

2.6.8. Information Leaking

Information leaks expose sensitive data to other applications on the device, like capability leaks. This can be due to storing sensitive data in unprotected areas; the application gives the information to anyone who knows how to ask.

2.6.9. Threats Based on Vulnerabilities

Malicious developers can exploit them, causing various damages, such as using root rights to access sensitive information. It is important to note that applications in this class of threats can be developed for no malicious actions but access more sensitive data than required by it to perform tasks.

2.7. Analysis of Android Apps

In general, program code analysis of applications divided into the dynamic and static analysis. It describes a research field which tries to extract information about the behavior of an application. Usually, static analysis involves an automated tool that takes the source code of a program as input and examines the code without executing it. Then it returns results after checking the code structure [22]; this often involves sequences of statements and how variables are processed throughout different API calls. The static analysis covers the entire code while requiring fewer resources [22]. Disadvantages are the lacking detection of dynamic code loading and issues with program code obfuscation.

The dynamic analysis runs the application in a prepared environment to examine its behavior during runtime. It is often costly to implement and expensive on resources during the run, but API calls, network traffic, and other measurements gathered. Dynamic analysis lacks code coverage because only code is executed, which meets specific conditions in the analysis environment [22].

Several libraries and complete toolkits exist to analyze Android apps. For the static analysis, the APK disassembled with tools like *Apktool* [23]. It helps to disassemble the classes.dex to human-readable syntax and resources like XML-files or assets nearly to their original format. An assembler/disassembler library for the Dex format is *SmaliBaksmali* [24]. It also contains DexLib2, a component that provides a Java presentation of the Dex bytecode. It has its syntax and fully supports Dex functionality. It is written and maintained by Ben Gruver. These tools to statically analyze Android applications, but do not provide a complete static analysis.

An example of a dynamic analysis tool is the *Mobile Security Framework (MobSF)* [25]. This framework provides security analysis for Android, iOS, and Windows applications. It is released under the GPL3. For Android, MobSF supports dynamic and static analysis. It uses Python 2.7

and Java 1.7 and provides a web interface and a simple API. The main functionality is provided by the web user interface; it is easy to choose a file and start an analysis. The results of static analysis are comprehensive. The dynamic analysis executed on a VirtualBox *virtual machine (VM)* or a connected device. Static analysis can be performed using the API; however, a dynamic analysis is not controllable through the API.

2.8. Machine Learning

This section deals with basic machine learning knowledge. Therefore two general approaches are described in subsection 2.8.1 and 2.8.2. Necessary information about feature engineering is described in subsection 2.8.1.1, followed by detailed machine learning algorithms is described in subsection 2.8.1.2. The upcoming lines will motivate and introduce the topic. Training is done based on the dataset, and the model that built is subsequently used to make predictions. *Figure 2. 8* [26]shows the general workflow of the machine learning process.

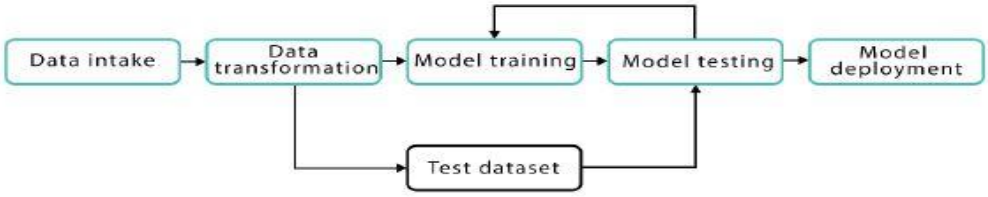


Figure 2. 8: The general workflow of the machine learning process

As can be seen, the learning process consists of five stages:

1. ***Data intake:*** the dataset is loaded from the file and saved in memory.
2. ***Data transformation:*** the data that loaded at *step 1* is transformed, normalized, and cleared to be suitable for the algorithm.
3. ***Model Training****:* a model is constructed using the selected algorithm.
4. ***Model Testing*****:** the model that constructed *step 3* is tested using the test data set
5. ***Model Deployment***: at this stage, an optimal model selected.

The machine learning algorithms can be divided into two main groups, which outlined in the next subsections.

2.8.1. Supervised Learning

The *supervised learning* approach can automate a decision process from the generalization of known examples and specific input data. Used to predict a specific result [27]. The data is labeled and split into training and testing data. The training data is fed into a supervised machine learning algorithm to train the model. Subsequently, the test data is used to verify the effectiveness of the model by comparing the predicted label with the test label of the data. Supervised learning divided into classification and regression algorithms.

Classification is used to predict a specific class out of a given a choice. An example is spam prediction, where two classes are defined: *spam* and *not spam*. Regression, in contrast, is used to predict continuous value like the income of a person [27]. To introduce standard terms in machine learning, a simple data approach is shown. The example is taken from the book of *Ethem Alpaydin* named *Introduction to Machine Learning - Third edition* [28]. The upcoming task should classify applications out of a set of applications. Therefore a training set with applications and the appropriate class membership are provided. Android applications are labeled as a positive example, whereas other applications are labeled as negative examples. Provided attributes for determining the class are permissions and *intents*. These input variables or features are defined as the *input representation*. Other features may be useful for this classification but are due to complexity [28].

An application is represented by a vector x and its label r, as shown in Equation 2.1 [27], and *Equation 2.2* [28] Vector x contains the permissions as $x1$ and the intents as $x2$. The label r is set to 1 in case of a positive example; otherwise, it is set to 0.

$$x = \begin{bmatrix} x1 \\ x2 \end{bmatrix} \quad \text{----------(2.1)}$$

$$r = \begin{cases} 1 & \text{if } x \text{ is a positive example} \\ 0 & \text{if } x \text{ is a negative example} \end{cases} \quad \text{----------(2.2)}$$

The entire training set is named X and shown in *Equation 2.3 [27]*. Every application has an index t with the appropriate vector x^t and labels r^t. The training set contains N examples in total and starts with index $t = 1$.

$$X = \{x^t, r^t\}_{t=1}^{N} \quad \text{----------(2.3)}$$

The data of a sample training set for the applications and their appropriate label depicted in *Figure 2. 9* [28]**.** The feature *permissions (x*1*)* can be found at the x-axis, whereas feature *intents (x*2*)* is represented by the y-axis. Every application depicted as a circle, which filled with a + if the application labeled as a standard application and − if not.

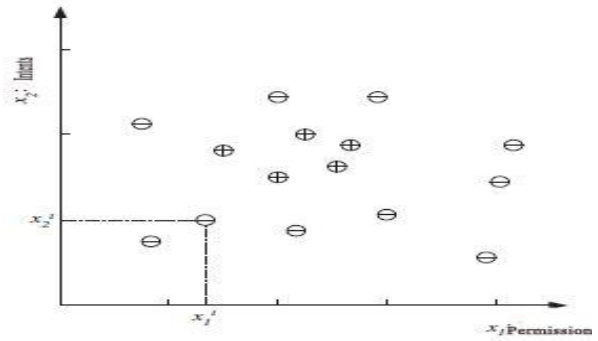


Figure 2. 9: A Training set with labels

2.8.1.1. Feature Engineering

In order to use machine learning, access to an application-specific data collection is mandatory. Every record in the collection contains information in the form of data, generally in a specific format. It is essential to understand the data and, for practical usage of machine learning, to represent data correctly, so it fits best for the application. This is done by *feature engineering*.

2.8.1.2. Classification Algorithms

Classification is a supervised learning concept in which the problem is identified and classified into their appropriate categories, on a given trained dataset containing instances whose category membership is specified. In supervised learning, the classification approach is based on input data to the computer program that learns from it. The upcoming subsections will introduce the frequently used machine learning algorithms in Android malware detection.

2.8.1.3. Decision Trees and Random Forests

This section deals with a supervised machine learning algorithm named *decision trees* and elaborates on the ensemble approach *random forests*. Decision trees are used for supervised learning. They can be applied to regression and classification problems. A decision tree is composed of decision nodes and terminal leaves. They are connected through edges. The number of outgoing edges to other child nodes can be in binary or non-binary form.

The process starts at the root node and for each node, which is not a leaf, one or several appropriate splitting features *fi .,.,.* are assigned. If one splitting feature is used, the tree is called univariate. If several splitting features are used, the tree is called multivariate. The assignment of splitting features *fi,.,.* is crucial for the decision tree construction algorithm. The splitting features will split the node into child nodes. Usually, this is done by an *impurity measure* calculated for the corresponding subset S_q of the training data set *S* [29]. At the top of the tree, the most significant features for decision nodes are used. The child nodes at the bottom will assign the data points to their categories in a more excellent way. A split is pure if after the split, for all branches, all the instances choosing a branch belonging to the same class [28]. This results in a leaf node, which terminates the branch of the tree.

The advantages of decision trees are its simplicity, little data preparation, included feature extraction and the interpretability of the model. This results in the ability to visualize the model [28]. Decision trees can handle numerical as well as categorical data. Disadvantages are overfitting and instability. If a small number of data changes, it can result in a completely new tree structure. If classes dominate the training set, it results in possibly in a biased tree.

One solution to mitigate overfitting is limiting the maximum depth of the tree. Another would be to set a threshold for the minimum number of samples to create a leaf node. A biased tree prevented by balancing the data set beforehand. Decision trees can be used in an *ensemble approach*. This means several models using the same machine learning algorithm are build and used for prediction. It will affect the learning error in comparison to one single machine learning algorithm. Two examples are *bagging (bootstrap aggregating)* and *boosting*. Bagging will randomly choose subsets from the data set. This is done by bootstrapping, which means a random sampling with replacement. The resulting bootstrap subset is used to train the specific model. This is done in parallel and independently [30]. For decision trees, *random forests* are a simplified bagging approach. The user-specified the number of used trees in the forest. Bagging helps to reduce the variance and therefore prevent overfitting of the model. A final prediction is given by averaging the results of the individual trees in the forest. They will often provide better results than an individual model.

Boosting in comparison works sequentially. A model chooses a subset of training data and trains the model. The training data is then evaluated on the model. At each step of boosting the training,

data is reweighted, such that incorrectly classified data elements get a higher weight in the new training set [30]. Data elements with higher weight preferred in the subset for the next boosting step. Boosting will improve bias. *Table 2. 3* [49] illustrate frequently used machine learning algorithms in malware detection

2.8.2. Unsupervised Learning

Unlike supervised learning, the results of the output data are unknown. There is no teacher for training the algorithm and no labels that are assigned. The machine learning algorithm receives only input data and the task to extract knowledge out of the data[28]. Imagine a collection of blog entries and the need to retrieve the topics. For this problem, the text fed into an unsupervised machine learning algorithm, which identifies the topics. In this case, one can limit the number of topics, but it is not possible to predict the allocations of the individual blog entries [27]. It is challenging to evaluate unsupervised learning algorithms.

Table 2. 3: Frequently used machine learning algorithms in malware detection

Algorithms	***Advantages***	***Disadvantages***
Naive Bayes	Even yet, the conditional independence assumption rarely holds true, NB models make surprisingly well in practice, particularly for how simple they are. They are relaxed to implement and can scale with the dataset.	Due to their sheer simplicity, NB models are often compacted
Support Vector Machines (SVM)	High accuracy and speedy, deal with significant dimensional and non-linear problems;	Heavy storage burden require to choose kernel function and repeatedly call parameters;
Decision Tree	They are robust to scalable, outliers, and naturally, model non-linear decision boundaries.	Easy to overfitting;
Random Forest	Overcome overfitting, deal with high dimensional data	Accuracy depends on tree number, Sensitive to the unbalanced sample set.
K-means	Rapid to converge, self-optimize capability;	Required to predefine value of K, sensitive to outlier;
K Nearest Neighbors	High precision and accuracy, nonlinear classification, no assumption of features;	Sensitive to unbalanced sample set, heavy computing burden;
Logistic Regression	the algorithm can be regularized to avoid overfitting, and Outputs have an excellent probabilistic interpretation	weak to handle high dimensional data;

2.9. Blockchain Technology

2.9.1. Overview of Blockchain

Blockchain-based systems offer the possibility for their users to insert their data into this distributed ledger. These so-called blocks inserted in chronological order, and a so-called block number is incremented continuously and can be used to assure global order [31]. The whole user base of a blockchain is guaranteed to have an identical data state of the blockchain. Each member of the blockchain is recognized by one or multiple unique identities called addresses. While in the foundation blockchain, where mainly a tool for transferring money, without relying on a centralized party, the focus is now shifting towards a broader usage. Some of the newer uses of blockchains include digital identity[32], Malware detection, e-voting, and governance.

2.9.2. Architecture of Blockchain

Each block points to the carefully previous block via a reference that is fundamentally a hash value of the former block called *parent block*. The initial block of a blockchain is called a *genesis block*, which has no parent block. *Figure 2. 10* [41] depicts the architecture of blockchain.

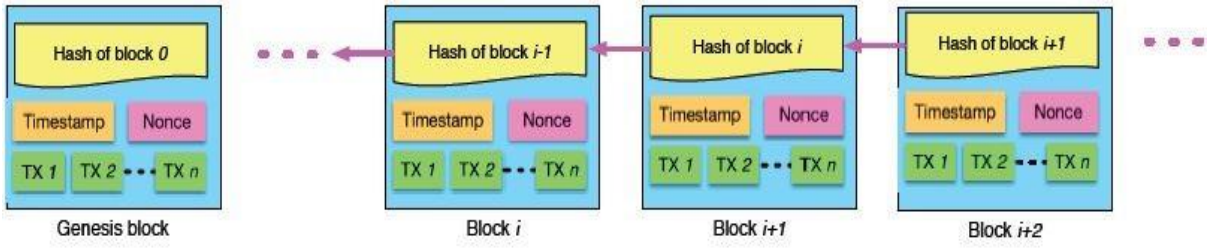


Figure 2. 10: Example of blockchain consists of a continuous sequence of blocks

2.9.3. Characteristics of Blockchain

In summary, blockchain has the following key characteristics [33].

- ***Decentralization****:* a transaction in the blockchain network can be conducted between any two peers (P2P) without the authentication by the central agency.
- ***Persistency****:* each broadcasted block validated by other nodes and transactions checked. So any falsification detected quickly.
- ***Anonymity****:* Each user can use the blockchain with a generated address.
- ***Auditability****:* users verify and trace the previous records easily by accessing any node in the distributed network.

2.9.4. Peer-to-peer (P2P) Network

Blockchain technology is on peer-to-peer transactions based; this means that there are no intermediaries in the transaction process. The introduction of peer-to-peer (P2P) networks has made a massive difference in the digital distribution of services and information sharing. The peer-to-peer system differentiates from the traditional client-server architecture, in which the data is stored in a particular server and distributed only when a related party demands it. This kind of system called one-to-many since the parties involved must rely on a centralized source of information. P2P network, on the other hand, is built upon the decentralized network, where different peers share information without the central.

2.10. Related Works

This subsection introduces related and most recent scientific research with a focus on Android malware detection. The papers use different analysis approaches and machine learning techniques. We discuss the gap and identify issues to improve all these works.

Behavioral-based detection systems may cause extra cost in the deployment environment because mobile handsets have limitations in terms of CPU and battery. To overcome this obstacle, Damopoulos et al. [34] present a proof-of-concept mobile Intrusion detection system architecture deployed both on the host and the cloud. Their architecture consists of event sensors that collect events from the device to construct behavioral signatures [like system calls, inter-process communications, hardware sensors, API calls, system services (e.g., user SMS), and any library call]. System managers to collect signatures and forward to decision engines, decision engines to fulfill detection mechanisms on device, cloud manager to apply decision algorithms on the cloud. Their system decides based on sensitivity, which changes according to the user preferences on system sources to provide flexibility. Four different detection mechanisms from the previous work, namely SMS Profiler, iDMA, iTL, and Touchstroke, and then the Random Forest algorithm is used as the classification algorithm. They use the Performance (CPU and memory consumption) and Timeliness (train and test time) metrics to evaluate the effectiveness of their real-time Intrusion detection system. Cloud-based detection is found to have lower training and testing time, but the overall detection time of the cloud is worse than host-based detection because of communication delays. However, CPU performance shows that battery life is affected by on-device detection. As a result, they conclude that a hybrid solution performing

the most heavyweight detection tasks on the cloud and the more time-sensitive ones on the host would be a better solution.

DroidMat, developed by Dong-Jie Wu1 et al.[35] a system used as a static feature-based mechanism to detect Android malware by considering requested permissions, intent messages passing, API calls, and components of applications (activity, service, and receiver) as static information. The dataset used in the study includes 238 Android malware collected from a free Android dataset, Contagio mobile, and 1500 benign applications downloaded from the official Android market and verified through the website of VirusTotal malware detection community. As the first step, K-means and EM clustering methods are used to enhance the malware modeling capability. Then kNN (k-nearest neighbor) and Naïve Bayes algorithms are trained and tested to detect unknown malware samples. Different combinations of static features tried, and the feature set consisting of permissions, intent messages, and API calls are found to be the most precise. Finally, the combination of k-means as the clustering and kNN with k=1 as a classification algorithm chosen as the best result.

Huang et al. [36] Used static analysis and evaluated four classifiers (i.e., AdaBoost, NB, DT48, and SVM) on application permissions to detect anomalies. They detected malware with a rate of 81% using the Naïve Bayes classifier. Thus, they requested that permission-based analysis is a quick filter for identifying abnormal applications. Though permission-based analysis is less effective involving malware such as Basebridge, which hides an up to date version within the standard application and, as a result, slips into a mobile device without the user's knowledge and bypasses the permission. Thus, due to the weaknesses as mentioned above in signature-based technique and static analysis, while in this research, hybrid analysis is used by combining static and dynamic techniques is adapted.

Hyo-Sik and Mi-Jung [37] in which they applied machine learning on selected features of Android applications. The weakness of their work is that they evaluated their proposed system with five malware families, whereas in this research, more than five malware families used, which in turn makes the result much reliable.

Kabakus et al.[38] Develop a permission-based Android malware detection framework that uses a static analysis malware detection system. APK Auditor consists of three components: (1) An Android client, (2) A signature database, (3) a central server that communicates with both the

Android client and the signature database, and handles the analysis process. APK Auditor software architecture has two phases: training and evaluation phase. In the training phase, both benign and malware applications are submitted to the train system (central server) and store the analysis result on the signature database. In the evaluation phase, the application is sent to be analyzed to a central server. Then, from the analyzed result, the classification result is reported to the user. APK Auditor uses static analysis techniques based on permissions in order to extract features for Android applications.

Liang and Du [39] present a similar approach to APK Auditor, except they combine permissions in sets. This approach brings difficulty for application classification when the number of permissions in a set increases. As the number of permissions in a set is increasing, more permissions are needed to declare an application as malicious. If one of the permissions in a set is lost, even the analyzed application contains malicious content, and the system does not classify the application correctly

Zhuang et al. [40] to address the problem of identifying malicious codes in malware and extracting the matching features in mobile devices, they build a consortium blockchain framework, which is composed of a detective work consortium chain shared by check members and a public chain shared by users. Primarily, given different malware families in the Android-based system, they perform feature modeling by utilizing a statistical analysis approach, to extract malware family features, including permission and application feature, software package feature, and performance decision feature.

Fetulhak [41] framework has been tested by analyzing both malicious and benign applications collected from different websites. The technique used is shown to be an effective means of detecting malware and alerting users about detection of malware, which suggests that it has the capability to stop the attack spread of detected malware. Experimental results show that the proposed framework has detection rate of 96.73% in RandomForest machine learning model which is used during the final development of our proposed framework as an Android application and low rate of false positive rate (0.01), performance impact on the Android system can also be ignored which is only 3.7-5.6% CPU overhead, 3-4% of RAM overhead and the battery exhaustion is only 2%.

2.10.1. Summary of Related Works

The present work is compared based on malware detection methods, machine learning techniques used, based on blockchain and the malware that is disclosed and overcome. Most of the time, computing algorithms such as Naïve Bayes, J48 Decision Tree, Random Forest, K Nearest Neighbors, SVM, and Logistic Regression are used to improve the classifier model. This study concludes that the Random Forest classifier has higher classification accuracy and detection performance. *Table 2. 4* illustrates the base papers and their gap.

