ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Προηγμένες Μέθοδοι Ανίχνευσης Κακόβουλου Λογισμικού στην Πλατφόρμα του Android

Συγγραφέας
Βασίλειος Κουλιαρίδης

Επιβλέπων
Καθ. Γεώργιος Καμπουράκης

ΔΙΑΤΡΙΒΗ
για την απόκτηση Διδακτορικού Διπλώματος
στο

Εργαστήριο Ασφάλειας Πληροφοριακών και Επικοινωνιακών Συστημάτων
Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων
Πολυτεχνική Σχολή
Πανεπιστήμιο Αιγαίου

Σάμος, Μάρτιος 2021

University of the Aegean

Doctoral Thesis

Advanced Methods for Android Malware Detection

Author
Vasileios Kouliaridis

Supervisor
Prof. Georgios Kambourakis

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy

at the

Laboratory of Information and Communication Systems Security
Department of Information and Communication Systems Engineering
School of Engineering
University of the Aegean

Samos, March 2021

Υπεύθυνη Δήλωση

Εγώ, ο Βασιλειος Κουλιαρίδης, δηλώνω ότι είμαι ο αποκλειστικός συγγραφέας της υποβληθείσας Διδακτορικής Διατριβής με τίτλο «Προηγμένες Μέθοδοι Ανίχνευσης Κακόβουλου Λογισμικού στην Πλατφόρμα του Android». Η συγκεκριμένη Διδακτορική Διατριβή είναι πρωτότυπη και εκπονήθηκε αποκλειστικά για την απόκτηση του Διδακτορικού διπλώματος του Τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων. Κάθε βοήθεια, την οποία είχα για την προετοιμασία της, αναγνωρίζεται πλήρως και αναφέρεται επακριβώς στην εργασία.

Επίσης, επακριβώς αναφέρω στην εργασία τις πηγές, τις οποίες χρησιμοποίησα, και μνημονεύω επώνυμα τα δεδομένα ή τις ιδέες που αποτελούν προϊόν πνευματικής ιδιοκτησίας άλλων, ακόμη κι εάν η συμπερίληψη τους στην παρούσα εργασία υπήρξε έμμεση ή παραφρασμένη. Γενικότερα, βεβαιώνω ότι κατά την εκπόνηση της Διδακτορικής Διατριβής έχω τηρήσει απαρέγκλιτα όσα ο νόμος ορίζει περί διανοητικής ιδιοκτησίας και έχω συμμορφωθεί πλήρως με τα προβλεπόμενα στο νόμο περί προστασίας προσωπικών δεδομένων και τις αρχές της Ακαδημαϊκής Δεοντολογίας.

Υπογραφή: _____

Ημερομηνία: Μάρτιος 7, 2021

Declaration of Authorship

I, Vasileios Kouliaridis, declare that this thesis entitled, "Advanced Methods for Android Malware Detection" and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: March 7, 2021

Advising Committee of this Doctoral Thesis:

Professor Georgios Kambourakis, Supervisor
Department of Information and Communication Systems Engineering
University of the Aegean, Greece

Professor Emmanouil Maragoudakis, Advisor
Department of Informatics
Ionian University, Greece

Assistant Professor Elisavet Konstantinou, Advisor
Department of Information and Communication Systems Engineering
University of the Aegean, Greece

University of the Aegean, Greece
2021

Approved by the Examining Committee:

Stefanos Gritzalis
Professor, University of Piraeus, Greece

Georgios Kambourakis
Professor, University of the Aegean, Greece

Emmanouil Maragoudakis
Professor, Ionian University, Greece

Elisavet Konstantinou
Assistant Professor, University of the Aegean, Greece

Panagiotis Rizomiliotis
Assistant Professor, Harokopio University, Greece

Dimitrios Skoutas
Assistant Professor, University of the Aegean, Greece

Marios Anagnostopoulos
Assistant Professor, Aalborg University, Denmark

University of the Aegean, Greece
2021

Vasileios Kouliaridis

Department of Information and Communication Systems Engineering
School of Engineering
University of the Aegean

Abstract

Department of Information and Communication Systems Engineering
School of Engineering
University of the Aegean

Doctor of Philosophy
by Vasileios Kouliaridis

Mobile devices are an integral part of our everyday life. From online social networks to mobile banking transactions, mobile devices are more or less trusted and used by billions of people worldwide. In the same trend, the number of vulnerabilities exploiting mobile devices are also augmented on a daily basis and, undoubtedly, popular mobile platforms, such as Android and iOS, represent an alluring target for malware writers. The Android operating system is currently the most widespread mobile platform worldwide. As a result, countless malicious applications are deployed every year to capitalize on the popularity of this platform. Indeed, the topic of mobile malware detection has attracted significant attention over the last several years. However, while notable research has been conducted toward mobile malware detection techniques, most state-of-the-art approaches lean on rather outdated mobile malware datasets and Android operating system versions. Therefore, current proposals to mobile malware analysis and detection cannot easily keep up with future malware sophistication.

This PhD thesis aims to devise, propose, and meticulously assess advanced, more robust mobile malware analysis schemes destined to the Android platform. Precisely, we examine the effect of classification features when dynamic instrumentation is applied, to demonstrate that the effectiveness of base classification models based on either static or dynamic analysis without instrumentation is clearly outperformed by models using dynamic instrumentation. This thesis also proposes and evaluates ensemble learning techniques, as well as a more sophisticated extrinsic ensemble approach to exhibit that ensemble models can further improve the performance of each individual base classifier. We also report on the effect of using either the entire feature set or a random subspace of classification features of instances to demonstrate how the latter assists an extrinsic malware detection ensemble to further augment its effectiveness. This thesis also aims to examine the usefulness of dimensionality reduction techniques, when exclusively applied on malware detection base verifiers, as well as ensembles. On top of everything else, the thesis at hand aims to analyze the most challenging and recent mobile malware

datasets and explore the most significant feature categories in terms of classification effectiveness.

Greek Abstract

(Εκτεταμένη Περίληψη)

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων
Πολυτεχνική Σχολή
Πανεπιστήμιο Αιγαίου

Διδακτορική διατριβή
του Βασίλειου Κουλιαρίδη

Οι φορητές συσκευές αποτελούν αναπόσπαστο κομμάτι της καθημερινής μας ζωής. Από διαδικτυακά κοινωνικά δίκτυα έως συναλλαγές τραπεζικής κινητής τηλεφωνίας, οι κινητές συσκευές είναι περισσότερο ή λιγότερο αξιόπιστες και χρησιμοποιούνται από δισεκατομμύρια ανθρώπους παγκοσμίως. Την ίδια στιγμή, ο αριθμός των ευπαθειών που εκμεταλλεύονται κινητές συσκευές αυξάνεται επίσης σε καθημερινή βάση, και αναμφίβολα, δημοφιλείς πλατφόρμες για κινητά, όπως το Android και το iOS, αντιπροσωπεύουν έναν ιδιαίτερα δελεαστικό στόχο για οποιονδήποτε συγγραφέα κακόβουλου κώδικα. Το λειτουργικό σύστημα Android είναι σήμερα η πιο διαδεδομένη πλατφόρμα για κινητά παγκοσμίως. Ως αποτέλεσμα, αμέτρητες κακόβουλες εφαρμογές αναπτύσσονται κάθε χρόνο για να εκμεταλλευτούν τη δημοτικότητα αυτής της πλατφόρμας. Πράγματι, το ζήτημα της ανίχνευσης κακόβουλου λογισμικού για κινητές συσκευές έχει προσελκύσει σημαντικές ερευνητικές προσπάθειες τα τελευταία χρόνια. Ωστόσο, ενώ έχει γίνει αξιοσημείωτη έρευνα σχετικά με τις τεχνικές ανίχνευσης κακόβουλου λογισμικού για κινητές πλατφόρμες, οι περισσότερες προηγμένες προσεγγίσεις βασίζονται σε ξεπερασμένα σύνολα δεδομένων (datasets) κακόβουλου λογισμικού και σε παρωχημένες εκδόσεις του λειτουργικού συστήματος Android. Επομένως, οι τρέχουσες προσεγγίσεις για την ανάλυση και τον εντοπισμό κακόβουλου λογισμικού για κινητές συσκευές δεν μπορούν εύκολα να συμβαδίσουν με τη μελλοντική εξειδίκευση κακόβουλου λογισμικού.

Η παρούσα διδακτορική διατριβή στοχεύει να προτείνει, εξετάσει, και αξιολογήσει νέες προσεγγίσεις ανάλυσης κακόβουλου λογισμικού για κινητές συσκευές στην πλατφόρμα του Android. Συγκεκριμένα, εξετάζουμε την επίδραση των χαρακτηριστικών ταξινόμησης όταν παρέχεται κώδικας ελέγχου (instrumentation code) κατά την εκτέλεση του προγράμματος, για να αποδείξουμε ότι η αποτελεσματικότητα των μοντέλων ταξινόμησης που βασίζονται είτε σε στατική είτε σε δυναμική ανάλυση χωρίς κώδικα ελέγχου υπερτερεί σε σχέση με τα μοντέλα που εφαρμόζουν δυναμική ανάλυση με τον κλασσικό τρόπο. Επιπλέον, η παρούσα

διατριβή προτείνει και αξιολογεί τεχνικές συλλογικής μάθησης, (ensemble learning) καθώς και μια πιο εξελιγμένη προσέγγιση εξωγενούς συλλογικής μάθησης για να καταδείξει ότι τα μοντέλα συλλογικής μάθησης μπορούν να βελτιώσουν περαιτέρω την απόδοση κάθε επιμέρους βασικού ταξινομητή. Επιπλέον, η διατριβή διερευνά την επίδραση της χρήσης είτε ολόκληρου του συνόλου χαρακτηριστικών ταξινόμησης είτε ενός τυχαίου αντιπροσωπευτικού υποσυνόλου με σκοπό να καταδείξει πώς το τελευταίο βοηθά ένα εξωγενές σύνολο εντοπισμού κακόβουλου λογισμικού για να αυξήσει περαιτέρω την αποτελεσματικότητά του. Η διατριβή στοχεύει επίσης να εξετάσει τη χρησιμότητα των τεχνικών μείωσης διαστάσεων, όταν εφαρμόζεται αποκλειστικά σε βασικούς ταξινομητές ανίχνευσης κακόβουλου λογισμικού, καθώς και σε μοντέλα συλλογικής μάθησης. Τέλος, η διατριβή στοχεύει στην ανάλυση των πιο απαιτητικών και πρόσφατων συνόλων δεδομένων κακόβουλου λογισμικού για κινητές πλατφόρμες και στη διερεύνηση της σημαντικότερης κατηγορίας χαρακτηριστικών ταξινόμησης.

Acknowledgements

First and foremost, i would like to express my deep gratitude to my supervisor Professor Georgios Kambourakis, for his guidance, advice, and continuous support throughout my PhD study. He has been of invaluable support throughout these years in many ways, which has really helped me to shape my research.

I would like to express my gratitude to Dr. Nektaria Potha, Dr. Konstantia Barbatsalou, and Dr. Geneiatakis Dimitrios, for their precious help and support during my research. Particularly, i am grateful to Dr. Potha for her valuable contributions to chapters 5, 6, and especially 7, of this PhD thesis. Additionally, i would like to thank the members of the examining committee, for investing time in reviewing this thesis and offering advice.

Last but not least, I would like to express my utmost gratitude to my family for their unconditional love and trust.

Dedicated to my family

Contents

List of Figures

List of Tables

Abbreviations

ADB	Android Debug Bridge
API	Application Programming Interface
APK	Android Application Package
ARM	Advanced RISC Machine
ART	Android Runtime
AUC	Area Under the Curve
AVG	Average
CA	Classification Accuracy
CA	Code Analysis
CI	Code Instrumentation
CPU	Central Processing Unit
CSV	Comma-separated values
DB	Database
DDoS	Distributed Denial of Service
DI	Dynamic Instrumentation
DR	Demensionality Reduction
EL	Ensemble Learning
ERBE	Extrinsic Random-based Ensemble
EC2	Ensemble Clustering and Classification
FI	Feature Importance
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
HLLE	Hessian Eigenmapping

ICC	Inter-Component Communications
IDS	Intrusion Detection System
IPC	Inter-Process Communication
k-NN	k-Nearest Neighbors
LDA	Linear Discriminant Analysis
LLE	Locally Linear Embedding
LR	Logistic Regression
MA	Manifest Analysis
MAC	Mandatory Access Control
MCA	Multiple Correspondence Analysis
MDS	Multi-dimensional Scaling
MIET	Mal-warehouse Information Extraction Tool
ML	Machine Learning
MLP	Multiple Layer Perception
MV	Majority Vote
NB	Naive Bayes
OEM	Original Equipment Manufacturer
OS	Operating System
RCA	Principal Component Analysis
ROC	Receiver Operating Characteristic
RPC	Remote Procedure Calls
SDK	Software development kit
RF	Random Forest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
SI	Static Instrumentation
TEE	Trusty Trusted Execution Environment
TLS	Transport Layer Security
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate
SCA	System Calls Analysis

SGD	Stochastic Gradient Descent
SRA	System Resources Analysis
SNE	Stochastic Neighbor Embedding
SMS	Short Message Service
SVM	Support Vector Machine
URL	Uniform Resource Locator
USB	Universal Serial Bus
UI	User Interface
UIA	User Interaction Analysis
VM	Virtual Macine

Chapter 1

Introduction

Modern smartphones combine the communication capability of cellphones with the functionality of a *Personal Computer (PC)*. Such devices allow users to access a large variety of on-line services, such as web navigation, message exchange, and mobile banking. In addition, most smartphones are equipped with an application-based interface, which facilitates the downloading of various programs that can perform a diversity of tasks. However, the availability of such mobile services increases their susceptibility to malware attacks [1]. Mobile malware poses undoubtedly a major threat to the continuously increasing number of mobile users worldwide. The Android operating system (OS) is currently the most prevalent mobile platform, with a market share that exceeds 74% [2]. In this context, the openness of the Android OS, as well as its immense penetration into the market makes it a hot target for malware writers. According to a Kaspersky report, 5,6M mobile malicious installation packages for Android have been discovered in 2019 [3]. Furthermore, while researchers have been trying vigorously to find optimal detection solutions, mobile malware is becoming more sophisticated and its writers are getting increasingly skilled in hiding malicious code [4]. To stay covert, malware writers deploy numerous techniques, such as encrypting code, thus making legacy signature-based detection increasingly harder. Furthermore, most Antivirus (anti-malware) systems use static signatures of previously identified malware to identify the existing malware, and while they do so quite successfully, they are far from achieving the same detection performance for new instances of malware. That is, mutated and zero-day malware requires continuous update of the signature database as malicious applications (apps) are constantly improved to circumvent the various detection methods.