Table 2. 4: Summary of related works

Ref#	***Title(Author)***	***Year***	***Methodology***	***Hybrid of***	***Tools***	***Achievement***	***Limitation or Gap***
[38]	Permission-based Android malware detection system (Kabakus etl.)	2015	Uses permissions as static analysis features	Static, On-device, Off-device	…….	Achieves 92.5%	it cannot detect the dynamic malicious Apps
[42]	Android Malware Characterization and Analysis Using Deep Learning (Zhenlong)	2016	Extracts permissions and sensitive API calls as static features	Static Analysis, Dynamic	Baksmali & DroidBox	Detects with 86.7% accuracy	Unrealistic dynamic analysis
[43]	Android malware Analys system using ensemble learning methods (Tom Elastic.)	2017	A combination of permissions and intents. Ensemble approaches to optimize the classification results.	Statistical & ML methods	Pscout & Stowaway	Provides 95.8% accuracy	it cannot detect the malicious dynamically Uses a small amount of data
[44]	A Novel 3-Level Hybrid Malware Analysis Model for Android OS (Arsha)	2018	Local and cloud Machine Learning	Static and Dynamic Analysis	Asset Packaging	Detect 91.6% accuracy	It cannot handle DoS It is not decentralized Used outdated dataset
[40]	Blockchain-Based Malware Detection in Mobile devices (Jingjing etl.)	2018	Perform feature modeling by utilizing statistical analysis method	statistical Method	FlowDroid, & Apache Tika tool kit	92.5% detecting accuracy	It cannot learn from existing data

CHAPTER THREE

3. The Research Methodology

This chapter discusses the research methodology and the process of developing an Android malware detection framework using machine learning and blockchain technology. It begins with a methodology for how the data were collected and processed to alternatively, in the Android malware classification. Finally, the following section clarifies the explanatory information about the detection method, classification, feature selection algorithms, development tools, and framework evaluation method.

3.1. Dataset Retrieval and Description

This section deals with the idea of retrieving the required apps as input data. One core requirement is to collect Android malware and benign apps of recent years and stored them on disk. First, the decision for the Android malware repository is described, followed by the benign app repository. Because *Drebin* and *Genome* Project Dataset repository contains outdated malware, not used within this study. Instead virushare repository requested. An invitation is needed in order to request the virushare service. It was granted with the help of personal email communication, including a short description of the use case and the research institute. Within this study, the zip file VirusShare_Android_APK_2018 and all Android Apk files collected in 2018, is used.

3.1.1. Apk Files

Apk is a file format that is created by Google, which is used to distribute and install an application on the Android operating system. Throughout this study, we refer to various components of an Android installation file, known as Apk. The Apk File component provides easy access to the Apk itself and the information stored in the AndroidManifest via the Manifest Parser. The normal applications collected from the Google Play Store and Appsapk for evaluation. virushare samples are the repository of malware samples to provide access for the researchers to collect a sample of malicious applications [45].

3.1.2. Apk File Repositories

This subsection introduces app repositories used for this study in order to get benign and malware Android applications. An extensive collection named Google Play Store is described, followed by Virusshare repositories.

Google Play Store is the app collection for Android applications. It offers several apps organized in categories. It is not possible to download the apps directly in APK format, but the third-party service *APK Downloader* provides this functionality. Other Android app repositories also allow a direct download of APKs, for example, *APKMirror* or Apkpure. Both websites offer downloads of benign apps from different categories.

Virusshare is a malware repository which provides samples to malware analysts, security researchers, and incident responders access to the samples. They can inspect and analyze the behavior and specifics of a particular malware themselves [46].

3.2. Android Malware Detection Methods

Currently, various Android malware detection techniques introduced to detect and prevent mobile users from attack. Many researchers have been conducting different Android malware detection techniques in different ways. Meanwhile, the hybrid analysis method has been selected for this research work because the hybrid analysis mitigates the drawback of each static and dynamic analysis method and also increases the detection accuracy.

3.3. Development Tools

This subsection outlines the essential tools and libraries used in this study. They are crucial for realizing the proposed functionality.

3.3.1. Design Tool

Drawio is a type of design tool which enables the software designer to reliably create and publish many kinds of diagrams to represent an idea. This design tool provides many features such as flowcharts, workflows, software, Unified Modeling Language diagrams, Business diagrams, database, and ERD [47]. That is why Drawio used in this study.

3.3.2. Android Debug Bridge and Xposed Framework

Android Debug Bridge used for interaction between the host developer machine and the Android device. The Android Debug Bridge (ADB) is a command-line tool as part of Android SDK platform-tools and enables communication with an emulated or physical Android device. It can perform various interactions with the device like installing apps, exchanging data, or launching apps.

The Xposed framework can be installed on rooted Android devices and provides an extensible environment through modules found in the Xposed Module Repository. In this study, the Droidmon module is used, which can change the system's behavior, for example, by adding functionality.

3.3.3. Elasticsearch

Elasticsearch is used in order to store hybrid analysis results; Elasticsearch is a software for building search-oriented applications. Elasticsearch has written in Java and, therefore, cross-platform compatible. It offers scalable indexing performance, powerful searching, and sorting functionality. Elasticsearch is currently released in version 7.1.1 and requires the Java Virtual Machine (JVM) to run on various operating systems. Kibana software is used to visualize data from Elasticsearch primarily and to navigate through the Elastic Stack. Currently, it requires Java 8. By default, Elasticsearch listens to http://localhost:9200.

3.3.4. Machine Learning and Blockchain Technology

The first approach can predict malicious and benign apps; this can be formulated as a binary classification problem and realized while using benign and malware as labels. A training data set involves benign and malicious apps and utilized to train the model.

The first step is to load the results of the hybrid analysis from the database. Afterward, the feature engineering performed, followed by the classification and collected static (see 4.3.1.3) and dynamic information (see 4.3.1.4), transformed into categorical features for feature engineering; additionally, the number of items within a category stored as a continuous feature. An example would be the permissions where every permission assigned to its category. The overall amount of used permissions is stored as a feature as well as continuous data values used

as continuous features, whereby categorical data transformed into categorical features using one-hot-encoding.

Random forest is a bagging ensemble approach of decision trees, used as the classification algorithm. More details can found in section 2.8.1.3. Random forests and Extra tree is used due to their simplicity, effectiveness, and simplicity. Also, do not require feature preprocessing and prevent overfitting, which usually occurs when decision trees without further configuration used. Finally, Blockchain is used as a distributed database for classification results and implemented by java programming language.

3.4. Evaluation Techniques and Model Improvement

3.4.1. Evaluation Techniques

This subsection deals with evaluating a machine learning approach and improving the used model by optimizing parameters. All the collected feature values stored in a matrix *X*. The rows contain the data points or observations, while the columns denote the used feature set. An additional vector *y* contains the labels for all observations in *X*. A simple way to evaluate a supervised model is by splitting the data *X, y* into a training data set *X_train, y_train*, and test data set *X_test, y_test*. The size of the training set is set to 70%, while the test size is set to 30%. The training data set used to train the model, whereby the testing data set is utilized for testing. Therefore only *X_test* is used to predict the labels. Afterward, the *y_test* is utilized to compare the results and return the prediction accuracy for the test set.

Cross-validation split the data set into several subsets, named folds. In 10-fold cross-validation, always one fold is used as a test data set and the other nine as a training set. Such standard cross-validation using ten classes is depicted in *Figure 3. 1*. In summary, there are ten possible compositions and ten results. They can be summarized by using the mean or median value. This approach is more robust and precise than a simple train-test split because it takes more different training input into account as well as other different test data.

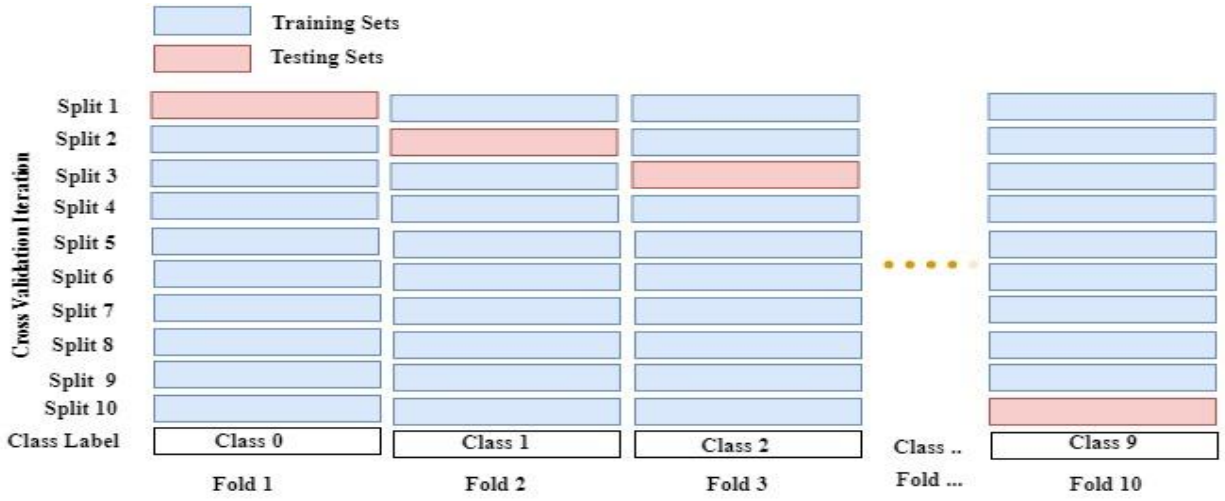


Figure 3. 1: Standard cross-validation with sorted class labels

In order to evaluate a classification, specific evaluation metrics are used. These values are the primary metrics for binary classification and can be depicted in a so-called confusion matrix *Equation 3. 1*:

$$\begin{bmatrix} True\ Negatives(TN) & False\ Posatives(FP) \\ False\ Negatives(FN) & True\ Posatives(TP) \end{bmatrix} \text{----------(3.1)}$$

The actual accuracy measurement can be defined by using the following equation

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \text{----------(3.2)}$$

Precision measures how much positive predictions are positive [48]. The recall is used as a measurement if the avoidance of false negatives is required. The definitions for both measurements depicted in equation 3.3 and 3.4.

$$Precision = \frac{TP}{TP + FP} \text{----------(3.3)}$$

$$Recall = \frac{TP}{TP + FN} \text{----------(3.4)}$$

A harmonic mean of the precision and recall is called *f*1 *score*. It is defined as

$$f1 = 2 * \frac{precision * recall}{precision + recal} \text{----------(3.5)}$$

3.4.2. Model Improvement

In order to improve the model, a possibility is the modification of the used parameters and observing the results. This parameter optimization is using a *grid search*. The first step is to

choose the parameters, which should be used for optimization. Then a predefined range of values is used until the highest score for a test data set is reached. The other parameter optimization used in this study is *nested cross-validation*. In this case, grid search is used with cross-validation for parameter optimization together with k-fold stratified cross-validation to test the prediction results. This would result in an even higher computational effort but results in a more precise evaluation of the optimized model.

CHAPTER FOUR

4. The Proposed Malware Detection Method

4.1. Overview

In this chapter, the high-level architecture of the proposed solution is presented in detail to show each component that exists within the proposed framework. Then, the design modules of the hybrid system using blockchain technology and machine learning discussed.

4.2. The Proposed Framework

The conceptual framework of the proposed solution shown in *Figure 4. 1*, which includes six layers: the application layer, service support layer, analysis layer, classification layer, the blockchain layer, and the network layer. In the application layer, a variety of applications services are provided. Users interact with several applications services without considering the bottom layer of the framework. Typical application features include request service, attach Apk, upload Apk and analyze, validate blockchain, synchronize status, a new block generated, and the associated facts broadcasted.

In the service support layer, the interface between the nodes and the fact-base of malicious codes are provided, including features of consensus mechanism, digital signature, encryption, key management, access control, and synchronization management. The consensus mechanism determines the legitimacy of nodes in the Blockchain, confirms the submitted information, and ensures the legality and validity of block data in the fact-base, and the data encryption encrypts the data. The digital signature makes the data submitted by nodes undeniable. The key management is used to control vital information safely. The access control prevents malicious nodes from illegally manipulating data, and finally, the synchronization block feature updates the fact-base.

In the analysis layer, static and dynamic analysis is done, and the result is stored in Elasticsearch, the next layer is the classification layer in this layer training and prediction is takes place then the classification result stored in the fifth layer which is blockchain layer finally in the network layer, link nodes communicate through a P2P network. *Figure 4. 1* depicted the layers and elaborated in the next sections.

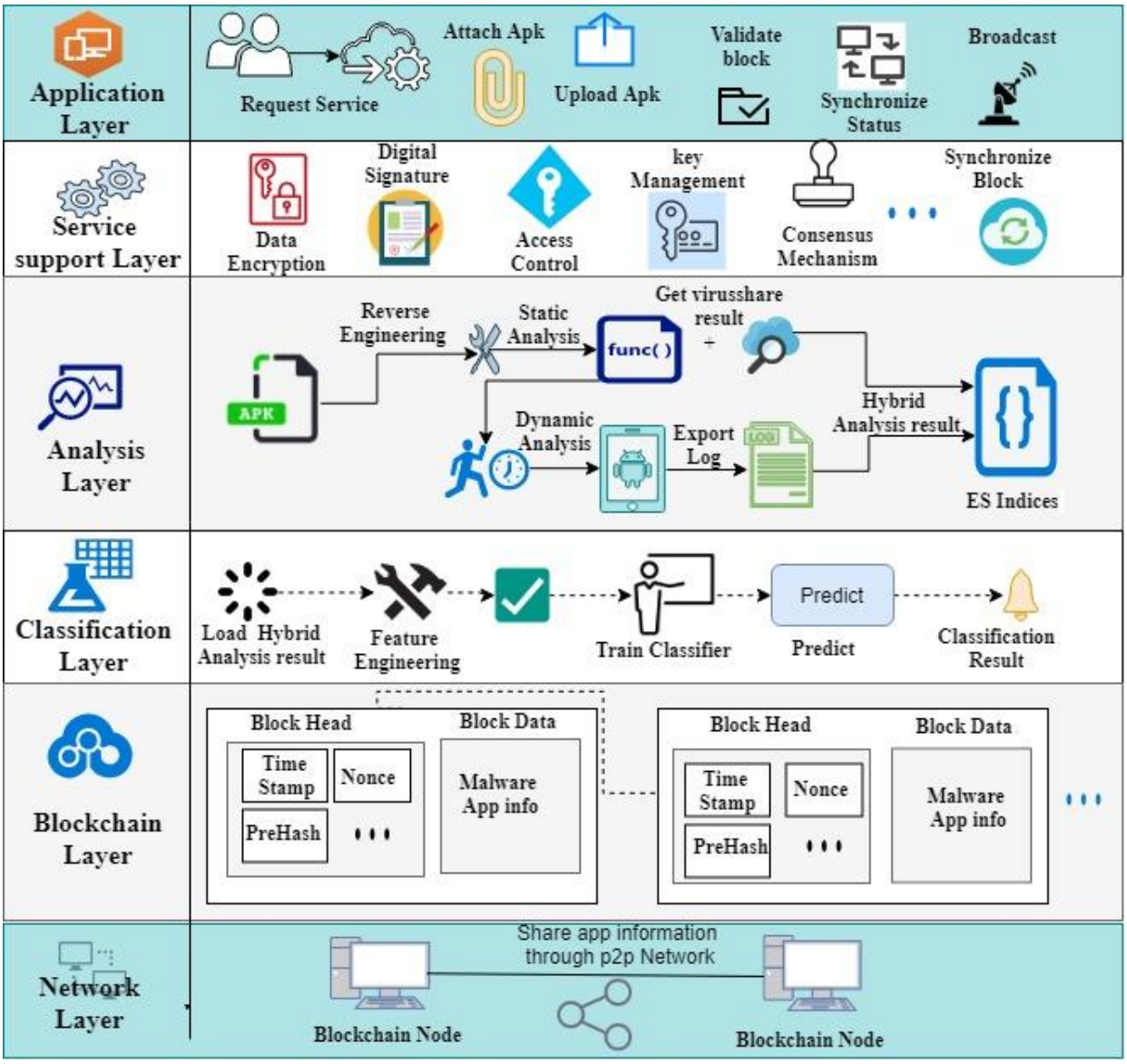


Figure 4. 1: Conceptual framework of the proposed solution

4.3. Proposed Analysis Module

As stated in the literature review part, the two primary methods when investigating the malicious behaviors of android applications are the static and dynamic analysis methods. The dynamic detection method is applied by running an application on an isolated environment and observing behaviors of the application in order to discover the matching behavior profiles of applications with known malware.

4.3.1. Hybrid Analysis

This section introduces the concept of both hybrid analysis approaches, which is layer three in the proposed framework. It is divided into a static and dynamic analysis part. Details can be found in the subsections 4.3.1.3 and 4.3.1.4. The upcoming lines will outline the general motivation of a hybrid analysis. Furthermore, it explains the depicted interconnections in dataset retrieval 3.1; this includes details about the database for storing the analysis results.

The main reasons for a hybrid approach are to utilize advantages of the static and dynamic approach and simultaneously neutralize disadvantages. The paybacks of static analysis are the performance, resource efficiency, and the high code coverage of an application. The main disadvantages are problems in detecting dynamic code loading and handling obfuscation. The main advantages of dynamic analysis, on the other hand, are the ability to handle dynamic code loading, obfuscation, and revealing actual behavior of an application. Disadvantageous is the code coverage and the required resources to perform an analysis.

The dynamic analysis concept discusses the two analysis approaches (see subsection 4.3.1.4). The hybrid approach guarantees the analysis results of both worlds. It is relatively rarely used [49] but enables the machine learning approach to utilize the best fitting features, and this is a significant advantage compared to stick with one approach beforehand.

The dynamic analysis requires the results of the static analysis, such as package names and components, so the static approach must be executed first. The approach performs the static analysis for all apps first, which is followed by the dynamic procedure. It is planned to implement both analyses in Java.

4.3.1.1. Virustotal Service

Besides the core analysis, the online service Virustotal used. A hash sum for every analyzed app will be created, which used to request aggregated virus scanner results. The result used to identify and label the collected apps as malware or benign apps. This is useful since there is no guarantee that Appsapk, google play only contains benign apps, and Virusshare only contains malware. The usage of several anti-virus scanners reduces the training noise for the upcoming machine learning process.

4.3.1.2. Database

A database used to store static, dynamic, and Virustotal results permanently. Because of the effort of the analysis and the value of the generated data, the database should support replication capabilities. The Virustotal service and the dynamic analysis are going to utilize the JSON format for results. Therefore, it makes sense to use a database, which supports this format natively. Therefore it is planned to use individual indices with specific mapping for every analysis approach, and this is going to improve the structure and loading time of the indices. To uniquely distinguish the analysis results of apps, a SHA256 hash sum for an Apk generated and used as a unique identifier within the database. Details about the static analysis outlined in the upcoming subsection.

4.3.1.3. Static Analysis

This subsection introduces decisions and details about the static analysis to provide the foundation for the implementation. It will first elaborate on static data, which can be extracted from an Android application. Afterward, suitable tools and libraries for a realization determined. The essential input for the static analysis is the *Android Package (APK)*. Due to planned machine learning, it is essential to focus on data that is usable as a feature. Therefore it is planned to extract bytecode, android manifest, and asset components features. For a well-structured approach, it makes sense to divide the required data into smaller components. The Apk contains three components. Each contains detailed information about what is going to be extracted and is going to be used as features in a later stage. The components are depicted in *Figure 4. 2.*

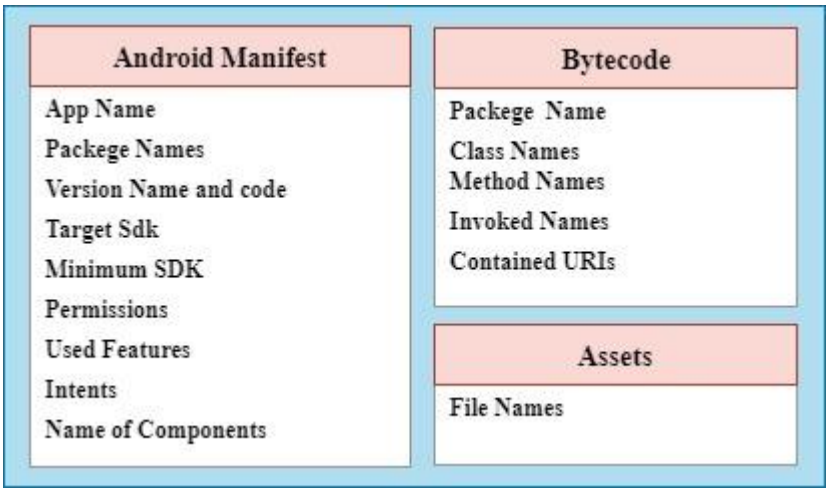


Figure 4. 2: Extracted features through static analysis

The Android manifest contains core information about the application, such as names of the components and permissions. Bytecode represents the classes.dex files. Information about class names, method names, invocations, and URIs can be extracted. Assets contain the names of the included assets. In this work, files from the assets and the lib folder processed. An improved approach would be the analysis of the content type (Media type or MIME) of the assets. Found native code files could be analyzed using fundamental analysis, and this would be an entire individual topic. The analysis of resource files could also be a useful feature. Though, this dropped at the moment due to feature size and the lack of expected benefits. Currently, the multidex feature not handled by static analysis. However, the missing code covered by the dynamic analysis.

In this study, a prototype is implemented using libraries. It is planned to cover the program code and the manifest file with two libraries. To cover the program code, the *DexLib2* library used, which provides much functionality for the Dalvik bytecode. It supersedes the predecessor DexLib. The *APK-Parser* decodes the binary-encoded manifest file and also supports processing the assets files. The final concept for the static analysis shown in *Figure 4. 3*.

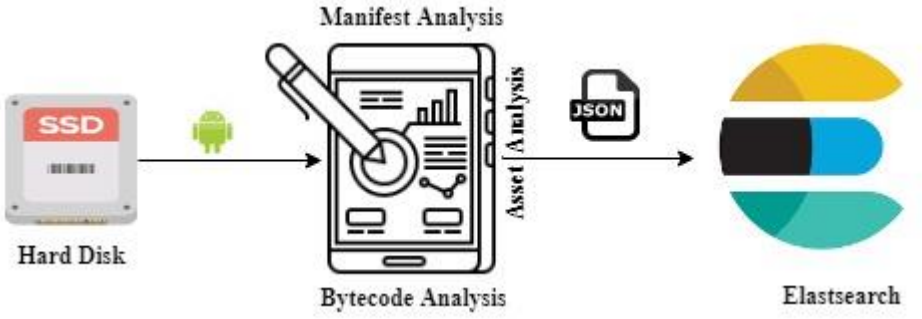


Figure 4. 3: Static analysis overview

It is planned to read the APKs from disk and hand them over to the static analysis shown in the central box of *Figure 4. 3*. The analysis is composed of three components. The *Manifest Analysis* collects the features shown in the appropriate box *in Figure 4. 2*. It uses the library APK-Parser. The *Bytecode analysis* realizes the data collection, mentioned in the Bytecode box in the depiction *Figure 4. 2*. Utilize the DexLib2 library. The last component of *Asset analysis* collects asset information. The results of all components are going to be merged into a single report and stored into the Elasticsearch database. *Algorithm 4. 1* shows a static analysis of the android application.

```
start
intialize analizers
start analizers
load Apk
for(file:apkfile)
    if(type=.apk)
        generate hash code
        if(has report)
          get report
          analyze apk
          update report
        else
          analyze apk
          get virustotla report
          merge the reports
          store report
        End if else
    else
       print invalid type
    End if else
    if(more apps exist)
      go to step 5
    else
      Stop static analysis
    End if else
End for
End
```

Algorithm 4. 1: General algorithm of the static analysis

4.3.1.4. Dynamic Analysis

This subsection deals with the concept of dynamic analysis. According to the goals and the research question, it is mandatory to realize two different approaches to dynamic analysis. The next lines describe what kind of feature is feasible to be extracted from the dynamic analysis. It also discusses the dependencies and tools for a realization. Afterward, the two approaches outlined in more detail.

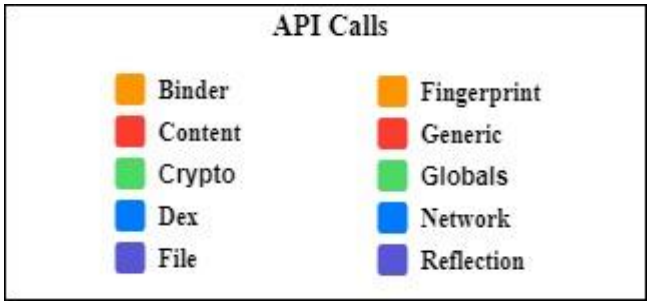


Figure 4. 4: Extracted features through dynamic analysis

In this study planned to monitor Android during run time and track specific system calls and network traffic. The defined API hooks of the default *hooks.json* used. It contains ten groups of API classes, including different assigned methods. The classes are depicted in *Figure 4. 4.*

The category *fingerprint* methods used to identify a device. The group *crypto* includes crypto operations and *reflection*, a method to perform reflection operations. The category *generic* contains miscellaneous operations, the *network* involves methods for establishing a network connection. The group *file* handles file operations, whereas *binder* contains receiver handling and Activity launching abilities. The group *content* covers access to content resolvers, media, and location data. Category *globals* contain methods that store data in global storage capabilities like Android shared preferences. Besides, *dex* covers functionality for dynamically loading Dex files.

The dynamic analysis tools *MobSF* and *CuckooDroid* already have been described in section 2.7. They both provide a comprehensive dynamic analysis. The results of *MobSF* are reliable, but no API is provided for further processing. The results are only visible in the web app or can be exported as PDF. *CuckooDroid* provides an API to retrieve reports and is more suited for the automated analysis. Both tools do not provide an automatic click runner as a feature.

Due to these reasons, an own implementation of dynamic analysis is realized. Specific requirements can always be directly integrated into the implementation. It is also possible to minimize the dependencies of third party code. This analysis can be integrated into the static analysis and would result in a merged project. *Figure 4. 5* shows the steps of dynamic analysis.

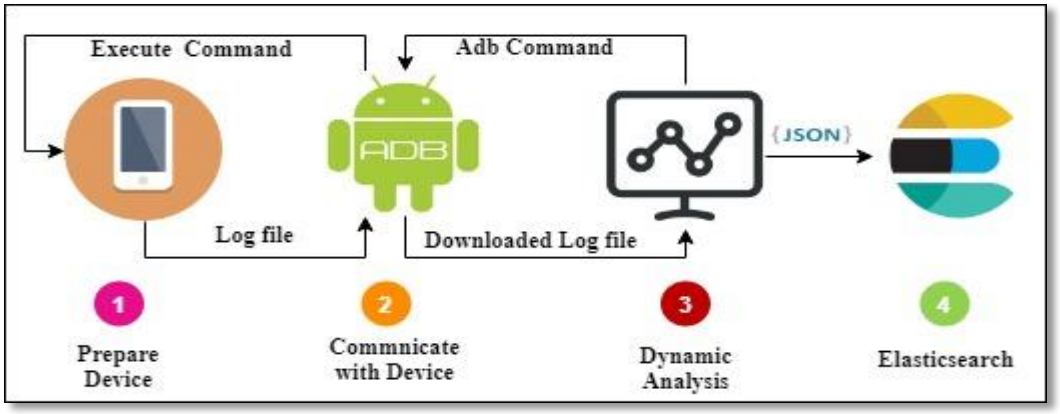


Figure 4. 5: Dynamic analysis overview

Instead of emulators, real devices used due to performance and fidelity of running applications. Since several devices used for dynamic analysis, it is crucial to use a concurrent design for parallel execution on the devices. The primary step is the preparation of the device for the dynamic analysis. The Samsung S5 used as the analysis device, which is easy to unlock and to root. It is widely supported by the community and is relatively cheap to purchase.

It is planned to prepare the device with the Xposed framework and monitor the API calls by using the *Droidmon* module, described in *section 3.3.2*. The *Android Debug Bridge (ADB)* used to interact with the device (*see section 3.3.2*). It is used for installing the app, launching components, clicking UI elements, and dialogs. Moreover, it also handles granting the permissions, rebooting the device, and transferring data.

To address the goals of this study and the research question, two dynamic analysis approaches implemented. They are required to determine the necessity of automated click tools for a machine learning approach. A click tool benefits from using real devices, due to more fluent interactions. The first one does not use any simulated click interactions. Instead, it relies on the defined activities and services of an app and is going to start them iteratively. Therefore the determined component names of the static analysis used. Thus this approach is called the *component approach*. Starting secondary components without the context of their main entry point could result in thrown exceptions, and this potentially prevents starting the components and lose valuable content for the dynamic analysis. The algorithm of the component approach is depicted in *Algorithm 4. 2.*

```
1  start
2  intialize component analyzer
3  get activities
4  for(appActivites: activities)
5     start activity
6     delay
7     check for errors and permission dialog
8     start services
9  End for
10 End component analyzer
```

Algorithm 4. 2: Algorithm of a component analyzer

However, the interactive strategy is going to use an automated click engine. It will launch the main component of the application and click through further linked components to increase code

coverage. It is planned to automatically store displayed elements of every displayed UI, and the elements further processed and clicked. In this way, the components of the UI invoked in a breadth-first search approach. The *dump* tool is going to be utilized to determine the displayed UI elements of the app components. The approach minimizes the probability of raising exceptions due to context lack. This *interactive approach* is going to use a breadth-first search to span a tree; this initially starts as many associated components as possible. Further, the algorithm follows the next branch and repeat the procedure for the containing components and its UI elements. The approach takes care that UI components are not processed twice. The key algorithm of the interactive approach is depicted in *Algorithm 4. 3*.

```
1   start
2   collect click events
3   if(is next and time available)
4      perform click events
5      get recent view
6      if( is view processed)
7        go to step 3
8      else
9         store new click
10     End if else
11  else
12     stop interactive analysis
13  End if else
14  End
```

Algorithm 4. 3: Algorithm of interactive approach

However, a guaranteed termination of this interactive approach remains problematic. A possible solution is the usage of a time threshold, which ensures the termination after expiration. The overall performance is reduced due to built-in timeouts between click events. However, the timeouts are required to act on UI interactions. The key algorithm of the dynamic analysis is depicted in *Algorithm 4. 4*.

```
start
get static report
locate Apk
install Apk
unlock the device
launch app
if(approach=interactive)
   perform interactive analysis
else
   perform component analysis
End if
get API logs
uninstall app
store report
if(more app available)
   go to step 2
else
   stop
End if
End
```

Algorithm 4. 4: General algorithm of the dynamic analysis

The depicted logic is part of a concurrent design, which is going to execute in parallel. One dedicated thread used for each connected device. The dynamic analysis depends on the static analysis results. It will initially process the static report and locate the appropriate APK file using the SHA256 hash. Afterward, the app installed on one of the devices. In the next step, it unlocked, and the main component of the app launched. Either the *interactive* or the *component* approach is going to execute. After executing the retrieval of the API, the log is performed. The log is processed, and the entries for the desired app filtered out. Afterward, the API log checked for successful entries. Otherwise, the process restarted. After API call extraction the app uninstalled, and the results are stored in the Elasticsearch database

4.4. Android Malware Detection Module

The proposed Android malware detection module depicted in *Figure 4. 1* consists of five major parts. The first component loads the hybrid analysis result from Elasticsearch. Then a feature engineering performs the feature extraction, particularly permission requested, from AndroidManifest.xml, and the vector generator carries out to convert extracted features into 0 and 1. After the vector generator extracted, feature selection is carried out in the third component of the framework. As a result of this component, each app represented as a single instance binary vector. Machine learning algorithms use this binary vector and learn from trained data to classify

unknown applications in the fourth component. Finally, the classification model is trained from collected data and ready for supervised classification algorithms in the last component.

4.4.1. Feature Extraction

Feature extraction performed for obtaining the feature data to predict the class of android application. Permission required is a feature extracted from Android manifest.xml to use them in analysis APK files. The feature extraction step performed while preparing the dataset before analysis.

4.4.2. Binary Feature Vector

Each app in the sample represented as a single instance with a binary vector of features, and a class label indicates whether the app is benign or malicious. If the feature is present in the app, it is represented by 1; if it is not present in the app, it is represented by 0.

4.4.3. Features Selection

The feature selection technique is employed to first-rate the most relevant features and to train the proposed classifiers. This method is also known as the mutual information method. Not all features are equally crucial in differentiating benign and malicious apps. Increasing the number of features will also increase the complexity of the model, and overfitting is more likely to happen. So it would be advantageous to know which the most relevant features for the desired scenario. In this study, feature selection made with a feature importance technique.

4.4.4. Classifier

We evaluate the feature selection procedure using different classification models. In this study, a Random Forest and Extra Tree classifier used. In the experiments, we use training and testing datasets: 70-30 and ten-fold cross-validation. Thus, the most accurate training and testing dataset distribution, also the best classifier is found.

In order to generate the model in this study, supervised learning algorithms used. There are different approaches for the classification; however, all of them require known classes for the training, testing, and also validation stages. Therefore, *Algorithm 4. 5* depicts the identification of labels for the presented applications.

```
start
Input: application's APK, threshold
Output: application label
Positive reports <- VirusTotal scan
     results of application's APK
If positive reports > threshold,
    application label = Malware
 Else
    application label = Bengin
End if else
End
```

Algorithm 4. 5: Label the class of applications

4.5. The Proposed Blockchain Module

The framework consists of a public blockchain, composed of the members in distributed malware detection nodes. These members build a fact-base of the distributed malicious codes. The public blockchain is the application chain, open for any user who needs to provide detection and evidence services for joining as a member node. *Algorithm 4. 6 s*hows how the block is created and added to the blockchain fact base.

```
start
load Malware data
for(id:id1)
  if(blockchain size==0)
    block= create genesis block
  else
    block= create block
  End if else
  get difficulty
  mine block
  Add block to blochain
End for
End
```

Algorithm 4. 6: Pseudocode for block adder

4.5.1. Blocks and the Blockchain

Malware features are grouped in blocks and then included in the blockchain. The blockchain is a data structure containing linked blocks of malware data. Each block holds the hash of the former block to create a chain that connects the genesis block to the current block. The hash identifies each block of its header, which is generated using SHA-256. *Figure 4. 1*, layer five shows how two blocks are linked by references in the previous block hash to form a chain.

4.5.2. Block Structure

A block is composed of block size, block header, block number, and the malware data. The block header contains, among other things, a hash of the previous block, a hash of the Merkle root of valid transactions to be included in this block, a timestamp, the current difficulty target for the block, and a nonce.

The blockchain goes back to the genesis block. Each block contains a summary of all transactions that it contains. The transactions are combined using a Merkle tree, also known as a binary hash tree. Merkle tree is a data structure that efficiently summarizes and verifies the integrity of large amounts of data.

4.5.3. The Fact-Base of Malicious Codes Based on Blockchain

In the case of detecting malicious codes, if a problem found in the result of the detection, the related information about the detection collected to a block data and submit to the fact-base, which can be used as the evidence of malicious codes and to update the feature base of the malware family. The information can also be analyzed to provide support for the new detecting rules. In the general framework, each data block includes malware id, API call feature, and static features of the app.

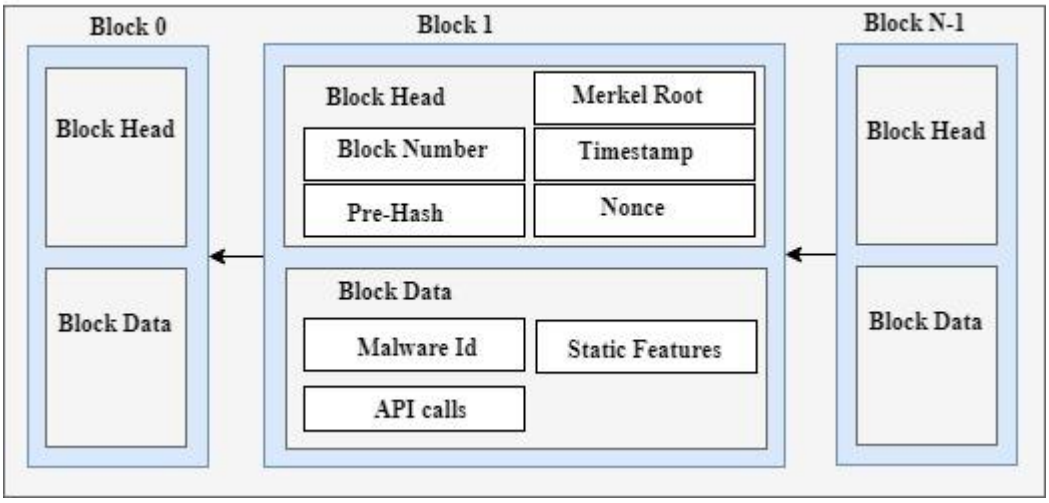


Figure 4. 6: Internal structure of the blockchain.

There is some necessary information, including timestamp, the hash value of the previous block, and a random number for verifying hash values. The data block structure is shown in *Figure 4. 6*, where Pre-Hash is the hash value of the previous block. The block number is used to track

software/protocol updates. Timestamp records the time that the block produces. Merkle root is a hash value calculated by all malicious codes detected in the block, which is used to check whether a test result exists in these blocks. A nonce is a random number before the block, which is recognized as a formal block. Each node calculates the hash value of the block when the block is received from the network. Whether the block is uploaded on the network or is stored on a permanent storage device of a node as part of the blockchain, the hash value of the block is unique, and it identifies a block.

4.5.4. Blockchain Validation

As per protocol, each node needs to verify the correctness of a block before forwarding it to its neighbors, ensuring that only valid block is propagated and included in a blockchain. Every new block gets broadcast to the network so that all the computing nodes are aware of that fact at the time it took place (to ensure the system is double spend resistant). *Figure 4. 7* depicts the process of blockchain validation.

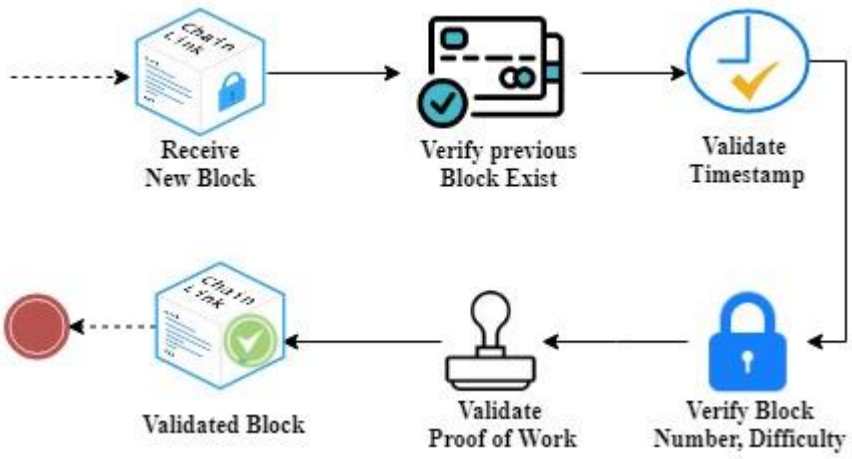


Figure 4. 7: Process of Block validation

In general, all miners independently verify all blocks they receive before propagating them to other nodes.