Machine learning (ML) has been exploited in the development of intelligent systems for several years. Supervised ML classifiers acquire a labeled dataset and generate an output model which is able to process new data. The adoption of ML classifiers has therefore been demonstrated to improve the precision of detection [5]. Consequently, machine learning has also become an important asset to mobile malware detection. Current mobile malware detection approaches lean primarily towards static anomaly-based detection [6, 7, 8, 9, 5], although methods based on dynamic analysis have started to proliferate [6, 9, 10, 11]. Naturally, the cardinal reason behind the popularity of static analysis techniques arises from the fact that they do not require the app to be running, hence they are usually faster and straightforward to implement. Generally, anomaly-based detection comprises two distinct phases; the training and the detection or testing one. It typically employs machine learning to detect malicious behavior, i.e., deviation from a model built during the training phase.

Up to now, a plethora of detection systems rely on machine learning to classify whether a mobile app is malware or not, and they do so with a high reported detection accuracy. However, various parameters can influence machine learning effectiveness. Precisely, the instances used to evaluate a model can have a major impact on the classification accuracy. The Android OS is a continuously updated operating system with major security enhancements, as well as new features being introduced every year. For the same reason, each year new mobile malware families emerge, exploiting different attack vectors, which mutates their behavior [12, 13]. A mobile app is flagged as malicious by the ML classifier which uses the features extracted during static or dynamic analysis as an input. A ML algorithm trained using outdated data is therefore not as reliable as it should have been.

To fill the above mentioned literature gaps, this doctoral thesis seeks for new methods that can address the reported limitations and empower detection systems with agile and adaptable characteristics. Under this prism, our work:

(a) Provides a thorough analysis on state-of-the-art mobile malware detection techniques for offering optimal countermeasures.

(b) Implements and evaluates new analysis techniques to improve current detection methodologies.

(c) Introduces and evaluates new methodologies in mobile malware detection through advanced machine learning properties.

As further explained in the next section, the PhD thesis at hand aims in introducing novel tools and methodologies to counter the latest and future mobile malware.

1.1 Motivation and Objectives

As previously mentioned, mobile malware detection approaches nowadays lean primarily towards static anomaly-based detection. In this context, a key point, which to our knowledge is not properly addressed in the literature, is the importance of each feature category, say, permissions and intents, in mobile app classification. Simply put, which group of features in general, and which features within each group in particular do contribute the most information when it comes to classification? And, is the answer to the previous question related to the employed dataset as the case may be? Furthermore, while significant research has been conducted towards the use of static approaches, insufficient attention has been paid to hybrid approaches for mobile malware detection. So, in cases where features stemming from static analysis cannot produce satisfactory results, is hybrid analysis a better direction for future mobile malware detection schemes? Overall, the galloping rise of mobile malware of any kind calls for more robust detection solutions by leveraging on ML.

Given the above mentioned observations, the goal of the PhD thesis at hand is to introduce and rigorously assess robust methodologies and tools to counter contemporary mobile malware, as well as to explore future needs and limitations of current mobile malware detection approaches. In a nutshell, the objectives, and simultaneously the research pillars of this PhD thesis are as follows:

Objective 1: With a focus on the Android platform, we intent to shed light on mobile malware detection techniques proposed in the literature so far and provide a thorough analysis over the analyzed features and the machine learning classification and detection methodologies used. Through this analysis we intent to identify open research challenges in mobile malware analysis and detection and pinpoint on future directions.

Objective 2: We aim to deliver novel methodologies which cater for highly accurate mobile malware detection mechanisms, and therefore can repel future malicious behavior with a high precision.

Objective 3: We aim to expand the second objective and further explore ML methodologies which can provide continuous improvement in mobile malware detection, by improving the gain of information on demand.

As detailed in the next subsection, the novelties of this work mainly lie in the last two objectives, while the first one basically explores the related literature for identifying possible gaps, shortcomings, and research directions.

1.2 Contributions

As already pointed out, the main intention of this PhD work is to devise advanced methodologies and tools as countermeasures to concurrent mobile malware threats, as well as to provide insight on current mobile malware analysis and detection approaches. More specifically, the contribution of this PhD thesis with respect to our publications in scientific journals and conferences is as follows:

- A survey on ML-powered mobile malware detection techniques[1][2]. The main axes of this contribution are:
  - Presents a thorough analysis of the latest mobile malware detection techniques.
  - Offers a comprehensive overview of the different approaches to mobile malware detection, in an effort to understand their detection method, explore their evaluation results, and possibly categorize each literature contribution under a novel classification scheme.
  - Identifies research challenges and future directions in mobile malware analysis and detection.

[1] V. Kouliaridis, G. Kambourakis, A comprehensive survey on machine learning techniques for Android malware detection. Information, 2020 (submitted)

[2] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, S. Chen. A survey on mobile malware detection techniques. IEICE Transactions on Information and Systems 103 (2), 204-211, 2020

  - Offers a detailed mapping of ML techniques utilized in this field of research and introduces a converging scheme to guide future research.

- Mal-warehouse: A data collection-as-a-service of mobile malware behavioral patterns[3]. The main contributions of this work are:

  - Presents Mal-warehouse, an open-source tool performing data collection-as-a-service for Android malware behavioral patterns.
  - Exhibits evaluation results by using ML techniques as a proof-of-concept of the detection capabilities of the proposed model.
  - Offers a publicly available database on the cloud, for further evaluation and processing by other researchers and interested parties.

- Feature importance in Android malware detection[4]. This work

  - Examines the hitherto most commonly used and modern datasets used in Android malware detection.
  - Demonstrates the most significant feature category, by using the average coefficients of permissions and intents for a large number of malware instances per corpus.
  - Reports the top ten features per dataset and discuss similarities between these corpora.

- Improving Android malware detection through dimensionality reduction techniques[5]. The contributions of this work are as follows:

  - Proposes a simple ensemble approach by aggregating the output of each malware instance separately, for a number of malware detection base models. The combination of base classification models achieves the best results in

[3]V. Kouliaridis, K. Barmpatsalou, G. Kambourakis and G. Wang. Mal-Warehouse: A Data Collection-as-a-Service of Mobile Malware Behavioral Patterns. 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 1503-1508, 2018, doi: 10.1109/SmartWorld.2018.00260.

[4]V. Kouliaridis, G. Kambourakis and T. Peng. Feature Importance in Android Malware Detection. 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1449-1454, 2020, doi: 10.1109/TrustCom50675.2020.00195.

[5]V. Kouliaridis, N. Potha, and G. Kambourakis. Improving Android Malware Detection Through Dimensionality Reduction Techniques. Machine Learning for Networking, Springer International Publishing, pp. 57–72, 2021, doi: 10.1007/978-3-030-70866-5_4.

comparison to each particular base model with reference to most challenging malware corpora. This evidently demonstrates that an ensemble of base classifiers based on a larger size and probably heterogeneous base models is the most appropriate and able to handle challenging malware detection scenarios, and thus can further improve the performance of each individual base classifier.

- Examines the usefulness of two well-known dimensionality reduction technique, namely, PCA and t-SNE when exclusively applied on malware detection base verifiers as well as ensembles, respectively. It is demonstrated that both these transformations can considerably increase the performance of each base model as well as the proposed ensembles. However, the implementation of t-SNE is more effective than PCA transformation and assists base models and malware detection ensemble methods to further increase their effectiveness in all the examined cases.
- Reports detailed experimental results on malware detection under the Androzoo dataset that are directly compared with state-of-the-art methods under the same settings.

- Two anatomists are better than one - Dual-level Android malware detection[6]. This work is dedicated to the design and implementation of a novel tool capable of applying both static and dynamic analysis of apps on the Android platform. Specifically, the main contribution axes of this work are as follows:
  - A methodology is presented that collects groups of static and dynamic features mapping the behavior of an app, including permissions, intents, API calls, Java classes, network traffic, and inter-process communication. Especially for the dynamic analysis part, among others, we improve the hitherto related work by means of contributing features emanating from the hooking of Java classes, which is made possible due to instrumentation.
  - We report experimental results on three different well-known mobile malware benchmark datasets that are directly compared with state-of-the-art methods under the same settings. The performance of the approaches presented in this work is quite competitive to the best results reported so far for these corpora,

[6] V. Kouliaridis, G. Kambourakis, D. Geneiatakis, N. Potha. Two Anatomists Are Better than One—Dual-Level Android Malware Detection. Symmetry 2020, 12, 1128, doi: 10.3390/sym12071128.

demonstrating that the proposed methods can be an efficient and effective alternative toward more sophisticated malware detection systems.

  - We propose an ensemble approach by averaging the output of all base models for each malware instance separately. The combination of all base models achieves the best average results across all three data sets examined. This demonstrates that ensembles of classifiers based on multiple, possibly heterogeneous models, can further improve the performance of individual base classifiers.

- An Extrinsic Random-based Ensemble Approach for Malware Detection[7].

  - We adopt predefined categories of external malware and benign instances and propose a more sophisticated extrinsic ensemble approach, which provides a positive or negative answer by averaging the output of the base models for each test instance separately. Again, it is demonstrated that ensemble models can further improve the performance of each individual base classifier.
  - We examine the effect of external instances when an ensemble malware detection method is provided combining different sizes and types of external instances. It is demonstrated that ensembles based on a larger and possibly homogeneous size of external instances are exceptionally effective alternative to ensembles included smaller sizes and feasibly more heterogeneous external instances.
  - We report experimental results on contemporary benchmark datasets and directly compare them against state-of-the-art methods under the same settings. The performance of the method presented in this study is quite competitive to the best results reported so far for these datasets, exhibiting that an extrinsic ensemble method is much more reliable and effective for the malware detection task.

1.3 Thesis Structure

The next chapter presents the key components of the Android's security model, as well as background information on mobile malware types and the different kinds of mobile

[7]Nektaria Potha, V. Kouliaridis, G. Kambourakis. An extrinsic random-based ensemble approach for android malware detection. Connection Science, doi: 10.1080/09540091.2020.1853056.

malware detection techniques. Furthermore, it offers a succinct overview of the diverse machine learning algorithms, the relevant evaluation metrics, and possible performance, in terms of detection accuracy, enhancing techniques.

Chapter 3 details on the design and implementation of "Mal-warehouse", a data collection-as-a-service of mobile malware behavioral patterns.

Chapter 4 presents results on the analysis of a critical mass of mobile apps from the hitherto most contemporary and prevailing datasets. Moreover, it provides insight on the importance of app classification features pertaining to permissions and intents, by ranking these feature categories using the Information Gain algorithm.

Chapter 5 presents "Androtomist", a novel open source tool capable of applying both static and dynamic analysis of apps on the Android platform. Unlike similar hybrid solutions, Androtomist capitalizes on a wealth of features stemming from static analysis along with rigorous dynamic instrumentation to dissect apps and decide if they are benign or not.

Chapter 6 examines the potential contribution of two known dimensionality reduction transformations namely, Principal Component Analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) in mobile malware detection.

Chapter 7 introduces a sophisticated Extrinsic Random-based Ensemble (ERBE) method where in a predetermined set of repetitions, a subset of external instances (either malware or benign) as well as classification features are randomly selected, and an aggregation function is adopted to combine the output of all base classification models for each test case separately.

Chapter 8 attempts to schematize the so far ML-powered malware detection approaches and techniques by organizing them under four axes, namely, the age of the selected dataset, the analysis type used, the employed ML techniques, and the chosen performance metrics. Moreover, based on these axes, this chapter introduces a converging scheme which can guide future Android malware detection techniques and provide a solid baseline to machine learning practices in this field.

Chapter 9 offers a discussion on the results and contributions of this PhD thesis. The same chapter elaborates on future research directions as well.

Chapter 2

Background

2.1 Android's Security Model

This section presents a brief discussion on the major components of the Android's security model. However, it primarily focuses on the permissions and inter-process communication, which are a key part of this thesis.