4.6. Network Module

As the last layer of the framework depicts, a peer-to-peer network is a representative distributed network. *Algorithm 4. 7* shows the pseudocode, which used to create a p2p network in the proposed framework.

```
start
initialize port number,node count
create thread
create socket
get Ip address
wait for node
if(node count<node size)
  accept node
  open socket
  start thread
else
  remove node
End if else
if(node join)
  notify all
else
  kill the process
End if else
End
```

Algorithm 4. 7: Pseudocode for P2P network Initializer

A distributed system consists of numerous nodes. A node can be defined as an individual user in the system. Because Blockchain is a decentralized distributed system, understanding distributed systems are requisite.

CHAPTER FIVE

5. Implementation of the Proposed Framework

5.1. Overview

This chapter describes the prototype implementation of the proposed framework. Each malware detection module implemented separately. Detail about the implementation of the proposed framework and further modules given in the following sections.

5.2. Working Environments

- Desktop Computer
  - Processor: Intel Core™ i7-5005U CPU @ 2.00GHz 2.0 GHz(Giga Hertz)
  - Installed memory (RAM): 8.00GB(Gigabyte)
  - System type: 64 –bit OS, x64-based processor
- Intellij IDEA: 2018.3.4 (community edition): it used as scripting editors for both java and python programming languages. The following are runtime execution environments that exist in IntelliJ IDEA (Integrated Development Environment).
  - JRE: 1.8.0_141
  - JVM: Open JDK 64-Bit Server
  - Python interpreter: python 3.7/Anaconda3/python.exe
- Elasticsearch: used as a database for the analysis results
- Xposed framework: used to get access privilege of mobile phone with the help of supersu
- Droidmon with hooks.json configuration file which used to read a log file

5.3. Implementation of Hybrid Analysis

The prototype is implemented in Java using Oracle JDK jdk1.8.0_141 and Jetbrains Intellij IDE on windows environment. Both the static and dynamic implementation is integrated into the same Java project. The prototype is called *AndroML_Hybrid _Blockchain* and is created as a Gradle project. The implementation mostly follows Test-driven development (TDD). The predefined package structure src is used for productivity, and tests utilized for test code. Dependencies of the applications are defined within the *build. gradle* file. *AndroML_Hybrid _Blockchain* uses the following package structure for the production code:

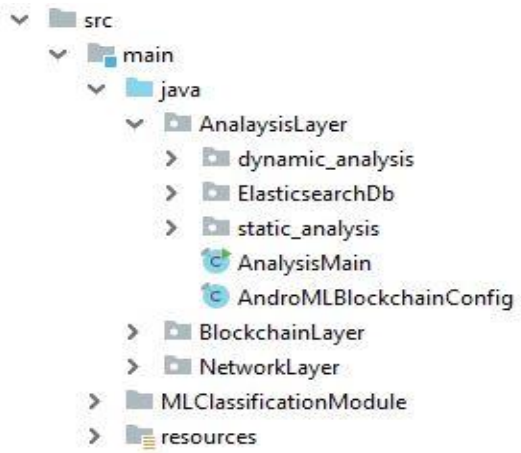


Figure 5. 1: Implementation package overview

The class *AnalysisMain* contains the main method of the application. This class initially creates a configuration instance from class *AndroMLBlockchainConfig* and a database instance from class *ElasticAdapter*. These are injected into an instance of *StaticAnalysis* and *DynamicAnalysis*. *AndroMLBlockchainConfig* class uses Apache Commons Configurations to read in a configuration file and provides specified settings through appropriate getter methods.

5.3.1. Static Analysis Module

The static analysis is located within the module *static_analysis*, which included under the package *AnalysisLayer*. The main entry point is the class *StaticAnalysis*. It contains dependencies on the external module *ElasticsearchDb*. Within the package *static_analysis*, the class *FileHasher* is utilized, and the inner packages analyzer and reports are accessed. They contain logic for the analysis processes and the collected analysis results. *Figure A. 4* depicts the hash generator code.

The interface *StaticAnalyzer* specifies general functionality for analyzers. Therefore the method *analyzes ()* and *getReport ()* is defined. Every analyzer takes a file and analyzes it. The results are stored in the appropriate report. *StaticAnalyzer* is implemented by three analyzers: *ApkMetaAnalyzer*, *AssetAnalyzer*, and *ByteCodeAnalyzer*.

To execute static analysis, the StaticAnalysis instance checks the global configuration by invoking *checkForStartingAnalysis().* If activated, it calls *initializeStaticAnalyzers()* to add the three described analyzers to the list *staticAnalyzers*. Afterwards *startStaticAnalysis()* is called. This procedure recursively iterates through a given directory and its subdirectories. If an Apk is

found, the method *processApkFile()* is invoked. Primary objects components and passed as a parameter. This method generates a SHA256 hash of the APK file using *FileHasher*. It checks if a report already exists. If not, the method *startAnalyzers()* with the Apk file and *storeReports()* invoked. The method *startAnalyzers()* works on the StaticAnalyzer interface and calls *analyze()* and *getReport()* in order to store the resulting report. Finally, *storeReports()* collects reports of the *analyzers*. The method *buildFinalReport()* merges the reports into a final report, which stored into the database with the help of the *ElasticsearchDb* package. *Figure A. 5* shows the static analysis process sample code.

5.3.2. Dynamic Analysis Module

The main entry point for the dynamic analysis is the class *DynamicAnalysis*. The class *ConcurrentAnalysis* implements the Callable interface, and an instance performs the dynamic analysis within a separate thread. The module has a dependency on the *ElasticsearchDb* package. The main package *dynamic_analysis* contains four modules: adb, runner, clicks, and analyzer.

The dynamic analysis started by invoking the method *checkForStartingAnalysis()*. Afterward *initializeDynamicAnalysis*() is called. This method receives the database ids and invokes the procedure *prepareApps().* It receives stored static analysis information of an app. Both are realized by using the *ElasticAdapter* within the *ElasticsearchDb* module. The target index for storing the dynamic analysis results is determined by using the configuration file. The index is stored in the class attribute targetIndex within *ConcurrentAnalysis*. *Figure A. 6* shows the method used to prepare apps for dynamic analysis.

An app is represented by an instance of the *App* using the collected data. They are appended to a *List<App>* and returned to *checkForStartingAnalysis().* The target index is retrieved by invoking the *receiveTargetIndex().* If necessary, the appropriate database is created by calling *createDynamicDatabase().* Afterward, the method *startDynamicAnalysis()* is called. It launches the analysis. Therefore a new *ExecutorService* with a fixed thread pool is created. Within this study, two devices are used. The collected apps in the *List<App>* are added to a *ConcurrentLinkedQueue* named *appQueue. Figure A. 7* depicts the dynamic analysis starter. For every device, an instance of *ConcurrentAnalysis* is created. As parameters, the appQueue,

device identifier, and *targetIndex* are used. Finally, all worker threads are started by invoking *invokeAll()* on the *ExecutorService*.

5.4. Elasticsearch Module

In order to interact with Elasticsearch, the *ElasticsearchDb* package used. The main entry point for the analysis logic is the *ElasticAdapter* class. It implements the interface Database, and this realizes the Dependency Inversion Principle (DIP), which simplifies a possible exchange of the database. The module provides communication with the database by directly invoking endpoints.

The *ElasticAdapter* uses the *AndroMLBlockchainConfig* in order to build the database endpoint of the database. The endpoint is stored within the class attribute *endPoint*. All required functionality is provided by the Database interface and implemented within the class ElasticAdapter. In order to communicate with the Elasticsearch, HTTP and REST are used. Sending and receiving HTTP requests utilized by the *Unirest* library. It provides convenient access to HTTP methods and has excellent documentation. The class *ElasticAdapter* provides the creation and deletion of indexes and documents. It is also able to check if an index or document exists. Moreover, *ElasticAdapter* can receive a specific range of ids from a given index. It is also possible to create, update, read, and remove documents within specific indices. *Figure A. 8* depicts the code used to retrieve data from Elasticsearch.

5.5. Machine Learning Module

This subsection describes the implementation of the machine learning prototype. The general dependencies for the prototype are *argparse, logging,* and *pytest*. The package *argparse* enables the parsing of arguments when starting the Python application. The *logging* package enables the logging feature. The package *pytest* is used as a testing framework for the application. *Figure 5. 2* depicts the Overview of the package structure for the machine learning module

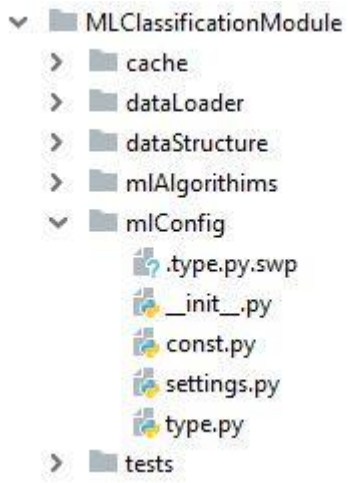


Figure 5. 2: Overview of the package structure for the machine learning module

The *dataLoader* module contains classes responsible for loading data. Package *mlAlgorithims* contains functionality for machine learning. Module *dataStructure* holds data structures for representing apps and their analysis data. The package *mlConfig* contains generic data structures necessary for implementation. The package holds four classes in total. The class *Const* contains important constants like the group of used index names for Elasticsearch and various strings in order to access the reports of the documents. The class *Settings* holds settings like the Elasticsearch configuration, including the endpoint IP and several default configurations, which can be changed by using arguments. As arguments, the number of processed apps or the used feature class like static, dynamic, or hybrid can be used. Also, the amount of malicious and benign apps can be set by using an argument. The classes *FeatureType* and *AppType* inherit the *Enum* class. The first one provides specific feature configurations, and the second one is used for the determination of the class label.

5.5.1. Classification Algorithms Modules

The classification is performed in the *mlAlgorithms* package and requests required data from the *dataLoader* package. The data package uses the *dataStructure* package to structure the apps and their analysis results. The class *Classifier* is initialized by calling *__init__()*. This method creates an instance of *Feature_Enginnering*. By calling the class, the X and y matrices for the classification process are returned. Variable X contains the data points, including the feature vectors, and y contains the appropriate label assignments. Afterward, *print_info()* is called to output information about the matrices. Calling the *Classifier* instance invokes the method

__call__. This method executes a grid search with a train/test data split or nested cross-validation using a grid search.

For the first approach the method *_test_train _grid_search()* is called. It splits the data using the sci-kit-learn method *train_test_split()* from the package *sklearn.model_selection*. The split size is divided into 70% train and 30% test data. The stratify option is used to retain the class distribution within the sets. A random seed value is set to be able to verify the results after multiple calculations. Afterward, the *RandomForestClassifier()* estimator and a parameter grid are created. They are used as arguments for *GridSearchCV()* object, also available in the model selection package of *sci-kit-learn*. Next, the model is fitted by invoking *fit()* and using the training data *X_train* and *y_train*. The best estimator is chosen and applied to the test set by calling *predict().* The results are plotted using a confusion matrix. The *ConfusionMatrix* object is provided by the *SciPy* package.

For nested cross-validation the method *_nested_cross_validation()* is invoked. Instead of manually split the data and call *fit()* and *predict()* with train and test sets, the estimator *GridSearchCV* is used together with X and y in the method *cross_val_score()* of the package *sklearn.model_selection*. It performs the cross-validation with a grid search and returns a score array, which is outputted together with the mean of all scores.

5.5.2. Feature Engineering

The *Feature_Enginnering* class is responsible for providing the X and y matrices. Therefore the method *_get_X_y()* is utilized. It uses a *FileCacher* instance, which is stored in the variable fc. The *Feature_Enginnering* checks if the matrices are already available by calling *fc.are_matrices_available().* If yes, *fc.load_matrices()* is invoked, and X and y are returned. If no, the *Feature_Enginnering* instance generates them. This is done the following way.

The first indices for benign and malware apps are received by invoking *load_benign_ids()* and *load_malware_ids()* of the *FileCacher*. The fc looks for cached data first and otherwise requests the data using the *DbAdapter* and *get_ids_from_db().* Afterward, *_get_features()* is called, which returns a list of numerical and categorical features together with the label matrix y. In order to receive these features, the method contains the invocation of *_iterate_apps()*. It processes the apps for the good and malware category.

The method *_iterate_apps()* iterates over all ids of an app category and build instances of the class *App* by invoking *fc.load_app(id, type)*. The method either returns the app by loading the cached app or using *AppBuilder* to build a fresh instance of *App*. To extract information *_get_numerical_features_from_app()* and *_get_categorical_features_from_app()* methods are invoked. They use sub-methods to separate static and dynamic features. A list of the features for every method is returned. Numerical features are returned as a list, and categorical features are as a dictionary per app.

5.5.3. The dataStructure Package

This package contains the classes *AppBuilder* and *App*. It is used if no cached app is available. An *AppBuilder* instance is created in the *FileCacher* and invoked by calling *build_app(id, type).* The code for this function is listed in *Figure B. 1*.

The method *_get_static_analysis_results()* requests the database using *get_doc_from_db()* and receives the static report document. An empty dictionary *static_content* is created to store the static information as key-values. The dictionary is passed as a reference to *_get_static_content()* and is filled with the analysis results. The method calls four methods to extract data from the APK-meta, asset, bytecode, and Virustotal report. The method *_get_dynamic_analysis_results()* invokes *_get_dynamic_analysis_as_json(),* which retrieves the document for the appropriate dynamic index. The result is stored and returned in JSON format. Afterwards *_classify_api_calls()* is called. This method takes the JSON and transforms it into a dictionary, which classifies the API calls.

Finally, the method build*_app()* builds an instance of *App*. The Virustotal report is used to determine the app type by calling *_determine_app_type().* The method decides if the app is classified as malicious or benign. Afterward, the app is created and returned. The else case is used if it is not possible to determine the app type due to problems with the Virustotal report. In this case, the app type is set to T*ype.UNKNOWN*.

5.6. Blockchain Module

The sections deal with the blockchain, which is executed after the classification. Block validation process is described in *Figure 4. 7*. All of the block's components are validated,

including the proof-of-work and state validation. Every full node in the network does this before they add the new block to their blockchain.

An instance of a *block* is created and called in the *FXMLController* class; this invokes the method to *createBlock()* from the blockchain instance. The classification result passed to the method as a parameter, and a block returned.

```
private void onAddBlock() {
    Optional<Transaction> transaction = FXUtil.showTransactionDialog(
            wallets.getItems());
    if (transaction.isPresent()) {
        String blockData = transaction.get().toString();
        Block block;
        if (blockChain.getBlocks().size() == 0) {
            block = blockChain.createGenesisBlock(blockData);
        } else {
            block = blockChain.createBlock(blockData);
        }
        Stage stage = FXUtil.showBlockGeneration(block);
        new Thread(() -> {
            block.mineBlock(blockChain.getDifficulty());
            Platform.runLater(() -> {
                stage.close();
                blockChain.addBlock(block);
            });
        }).start();

    }
}
```

Figure 5. 3: Block adder method

In this study, the SHA256 cryptographic algorithm is used to generate a digital signature, *java.security.MessageDigest* package.

Moreover, in order to implement the SHA256 algorithm. Then the *calcHash ()* method of *Block* instance invoked, in *isChainValid ()* method of *Blockchain class*, to calculate the hash. The hash must be calculated from all attributes of the block we do not need to be tampered with. So for the block, the previous Hash, data, Nonce, and timestamp are included. Now we need a technique to check the integrity of the blockchain. A method *isChainValid()* in the *Blockchain* class is used to validate the block, which iterates through all blocks in the blockchain and compare the hashes. This method used to check the hash variable is equivalent to the calculated hash, and the previous block's hash is equivalent to the *previousHash* variable.

5.7. Communication Module

In this module, the communication between nodes and how a block is broadcasted addressed. Initially, a node creates a network then waits for other nodes until they join the peer to peer network while the nodes join, they will notify each other. After the connection is established, a node synchronizes blocks of data from the neighbor node. *Figure 5. 4* shows the source code used to broadcast a block for all nodes that are active in the peer to peer network.

```
public void Announce(String type, String sender, String content) {
    NetworkLayer.socket.BlockData Data = new NetworkLayer.socket.BlockData(type,
            sender, content,  recipient: "All");
    for (int i = 0; i < clientCount; i++) {
        clients[i].send(Data);
    }
}
```

Figure 5. 4: Sample code used to broadcast a Block

CHAPTER SIX

6. Evaluation, Result, and Discussion

This chapter presents an overview of this study, discusses the evaluation and results of the research. First, it provides the result of this research in the form of a discussion. Then, the research questions raised in section 1.3 are discussed based on the result of the study. Finally, it reflects the evaluation process, which emerged during this study.

6.1. Results of the study

6.1.1. Feature Selection

In this study, features selected by using the feature importance approach, which is a bagged decision tree, like the random forest or Extra tress, can be used to determine the features' importance approximately. The grid search is executed using the training data. Parameters are finely adjusted, and 2240 apps are processed in total. The results for the optimal parameters are depicted in *Table 6: 1*.

Table 6: 1: Used feature and optimized parameters for training the model

Algorithms	*Analysis*	*No. of Features*	*n_estimators*	*max_features*
Random Forest	Hybrid Interactive	46625	[140, 143, 145,160]	['sqrt', 'log2']
	Hybrid Component	49554	[140, 143, 145,160]	['sqrt', 'log2']
Extreme Trees	Hybrid Interactive	46625	[140, 143, 145,160]	['sqrt', 'log2']
	Hybrid Component	49554	[140, 143, 145,160]	['sqrt', 'log2']

To evaluate and discuss the results of the developed prototype. For results purposes, four different categories of features and apps parameters are collected during the execution time. The results are:

Table 6: 2: Selecting the number of features process

Execution Time	*No of Apk file*		*No. of Features*		*Accuracy%*	
	Malware	*Benign*	*Total*	*Selected*	*RF*	*ET*
1	1356	1570	31568	12	88.8	89.58
2	1356	1570	31568	10	88.8	89.58
3	1000	1080	27855	12	90.92	91.98
4	750	1502	49554	10	91.92	92.56

Based on *Table 6: 2*, the number of features during feature engineering is selected. The top ten features of the component approach are shown in *Figure 6. 1*. The selected features depicted with their importance score, which is calculated by the *feature_importances_* method of *forest* class. These values have the most impact on the training models; thus, it makes up the criteria for the evaluation process.

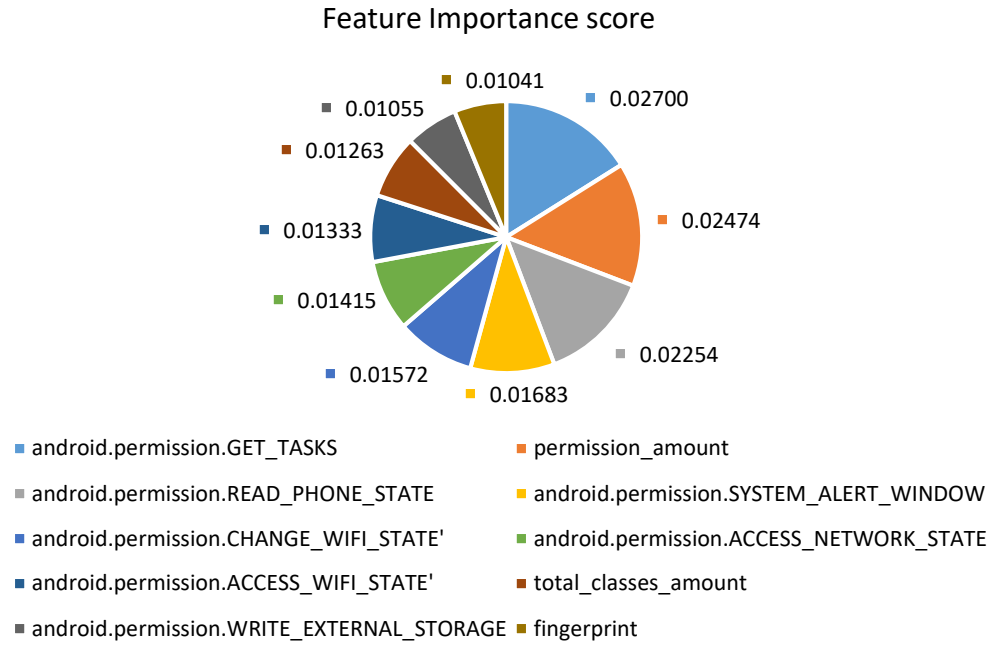


Figure 6. 1: Selected features and importance score using the component approach

Figure 6. 1 depicts the repeatedly occurring features in the analyzed Android apps. Only ten features that are highly frequent is selected. A tenfold cross-validation strategy is adapted for the experiments. We trained the model using nine folds and tested on the remaining fold. The experiment repeated ten times and reported average accuracy. This ensures a broader range of samples for testing the classifier.

When we performed feature selection before modeling, we get the following benefits:

- Reduces Overfitting: less opportunity to make decisions based on noise.
- Improves Accuracy: improves modeling accuracy
- Reduces Training Time: algorithms train faster when the data is fewer

6.1.2. Performance of Classification Models

Random forest and Extra Trees machine learning algorithms used for classification. These machine learning algorithms have different detection performance. The algorithm with high

classification accuracy is selected to provide a better Android malware detection system. The true positive (TP) value, false positive (FP) value, accuracy, precision, and recall have been calculated. The parameters from *Table 6: 1* are used to predict the labels for the apps in the test set X_test and compare the results with the true labels y_test. The results are presented as confusion matrices shown in the depiction of *Figure 6. 2*. Matrix a), and b) represents a component approach with random forest and Extra tree, c) and d) represents a matrix of interactive approach with an Extra tree classifier. The number of apps per category is stated in the cells.

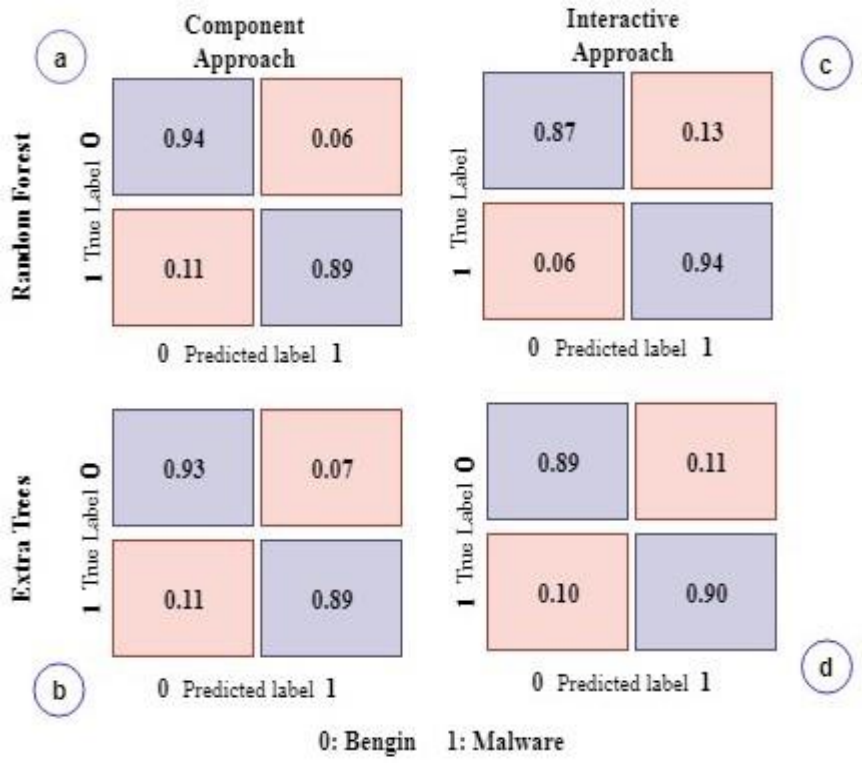


Figure 6. 2: Normalized confusion matrix due to train –test split

These confusion matrices are used to calculate the final metrics for the results of both approaches. *Table 6: 3* presents the results obtained with the most efficient independent variable settings. Using these settings, the Random forest achieved accuracy rates higher than 92%. The proposed classifier adds value compared with existing applications, by detecting the malware applications that exist not only on Google play store but also that exists on the third-party Android application markets.

Table 6: 3: Results of the train-test split method

Classifier	Approach	*Recall*	*Precision*	*F1 Score*	*AUC*	*Accuracy*
Random Forest	Component	0.885	0.851	0.868	0.962	0.922
	Interactive	0.935	0.805	0.865	0.897	0.895
Extra Trees	Component	0.893	0.860	0.876	0.958	0.931
	Interactive	0.903	0.823	0.861	0.905	0.896

In particular, the best classifier, in terms of accuracy, was Extra Trees with an accuracy rate of 93%. Regarding the recall results, again, Extra Trees trained with 143 trees was the best classifier with a recall of 89%. In terms of AUC, Random Forest was the best classifier with an AUC of 96.2%.

The tradeoff between recall and precision for different thresholds illustrated using a precision-recall curve. High scores for both metrics show that the proposed classifiers are returning more accurate results. *Figure 6. 3(a, b)* shows the tradeoff between precision and recall of Random Forest and Extra Trees, respectively.

a) Random Forest b) Extra Trees

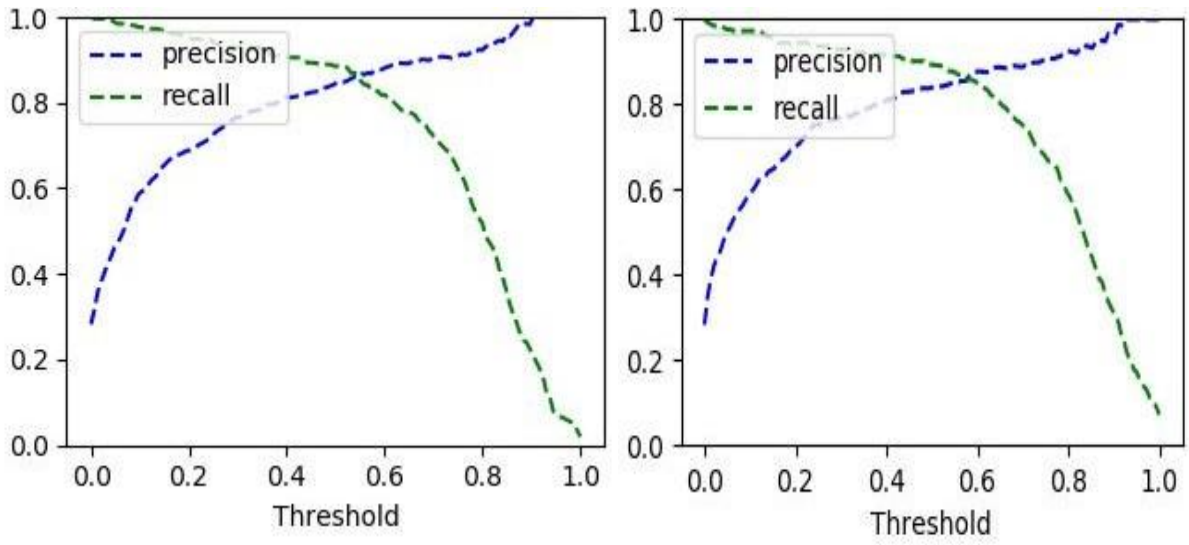


Figure 6. 3: Precision-recall curve of the models for the component approach

6.2. Evaluation

6.2.1. Evaluation of the Proposed Classifiers

To evaluate the performance of machine learning classifiers, k-fold cross-validation is usually used. The right metric for this problem is the F1 score since it combines precision and recall. Therefore, for each classifier that has been used, we apply k-fold cross-validation with k =10, i.e., the dataset is partitioned ten times into ten different sets. This way, each time we use 90% of the data for training and 10% for testing.

An improved approach is the usage of the nested cross-validation. It performs parameter optimization on different training, validation, and test folds. This will result in improved accuracy compared to the previous evaluation approach. It is caused by the utilization of cross-validation instead of a train/test split. The results of nested cross-validation are presented as confusion matrices shown in the depiction of *Figure 6. 4*. Matrix a), and b) represents a component approach with random forest and Extra tree, c) and d) represents a matrix of interactive approach with an Extra tree classifier. The number of apps per category is stated in the cells.

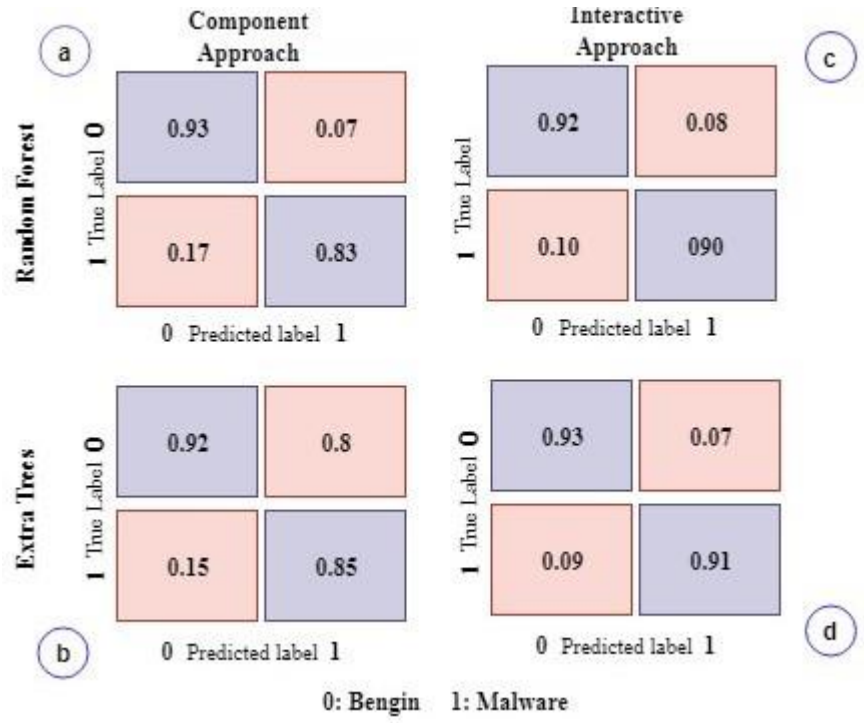


Figure 6. 4: Normalized confusion matrix due to Nested cross-validation

The results are basic measurements for the nested cross-validation with parameter optimization. The method *cross_val_score* of scikit-learn is used in combination with the Meta estimator GridSearchCV. The variables X and y are handed over as parameters as well. The number of folds is set to 10, while the parameter n_estimators is set to a list of [100, 120, 140, 160] and max_features to the list [' sqrt,''log2']. The increased computational effort of this evaluation approach results in a reduced number of folds and a reduced range of values for n_estimators. The results of the nested cross-validation are shown in *Table 6: 4.*

Table 6: 4: Results of Nested Cross-Validation

Classifier	Analysis Approach	*Score Mean*	*Error*
Random Forest	Component	0.908	0.092
	Interactive	0.903	0.097
Extra Trees	Component	0.912	0.088
	Interactive	0.910	0.090

6.2.2. Comparison and Validation of Models

This section determines which model offers better results. *Figure 6. 5* illustrates the AUC results of the proposed models. According to AUC, the component analysis approach of the random forest model outperforms.

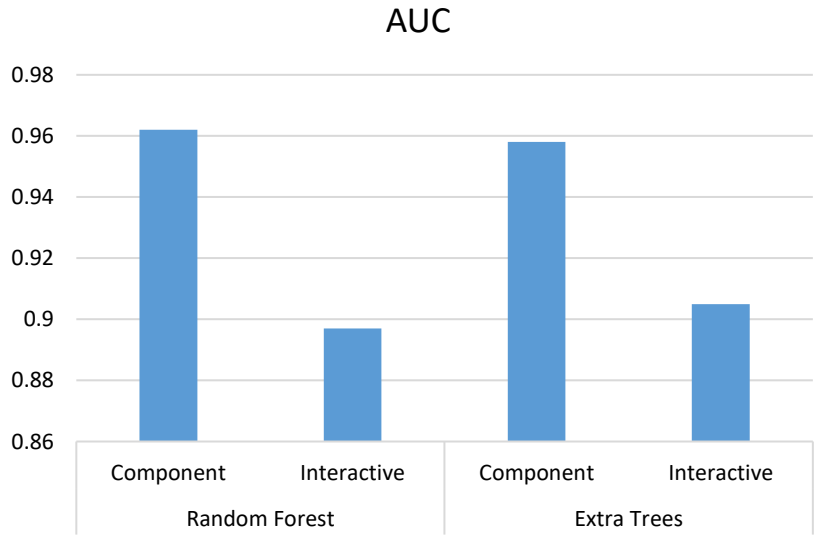
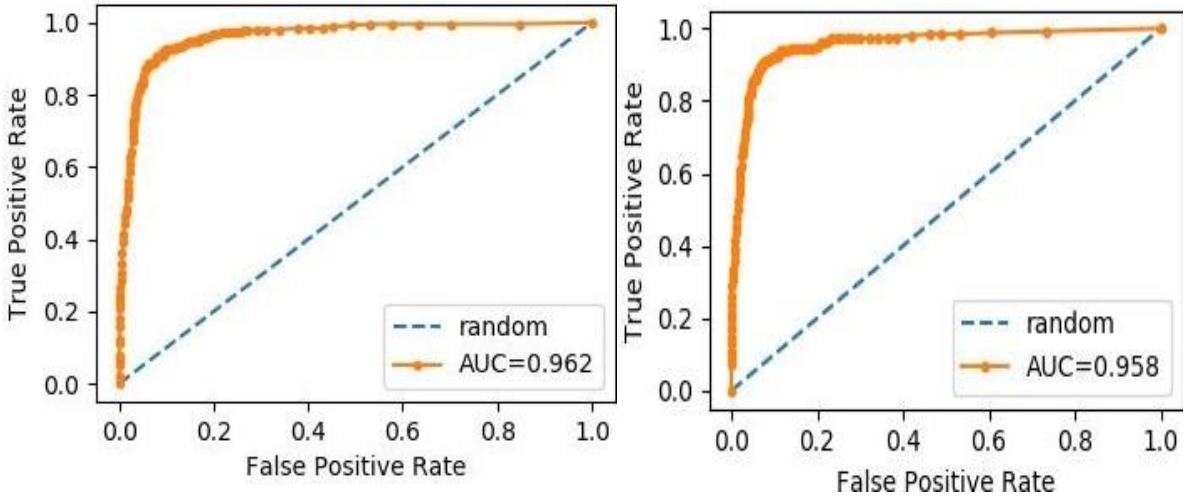


Figure 6. 5: Comparison of proposed models

The models are validated based on the validation metrics used to determine which model offers better results. *Figure 6. 6* (a, b and c, d) illustrates respectively ROC and Validation curve results on these models for the best classifier.

a) ROC of Random Forest b) ROC of Extra Tree



c) Validation curve of Random Forest d) Validation curve of Extra Tree

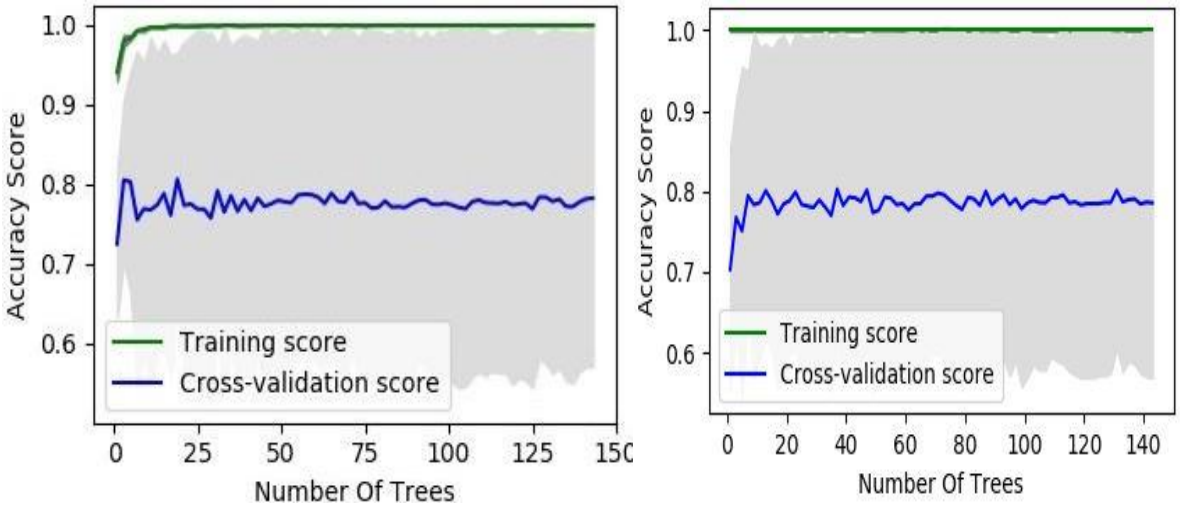


Figure 6. 6: A ROC and curve validation of proposed classifiers

*Figure 6. 6 (a and b)*shows the component approach ROC of used classifiers in this study; used to compare results obtained from classifiers with Random Forest and Extra Trees to verify the selection of correct machine learning algorithms. From the test results, we can see that the component approach analysis with the Random Forest and Extra Trees Algorithm with 143 trees, has given an AUC value closer to 1.0. The ROC of the interactive approach is depicted in

Figure C. 2. Figure 6. 6(c and b) depicts the training and cross-validation score of Random forest and Extra Trees, respectively. The training score of Extra Trees is precisely 1.0.

6.2.3. Analysis Module Evaluation

This section describes how the analysis environment and the devices were prepared for the dynamic analysis evaluation. The first part starts by describing the preparation of the devices, followed by the evaluation of dynamic analysis.

The interaction with the Android devices and the installation of the Xposed framework needed some preparation. On the device, a custom recovery TWRP should be installed in order to be able to root the devices and enable backup and restore capabilities. In order to install TWRP, the devices were unlocked by using *fastboot oem unlock* within the fast boot mode. Subsequently, the TWRP image was flashed onto the device by using *fastboot* flash recovery *twrp-<version>-hammerhead.img*. After booting into recovery, the device was rooted. The app *SuperSU* was used to control the root access.

In the next step, the Xposed Framework was installed via adb install. Afterward, the Droidmon module was installed and was activated in the Xposed app. It was required to enable USB debugging to communicate through adb. Therefore activation of the developer settings was required. The *SuperSU* app was configured so only Xposed, and adb was able to request root permissions. The device was connected to the *analyzer* and was tested using the analysis tests.

Table 6: 5: Comparison of existing analysis frameworks

Analysis Tools	*Drawback*	*Environment*	*Report Format*
Apktool	Used only for compile and decompile	Windows	Java code
MobSF	The report not provided in an applicable format	Sandbox	PDF
CuckooDroid	Do not provide an automatic click runner as a feature	Sandbox	Web
Proposed		Windows and Mobile	JSON

Table 6: **5** shows the proposed analysis module compared with the existing tools. As we compare, the proposed framework was better than the existing tools because, beyond decreasing dependencies, it implements hybrid analysis while the compared tools do not address it.

To evaluate the proposed model, we identify a tool that performs the most fundamental transformation of the Android executables and compares with the framework; we carry out parametric two-sample t-tests on the log-transformed data of Monkey test result. *Table C. 1* and *Figure 6. 7* shows the number of apps failing the monkey test for twenty-six categories tested.