2.1.1 Application Sandboxing

One of the most important security features of Android is app sandboxing, also known as app isolation. It works by taking advantage of the Linux OS user-based protection to identify and isolate app resources. To do so, Android assigns a unique user ID (UID) to each Android app and runs it in its own process. It then executes each app in a dedicated process (process-level isolation) with that UID. The Application Sandbox is located in the kernel, thus it extends to both native code and OS apps. Specifically, all of the software running after loading the kernel, run within the app sandbox.

2.1.2 Permissions

App permissions support the user's privacy by protecting access restricted data, such as the user's contacts, and restricting actions, such as making a phone call. Android categorizes permissions into different types, including install-time permissions, runtime permissions, and special permissions [14].

- Install-time permissions give an app limited access to restricted data and actions. The system automatically grants an app install-time permissions upon installation.

- Runtime permissions, also known as "dangerous" permissions, grant an app additional access to restricted data and restricted actions. Therefore, an app must request runtime permissions before it can access the restricted data or perform restricted actions. Since API level 23, apps which use runtime permissions prompt users to accept permissions at runtime rather than at installation. Since API level 29, users are prompted by the runtime permissions dialog to either always allow, allow while in use, or deny permissions.

  So far, there are 30 dangerous permissions listed in the Android API, namely, API version 30 [14]:

  1. ACCEPT_HANDOVER
  2. ACCESS_BACKGROUND_LOCATION
  3. ACCESS_COARSE_LOCATION
  4. ACCESS_FINE_LOCATION
  5. ACCESS_MEDIA_LOCATION
  6. ACTIVITY_RECOGNITION
  7. ADD_VOICEMAIL
  8. ANSWER_PHONE_CALLS
  9. BODY_SENSORS
  10. CALL_PHONE
  11. CAMERA
  12. GET_ACCOUNTS
  13. PROCESS_OUTGOING_CALLS
  14. READ_CALENDAR
  15. READ_CALL_LOG
  16. READ_CONTACTS
  17. READ_EXTERNAL_STORAGE
  18. READ_PHONE_NUMBERS
  19. READ_PHONE_STATE
  20. READ_SMS
  21. RECEIVE_MMS
  22. RECEIVE_SMS
  23. RECEIVE_WAP_PUSH
  24. RECORD_AUDIO
  25. SEND_SMS
  26. USE_SIP
  27. WRITE_CALENDAR
  28. WRITE_CALL_LOG
  29. WRITE_CONTACTS
  30. WRITE_EXTERNAL_STORAGE

- Special permissions correspond to particular app operations. Only the platform and original equipment manufacturers (OEMs) can define special permissions.

2.1.3 Inter-Process Communication

Process isolation improves the security of the apps which are running on the device. Nevertheless, there are some cases in which one process may need to provide useful service to other processes. Android offers a mechanism for inter-process communication (IPC) using remote procedure calls (RPCs), in which a method is called by an activity or other app component, but executed in another process, with any result returned back to the caller [15]. An Intent provides a facility for performing runtime binding between the code in different apps.

Specifically, Intents carry information that the Android OS uses to select the appropriate component to use, such as the exact component name or component category to receive the intent. They also carry information which specifies the generic action to perform, such as view or pick. For example, if an app wants to view a web page, it expresses its "Intent" to view the URL by creating an Intent instance and handing it off to the system. The system locates some other piece of code, in this case the web browser, that knows how to handle that Intent, and runs it. There are two forms of intents used, namely explicit and implicit. The former specify a component which provides the exact class to be run. Implicit Intents on the other hand do not directly specify a software component; instead, they must include enough information for the system to determine which of the available components is best to run for that intent [16]. Listing 2.1 displays a piece of the manifest file of an app which uses intents.

```
<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

LISTING 2.1: Example of an intents declaration in an app's manifest file

2.1.4 SELinux

Android uses the Security-Enhanced Linux (SELinux) to enforce mandatory access control (MAC) over all processes, even processes running with root/superuser privileges. SELinux operates on the principle of default denial. In other words, anything not explicitly allowed is denied. SELinux can operate in two modes: "permissive", in which permission denials are logged but not enforced, and "enforcing", in which permissions denials are both logged and enforced. Android operates SELinux in enforcing mode [17].

2.1.4.1 Seccomp filter

In addition to SELinux, Android uses *Seccomp* [18] to further restrict access to the kernel by blocking access to certain system calls. As of Android 7.0, Seccomp was applied to processes in the media frameworks. As of Android 8.0, a seccomp filter is installed into *zygote*, the process from which all apps are derived. It blocks access to certain system calls, which have been implicated in some security attacks. It also blocks the key control system calls, which are not useful to apps [19].

2.1.5 Application Signing

Android requires that all apps be signed with a certificate before they are installed on a device or updated. Apps which attempt to install without being signed are rejected by the package installer of the Android device. One important advantage of code signing is that the OS can check if an installed app is coming from the same source upon their update by comparing its signing certificates.

2.1.6 Trusty Trusted Execution Environment (TEE)

Trusty is a secure OS kernel derived from Little Kernel, which facilitates a Trusted Execution Environment (TEE) for the Android OS. Trusty runs on the same processor

as the Android OS, but it is isolated from the rest of the system by both hardware and software. Trusty and Android run parallel to each other. Trusty's isolation aims to protect itself from malware or potential vulnerabilities exploited. There are many potential uses for a TEE such as mobile payments, multi-factor authentication, device reset protection, secure PIN and fingerprint processing [20].

2.1.7 Verified Boot

Verified Boot ensures that all executed code comes from a trusted source, such as the device's OEM, rather than from an attacker or corruption. It works by establishing a full chain of trust, starting from a hardware-protected root of trust to the bootloader, to the boot partition and other verified partitions [21].

2.2 Mobile Malware

Despite Android's well-thought and engineered security model, its great market share as well as its openness have provided an interesting opportunity for attackers. Over the years, Android malware have spread and evolved more than in any other platform [3, 4]. Any type of software segments that lead to a device usage without the owner's awareness can be characterized as malware. There exist many different types of mobile malware, including Trojans, worms [22], botnets [23], spyware and ransomware. Each of them follows a diverse behavior pattern, but all of them are in scope of this thesis. The latest mobile malware types are summarized in the following subsections.

2.2.1 Trojans

Malware can come packaged as a Trojan, a piece of software that appears to provide some functionalities, but instead, contains a malicious functionality. Namely, its basic feature is the creation of backdoors, so as to provide entry points to the remote attackers. Contrary to viruses and worms, a trojan does not create replications.

When installed on a device, Trojans may steal user's confidential information without the user's knowledge. They can also usurp browsing history, messages, contacts, and even banking credentials. According to the Kaspersky Mobile Evolution Report [3],

more than 156K mobile banking Trojans, such as BankBot [24], have being detected in 2020. End-user devices get infected by fake updates, email, and SMS phishing.

2.2.2 Worms

A worm is a program that makes copies of itself, typically from one device to another, using different transport mechanisms through an existing network without the user's intervention. To do so, contrariwise to computer viruses, worms are equipped with networking capabilities. The entities that can be affected by worms are web sites, internet traffic and sometimes even the victim's device can be remotely seized and operated by the programmer of the worm.

2.2.3 Rootkit

A Rootkit is a type of program designed to hide itself in the compromised devices and provide continued privileged access to a device. A rootkit was originally a collection of tools that provided administrator-level access to a device or network [25].

2.2.4 Botnet

The word botnet is made up of two words: "bot" and "net". The former term is short for Robot and the latter for network. As shown in Figure 2.1 a botnet is a set of devices linked together with the ability to be remotely controlled by the so called command and control (C&C) server. Attackers often use botnets to launch large-scale attacks, such as a distributed denial of service attack (DDoS), orchestrate massive spam mail campaigns, conduct BitCoin mining, or to illegally collect information.

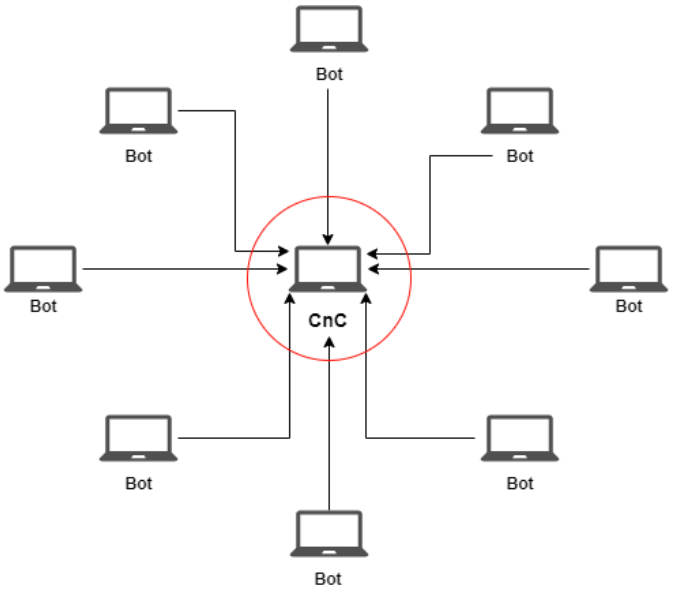

FIGURE 2.1: Typical centralized botnet architecture

2.2.5 Cryptocurrency Mining

While not as sophisticated as their desktop counterparts, mobile malware related to Bitcoin mining are increasing every year [26, 4]. According to Kaspersky Security Network [27], most malware of this type was hidden within popular apps, that were secretly mining cryptocurrency while showing soccer videos.

2.2.6 Spyware

The purpose of spyware is not dedicated to causing direct system damage, but rather to collecting information to be used for advertisement by third party entities, who keep their identity secret [28]. Spyware apps may acquire, store and transfer personal information (credentials, credit card numbers, phone numbers, e-mail addresses etc.), but are also capable of sniffing network traffic and associated users' activity, generating pop-up messages, and altering default web pages in browsers. When installed on a device, this piece of software spies the user's activity and forwards it to remote servers. The spyware does not display any app icon in the Application Launcher, and could therefore stay unnoticed by users. One way to detect this malware is by observing the Application Manager, where a so-called "Android System Message" app shows. Nickispy [29] is a characteristic example of this type of malware.

2.2.7 Ransomware

As illustrated in Figure 2.2, this type of malware, prevents users from accessing the data on their devices by encrypting them, until a considerable ransom amount, typically in some cryptocurrency, is paid. In 2020 more than 20K mobile ransomware have been detected [3]. Ransomware present an alarming threat to users and organizations who must choose to either pay the ransom or end up with possibly valuable encrypted data.

FIGURE 2.2: Typical ransomware operation

2.2.8 Hybrid

This type of mobile malware is very common nowadays. For example, *Android/LokiBot* [26] combines the functionality of a banking trojan with crypto ransomware. It can encrypt files, but it might also send bogus notifications in an attempt to trick users into logging in to their bank account. During 2018, Android/LokiBot has targeted more than 100 financial institutions and kit sales on the dark web generated a profit of up to 2$ million [26].

2.3 Mobile Malware Penetration Techniques

This section succinctly presents the four most well-known malware penetration techniques used by mobile malware. Note however that such techniques is not the focus of the current PhD thesis.

2.3.1 Repackaging

Repackaging is often used to disassemble an app for appending malicious content and then reassembling it. This process is done by reverse engineering tools, and as shown in Figure 2.3, the result is a cloned app which operates as the original, but embeds hidden malicious operations. Malware authors repackage the popular apps of Android official market, namely Google Play [30], and distribute them on other less monitored and policed third party app-stores. During repackaging, malicious authors change the signature of repackaged app and so the app seems new to virus detectors. The main steps involved in app repackaging are:

- Download the popular free/paid app from the popular app-store(s).
- Disassemble the app with a disassembler such as *apktool* [31].
- Generate a malicious payload in Java and convert it to bytecode using the *dx* (part of the Android SDK build tools [32]) tool.
- Add the malware payload into the benign app. Modify the AndroidManifest.xml and/or resources, if required.
- Assemble the modified source again using apktool [31].
- Distribute the repackaged app by self-signing with another certificate to the less monitored third party app market.

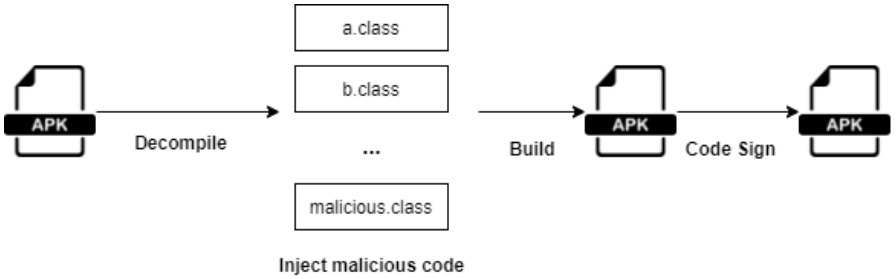

FIGURE 2.3: General APK repacking procedure

Furthermore, repacking and repackaging techniques can be used to generate a large number of malware variants. It can also be used to produce a number of unseen variants of an already known malware. As the signature of each malware variant varies, the commercial anti-malware cannot detect the mutated malware. Repackaging is a big threat

as it can pollute the app distribution market places and also hurts the reputation of the original third party developer. On top of everything else, malware writers can divert advertisement revenues by replacing the advertisements of the original developers. The *AndroRAT APK Binder* [33] tool repackages and generates a trojanized version of a popular and legitimate app, equipping it with Remote Access Trojan (RAT) functionality. That is, the adversary can remotely force the infected device to send SMS messages, make voice calls, access the device location, record video and/or audio, and access the device files using the hidden remote access service.

2.3.2 Drive by download

This technique refers to an unintentional download of malware in the background. As shown in Figure 2.4, drive by download attacks occur when an unsuspected user visits a nefarious website, which in turn injects malware into the victim's device without the user's knowledge.

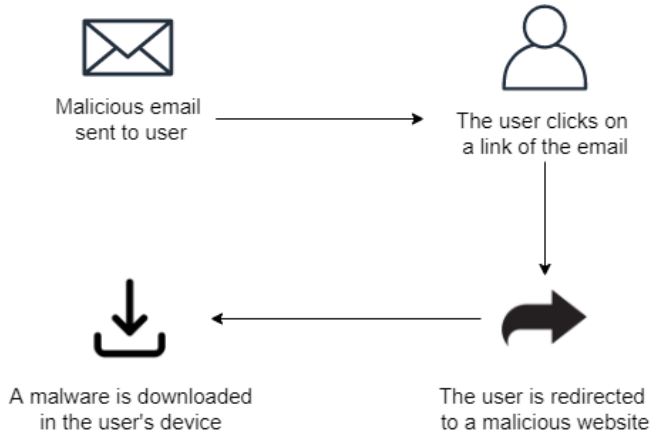

FIGURE 2.4: Drive by download method

2.3.3 Dynamic payloads

An app can also incorporate malicious payload as an executable APK/jar in encrypted or plain format within its APK resources [34]. Once installed on the device, the app decrypts the payload. If the malicious payload is a jar file, the malware loads the DexClassLoader API and executes dynamic code. However, it can trick the user to install the embedded APK by disguising as an important update. The app can execute native binaries using the Runtime.exec API, an equivalent of Linux fork()/exec(). Some

malware, instead of embedding payload as a resource, download the malicious content from remote servers dynamically, and thus are infeasible to detect through static analysis methods.

2.3.4 Stealthy malware techniques

Android device malware scanners cannot perform deep analysis because of the availability of limited resources such as memory and battery. Malware writers exploit these hardware constraints limiting the anti-malware defences, and use stealth techniques, including code encryption, key permutations, dynamic loading, reflection code, and native code execution to effectively attack the victim's device.

2.4 Mobile Malware Detection Techniques

2.4.1 Introduction

Until now, mobile malware detection techniques has been surveyed by several works. Yan et al. [7] compare mobile malware detection methods based on several different evaluation criteria and metrics but mainly focus on the Android OS. La Polla et al. [35] survey the evolution of mobile threats, vulnerabilities and intrusion detection systems over the period 2004-2011. While this is one of the most comprehensive works on the topic, by now it misses current developments. Gandotra et al. [36] examine techniques for analyzing and classifying mobile malware. Furthermore, they list several works for each detection technique. Nevertheless, they do not discuss the effectiveness of each work based on their evaluation results. Yan et al. [37] report on mobile malware categories, taxonomy and attack vectors. Furthermore, they provide a comparison of dynamic mobile malware detection methods and discuss future research trends.

This section aims to provide state-of-the-art information on current mobile malware trends. Furthermore, it offers a comprehensive overview of the different approaches to mobile malware detection, in an effort to understand their detection method, discuss their evaluation results, and possibly categorize each contribution under a novel classification scheme.

2.4.2 Mobile malware detection classification

Mobile malware detection methods serve as countermeasures for the existing malware. However, their functionality differs according to variables related to the focus of each method. The main categorization vector in malware detection methods is related to the detection type. As shown in Figure 2.5 the two main detection techniques are the signature-based and anomaly-based [6]. In regards to the analysis part, the static and dynamic methods are used.

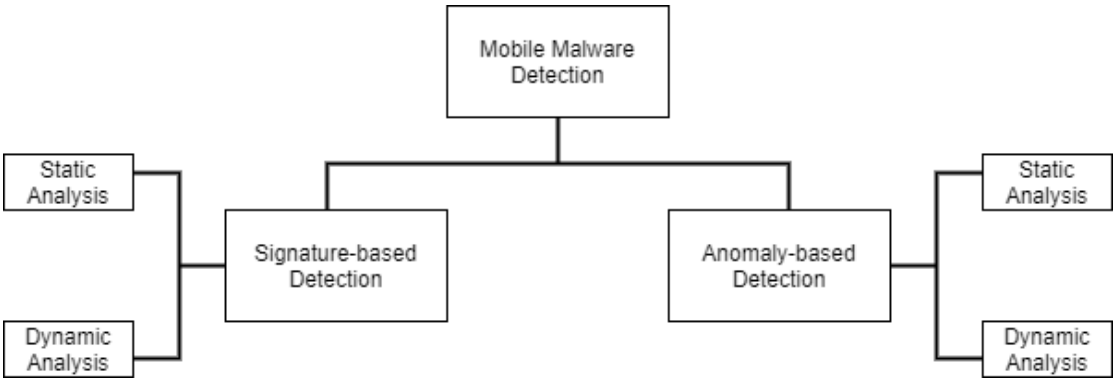

FIGURE 2.5: Mobile malware detection techniques

2.4.2.1 Signature-based detection

Generally, signature-based detection (also known as misuse detection) relies on known signatures, that is, detection rules aiming to discern between benign and malicious pieces of code. More specifically, a signature represents a collection of features which can be used to model the malicious behavior of malware. Ideally, a signature should be able to identify any malware exhibiting the malicious behavior specified by the signature [38]. While signature-based detection systems are able to identify previously encountered malicious software and may have a high degree of portability between platforms, they miss to recognize novel instances of malware or variations of known ones. Thus, the detection ability of a misuse detection system primarily depends on the newness of the detection rules the system has been configured with.