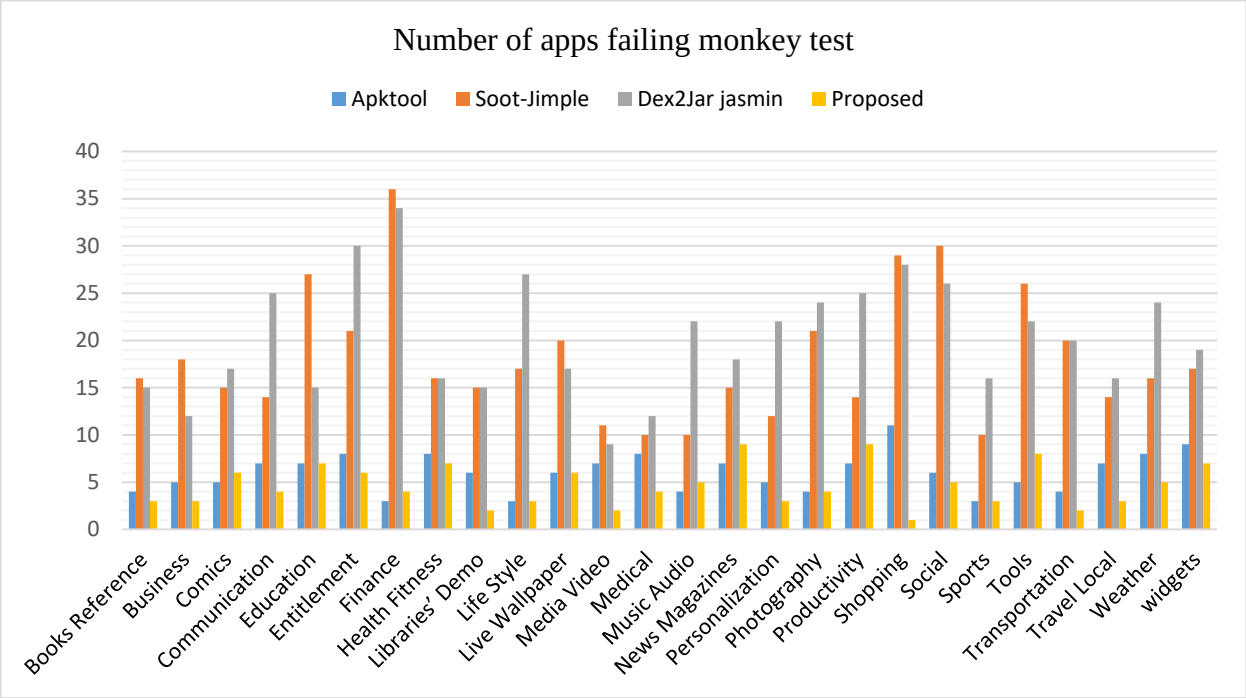


Figure 6. 7: Number of apps failing Monkey test

We perform a two-sample t-test to identify whether the apps, which are assembled from Smali, Jasmin, and Jimple intermediate languages, have introduced any notable differences in their behaviors comparing to the original ones.

Table 6: 6: Result of Two-sample t-test

Tools	*Total Categories*	*T-value*	*P-value*	*Significance Leve1*
Apktool	26	-1.05	0.093	0.05
Soot-Jimple	26	-1.06	0.002	0.05
Dex2Jar	26	-3.99	0.001	0.05
Proposed	26	0.75	0.45	0.05

In *Table 6: 6*, we show the results of two-sample t-tests for Apktool, Smali, Dex2jar–Jasmin, and Soot–Jimple and proposed. Comparing the obtained P-values for each tool with the significance level, we can see that the obtained P-value (P = 0.45) for proposed is higher than

the required level of significance ($\alpha = 0.05$). Therefore, we conclude that the proposed framework performs the most accurate transformation of the apps.

6.2.4. A Blockchain Use Case Evaluation

To evaluate and identify the right use case for blockchain in this study, an evaluation is done based on *Chili Sauce Blockchain Evaluation Framework* [50]. *Figure 6. 8* illustrates the use cases and steps used to evaluate the proposed module of this study. As the evaluation framework shows clearly, using blockchain technology in android malware analysis and detection is an optimal solution. The applicability of use cases is analyzed and assessed against Blockchain core characterizes.

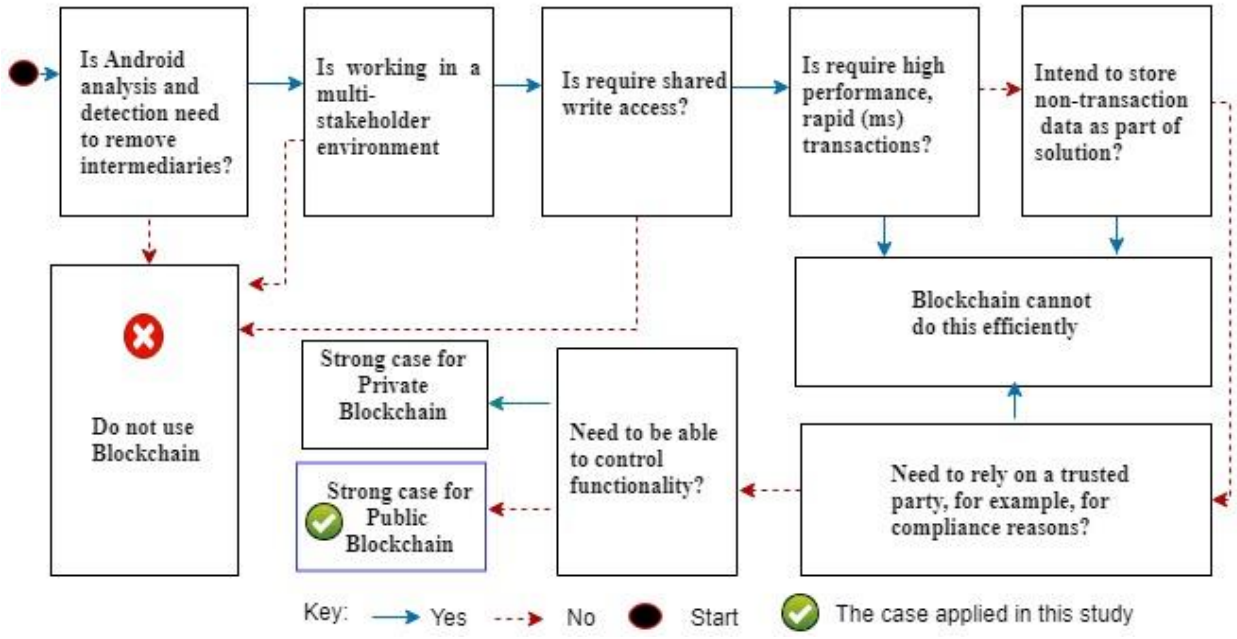


Figure 6. 8: Used use cases for evaluation of blockchain

Based on the introduced Blockchain characterization and early learning from the Android malware engines, evaluation use cases can be categorized in the following cases:

Case 1: If the hash code of the given android application exists in Elasticsearch:

The process of extracting hash code and comparing the value with malicious lists found on Elastic search is a type of static-based Android malware detection. In this study, the system can generate the SHA-256 value of the given application to detect the incoming application fast. In this case, the user can identify the malicious application within a short period, which means the

hashing process not be executed to extract the features and update the document stored in Elasticsearch with a unique hash code.

Case 2: If the hash code of the given android application does not exist in Elasticsearch:

In this case, the system has been checked, the generated hash code in Elasticsearch, but if the hash code document is not found in the Elasticsearch. Even though the hash code of the application is not found in the Elasticsearch, it is difficult to decide that the application is a benign android application or not. In such a case, the android application is analyzed, and the hybrid analysis result stored in Elasticsearch.

Case 3: Approaches based on hybrid analysis

Research works on dynamic analyses [38], [39], and [41] are based on system calls, API calls, CPU consumption, packets sent through network, data, and control flow. The proposed has exciting results compared to these works, although it is limited only to static analysis of the Manifest file. They indicate that their approach is not efficient for predicting malware compared to the TPR of the proposed framework providing 93% in predicting unknown malware. This approach is limited to some applications for testing. The proposed framework gives better results with more massive data.

Case 4: Trying to modify malicious code in the blockchain

In this case, when any node tries to update the malicious data in the blockchain, the blockchain not allows modifying the data because every block of data is validated by every node before broadcasting to the neighbor's node and after receiving from any node this ensures the security of the proposed system.

6.3. Discussion

In order to evaluate the classification, two different procedures for the evaluation applied: the component and interactive analysis. The first one compares the analysis prediction using a train/test split with parameter optimization. This approach provides insights into the mechanism and enables subsequent use of the model. The second procedure is a nested, cross-validation approach. This is executed for both hybrid approaches, both dynamic analyses, and the static analysis. For the prediction, the *RandomForestClassifier* and *ExtremTreesClassifier* methods are used.

The first evaluation approach returns confusion matrices and the second one accuracy values. The train/test split is used to shuffle the data and to split them into 70% training data and 30%

test data. The training data is divided into a training set *X_train* and *y_train* for labels. The test data is composed of a test set *X_test* and *y_test*. For the execution, the scikit-learn method *test_train_split* from sklearn.model_selection is used. The ratio of malware and benign apps is preserved by setting a *stratify=self.y* option.

The training set prevents leakage of knowledge from the test set into the model, which would result in an overfitted model. For the parameter optimization, cross-validation is used. This functionality is provided by the scikit-learn meta estimator *GridSearchCV*. Grid search uses 3-fold stratified cross-validation by default.

Two parameters are considered to optimize the performance of the random forest and Extra tree. The first parameter is called *n_estimators* and defines the number of used trees in the forest. The second parameter is named *max_features* and defines a specific amount of features when looking for the best split. For *n_estimators,* a range of [20, 160] is used. Values lower than 20 results in poor prediction performance. Higher values require considerably more computational effort without increasing performance. For *max_features,* the two most commonly applied parameters *sqrt* and *log2* are used. The GridSearchCV needs to operate on the random forest estimator, the parameter dictionary, and the number of folds. For this approach, ten folds are used. This is a balanced trade-off between evaluation precision and performance.

The stated questions in this study are addressed based upon the result, and evaluation metrics have been done. *How to scheme a better approach for detecting Android malware by integrating machine learning and blockchain?* Section 6.2 illustrates the answers to this and other research questions of the study. And *what are the limitations of the existing Android malware detection solutions?* To answer this question on existing solutions literature review applied and their gap stated in section 2.10.this makes sure by comparing the proposed model with the existing solution of android malware detection, section 6.2 illustrates the answers to this and other research questions of the study.

The second research question is *How to improve the performance of the Android malware detection system on a peer-to-peer node by the decentralized solution?* In the proposed solution, users share their detection results to whole users who are active in the peer to peer network. Once a given android application is analyzed by a given node, no need to analyze that application by other nodes because they have already the analysis result, this scenario is addressed by

integrating the blockchain technology with designed analysis algorithms and evaluated in section 6.2.4.

Finally, the main research question for this study is how *the malware detection of recent Android malware is precise by comparing two different analysis approaches?* In order to reach to answer the research question, two dynamic analysis approaches implemented: *component and Interactive approach.* A sub-question was formulated to give the first answer to this question: *What is the code used to represent the hybrid analysis process*? Intermediate code is used to represent the analysis process, and JSON is used to represent the result of the hybrid analysis. *What are the main techniques used in the hybrid analysis of Android apps?* They are required to determine the necessity of automated click tools for a machine learning approach. The click tool benefits from using real devices due to more fluent interactions. The first one does not use any simulated click interactions. Instead, it relies on the defined activities and services of an app and is going to start them iteratively. Therefore the determined component names of the static analysis used. Section 4.3.1.4 and 6.2.1 answers this question in detail.

CHAPTER SEVEN

7. Conclusion and Future Work

This chapter contains recommendations for additional and useful tasks in the future. It covers important thoughts and ideas, which were not included in this study, due to time and scope restrictions.

7.1. Conclusion

A framework to detect and classify android malware through machine learning and Blockchain technology is constructed. The results of the classification show a slightly better performance of the hybrid analysis when comparing the accuracy and f_1 measurement. However, if nested cross-validation is used, the hybrid approach performs slightly better. Since the nested approach is more precise, its results should be given more weight. Therefore the final results for both approaches can be considered comparable good. In the scenario, without further modification, the hybrid analysis should be preferred since it has a better performance and is less error-prone compared to the individual analysis approach. The interactive approach might be more precise if the analysis threshold would be increased to more than ten minutes. Increasing the feature number of the dynamic analysis could change the impact and accuracy of the dynamic approach. Moreover, a higher time threshold for the interactive approach could improve the code coverage. This, in turn, could enhance the prediction accuracy using this approach.

In order to answer the central research questions of this study, one can denote that the malware detection for detecting recent Android malware lies approximately 93% detection accuracy. Both analysis approaches do not differ noteworthy for the inspected scenario. This result seems to keep up and sometimes outperform other machine learning approaches described in section 2.10.1. However, various approaches achieve outstanding precision close to 100%. This might lie in the usage of outdated malware, which could be more comfortable to identify.

Perhaps a large number of unwanted programs and generic malware in the recent malware collection could be harder to identify than the utilized collections of the related work. A reason for that is improved APIs and a more hardened Linux kernel. The malware focuses more on tricking the user. Interesting is the increased usage of the target SDK version prior to the runtime permission feature. Due to backward compatibility, these apps are still working using the prior

installing permission dialog. In order to prevent the installing of dynamically loaded untrusted applications. This, however, damages the more open approach of the ecosystem in comparison to other mobile operating systems. Within this study, malware is labeled as malware if it has at least two detections of Virustotal tools. The other apps are labeled as benign. This should minimize false positives and reduce training outliers.

The contribution of this study is developing a decentralized Android malware detection system by integrating blockchain and machine learning together. Mitigate the drawbacks of the static and dynamic analysis approach, and this study implements a hybrid analysis prototype to generate a more available system that detects malicious applications with minimum computational resources and time. To address this goal, a decentralized system with a blockchain approach developed. Precisely for this study, the static and dynamic analyzer is developed, which reduces the dependence.

7.2. Future Work

This section deals with improvements for static as well as dynamic analyses. For the dynamic analysis, the number of dynamic features could be increased, for example, by utilizing arguments of the API calls. Additionally, logged internet traffic could be inspected and used as features. Meta-information about the network connection like source and target address or used protocols can be transformed into features. Also, information about the payload can be included. For the interactive approach, the click runner can be enhanced, for example, by generating input for specific input fields. Also, new swipe movements can be added.

For the static analysis, an improvement would be the implementation of a multithreaded static analyzer. Besides, the analysis of assets can be enhanced by analyzing the mime type and size of a specific asset. Maybe specific malware can be identified. An example would be dynamic code loading hidden in a concealed picture file. Moreover, an improvement would be adding a native code analysis.

The functionality could be provided via an API or a web application. Also, possible to the deployment of the static analysis and a machine learning model on a mobile device in order to enable malware recognition as an Android application. Moreover, it is recommended to integrate with an analysis tool which is called Androlyzer.

References:

[1] R. Unuchek, “Mobile malware evolution 2017,” *Kaspersky Lab*, 2017. [Online]. Available: ttps://securelist.com/mobile-malware-review-2017/84139/. [Accessed: 03-Aug-2018].

[2] K. Shibija and R. V. Joseph, “A Machine Learning Approach to the Detection and Analysis of Android Malicious Apps,” *2018 Int. Conf. Comput. Commun. Informatics, ICCCI 2018*, pp. 1–4, 2018.

[3] Q. Zhang, X. Jiang, M. Grace, S. Zou, and Y. Zhou, “RiskRanker,” p. 281, 2012.

[4] L. Wu, X. Du, and X. Fu, “Security threats to mobile multimedia applications: Camera-based attacks on mobile phones,” *IEEE Commun. Mag.*, vol. 52, no. 3, pp. 80–87, 2014.

[5] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,” *Proc. - 2017 IEEE 6th Int. Congr. Big Data, BigData Congr. 2017*, pp. 557–564, 2017.

[6] K. R. Fleischmann and T. Srikantaiah, “SWOT analysis of mobile phones in four countries: Comparing India, Ethiopia, Kuwait, and the United States,” *Proc. ASIST Annu. Meet.*, vol. 48, 2011.

[7] S. Hanna, A. P. Felt, E. Chin, M. Finifter, and D. Wagner, “A survey of mobile malware in the wild,” p. 3, 2011.

[8] T. A. Abdurrahman Pekta, MahmutC¸avdar, “Android Malware Classification by Applying Online Machine Learning,” *Comput. Inf. Sci.*, vol. 9, no. 1, pp. 15–15, 2016.

[9] B. Kitchenham and S. Charters, “Procedures for Performing Systematic Literature Reviews in Software Engineering,” *Keele Univ. Durham Univ. UK*, 2007.

[10] Statista, “Leading Android app categories worldwide 2018,” 2018. [Online]. Available: https://www.statista.com/statistics/200855/favourite-smartphone-app-categories-by-share-of-smartphone-users/. [Accessed: 03-Jan-2019].

[11] B. Padhya, P. Desai, and D. Pawade, “Mobile Operating Systems and Application Development Platforms: A Survey,” *IVNC IFES 2006 - Tech. Dig. - l9th Int. Vac. Nanoelectron. Conf. 50th Int. F. Emiss. Symp.*, vol. 6, no. 1, pp. 65–66, 2006.

[12] W. Enck, M. Ongtang, and P. Mcdaniel, “On Lightweight Mobile Phone Application Certification,” pp. 235–245.

[13] A. O. S. Project., “The Android Source Code.” [Online]. Available: https://source.android.com/setup. [Accessed: 14-Mar-2019].

[14] A. O. S. Project., “Content License.” [Online]. Available: https://source.android.com/setup/start/licenses. [Accessed: 14-Mar-2019].

[15] A. O. S. Project, “ART and Dalvik.” [Online]. Available: https://source.android.com/devices/tech/dalvik/. [Accessed: 16-Mar-2019].

[16] A. O. S. Project, “Security.” [Online]. Available: https://source.android.com/security/.

Accessed: 16-Mar-2019].

[17] A. O. S. Project, “System and kernel Security.” [Online]. Available: https://source.android.com/security/overview/kernel-security,. [Accessed: 16-Mar-2019].

[18] A. Developers, “Requesting Permissions at Run Time.” [Online]. Available: https://developer.android.com/training/permissions/requesting.html. [Accessed: 17-Mar-2019].

[19] Google Inc, “Application Fundamentals - Android Developers.” [Online]. Available: https://developer.android.com/guide/components/fundamentals.html,. [Accessed: 17-Mar-2019].

[20] L. Wen and H. Yu, “An Android malware detection system based on machine learning,” *AIP Conf. Proc.*, vol. 1864, 2017.

[21] M. Chandramohan and H. B. K. Tan, “Detection of mobile malware in the wild,” *Computer (Long. Beach. Calif).*, vol. 45, no. 9, pp. 65–71, 2012.

[22] L. Li *et al.*, “Static analysis of android apps: A systematic literature review,” *Inf. Softw. Technol.*, vol. 88, pp. 67–95, 2017.

[23] C. Tumbleson, “Apktool v2.4.0 Released.” [Online]. Available: https://connortumbleson.com/2019/03/03/apktool-v2-4-0-released/. [Accessed: 17-Mar-2019].

[24] smali, “smali/baksmali.” [Online]. Available: https://github.com/JesusFreke/smali. [Accessed: 17-Mar-2019].

[25] MobSF, “Mobile-Security-Framework-MobSF.” [Online]. Available: https://github.com/MobSF/Mobile-Security-Framework-MobSF. [Accessed: 17-Mar-2019].

[26] C. Hero, “general workflow process,” 2018. [Online]. Available: https://www.coursehero.com/file/p6hkghop/Figure-1-General-workflow-process-As-it-can-be-seen-the-process-consists-of-5/. [Accessed: 16-Jun-2019].

[27] S. Guido, *Introduction to Machine Learning with Python*. 2017, European Union.

[28] European Union, “Introduction to Machine learning,” no. July, p. 213, 2015.

[29] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, “The CART decision tree for mining data streams,” *Inf. Sci. (Ny).*, vol. 266, pp. 1–15, 2014.

[30] M. Skurichina and R. P. W. Duin, “Bagging, boosting and the random subspace method for linear classifiers,” *Pattern Anal. Appl.*, vol. 5, no. 2, pp. 121–135, 2012.

[31] B. Singhal, G. Dhameja, and P. S. Panda, *Beginning Blockchain : A Beginner’s Guide to Building Blockchain Solutions*. 2018.

[32] S. B. (IBM), “It Was Only a Matter of Time Digital Identity on Blockchain,” 2018. [Online]. Available: https://www-01.ibm.com/common/ssi/cgi-

bin/ssialias?htmlfid=GIL12346USEN&,. [Accessed: 24-Mar-2018].