In static analysis, the acquisition of signatures occurs during the decomposition and analysis of the malware source code. On the other hand, signatures in dynamic analysis are acquired after the execution of the malicious code. More specifically, information is

gathered during app execution to decide its maliciousness. This is done using preconfigured and predetermined attack patterns that are given beforehand by experts to build a signature database or a pattern set [6]. Finally, hybrid analysis incorporates both static and dynamic signature-based detection.

2.4.2.2 Anomaly-based detection

Anomaly-based methods use a less strict approach. This is done by observing normal behavior of a device for a certain amount of time and using the metrics of that normal model as a comparison vector to deviant behavior. In regards to the analysis part, the static and dynamic methods are used. The static analysis examines an app before installation by dissecting it, whereas the dynamic performs the analysis during the app execution, by gathering data such as system calls and events. Either in the static or the dynamic version, anomaly-based detection techniques comprise two parts, the training and detection phase. During the former, a non-infected system is operating normally and this procedure is observed and tracked. On the other hand, the detection phase serves as a testing period, when deviations from the training period model are considered anomalies. A key advantage of anomaly-based detection is its ability to detect zero-day attacks [38]. Zero-day attacks are attacks that are previously unknown to the malware detection system.

Static analysis does not require the execution of the malicious payload. Its function is to check the code of the potentially malicious app for specific snippets of code, suspicious functionality, and other behavioral traits. In dynamic analysis, the training and detection phases happen during the execution of the app. Apart from the capability of detecting unknown malware, this trait also enables the detection of zero-day attacks. However, as already mentioned before, the false positive rate issues are rather intense. In order to soften this incident, accurate normal behavioral models have to be constructed during the training sessions. Hybrid analysis is rather popular among anomaly-based detection solutions [6].

2.4.3 Survey of works

Mobile malware detection methods serve as countermeasures for the existing malware. However, their functionality differs according to variables related to the focus of each method. This section aims to classify the existing research works, according to the detection techniques reported by the authors, and review their functionality and effectiveness. This survey focuses on research papers dated no more than 10 years ago. The considered works have been categorized in the following subsections in chronological order.

2.4.3.1 Signature-based detection

Enck et al. [39] proposed a security service for the Android Operating System (OS), named *Kirin*. *Kirin* certificates an app at install time, using a set of security rules, which are templates designed to match suspicious properties in apps' security configuration. More specifically, after the installer extracts security configuration from the package manifest, *Kirin* evaluates the configuration against a collection of predefined security rules.

Chen et al. [40] proposed a detection approach which identifies threat patterns. It analyzes the function invocation, as well as the data flow to detect malicious behaviors in Android devices. More specifically, their scheme uses reverse engineering to recreate the source code and class files from each app and builds the corresponding API invocation and dependency graphs. Based on these two graphs, their system can detect threat patterns, which may reveal whether an app attempts to access confidential information or perform any illegal access. Their experiments show 91.6% detection rate over 252 malicious samples.

Papamartzivanos et al. [11] proposed a host and cloud-based system that operates under a crowdsourcing logic. Their system includes 3 main services, namely privacy-flow tracking, crowdsourcing, and detection and reaction against privacy violations. The client communicates with the cloud services via a TLS connection and is relieved from resource demanding tasks. More specifically, the client consists of 3 modules, namely privacy inspection, response, and event sensor. The cloud side also consists of 3 modules, namely crowdsourcing, detection, and hook update.

2.4.3.2 Anomaly-based detection

Wu et al. [41] implemented *DroidMat*, which provides malware detection through manifest and API call tracing. The authors extract app information from its manifest file and disassembly codes. More specifically, they collect information from the app's manifest file such as "intent", which is an abstract description of an operation to be performed, and Inter-Component Communications (ICC) and API calls related to permissions. The authors collected 238 Android malware and 1,500 benign apps to test *DroidMat* and their results show an up to 97.87% accuracy rate in detecting mobile malware.

An approach which analyses an app's permissions to detect malware in Android *(PUMA)*, was presented by Sanz et al. [42]. The authors gathered 1,811 benign Android apps, as well as 4,301 malware samples. The authors state that they observed several differences in permissions usage by malware apps. More specifically, they noticed that malware often requires only one permission, while benign apps usually ask for 2 or 3 permissions. The authors used several machine learning techniques for malware detection, including SimpleLogistic, NaiveBayes, BayesNet, SMO, IBK, J48, RandomTree and RandomForest. Finally, they performed analysis on the extracted permissions from mobile apps and observed a detection accuracy of 92%.

Peiravian et al. [43] proposed the combination of permissions and API calls and the use of machine learning methods to detect malicious Android apps. Their framework consists of 4 components. The first one decompresses the APK file of an app to extract the manifest and class files. The second characterizes apps based on the requested permissions and API calls. The third one carries out feature extraction on the permissions and API calls. The latter employs the training of the classification models from the collected data. The authors state that during the evaluation tests, the proposed method achieved a promising detection rate, while holding precision up to 94.9%.

In an attempt to address the issue of removing malicious apps from mobile app markets, Chakradeo et al. [44] proposed an approach for market-scale mobile malware analysis *(MAST)*. *MAST* analyzes attributes extracted from the app package and uses Multiple Correspondence Analysis (MCA) to measure the correlation between multiple categorical data. Furthermore, only easily obtained attributes are extracted to keep *MAST* less costly than deep analysis. These attributes are permissions included in the manifest file, intent filters and pre-agreed upon action strings (also included in the manifest file),

native libraries inside the source code and malicious payloads hidden in zip files inside the app package. During the training phase 15,000 apps from Google Play [30] and a dataset of 732 known-malicious apps were used to train *MAST*. According to the authors, *MAST* triage processes mobile app markets in less than a quarter of the time required to perform signature detection.

Liang et al. [45] proposed a permission combination-based scheme for Android mobile malware detection. The authors collected permission combinations declared in the app manifest file, which are requested frequently by mobile malware, but rarely by benign apps. More specifically, a tool called *k-map* was developed in order to find permission combinations extracted from the app manifest file, which are frequently used by malware apps. Moreover, they calculated the permission request frequencies out of the permission combinations extracted. Their experiments showed that the system was able to detect malware with low false positive and negative rates, that is, malware detection rate up to 96%, and the benign app recognition rate was up to 88% [45].

Canfora et al. [46] proposed mobile malware detection using op-code frequency histograms. Their approach classifies malware by focusing on the number of occurrences of a specific group of op-codes. More specifically, the authors used a detection technique, which uses a features vector obtained from 8 Dalvik op-codes. These op-codes are usually used to alter the app's control flow. After training the classifier, the authors tested their proposed method to conclude that these features are able to classify a mobile app as trusted or malicious with a precision rate of 93.9%.

Yusof et al. [47] proposed a mobile botnet classification based on permissions and API calls. During the training phase, 5,560 malware from 179 different mobile malware families were collected. The authors examined 50 Android botnet samples using static analysis and reverse engineering to extract the 16 most important permissions and 31 API Calls from the botnet samples. Finally, they chose 800 random apps from Google Play [30] to test their classification using Naive Bayes, K-nearest Neighbour, Random Forest, and Support Vector Machine algorithms. Their results achieved 99.4% detection rate and 16.1% false positive rate.

Li et al. [48] proposed *SIGPID*, a malware detection system based on permission usage analysis on the Android platform. To test their detection model, the authors collected 3 different datasets which contain 2,650, 5,494 and 54,694 malware apps respectively.

Their detection model uses 22 out of 135 permissions to improve the runtime performance by 85.6%. Finally, they used machine learning algorithms to evaluate their results, including RandomForest, PART, FT, RotationForest, RandomCommittee, and SVM, and achieved a detection rate of 93.62%.

Tao et al. [49] proposed *MalPat*, an automated malware detection system which scans for malicious patterns in Android apps. During the training phase, the authors were able to aquire hidden patterns from malware and extract APIs that are widely used in Android malware. The authors collected 31,185 benign apps and 15,336 malware samples and extracted features from the source code of decompiled files. To evaluate *MalPat*, the authors followed a repeated process, in which they randomly selected a percentage of both malicious and benign datasets as the training set, and the remaining part is regarded as the testing set. The average of their results show that *MalPat* can detect malware with 98.24% F1 score.

Shen et al. [50] proposed a malware detection approach based on information flow analysis. The authors proposed complex-flow as a new representation schema for information flows. According to the authors, complex-flow is a set of simple flows that share a common portion of code. For example, if an app is able to read contacts, store them and then send them over the Internet, then these two flows would be (contact, storage) and (contact, network). The authors state that their approach can detect if an information flow is malicious or not based on the app's behavior along the flow. When a new app is installed their system compares its behavior patterns (obtained from the complex-flows representation of the app) to decide whether it is more similar to benign or malicious apps from the training set using two-class SVM classification. During the evaluation process, the authors used 4 different data sets, totaling 8,598 apps, to test the precision of their detection approach.

Shabtai et al. [51] presented a system for detecting meaningful deviations in a mobile app's network behavior. The system monitors the running apps to create their "normal" network behavior. It is then able to detect deviations from the learned patterns. According to the authors, their main goal was "to learn user-specific network traffic patterns for each app and determine if meaningful changes occur". For this reason, semi-supervised machine learning methods were used to create the normal behavioral patterns and to detect deviations from the app's expected behavior.

Damopoulos et al. [52] proposed a tool which dynamically analyzes iOS apps in terms of method invocation. The authors designed and implemented an automated malware analyzer and detector for the iOS platform, namely *iDMA*. *iDMA* is able to generate exploitable results, which can be used to trace app's behavior to decide if it contains malicious code. Also, Damopoulos et al. [52] proposed an IDS framework that supports both host- and cloud-based protection mechanisms. Their framework employs diverse anomaly-based mechanisms. To evaluate their architecture, the authors developed a proof-of-concept implementation of the framework, equipped with 4 smartphone detection mechanisms. "The first two detection mechanisms, namely SMS Profiler and *iDMA*, aim to detect the illegitimate use of system services and identify unknown malware. The other two, coined iTL and Touchstroke, can provide (post) authentication to ensure the legitimacy of the current user" [52].

Jang et al. [53] presented Andro-AutoPsy, an anti-malware system based on similarity matching of malware information. During the training phase, the authors gathered malware-centric and malware creator-centric information from anti-virus technical reports, malware repositories, community sites and web crawling. The authors chose 5 footprints as features: "the serial number of a certificate, malicious API sequence, permission distribution (critical permission set, likelihood ratio), intent and the intersection of the usage of system commands and the existence of forged files" [53]. Andro-Autopsy consists of a client app running on the device and a remote server. The client app sends the app package file (.apk) to the remote server. The latter entity then analyzes the app and decides whether it is malicious or not, based on integrated footprints. The authors state that Andro-AutoPsy "successfully detected and classified malware samples into similar subgroups by exploiting the profiles extracted from integrated footprints" [53], while it is able to detect zero-day exploits at the same time. Furthermore, Andro-AutoPsy allows anti-virus vendors to conduct similarity matching on previously detected samples.

Chen et al. [54] aimed to combine network traffic analysis with machine learning methods to identify malicious network behavior in highly imbalanced traffic. The authors captured traffic from over 5,560 mobile malware samples. Furthermore, they designed a tool to convert mobile traffic packets into traffic flows. According to the authors, the accuracy rate of the machine learning classifiers can reach up to 99.9%. However, the performance of the classifiers declines when the imbalanced problem gets worse.

As discussed further in Chapter 3, Kouliaridis et al. [55] proposed *Mal-warehouse*, an open-source tool performing data collection-as-a-service for Android malware behavioral patterns. Specifically, the authors collected 14 malware samples to analyze their effects on the Android platform. The authors developed an open source tool called *"MIET"*, which extracts usage information, over a period of time, from the Android device for each malware installed on the device. Finally, *Mal-warehouse* is enhanced with a detection module, which the authors evaluated via the use of machine learning techniques.

Wang et al. [56] proposed a method which combines analysis of network traffic with the c4.5 machine learning algorithm which according to the authors is capable of identifying Android malware with high accuracy. During the evaluation process the authors tested their model with 8,312 benign apps and 5,560 malware samples. Furthermore, their results show that the proposed model performs better than state-of-the-art approaches. Finally, when combining two detection mechanisms, it achieves a detection rate of 97.89%.

Alam et al. [57] proposed *DroidNative* for the detection of both bytecode and native code Android malware. According to the authors, *DroidNative* is the first scheme to build cross-platform (x86 and ARM) semantic-based signatures for Android and operates at the native code level. When apps are analyzed, bytecode components are passed to an Android Runtime (ART) [58] compiler to produce a native binary. The binary code is disassembled and translated into Malware Analysis Intermediate Language (MAIL) code. After MAIL code is generated, *DroidNative* operates in two phases, training and testing. To evaluate *DroidNative*, the authors performed a series of tests with over 5,490 Android apps. Their results demonstrated a detection rate of 93.57% with a false positive rate of 2.7%. Unfortunately, as with all static analysis detection techniques, *DroidNative* cannot detect compressed or encrypted malicious code.

Fei et al. [59] propose a hybrid approach for mobile malware detection. The authors collect information pertaining to runtime system calls over a set of known malware and benign apps using a dynamic approach. More specifically, they gather system-calling data during runtime by modifying the Android OS source code. Furthermore, they process and analyze the collected information to create malicious patterns and normal patterns from both system calls and sequential system calls. That is, malicious and normal patterns are produced "by calculating the ratio of the average frequency of a

sequential system call in the set of malware and the average frequency of the same sequential system calls in the set of benign apps" [59]. According to the authors, the accuracy rate of their detection approach exceeds 90%.

2.4.4 Discussion and future directions

This section presents a comprehensive comparison of the 22 mobile malware detection approaches surveyed in Section 2.4.3. Figure 2 illustrates the timeline of the research works included in this survey. As already mentioned, papers are dated between 2009 and 2018. Different kinds of geometrical shapes refer to detection classification (e.g., square to static signature-based, trapezium to behavior signature-based, parallelogram to hybrid signature-based, circle to static anomaly based, diamond to dynamic anomaly-based, and hexagon to hybrid anomaly-based). The various works are placed within the diagram in chronological order (top to bottom). Numbers inside them correspond to the matching reference. The letter on the left refers to OS type (A is for Android, I is for iOS), while the letter on the right refers to the detection method. The selection of letters is as close to the first letter of each detection method as possible. Solid lines between two shapes imply influence (of a given work vis-a-vis to another), while dashed ones imply compliance or reference to previous work.

As shown in Figure 2, Enck et al. [39] and Wu et al. [41] had an important impact on the evolution of mobile malware detection. Furthermore, while there is a variation in detection methods used during the previous 8 years, latest contributions lean towards anomaly-based detection. More specifically:

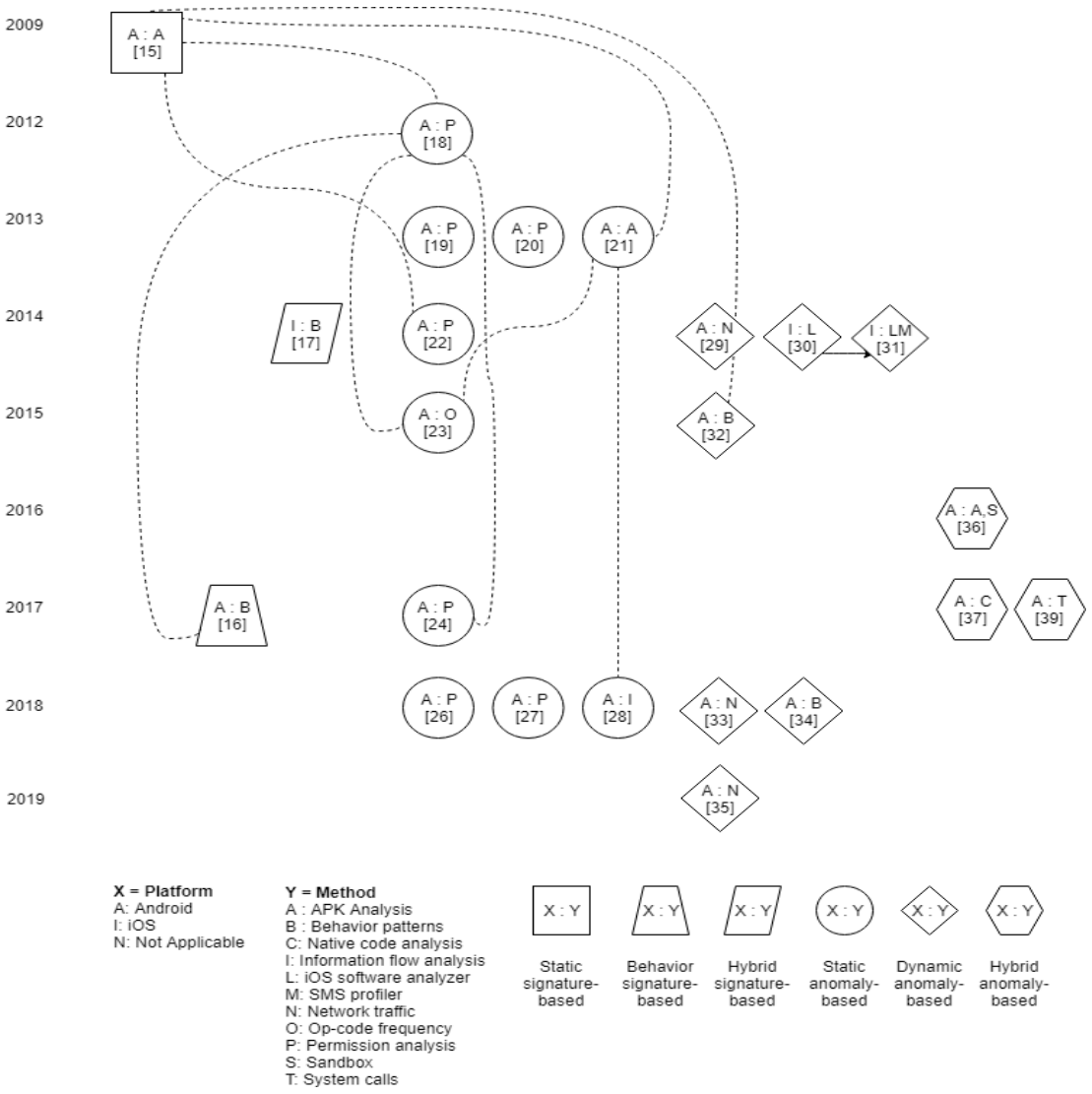

FIGURE 2.6: Malware Detection Techniques in Chronological Order

- At least 9 out of 22 approaches depend on the app's manifest file for their detection process, including [39], [41], [60], [45], [42], [43], [47], [48], [49]. Permission analysis is a popular detection technique among these approaches and it is the most popular detection technique since 2014. According to evaluation results from these contributions, permission-based detection can produce results with high detection rate, but also in some cases high false positive rate (FPR).

- Schemes which utilize native code analysis, such as Alam et al. [57], can produce a high detection rate of up to 93.57% and 2.7% FPR. Unfortunately, this approach cannot detect compressed or encrypted code.

- Complex-flow analysis is a new type of information flow analysis proposed by Shen et al. [50], which according to the authors, produced 86.5% detection rate. Unfortunately, their method cannot detect malicious behavior that is present in native code, which is the case for some of the latest mobile malware.

- Chen et al [54] produced the highest accuracy rate among dynamic anomaly-based approaches. However, while this approach can be highly accurate, it can only detect a subset of malware samples, i.e., those that generate considerable network traffic.

- iOS Detection approaches, such as the work proposed by Damopoulos et al. [61] [52], produce high accuracy results, however these approaches require jailbreaking [62], which could put the device at risk and make the end-user reluctant to employ it.

- Hanlin et al. [60] use sandboxing to safely analyze malware behavior. Although this is a rather promising approach, previous research has shown that some mobile malware are able to detect emulators by looking into several device features [63].

- Some methods combine 2 detection categories into a hybrid solution so as to detect a wide range of malware types. Several of these hybrid solutions carry out mobile malware detection on both the host and cloud, including [60], [52], [11], [59]. While hybrid solutions could offer many benefits, the small amount of reported results from the works included in Section 2.4.3, as well as previous work [64] suggests that these benefits should be subject to careful examination.

Some approaches were rendered as inconclusive during this survey due to doubtful methodologies or metrics. These approaches are:

- Canfora et al. [46] showed a promising accuracy rate of up to 95% using OP-code frequency analysis, but their results are doubtful due to outdated app samples dated from 2012.

  Tao et al. [49] showed high F1 score, but the authors used an outdated Android OS version and malware samples.

- Damopoulos et al. [61] proposed a promising approach for the iOS platform and reported zero FPR, but failed to report on essential data, such as the number of non-malware samples used.

Most of the techniques surveyed in Section 2.4.3 still lack in detecting zero-day malware. Furthermore, with the current sophistication of malware, it is difficult to detect it through traditional rule matching using existing technologies [65, 66]. This may be the reason behind the large number of malicious apps still on the loose in official app stores. Therefore, future research efforts should concentrate on clarifying how to efficiently join detection techniques into hybrid solutions with the purpose of increasing the subset of malware which can be detected, as proposed in previous work [67], but also offer actual detection improvement [64].

2.4.5 Conclusions

This section provides a state-of-the-art survey on the timely topic of mobile malware detection techniques. It does so, by categorizing and succinctly analyzing the various detection schemes as proposed in the literature during the years 2011 to 2018, based on their detection method. It also highlights on the benefits and limitations per category of techniques and per examined scheme where applicable, in an effort to offer a comprehensive overview of this challenging topic. As a side contribution, it elaborates on the existing interrelations between the examined works, which not only reveals the major influencers in this fast evolving research area but also the chief challenges to be addressed in the near future.

2.5 Machine Learning-based Classification

In machine learning, classification is a supervised learning approach in which an algorithm learns from the input data and then uses this learning to classify new observations. In other words, it is the process of predicting the class of given instances [68]. Classes are also called targets or labels. The dataset may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class. Some practical examples of classification problems are: speech recognition,

handwriting recognition, bio metric identification, document classification, and malware detection.

2.5.1 Machine learning classifiers

There are a lot of classification algorithms available but it is not possible to conclude which one is superior to other. It depends on the application and nature of the examined dataset. The best performing classification algorithms used in this thesis are detailed below.

Naive Bayes Classifier (Generative Learning Model) It is a classification technique based on Bayes' Theorem with the assumption of independence among predictors. In other words , a Naive Bayes classifiers assume that the presence of a particular feature in a class is unrelated to the presence of any other feature or that all of these properties have independent contribution to the probability [69]. This family of classifiers is relatively easy to build and particularly useful for very large data sets as it is highly scalable. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Nearest Neighbor The k-nearest-neighbors algorithm is a supervised classification technique that uses proximity as a proxy for 'sameness'. The algorithm takes a bunch of labelled points and uses them to learn how to label other points. To label a new point, it looks at the labelled points closest to that new point (those are its nearest neighbors). Closeness is typically expressed in terms of a dissimilarity function. Once it checks with 'k' number of nearest neighbors, it assigns a label based on whichever label the most of the neighbors have [70]. Figure 2.7 depicts the k-NN operation. Using the geometric distance to decide which is the nearest item may not always be reasonable or even possible: the type of the input may, for example, be text, where it is not clear how the items are drawn in a geometric representation and how distances should be measured. You should therefore choose the distance metric on a case-by-case basis.

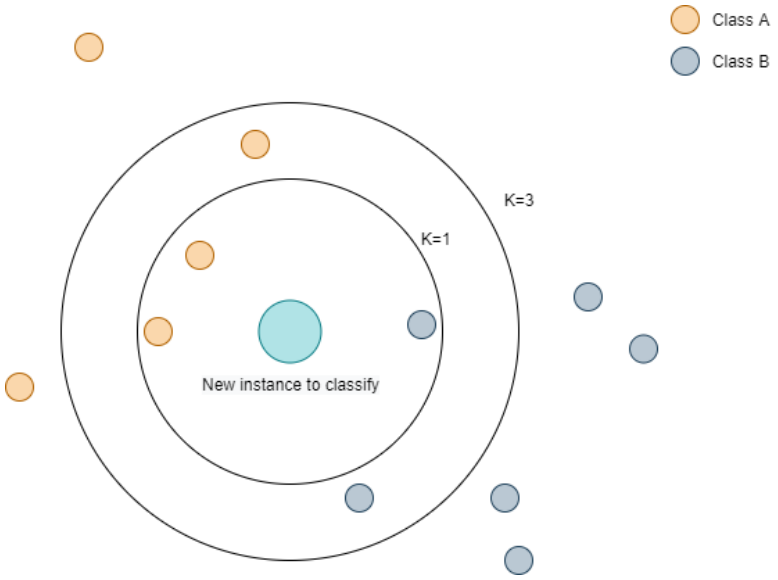

FIGURE 2.7: K-Nearest Neighbor

Logistic Regression (Predictive Learning Model) The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables [71]. This is better than other binary classification like nearest neighbor, since it also explains quantitatively the factors that lead to classification. Overall, Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using both continuous and discrete datasets. Figure 2.8 depicts logistic function's curve which indicates the likelihood of something, such as whether a person is male or female.

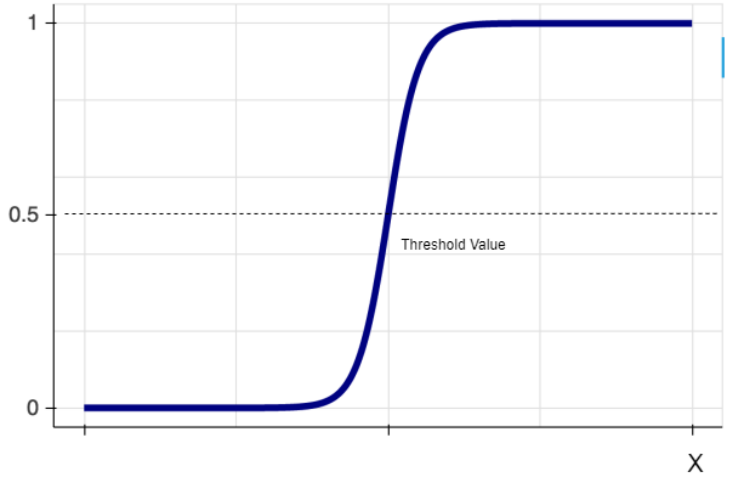

FIGURE 2.8: Logistic Regression

Decision Trees Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed [72]. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches and a leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data. Figure 2.9 depicts a simple example of a decision tree.

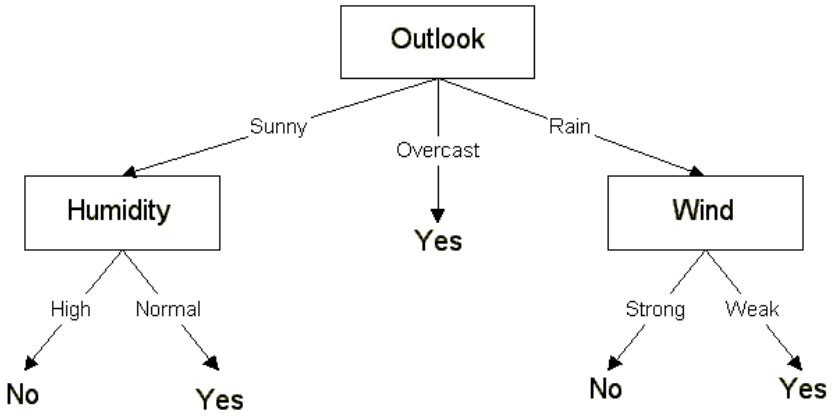

FIGURE 2.9: Decision Tree

Random Forest Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set. Figure 2.10 illustrates the Random Forest's operation.

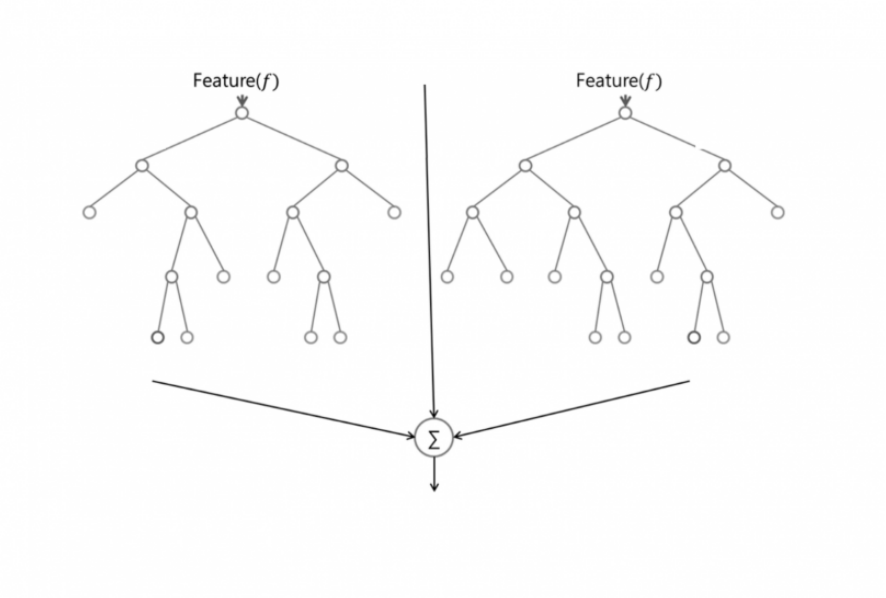

FIGURE 2.10: Random Forest

Neural Network As shown in Figure 2.11, a neural network consists of units (neurons), arranged in layers, which convert an input vector into some output. Each unit takes an input, applies a (often nonlinear) function to it and then passes the output on to the next layer. Generally the networks are defined to be feed-forward: a unit feeds its output to all the units on the next layer, but there is no feedback to the previous layer. Weightings are applied to the signals passing from one unit to another, and it is these weightings which are tuned in the training phase to adapt a neural network to the particular problem at hand.

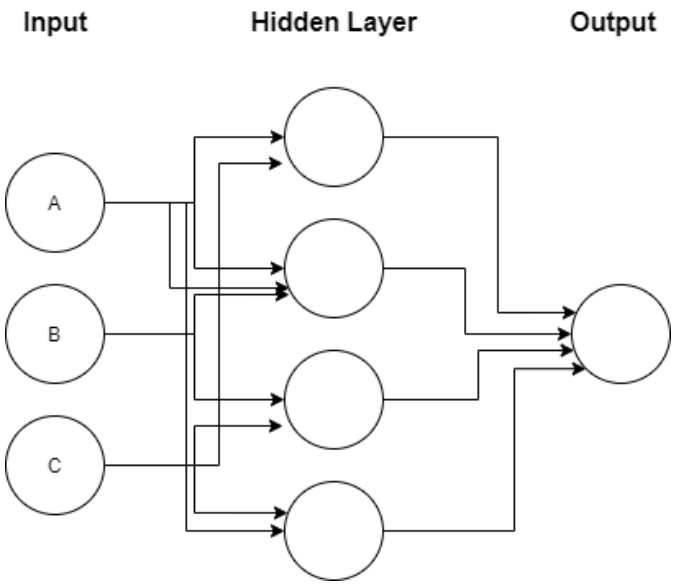

FIGURE 2.11: Neural Network

2.5.2 Evaluation metrics

After training the model the most important part is to evaluate the classifier to verify its applicability. We consider the following classification performance metrics, where TP, TN, FP, and FN represent correspondingly true positives, true negatives, false positives, and false negatives.

- Accuracy $(CA) : \frac{TP}{TP+TN+FP+FN}$. The number of correctly classified patterns over the total number of patterns in the sample.
- Precision $(P) : \frac{TP}{TP+FP}$. The ratio of TP values over the sum of TP and FP.
- Recall $(R) : \frac{TP}{TP+FN}$. The ratio of TP over the sum of TP and FN.
- Area Under Curve (AUC): The higher positive-over-negative value ranking capability of a classifier.
- $F1 : 2\frac{P \times R}{P+R}$.

2.5.2.1 Precision and Recall

Precision is the fraction of relevant instances among the retrieved instances, while recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Precision and Recall are used as a measurement of the relevance.

2.5.2.2 ROC curve (Receiver Operating Characteristics)

ROC curve is used for visual comparison of classification models which shows the trade-off between the true positive rate and the false positive rate. The area under the ROC curve is a measure of the accuracy of the model. When a model is closer to the diagonal, it is less accurate and the model with perfect accuracy will have an area of 1.0

2.5.2.3 Area Under the Curve (AUC)

Generally, AUC quantifies the effectiveness of each examined approach for all possible score thresholds. As a rule, the value of AUC is extracted by examining the ranking of scores rather than their exact values produced when a method is applied to a dataset. Noticeably, the estimation of the AUC measure is based on all possible thresholds. Besides, this evaluation measure does not depend on the distribution of positive and negative samples and therefore can produce more robust results when using non-balanced datasets i.e., datasets with unequal number of instances per class. Figure 2.12 comprises four exaples of a ROC curve and the corresponding AUC value.

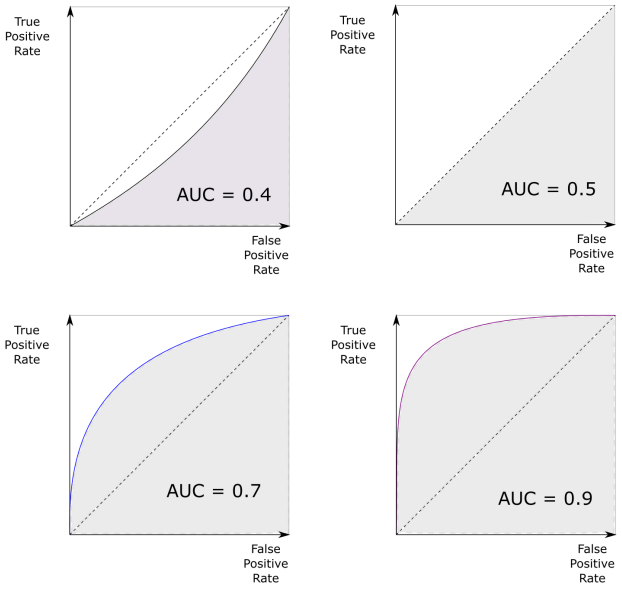

FIGURE 2.12: AUC exaples

2.5.3 Training and validation

In a dataset, a training set is implemented to train a model, while a test (or validation) set is to validate the model. Instances in the training set are excluded from the test (validation) set. Usually, a dataset is divided into a training set and a validation set (or testing set) in each iteration. There are more than one ways one can split a dataset into a training and testing set, the most sampling methods are listed below.