[33] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain Challenges and Opportunities : A Survey Shaoan Xie Hong-Ning Dai Huaimin Wang," *Int. J. Web Grid Serv.*, pp. 1–24, 2017.

[34] D. Damopoulos, "The Best of Both Worlds . A Framework for the Synergistic Operation of Host and Cloud Anomaly-based IDS for Smartphones," *Eurosec*, 2014.

[35] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P. Wu, "DroidMat: Android malware detection through manifest and API calls tracing," *Proc. 2012 7th Asia Jt. Conf. Inf. Secur. AsiaJCIS 2012*, pp. 62–69, 2012.

[36] J. S. Pan, C. N. Yang, and C. C. Lin, "Advances in Intelligent Systems and Applications - Volume 2," *Smart Innov. Syst. Technol.*, vol. 21, pp. 779–788, 2013.

[37] H. S. Ham and M. J. Choi, "Analysis of Android malware detection performance using machine learning classifiers," *Int. Conf. ICT Converg.*, pp. 490–495, 2013.

[38] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," *Digit. Investig.*, vol. 13, pp. 1–14, 2015.

[39] S. Liang and X. Du, "Permission-combination-based scheme for Android mobile malware detection," *2014 IEEE Int. Conf. Commun. ICC 2014*, pp. 2301–2306, 2014.

[40] J. Gu, B. Sun, X. Du, and S. Member, "Consortium Blockchain-Based Malware Detection in Mobile Devices," *IEEE Access*, vol. 6, pp. 12118–12128, 2018.

[41] F. Abdurahman, "ADDIS ABABA UNIVERSITY SCHOOL OF GRADUATE STUDIES INSTITUTE OF TECHNOLOGY," 2014.

[42] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Sci. Technol.*, vol. 21, no. 1, pp. 114–123, 2016.

[43] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "PIndroid: A novel Android malware detection system using ensemble learning methods," *Comput. Secur.*, vol. 68, pp. 36–46, 2017.

[44] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," *IEEE Access*, vol. 6, no. c, pp. 4321–4339, 2018.

[45] Ashishb, "Collection of android malware samples." [Online]. Available: https://github.com/ashishb/android-malware. [Accessed: 03-Jan-2019].

[46] Virustotal., "How it works," 2019. [Online]. Available: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works. [Accessed: 22-Jul-2019].

[47] "Drawio." [Online]. Available: https://www.draw.io/. [Accessed: 18-Apr-2019].

[48] J. Sahs and L. Khan, "A machine learning approach to android malware detection," *Proc. - 2014 Eur. Intell. Secur. Informatics Conf. EISIC 2014*, pp. 141–147, 2014.

[49] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, “A review on feature selection in mobile malware detection,” *Digit. Investig.*, vol. 13, pp. 22–37, 2015.

[50] L. Jiang, “A Framework to Evaluate Blockchain Use Cases,” 2018. [Online]. Available: https://hackernoon.com/breaking-blockchain-a-framework-to-evaluate-blockchain-use-cases-9efbc30a3fa7. [Accessed: 09-Nov-2019].

[51] W. Meng, E. W. Tischhauser, Q. Wang, Y. Wang, and J. Han, “When intrusion detection meets blockchain technology: A review,” *IEEE Access*, vol. 6, pp. 10179–10188, 2018.

[52] D. Puthal, N. Malik, S. P. Mohanty, E. Kougianos, and C. Yang, “The Blockchain as a Decentralized Security Framework [Future Directions],” *IEEE Consum. Electron. Mag.*, vol. 7, no. 2, pp. 18–21, 2018.

[53] Y. F. Lu, C. F. Kuo, H. Y. Chen, C. W. Chen, and S. C. Chou, “A SVM-Based Malware Detection Mechanism for Android Devices,” *2018 Int. Conf. Syst. Sci. Eng. ICSSE 2018*, pp. 1–6, 2018.

[54] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, “A novel dynamic android malware detection system with ensemble lFeng, P., Ma, J., Sun, C., Xu, X., & Ma, Y. (2018). A novel dynamic android malware detection system with ensemble learning. IEEE Access, 6, 30996–31011. https://doi.org/10.1109/ACCESS.2018.2,” *IEEE Access*, vol. 6, pp. 30996–31011, 2018.

[55] J. Garcia, M. Hammad, and S. Malek, “Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware,” *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 3, pp. 1–29, 2018.

[56] I. Martín, J. A. Hernández, A. Muñoz, and A. Guzmán, “Android Malware Characterization Using Metadata and Machine Learning Techniques,” *Secur. Commun. Networks*, vol. 2018, 2018.

[57] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, “Improving dynamic analysis of android apps using hybrid test input generation,” *2017 Int. Conf. Cyber Secur. Prot. Digit. Serv. Cyber Secur. 2017*, 2017.

[58] S. Raje, S. Vaderia, N. Wilson, and R. Panigrahi, “Decentralised firewall for malware detection,” *Int. Conf. Adv. Comput. Commun. Control 2017, ICAC3 2017*, vol. 2018-Janua, pp. 1–5, 2018.

[59] N. McLaughlin *et al.*, “Deep Android Malware Detection,” pp. 301–308, 2017.

[60] P. Ravi Kiran Varma, K. P. Raj, and K. V. Subba Raju, “Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms,” *Proc. Int. Conf. IoT Soc. Mobile, Anal. Cloud, I-SMAC 2017*, pp. 294–299, 2017.

[61] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, “Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers,” *Futur. Gener. Comput. Syst.*, vol. 78, pp. 987–994, 2018.

[62] B. Nuray, “A Comparison of Classification Algorithms for Mobile Malware Detection :

Market Metadata As Input Source," no. September, pp. 1–77, 2014.

[63] J. Song, C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang, "An integrated static detection and analysis framework for android," *Pervasive Mob. Comput.*, vol. 32, pp. 15–25, 2016.

[64] U. Kanonov, C. Glezer, Y. Weiss, Y. Elovici, and A. Shabtai, "'Andromaly': a behavioral malware detection framework for android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2011.

[65] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in Android," *J. Parallel Distrib. Comput.*, vol. 103, pp. 22–31, 2017.

[66] I. Muttik, S. Y. Yerima, and S. Sezer, "High accuracy android malware detection using ensemble learning," *IET Inf. Secur.*, vol. 9, no. 6, pp. 313–320, 2015.

[67] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "AppContext : Differentiating Malicious and Benign Mobile App Behaviors Using Context," *IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, pp. 303–313, 2015.

[68] X. Liu and J. Liu, "A two-layered permission-based android malware detection scheme," *Proc. - 2nd IEEE Int. Conf. Mob. Cloud Comput. Serv. Eng. MobileCloud 2014*, pp. 142–148, 2014.

[69] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," no. February, pp. 23–26, 2014.

[70] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," *2013 9th Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2013*, pp. 1666–1671, 2013.

[71] N. Peiravian and X. Zhu, "Machine learning for Android malware detection using permission and API calls," *Proc. - Int. Conf. Tools with Artif. Intell. ICTAI*, pp. 300–305, 2013.

[72] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra, "A multi-criteria-based evaluation of android applications," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7711 LNCS, no. 256980, pp. 67–82, 2012.

Appendixes

Appendix A: Java Source code

```
public abstract class DynamicAnalyzer {
    static final int INITIAL_LAUNCH_WAITING_TIME = 6000;

    Logger LOG;
    AdbAdapter aa;
    ClickRunner clickRunner;

    public abstract void analyze(App app, int reboots);

    public abstract Report getReport();

    void initializeAnalysis(String packageName) {
        LOG.info("Stop app {} and delete all logs.", packageName);
        aa.stopApp(packageName);
        //aa.deleteAllLogs();
        LOG.info("Check for unlocking of device.");
        aa.unlockDevice();
        LOG.info("Start initial launch of app.");
        aa.launchApp(packageName);
        checkForErrorAndPermissionDialog();
        aa.sleep(INITIAL_LAUNCH_WAITING_TIME);
    }


    void fixEmptyApiLog() {
        aa.rebootDevice();
        if (aa.isDeviceReady()) {
            aa.unlockDevice();
        }
    }

    void checkAndReboot(App app, int reboots, String packageName) {
        if (app.getApiCalls().isEmpty() && reboots == 0) {
            LOG.info("API LOG is empty. Rebooting the device...");
            fixEmptyApiLog();
            reboots++;
            LOG.info("Device is rebooted. Try app {} again.", packageName);
            analyze(app, reboots);
        }
    }
```

Figure A. 1: Unlock the mobile device and run dynamic analysis

```java
import AnalaysisLayer.ElasticsearchDb.Database;
import AnalaysisLayer.ElasticsearchDb.ElasticAdapter;
import AnalaysisLayer.dynamic_analysis.DynamicAnalysis;
import AnalaysisLayer.static_analysis.StaticAnalysis;
import org.python.util.PythonInterpreter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

final public class AnalysisMain {
    private static final Logger LOG = LoggerFactory.getLogger(AnalysisMain.class);
    public static void main(String[] args) {
        AndroMLBlockchainConfig config = new AndroMLBlockchainConfig();
        Database db = new ElasticAdapter(config);
        initializeCoreIndex(config, db);
        StaticAnalysis sa = new StaticAnalysis(config, db);
        sa.checkForStartingAnalysis();
        DynamicAnalysis da = new DynamicAnalysis(config, db);
        da.checkForStartingAnalysis();
        //Execute Python code**************************
        PythonInterpreter interp = new PythonInterpreter();
        interp.execfile( filename: "src/main/MLClassificationModule/classify.py");
    }

    public static void initializeCoreIndex(AndroMLBlockchainConfig config, Database db) {
        if (!db.isDatabasePresent(config.getElasticIndex())) {
            db.createDatabase(config.getElasticIndex());
        }
    }
}
```

Figure A. 2: Start a hybrid analysis and integrate python with java program

```java
private void extractIds(List<String> databaseIds, HttpResponse<JsonNode> response) {
    try {
        JSONArray hits = response.getBody().getObject().getJSONObject("hits")
                .getJSONArray( key: "hits");
        Iterator it = hits.iterator();
        while (it.hasNext()) {
            JSONObject entry = (JSONObject) it.next();
            databaseIds.add(entry.getString( key: "_id"));
        }
    } catch (Exception e) {
        System.out.println("Iterating over ids of index failed." + e);
    }
}
```

Figure A. 3: Extract Malware id from Elasticsearch document

```
public synchronized static String hashFile(File file, String algorithm) {
    byte[] byteHash = new byte[0];
    try {
        FileInputStream in = new FileInputStream(file);
        MessageDigest messageDigest = MessageDigest.getInstance(algorithm);
        byteHash = processMessageDigest(in, messageDigest);
    } catch (NoSuchAlgorithmException | IOException e) {
        LOG.error("Calculation of byteHash for file failed.", e);
    }
    return convertByteArrayToString(byteHash);
}
```

Figure A. 4: Sample code for the SHA-256 generation

```
private void processApkFile(File file) {
    List<Report> reports = new ArrayList<>();
    try {
        String apkHash = FileHasher.hashFile(file, FileHasher.SHA256);
        String checkForVirusTotalReport = apkHash;
        JSONObject response = db.receiveDocument(checkForVirusTotalReport, config.getElasticIndex()
        if (hasReport(response)) {
            LOG.info("The report with key {} is already exist.", apkHash);
            startAnalyzers(file, reports);
            updateReports(reports, apkHash);
        } else {
            startAnalyzers(file, reports);
            storeReports(reports, apkHash);
        }
    } catch (Exception e) {
        LOG.error("Could not parse APK: {} ", file.getName());
    }
}
```

Figure A. 5: Sample code for the static analysis

```
public Set<App> prepareApps(List<String> ids) {
    Set<App> apps = new HashSet<>();
    for (String id : ids) {
        if (!id.startsWith(VIRUS_SHARE_PREFIX)) {
            JSONObject result = db.receiveCustomSourcesFromID(config.getElasticIndex(), id, SOURCES);
            System.out.println("prepare Apps:" + config.getPath());
            App app = new App(result, config.getPath());
            apps.add(app);
        }
    }
    return apps;
}
```

Figure A. 6: Method to prepare apps to be analyzed

```
private void startDynamicAnalysis(Set<App> apps, String targetIndex) {
    ExecutorService es = Executors.newFixedThreadPool(DEVICE_AMOUNT);
    List<Callable<Object>> worker = new ArrayList<~>();
    Queue<App> appQueue = new ConcurrentLinkedQueue<>();
    appQueue.addAll(apps);
    for (String deviceID : DEVICES) {
        ConcurrentAnalysis daThread = new ConcurrentAnalysis(appQueue, targetIndex, deviceID);
        worker.add(daThread);
    }

    try {
        List<Future<Object>> results = es.invokeAll(worker);
    } catch (InterruptedException e) {
        LOG.error("InvokeAll worker failed.");
    }
}
```

Figure A. 7: Source code for a start dynamic analysis

```
@Override
public List<String> receiveAllDatabaseIds(String databaseName) {
    List<String> databaseIds = new ArrayList<>();
    HttpResponse<JsonNode> response = null;
    try {
        response = Unirest.get(endpoint + databaseName + "/_search") GetRequest
                .queryString( name: "q",  value: "*") HttpRequest
                .queryString( name: "_source",  value: "_id") HttpRequest
                .asJson();
    } catch (UnirestException e) {
        LOG.error(REST_ERROR, e);
    }
    extractIds(databaseIds, response);
    return databaseIds;
}
```

Figure A. 8: Sample code for retrieving the analyzed result from Elasticsearch

Appendix B: Python Source code

```
def _get_static_numerical_features_from_app(self, app):
    features = [app.permission_amount,
                app.used_features_amount,
                app.additional_permissions_amount,
                app.activities_amount,
                app.services_amount,
                app.receivers_amount,
                app.assets_amount,
                app.filtered_class_names_amount,
                app.filtered_invoke_refs_amount,
                app.filtered_method_names_amount,
                app.filtered_uris_amount,
                app.total_classes_amount,
                app.total_method_amount,
                app.total_package_amount,
                app.total_system_classes_amount]
    return features
def _get_dynamic_numerical_features_from_app(self, app):
    features = [app.api_globals_amount,
                app.api_reflection_amount,
                app.api_generic_amount,
                app.api_content_amount,
                app.api_network_amount,
                app.api_dex_amount,
                app.api_crypto_amount,
                app.api_file_amount,
                app.api_binder_amount,
                app.api_fingerprint_amount]
    return features
```

Figure B. 1: Code snippet used to get hybrid features from the app

```
def _get_adjusted_features(self, app):
    categories = dict()
    numerical = []
    if Settings.FEATURE_TYPE == FeatureType.HYBRID_S.value or \
            Settings.FEATURE_TYPE == FeatureType.HYBRID_I.value:
        categories = self._get_categorical_features_from_app(app)
        numerical = self._get_numerical_features_from_app(app)
    elif Settings.FEATURE_TYPE == FeatureType.DYNAMIC_S.value or \
            Settings.FEATURE_TYPE == FeatureType.DYNAMIC_I.value:
        categories = self._get_dynamic_categorical_features(app)
        numerical = self._get_dynamic_numerical_features_from_app(app)
    elif Settings.FEATURE_TYPE == FeatureType.STATIC.value:
        categories = self._get_static_categorical_features(app)
        numerical = self._get_static_numerical_features_from_app(app)
    return categories, numerical
```

Figure B. 2: Code snippet used to get adjusted features of an app

```
def _random_forest_export(self):
    self.rf = RandomForestClassifier(n_estimators=140, max_depth=None,
                                     max_features="sqrt")
    self.rf.fit(self.X_train, self.y_train)
    fimportance = self.rf.feature_importances_.tolist()
    LOGGER.info('Feature impotance.............. ')
    LOGGER.info(fimportance)
    self.fdesc = fc.load_feature_description()
    LOGGER.info('feature description.............')
    LOGGER.info(self.fdesc)
    importance = heapq.nlargest(10, fimportance)
    LOGGER.info(importance)
    importance_names = [self.fdesc[fimportance.index(value)] for value in importance]
    LOGGER.info('Selected Feature impotance and Importance score..... ')
    LOGGER.info(importance_names)
    for feature in zip(importance_names, importance):
        LOGGER.info(feature)
    self.plot_chart(fimportance)
```

Figure B. 3: Code snippet used to feature selection

```
def train_extra_trees(self):
    LOGGER.info("-------------Extra Trees Model---------------")
    extra_trees = ExtraTreesClassifier()
    param_grid = {
        'n_estimators': [140, 143, 145],
        'max_features': ['sqrt', 'log2']}
    clf_extra_trees = GridSearchCV(extra_trees, param_grid, cv=5, n_jobs=2)

    # ET Fit
    clf_extra_trees.fit(self.X_train, self.y_train)
    # ET Predict
    self.y_test_pred = clf_extra_trees.best_estimator_.predict(self.X_test)
    # ET Matrices
    LOGGER.info("Extra Trees Evaluation parameters:")
    self.print_metrices_out()
    p = plotter()
    p.plot_confusion_matrix(self.y_test, self.y_test_pred, normalize=True,
                            title='Normalized confusion matrix')
    # """ plot precison recall test/split"""
    y_scores = clf_extra_trees.predict_proba(self.X_test)
    precision, recall, thresholds = precision_recall_curve(self.y_test, y_scores[:, 1],)
    self.plot_prec_recall_vs_tresh(precision, recall, thresholds)
    self.roc_plot(self.X_test, self.y_test, clf_extra_trees,
                  title="ROC Curve ET Test/split")
```

Figure B. 4: Code snippet used for model training and classification

Appendix C: Prototype Evaluation Sample Results

C.1. Hybrid analysis sample execution output

```
12 recieve custom:ApkMetaReport.app_file_name,ApkMetaReport.
   pkg_name,ApkMetaReport.activities,ApkMetaReport.services
13 prepare Apps:F:/Bengin_Android_APK_2018/Apk2/Apk111/
14 Connection Established
15 C:\ADT\sdk\platform-tools/adb -s f7edc013 pull /data/data/
   de.robv.android.xposed.installer/log/error.log  C:/ADT/sdk
   /
16 Recieve document : http://elastic:changeme@localhost:9200/
   component_analysis_bengin_2019/report/
   b231ac81bd1af0164fc4ffdaed5ce32ae54619292dc3a726200054da7e
   bbe2ce
17 response recieve doc:Not Found
18 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
   analyzer.ComponentAnalyzer - Start component (non-
   interactive) dynamic analysis of app com.truecaller.
19 command1 :C:\ADT\sdk\platform-tools/adb -s f7edc013
   install  F:/Bengin_Android_APK_2018/Apk2/Apk111/
   AF3DWBexsd0viV96e5U9-
   SkM_V5zGHbwfGW_98TjGgFhws6OC7KrgkqJLkwcC21mygf2XRGKzKnqOhl
   K2lA6dlWvjmoUhPLNAZ5pyHCHxr22mGI5lgnn6Mo.apk
20 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
   analyzer.ComponentAnalyzer - Initialize component Analysis
   .
21 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
   analyzer.ComponentAnalyzer - Stop app com.truecaller and
   delete all logs.
22 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
   analyzer.ComponentAnalyzer - Check for unlocking of device
   .
23 command1 :C:\ADT\sdk\platform-tools/adb -s f7edc013  shell
    dumpsys gfxinfo
24 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
   analyzer.ComponentAnalyzer - Start initial launch of app.
25 command1 :C:\ADT\sdk\platform-tools/adb -s f7edc013  shell
    dumpsys gfxinfo
26 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
   analyzer.ComponentAnalyzer - Checking for permission
   dialog.
27 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
   analyzer.ComponentAnalyzer - Checking for error dialog.
28 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
   analyzer.ComponentAnalyzer - Launching activity com.
   truecaller.messaging.smspermission.SmsPermissionActivity.
29 command1 :C:\ADT\sdk\platform-tools/adb -s f7edc013  shell
    dumpsys gfxinfo
```

```
1573 command1 :C:\ADT\sdk\platform-tools/adb -s f7edc013
     shell dumpsys gfxinfo
1574 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
     analyzer.ComponentAnalyzer - Checking for permission
     dialog.
1575 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
     analyzer.ComponentAnalyzer - Checking for error dialog.
1576 [pool-2-thread-1] INFO AnalaysisLayer.dynamic_analysis.
     analyzer.ComponentAnalyzer - Finished component (non-
     interactive) dynamic analysis of app com.truecaller.
1577 Recieve document : http://elastic:changeme@localhost:
     9200/component_analysis_bengin_2019/report/
     b231ac81bd1af0164fc4ffdaed5ce32ae54619292dc3a726200054da
     7ebbe2ce
1578 response recieve doc:Not Found
1579 Handle response: Created
1580 [pool-2-thread-1] INFO AnalaysisLayer.ElasticsearchDb.
     ElasticAdapter - Document '
     b231ac81bd1af0164fc4ffdaed5ce32ae54619292dc3a726200054da
     7ebbe2ce' created successfully.
1581 Dynamic Analayis completed
1582
```