- Cross-validation: Over-fitting[1] is a common problem in machine learning which can occur in most models. k-fold cross-validation can be conducted to verify that the model is not over-fitted. In this method, the data-set is randomly partitioned into k mutually exclusive subsets, each approximately equal size and one is kept for testing while others are used for training. This process is iterated throughout the whole k folds [73].
- Random sampling: Random sampling randomly splits the data into the training and testing set in the given proportion (e.g. 70:30); the whole procedure is repeated for a specified number of times.
- Leave-one-out: Leave-one-out is similar to random sampling, but it holds out one instance at a time, inducing the model from all others and then classifying the held out instances. This method is usually more stable and reliable but also very slow.

2.5.4 Ensemble learning

Ensemble methods use multiple learning algorithms to obtain better classification performance than could be obtained from the use of a single learning algorithm [74]. More specifically, ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking).

[1] overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably

2.5.4.1 Ensemble learning techniques

There are many known ensemble techniques, some of them are easy to implement while others are more complex. The most common ensemble techniques are detailed below.

Taking the minimum or maximum of the results The most simple ensemble learning implementation is taking the minimum or maximum predicted value from all classifiers to make the final prediction.

Taking the mode of the results The mode is a statistical term that refers to the most frequently occurring number found in a set of numbers. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a separate vote. The prediction which we get from the majority of the models is used as the final prediction.

Taking the average of the results In this technique, the average score from all of the classifiers' scores is calculated to make the final prediction.

Taking the weighted average of the results This is an extension of the averaging technique. All models are assigned different weights defining the importance of each model for prediction.

Bagging In Bagging, random samples of the training data are created (sub sets of training data set). Then, we build a model (classifier or Decision tree) for each sample. Finally, results of these multiple models are combined using average or majority voting. As each model is exposed to a different subset of data and we use their collective output at the end, so we are making sure that problem of overfitting is taken care of by not clinging too closely to our training data set. Thus, Bagging helps us to reduce the variance error.

Boosting Boosting is an iterative technique which adjusts the weight of an observation based on the last classification. If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa. Boosting in general decreases the

bias error and builds strong predictive models. Boosting has shown better predictive accuracy than bagging, but it also tends to over-fit the training data as well. Thus, parameter tuning becomes a crucial part of boosting algorithms to make them avoid overfitting. Boosting is a sequential technique in which, the first algorithm is trained on the entire data set and the subsequent algorithms are built by fitting the residuals of the first algorithm, thus giving higher weight to those observations that were poorly predicted by the previous model. It relies on creating a series of weak learners each of which might not be good for the entire data set but is good for some part of the data set. Thus, each model actually boosts the performance of the ensemble.

Stacking Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features. The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous.

2.5.5 Dimensionality reduction

Dimensionality reduction is the process of reducing the dimension of the feature set. As the number of features increases, the model becomes more complex. The more the number of features, the more the chances of overfitting. A machine learning algorithm which is trained on a large number of features, can get increasingly dependent on the data it was trained on. This can lead to overfitted, resulting in poor performance on real data, beating the purpose [75].

Avoiding overfitting is a major motivation for performing dimensionality reduction. The fewer features our training data has, the lesser assumptions our model makes and the simpler it will be. The most important advantages of dimensionality reduction are listed below:

- Removes redundant features and noise.
- By decreasing misleading data the model accuracy improves [76].
- Less dimensions require less computing, thus requiring less training time.

- Less dimensions allow usage of algorithms unfit for a large number of dimensions
- Less data requires less storage space.

Linear dimensionality reduction methods The most common and well known dimensionality reduction methods are the ones that apply linear transformations, like

PCA (Principal Component Analysis): Popularly used for dimensionality reduction in continuous data, PCA rotates and projects data along the direction of increasing variance. As shown in 2.13, the features with the maximum variance are uses as principal components.

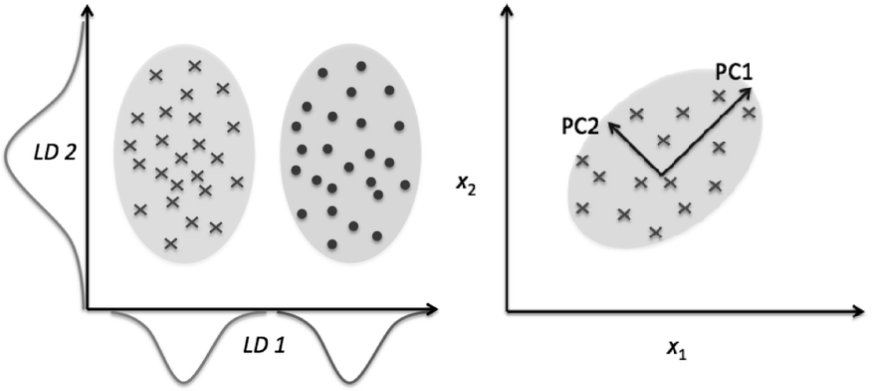

FIGURE 2.13: PCA method

Factor Analysis: a technique that is used to reduce a large number of variables into fewer numbers of factors. The values of observed data are expressed as functions of a number of possible causes in order to find which are the most important. The observations are assumed to be caused by a linear transformation of lower dimensional latent factors and added Gaussian noise.

LDA (Linear Discriminant Analysis): projects data in a way that the class separability is maximised. Examples from same class are put closely together by the projection. Examples from different classes are placed far apart by the projection.

Non-linear dimensionality reduction methods Non-linear transformation methods or manifold learning methods are used when the data does not lie on a linear subspace. It is based on the manifold hypothesis which says that in a high dimensional structure, most relevant information is concentrated in small number of low dimensional

manifolds. If a linear subspace is a flat sheet of paper, then a rolled up sheet of paper is a simple example of a nonlinear manifold. Informally, this is called a Swiss roll, a canonical problem in the field of non-linear dimensionality reduction. Some popular manifold learning methods are:

Multi-dimensional scaling (MDS) : A technique used for analyzing similarity or dissimilarity of data as distances in a geometric spaces. Projects data to a lower dimension such that data points that are close to each other (in terms if Euclidean distance) in the higher dimension are close in the lower dimension as well.

Isometric Feature Mapping (Isomap) : Projects data to a lower dimension while preserving the geodesic distance (rather than Euclidean distance as in MDS). Geodesic distance is the shortest distance between two points on a curve.

Locally Linear Embedding (LLE): Recovers global non-linear structure from linear fits. Each local patch of the manifold can be written as a linear, weighted sum of its neighbours given enough data.

Hessian Eigenmapping (HLLE): Projects data to a lower dimension while preserving the local neighbourhood like LLE but uses the Hessian operator to better achieve this result and hence the name.

Spectral Embedding (Laplacian Eigenmaps): Uses spectral techniques to perform dimensionality reduction by mapping nearby inputs to nearby outputs. It preserves locality rather than local linearity

t-distributed Stochastic Neighbor Embedding (t-SNE): Computes the probability that pairs of data points in the high-dimensional space are related and then chooses a low-dimensional embedding which produce a similar distribution.

Chapter 3

Mal-warehouse: A data collection-as-a-service of mobile malware behavioral patterns

3.1 Introduction

In the last few years, the expanding Android market share has become an increasingly attractive target for malicious attacks. According to a report by F-Secure [77], 99% of all mobile malware targets Android devices. Given the severity of this threat, there is a need for the development of reliable detection solutions. A robust detection solution in such platforms requires a rather detailed knowledge background, where malware behaviour will be tracked and further observed. To this end, this chapter concerns the design and implementation of Mal-warehouse, a data collection-as-a-service of mobile malware behavioral patterns. Firstly, a series of tests is performed on an Android device. The initial measurement takes place after a clean installation of the OS. After installing a different malware per test, CPU, memory and network usage, as well as process and network statistics are measured over time and then stored in a cloud database. The database will become publicly available to the community and will allow for users' contribution of experimental metrics. Up to now, CPU and memory usage were selected for further evaluation, due to their immediate feature availability that required only minor pre-processing. Machine learning algorithms were then employed so as to compare the

aforementioned metrics of each infected session with the clean instance. The results of the evaluation procedure showed that malicious behaviour can be successfully identified in the majority of the cases.

Despite the rather satisfactory results, there were some unavoidable limitations. First, there was a small amount of extracted patterns per spec and per malware, fact that resulted in some poorly calculated metrics. Despite the fact that these metrics were excluded, they can still be used in the future, when more information per malware is collected. Furthermore, the mobile device we employed is running the relatively new Android Nougat (7.0) version. This resulted in unexpected and restricted behavior for some malware, especially for those dating back to 2012-2015. Finally, the device does not support SIM cards, which means that certain malicious activities, such as SMS sending could not be observed during the testing phase.

3.2 Proposed Methodology

For testing our tool, a total of 14 malware samples were collected to analyze and experiment on their effects on a target Android device. In addition, *Mal-warehouse Information Extraction Tool (MIET)*, an open-source tool written in Python 3.4 was developed for automating the process of collecting hardware usage information, such as CPU, memory and network usage, running processes, and network traffic.

For this initial experiment, a NVIDIA Shield K1 tablet running the Nougat (7.0) Android version was used. The device runs on a 2.2 GHz quad-core ARM Cortex A15 CPU and carries 2GB of DDR3 RAM. At first, the aforementioned metrics were collected upon a fresh installation of the OS. All collected malware were then tested on the device and all the metrics were exported using MIET. Our approach demands that we revert the device to its original state, after the installation and testing of each malware. This guarantees the best possible comparison results and simultaneously ensures the forensic soundness of the data. The metrics collected were logged in a behavioral database in the cloud. The overall process is depicted in Fig. 3.1.

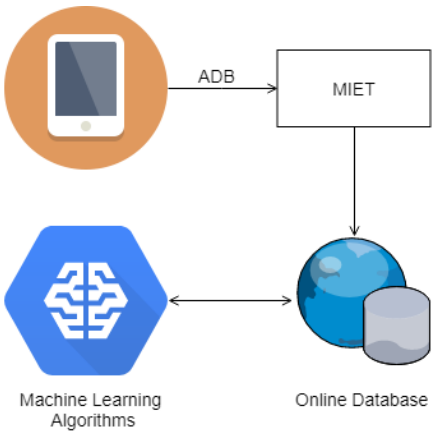

FIGURE 3.1: Mal-warehouse Operation

The aim of the aforementioned testing process is not only to contribute information on how different malware affect devices, but also the fact that further statistics related to hardware usage exported by MIET can be evaluated for detection purposes. This is achieved by using various machine learning algorithms (in our case kNN, Random Forest, SVM, Naive Bayes, AdaBoost) to determine whether a device is infected by malware, based on changes on the device behavior, instead of, say, just scanning for known file hashes. Finally, after each malware was tested, a device image was saved for further analysis and comparison. The MIET interface is depicted in Figure 3.2, where the following operations are available.

- Scan device: Search and list available connected devices.
- Dump CPU info: Extract CPU usage information.
- Dump memory info: Extract memory usage information.
- Dump netstat details: Extract network statistics.
- Dump battery stats: Extract battery statistics.
- Dump process stats: Extract running processes and associated details.
- Run full device scan: This operation lasts for approximately 30 minutes. It triggers 4 iterations, each running 60 seconds after the previous run has finished and executes each action listed above to get a complete device scan over time.

FIGURE 3.2: Mal-warehouse Information Extraction Tool

3.2.1 Data Collection

Device data were collected by using the ADB [78]. ADB is a command-line tool that enables communication between a device and a computer, and also provides access to a Unix shell. Enabling the "USB Debugging" option under "Developer Options" is important to allow MIET to communicate with the Android device via ADB. Rooting the mobile device is needed in order to gain administrator-level access and perform activities such as browsing the directories of the device's internal storage, copying necessary files, mounting partitions, etc. The necessary tools for rooting the device were the Android SDK platform tools [79], as well as the recovery image provided by NVIDIA [80]. One of the reasons for choosing the specific device was that NVIDIA provides an official rooting mechanism, without the need of third party software.

MIET runs in any environment able to execute Python scripts. It uses ADB to run shell commands and export CPU usage (overall and per process), memory usage (overall and per process), network traffic data, and battery usage. For each malware, MIET measures hardware usage statistics multiple times every few minutes and then stores the results in a local .csv file. Each file contains different measurements over time. In other words, multiple measurements are needed to capture all functions and effects of malware on a certain device, over a certain period of time. It is important to understand that metrics exported by MIET include more than just CPU or Memory usage as a percentage, but thousands lines of process information per malware over time. More specifically, for

CPU usage, MIET is able to log user and kernel usage per process, as well as minor and major faults per process. Furthermore, for memory usage, MIET is able to log memory usage (measured in Kbyte) per process and memory usage for each process category (e.g. System, Visible, Cached etc.). These results are then stored manually in the database, while automation of the specific procedure is one of the authors' future plans. The following subsection enlists the malware used on the device.

3.2.2 List of Malware