C2. A Hybrid analysis classification sample execution output

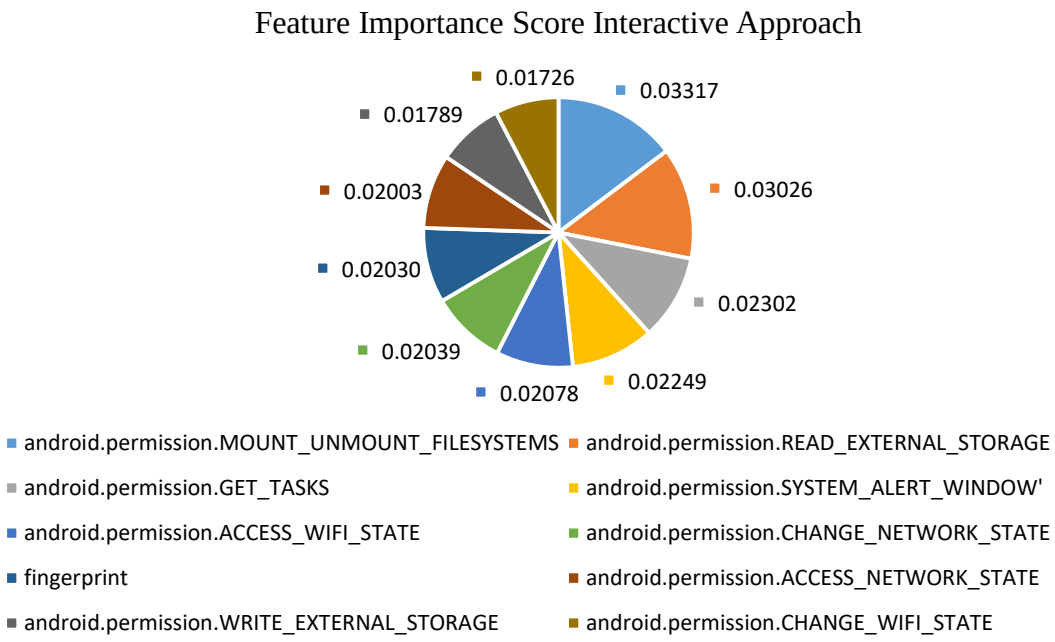


Figure C. 1: Feature importance score interactive approach due to random forest

a) Due to Random Forest b) Due to Extra Trees

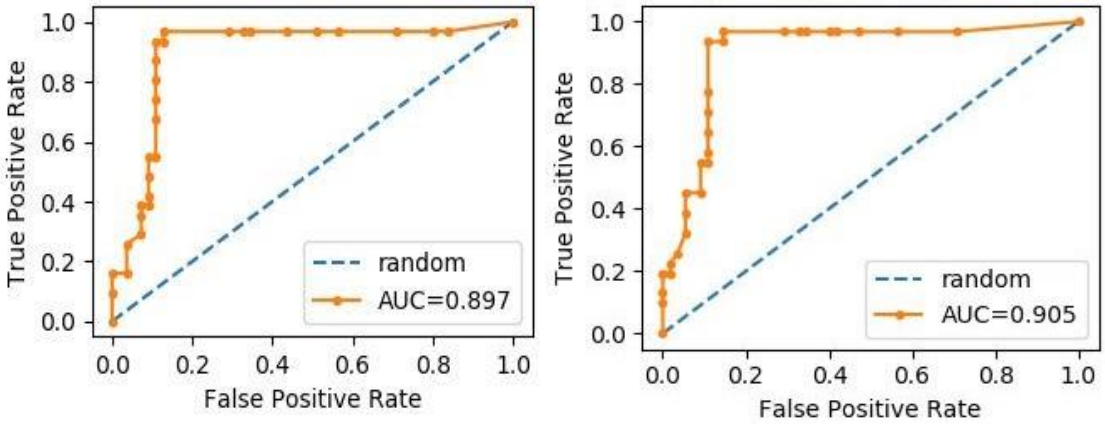


Figure C. 2: ROC of classifiers used in the interactive approach

a) Due to Random Forest b) Due to Extra Trees

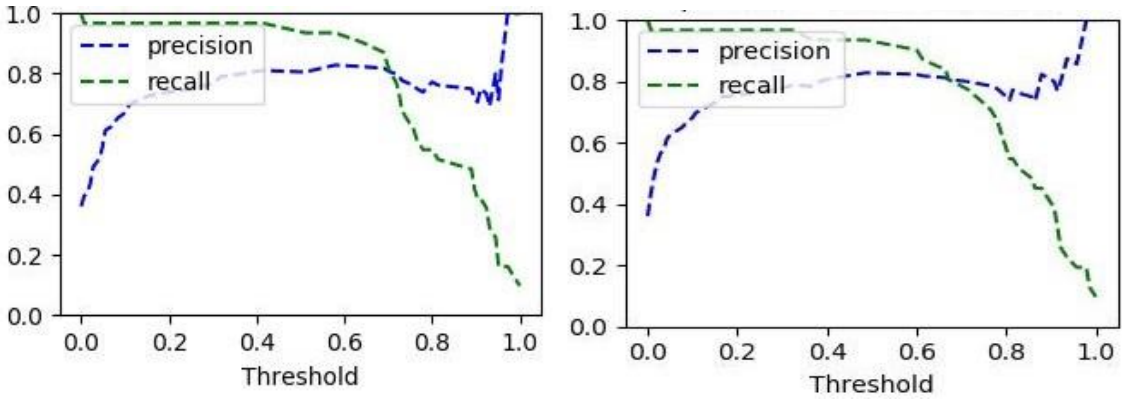


Figure C. 3: Precision-recall threshold of an interactive approach

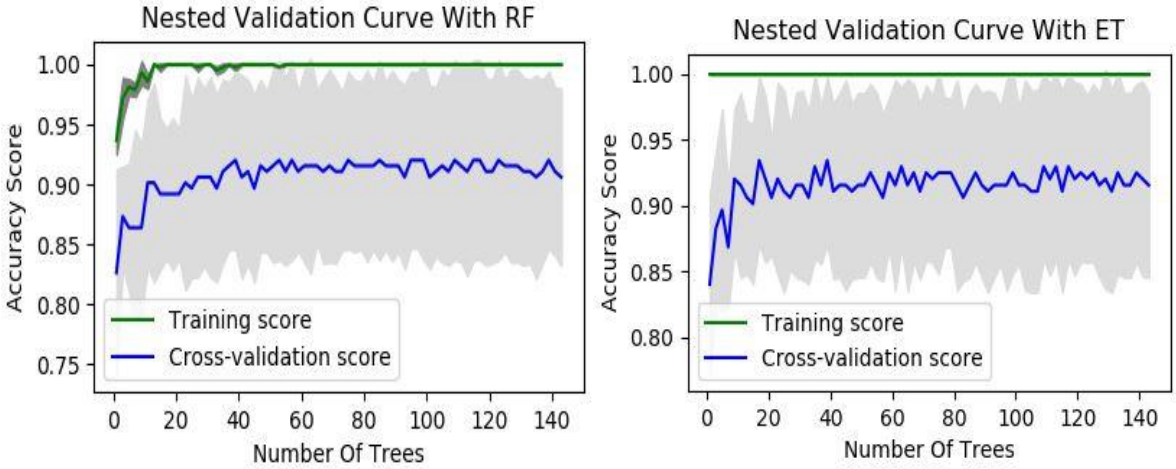


Figure C. 4: Validation curve of nested cross-validation interactive approach

Table C. 1: Number of apps failing Monkey test for twenty-six categories tested

No	*App Category*	*Tool Names*				**Original**
		Apktool	***Soot-Jimple***	***Dex2Jar***	***Proposed***	
1	Books Reference	4	16	15	3	3
2	Business	5	18	12	3	3
3	Comics	5	15	17	6	4
4	Communication	7	14	25	4	6
5	Education	7	27	15	7	5
6	Entitlement	8	21	30	6	4
7	Finance	3	36	34	4	2
8	Health Fitness	8	16	16	7	8
9	Libraries' Demo	6	15	15	2	5
10	Life Style	3	17	27	3	2
11	Live Wallpaper	6	20	17	6	4
12	Media Video	7	11	9	2	7
13	Medical	8	10	12	4	5
14	Music Audio	4	10	22	5	3
15	News Magazines	7	15	18	9	7
16	Personalization	5	12	22	3	3
17	Photography	4	21	24	4	2
18	Productivity	7	14	25	9	4
19	Shopping	11	29	28	1	9
20	Social	6	30	26	5	3
21	Sports	3	10	16	3	3
22	Tools	5	26	22	8	4
23	Transportation	4	20	20	2	3
24	Travel Local	7	14	16	3	5
25	Weather	8	16	24	5	7
26	widgets	9	17	19	7	8
Total Failure		157	470	526	121	119

Appendix D: Selected Literature and Repositories for SLR

Table D. 1: The top ten venues in security & privacy fields

Acronym	*Full Name*
COMPSEC	Computers & Security
CCS ACM	ACM Conference on Computer and Communications Security
S&P	IEEE Symposium on Security and Privacy
ACSAC	Computer Security Applications Conference
SEC	USENIX Security Symposium
TDSC	IEEE Transactions on Dependable and Secure Computing
IEEE TIFS	IEEE Transactions on Information Forensics and Security
NDSS	Network and Distributed System Security Symposium
ASIACRYPT	International Conference on The Theory & Application of Cryptology and Information Security
SOUPS	USENIX Symposium On Usable Privacy and Security

Table D. 2: Selected Literature for SLR

Ref#	*Author(s)*	*Year*	*Title*	*Repository*
[51]	Weizhi Meng, Elmar Wolfgang Tischhauser, Qingju Wang, Yu Wang, And Jinguang Han	2018	When Intrusion Detection Meets Blockchain Technology: A Review	IEEE
[40]	Jingjing Gu, Binglin Sun, Xiaojiang Du, Jun Wang, Yi Zhuang, and Ziwang Wang	2018	Consortium Blockchain-Based Malware Detection in Mobile Devices	IEEE
[52]	Deepak Puthal, Nisha Malik, Saraju P. Mohanty, Elias Kougianos, and Chi Yang	2018	The Blockchain as a Decentralized Security Framework	Google Scholar
[53]	Yung-Feng Lu, Chin-Fu Kuo, Hung-Yuan Chen, Chang-Wei Chen, and Shih-Chun Chou	2018	An SVM-based Malware Detection Mechanism for Android Devices	Science Direct
[54]	Feng, Jianfeng Ma1, Cong Sun, Xinpeng Xu, And Yuwan Ma	2018	A Novel Dynamic Android Malware Detection System With Ensemble Learning	IEEE
[55]	Joshua Garcia, Mahmoud Hammad, and Sam Malek	2018	Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware	ACM
[56]	Ignacio Martín , José Alberto Hernández, Alfonso Muñoz and Antonio Guzmán	2018	Android Malware Characterization Using Metadata and Machine Learning Techniques	ACM

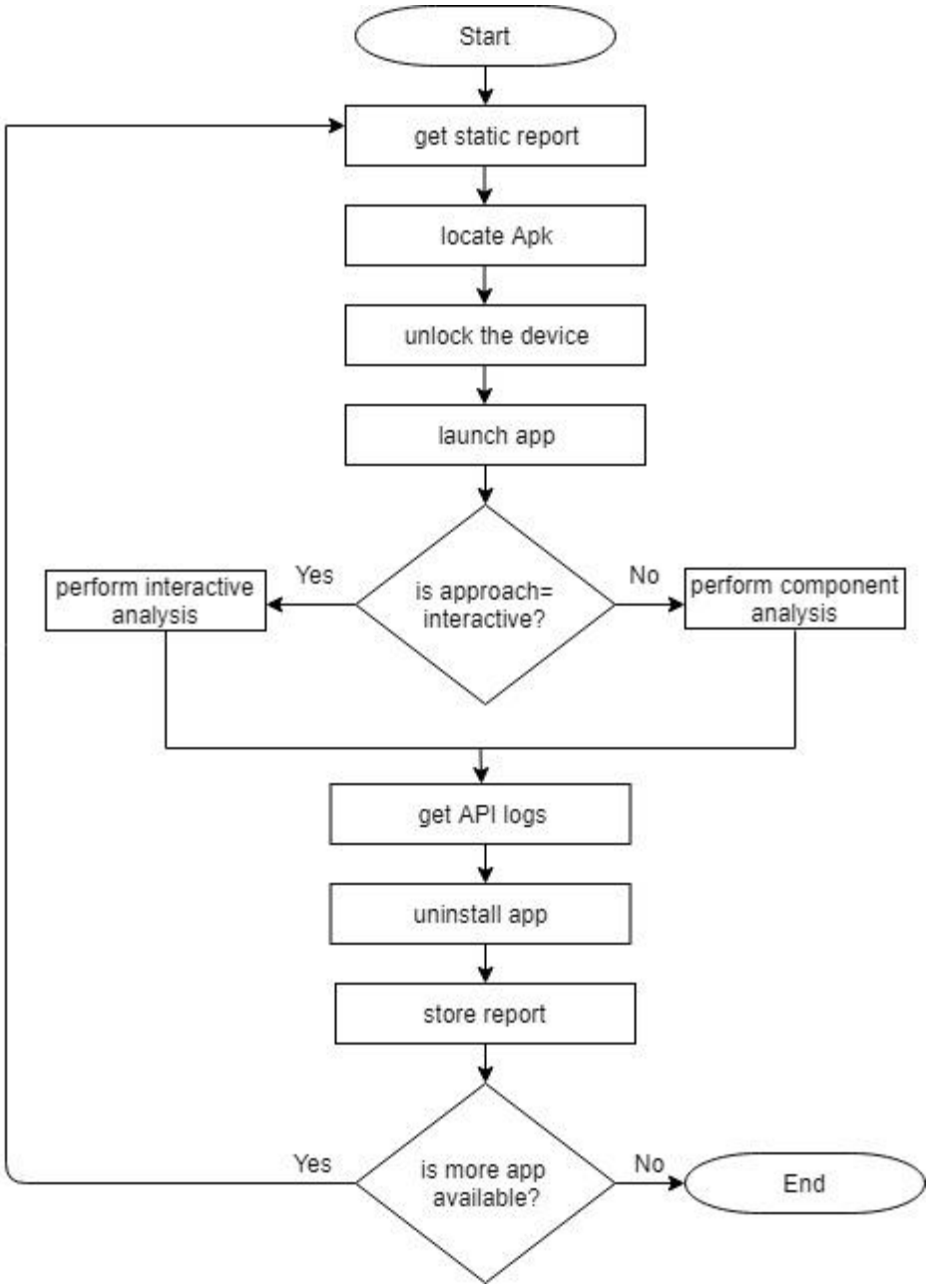
[57]	Mohammed K. Alzaylaee, Suleiman Y. Yerima, Sakir Sezer	2017	Improving Dynamic Analysis of Android Apps Using Hybrid Test Input Generation	IEEE
[58]	Saurabh Raje, Shyamal Valeria	2017	Decentralized firewall for malware detection	ACM
[59]	Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer	2017	Deep Android Malware Detection	ACM
[44]	Saba Arshad, Munam A. Shah, Abdul Wahid, Amjad Mehmood, and Song	2017	A Novel three-Level Hybrid Malware Detection Model for Android Operating System	IEEE
[60]	Ravi Kiran Varma P, Kotari Prudvi Raj, and K. V. Subba Raju	2017	Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms	IEEE
[61]	Wei Wanga, Yuanyuan Lia, Xing Wanga, Jiqiang Liu, Xiangliang Zhang	2017	Detecting Android Malicious Apps and Categorizing Benign Apps with Ensemble of Classifiers	Google Scholar
[43]	Fauzia Idrees, Muttu Krishnan Rajarajan, Mauro Conti, Tom Chen, and Yogachandran Rahulamathavan	2017	A novel Android malware detection system using ensemble learning methods	IEEE
[62]	Anusha Damodaran, Fabio Di Troia, and Visaggio Aaron Corrado	2017	A Comparison of Static, Dynamic, and Hybrid Analysis for Malware Detection	Google Scholar
[22]	Li Li, Tegawend, F. Bissyand, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, Yves Le Traon	2017	Static analysis of android apps: A systematic literature review	Google Scholar
[63]	Jun Songa, Chunling Hana, Kaixin Wang, Jian Zhao, Rajiv Ranjan, Lizhe Wanga	2016	An integrated static detection & analysis framework for android	Science Direct
[42]	Yuan, Zhenlong Lu, Yongqiang Xue, Yibo	2016	Android Malware Detection Using Deep Learning	Science Direct
[64]	Wei-Ling Chang, Wei Wu, and Hung-Min Sun	2016	An Android Behavior-Based Malware Detection Method using Machine Learning	IEEE
[65]	Fei Tong and Zheng Yana	2016	A hybrid approach of mobile malware detection in Android	Science Direct
[66]	Suleiman Y. Yerima, Sakir Sezer, Igor Muttik	2015	High Accuracy Android Malware Detection With Ensemble Learning	Spring
[38]	Kabaka Abdullah Talha, Dogru Ibrahim Alper, and Cetin Aydin	2015	APK Auditor: Permission-based Android malware detection system	Science Direct

[67]	Wei Yang, Xusheng Xiao, Benjamin Andowz, Sihan Li, Tao Xie, William Enck	2015	AppContext: Differentiating Malicious and Benign Mobile App Behaviors Using Context	IEEE
[49]	Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Ainuddin Wahid Abdul Wahab	2015	Evaluation of feature selection in mobile malware detection	Science Direct
[68]	Xing Liu and Jiqiang Liu	2014	A Two-layered Permission based Android Malware Detection Scheme	IEEE
[34]	Dimitrios Damopoulos, Georgios Kambourakis, and Georgios Portokalidis	2014	The Best of Both Worlds. A Framework for the Synergistic Operation of Host and Cloud Anomaly-based IDS for Smartphones	Google Scholar
[69]	Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck	2014	Drebin: Effective and Explainable classification of Android Malware	ACM
[70]	Brandon Amos, Hamilton Turner, Jules White	2013	Applying ML classifiers to dynamic Android malware detection at scale	IEEE
[71]	Naser Peiravian and Xingquan Zhu	2013	ML for Android Malware Detection Using Permission and API Calls	IEEE
[72]	Gianluca Dini1, Fabio Martinelli, Ilaria Matteucci, Marinella Petrocchi, Andrea Saracino, and Daniele Sgandurra	2012	A Multi-criteria-Based Evaluation of Android Applications	Spring

Table D. 3: Number of papers published in the selected publication repository

Name of venue	***Numbers of papers***	***Percentage***
ACM	5	16.67 **%**
Google Scholar	5	16.67**%**
IEEE	12	40 ***%***
Science Direct	6	20 ***%***
Spring	2	6.67 ***%***

Appendix E: Proposed Algorithms and UI



Algorithm E. 1: Dynamic analysis flowchart

Figure E. 1: The user interface of the proposed framework