This subsection includes a list of all malware analyzed and tested, as well as a succinct description for each of them. The first 10 malware samples, dating from 2011 to 2014 were acquired from the Contagio Minidump depository [81], whereas the rest 4 from Bhatia' s Github depository [82].

Android Armor It is a potentially unwanted application (PUA), a type of malware that is not essentially responsible for causing system damage, but is still not wanted in a system, since it monitors user behavior to pick pop-up ads. Pop-ups are likely to affect the device's performance but they definitely have an effect on users' experience, due to their personalized character. Similarly to most PUAs, Android Armor is available for download by third party application platforms or websites. When installed, it is invoked when the user opens a browser application. It begins running in the background, cascaded as a user experience improvement utility. Overall, the aim of this malware is to advertise more unwanted applications. Android Armor was spotted in January 2013 [83].

BadNews This malware is masqueraded as an advertisement and upon its installation, it starts spreading various other malware. Its actions then range from spreading fake news messages and sending the device's phone number to a command and control (C&C) server, to prompting users to install other malicious applications. BadNews is also known to disguise itself as application updates to other popular applications. The malware was initially propagated from the Google Play store and an estimated 2 to 9 million users had already downloaded this application from the store before its removal. BadNews was discovered in April 2013 [84].

Xsser It is a mobile Remote Access Tool (RAT). It is spreading through Man-In-The-Middle (MITM) and phishing attacks [85]. Researchers believe that this malware is being used by an organized group which targets owners of specific devices and software vendors with the goal of stealing credentials, hijacking browsing sessions, and executing code at the infected devices. Attacks have focused on software vendors, software-as-a-service (SaaS) providers and Internet Service Providers (ISPs). They also serve other malware via phishing or by impersonating legitimate websites. The malware was originally created so as to infect the Android OS, but later versions of this malware affect jailbroken iOS devices. Surveillance, theft of login credentials and Distributed Denial of Service (DDoS) attacks are malicious activities caused by Xsser infections. Xsser was detected in October 2014 [86].

CoinKrypt It uses the maximum computational power of a device for generating virtual currency [87]. This results in device overheating and battery drain. The malware appears to be targeting only newer virtual currencies such as Litecoin, Dogecoin, and Casinocoin. The threat was detected in March 2014.

NotCompatible This malware operates like a drive-by download threat [88]. It is downloaded by visiting an infected website, which tricks victims into downloading a package, namely update.apk. Once installed, the malware attempts to communicate with C&C servers. NotCompatible was first discovered in January 2012.

Nickispy It is a spyware circulating in unofficial application markets in China. When installed on a device, it spies the user's activity and forwards it to remote servers. The spyware does not display any application icon in the Application Launcher, and could therefore stay unnoticed by users. One way to detect this malware is by observing the Application Manager, where a so-called "Android System Message" application shows. This spyware was detected in October 2011 [89].

SpamSoldier It is a type of mobile malware that turns the victim's device into a bot and sends spam text messages, while pretending to be a game application. It proceeds to the actual game .apk installation, but also engages in malicious activity in the

background and sends spam messages that attempt to spread the malware. SpamSoldier was detected in December 2012 [90].

AngryBirds - LeNa Once activated, the malware connects with a C&C server, installing launch packages without the owner's approval. This malware is targeted for devices running the Android OS. Lookout researchers have stated that they have discovered this exploit on a number of alternative application marketplaces, hiding as a fake version of Rovio's popular Angry Birds in Space game. This malware was detected in April 2012 [91].

Angrybirds – Transformers Researchers of the Russian antivirus vendor Doctor Web [92] have analyzed the behavior of this malware, which is now identified as Android.Elite.1.origin, and discovered that it is designed to disrupt certain communication on the device, as well as to delete data stored on the SD card. The Elite Android Trojan requires administrative rights, while misleading the user that it is necessary for the completion of the installation process. The moment it gains higher privileges, it automatically starts the wiping process of the storage unit. Furthermore, it prevents communication via texting by blocking communication applications and access to them and the message "Obey Or Be Hacked" pops up. This malware was detected in October 2014 [93].

Feabme It is a remote access Trojan which appears to be a popular Android game such as the "Cowboy adventure". This Trojan can steal files and take over services or applications. Furthermore, it includes ransomware code, which can encrypt files, lock the screen and blackmail the user for a payment up to 500 USD to unlock the device. This threat was detected in July 2015 [94].

Rumms It refers to a family of mobile malware targeting users from Russia. Attackers lure their victims into downloading this malware by sending fake SMS messages containing malicious links. The infected app will request administrator privileges, hide itself and then send device information to a remote C&C server. Furthermore, infected phones upload incoming SMS messages to remote servers, forward incoming calls, and

send SMS messages to contacts. The earliest identified sample was traced in January 2016 [95].

TrojanDownloader.Agent.JI This malicious application imitates an Adobe Flash Player update. It is distributed by compromised websites such as adult services. The application creates pop ups related to increased battery consumption until the user enables the service. Further actions include sending device information to C&C servers and downloading other malicious applications. The application was detected in February 2017 [96].

3.3 Evaluation

This section is concerned with our initial machine learning evaluation procedure. Two metrics, namely CPU and memory usage of infected devices, are tested using a variety of classification algorithms against the equivalent measured values when the device was in a "clean" state.

3.3.1 Machine Learning Results

MIET exports metrics in .csv format. The metrics are then imported into the database, which contains the following tables: malware, CPU usage, memory usage, device behaviour, hardware behaviour. The table named "Malware" lists all the malware and includes a short description of each one. It also provides each malware with a unique id. The table entitled "CPU usage" includes usage (%), process, user and kernel usage (%), minor and major faults and a malware id field, which interconnects each row with the "Malware" table. The table entitled "Memory usage" includes usage (Kbyte), process, tag, and a malware id field. The "Hardware behavior" table lists only the increase in CPU and memory usage, as well as a malware id field. The "Device behavior" table includes a malware id field and a set of device behavior qualitative indications, such as the deleted files, file replication, pop-up windows, and the restricted services. Tables 3.1 and 3.2 are snapshots of the Mal-warehouse database.

Usage	Process	User	Kernel	Minor	Major	Malware
4.5	mediaserver	2.8	1.6	3	0	4
4.0	system_server	2.6	1.4	731	0	4
3.6	com.android.systemui	1.2	2.3	542	1	4
0.9	android.process.media	0.5	0.3	766	1	4
0.6	rm_ts_server	0.6	0.0	14	0	4

TABLE 3.1: Mal-warehouse Database - CPU Usage Table

Usage	Process	Tag	Malware
131.832	surfaceflinger	1	2
110.266	com.android.systemui	1	2
98.66	system	1	2
64.322	com.google.android.gms	1	2
56.996	com.android.chrome	1	2

TABLE 3.2: Mal-warehouse Database - Memory Usage Table

3.3.2 CPU Usage Results

Data analysis on the results was carried out using the open source data mining tool Orange [97] and included only quantitative metrics. More specifically, each row of the imported files contains a column pointing to the target variable, which represents the ground truth state of the tuple (infected-clean) and the rest of its columns are either CPU or memory usage attributes.

As already pointed out, in our evaluation, 5 different machine learning algorithms, namely kNN, Random Forest, Support Vector Machine, Naive Bayes and AdaBoost, were used. The selected sampling technique was 10-fold cross-validation. Table 3.3 shows the best performance results per malware for the classification related to CPU usage. AdaBoost and Naive Bayes showed the best performance over the rest of the algorithms. It is also observable that contemporary malware exhibited lower successful detection rates when compared to the respective older instances. One possible interpretation of this phenomenon is that their mechanisms may consume less resources when compared to their ancestors. However, this is not a safe conclusion and more data have to be collected in order to give a more definitive answer.

Malware	AUC	CA	F1	Precision	Recall	Algorithm
Android Armor	69.2%	79.1%	88.3%	81.3%	96.6%	AdaBoost
Angry Birds Lena	80.3%	73.0%	81.8%	69.8%	98.7%	AdaBoost
Bad News	80.3%	82.4%	89.5%	82.3%	98.0%	AdaBoost
Android CI4	79.2%	78.1%	86.5%	77.8%	97.5%	AdaBoost
Coinkrypt	53.7%	74.4%	85.0%	77.0%	94.7%	AdaBoost
Nickyspy	51.8%	74.0%	84.7%	77.7%	93.2%	AdaBoost
Spamsoldier	56.6%	75.8%	85.8%	77.7%	96.6%	AdaBoost
Xsser	78.7%	83.9%	90.6%	83.2%	99.4%	AdaBoost
Feabme	74.4%	69.5%	69.2%	73.6%	65.2%	Naive Bayes
Mazar bot	63.3%	67.1%	79.8%	73.8%	86.7%	Naive Bayes
Rumms	77.2%	71.9%	71.6%	73.5%	69.8%	Naive Bayes
TrojanDownloader.Agent.JI	76.1%	73.1%	73.0%	76.3%	69.9%	Naive Bayes

TABLE 3.3: CPU Usage Evaluation

Furthermore, it is notable that the "Rumms" malware is among the toughest against CPU usage detection, scoring the lowest recall and one of the lowest precision rates. "Feabme" on the other hand, scored the lowest precision and one of the lowest recall rates. "Xsser" has the highest precision, recall and accuracy rates. Finally, the ratio between the number of TP and TN results is slightly unbalanced. More specifically, the detection of non-infected patterns was less accurate, which can be seen in some of the AUC rates.

3.3.3 Memory Usage Results

Tests and score of each malware analysis for memory usage results are shown in Table 3.4. From the Table 3.4, it can be observed that AdaBoost had the best performance.

Malware	AUC	CA	F1	Precision	Recall	Algorithm
Android Armor	67.9%	80.1%	87.7%	82.5%	93.5%	AdaBoost
Angry Birds Lena	85.4%	83.6%	83.0%	84.7%	81.4%	AdaBoost
Bad News	76.8%	88.0%	92.9%	88.0%	98.4%	AdaBoost
Android CI4	71.4%	84.4%	90.8%	85.6%	96.6%	AdaBoost
Coinkrypt	79.1%	89.3%	93.7%	88.6%	99.5%	AdaBoost
Nickyspy	74.9%	87.4%	92.6%	87.3%	98.6%	AdaBoost
Spamsoldier	72.7%	85.8%	91.6%	86.6%	97.3%	AdaBoost
Xsser	75.0%	86.1%	91.7%	87.0%	96.9%	AdaBoost
Feabme	84.7%	84.0%	90.5%	86.1%	95.4%	AdaBoost
Mazar bot	75.8%	79.1%	86.8%	83.3%	90.6%	AdaBoost
Rumms	80.6%	83.5%	90.2%	86.0%	94.8%	AdaBoost
TrojanDownloader.Agent.JI	78.3%	80.6%	88.6%	83.5%	94.3%	AdaBoost

TABLE 3.4: Memory Usage Evaluation

Most malware have a satisfying result in memory usage, with "Android Armor" having the lowest recall and precision, as well as AUC rates. On the other hand, "Coinkrypt" presented the highest recall, precision and accuracy detection rates.

3.4 Discussion

It is obvious that the overall quality of the data collected by MIET strongly depends on the amount and diversity of the available malware. Further data extractions from the same and other malware would improve the respective findings. In addition, due to hardware limitations, a large amount of malicious applications were never tested as intended, while different versions of the Android OS would result in different device behaviors. Older Android OS versions would increase malware activity, while newer versions would further restrict malware activity on the device. Additionally, enhancing the scenarios with device usage simulations would also alter malware activity on the device and help improve the results. Finally, devices that support SIM cards would also increase activity with certain malware.

3.5 Related Work

Christodorescu et al [98] tested 3 commercial virus scanners to conclude that simple obfuscation transformations were able to successfully cascade the engineered malware.

The authors proposed *Static Analysis of Executables (SAFE)* in order to detect malicious patterns in executables and tested the aforementioned solution in 4 different samples. Other approaches, such as Kolter and Maloofs' research [99] focus on machine learning for detecting malicious executables.

Blasing at al. [100] presented AASandbox, a method based on the native Android sandbox mechanism, for performing static and dynamic analysis of executables.

Damopoulos et al. [101] proposed an anomaly-based Intrusion Detection System (IDS) based on users' behavioral profiles on mobile devices. Furthermore, they used 4 machine learning algorithms to detect misuse in phone calls, SMS and Web browsing service logs. According to the authors, at least one algorithm was able to detect misuses with a very high success rate.

An approach to permission-based Android malware detection called PUMA, was presented by Sanz et al. [42]. The authors used machine learning techniques for malware detection, by analyzing the extracted permissions from the mobile applications.

Wu et al. [41] implemented DroidMat, which provides malware detection through manifest and API calls tracing. Its static feature-based mechanism "includes permissions, deployment of components, intent messages passing, and API calls for characterizing the Android applications behavior".

Damopoulos et al. [61] proposed a tool for dynamically analyzing any iOS software in terms of method invocation. The same authors also detailed on the synergistic operation of host and cloud anomaly-based IDS for smartphones.

Papamartzivanos et al. [11] proposed a cloud-based system that operates under a crowdsourcing logic. Their system includes three main services, namely privacy-flow tracking, crowdsourcing, and detection and reaction against privacy violations.

Jang et al. [53] presented Andro-AutoPsy, an anti-malware system based on similarity matching of malware information. The authors state that Andro-AutoPsy succeeded in detecting and classifying malware samples into similar subgroups "by exploiting the profiles extracted from integrated footprints, which are implicitly equivalent to distinct characteristics".

3.6 Conclusions

In this chapter we presented a collection of mobile malware behavioral patterns by extracting CPU and memory usage information per process from a mobile device using the open-source tool MIET, which is available at the following repository: https://gitlab.com/billkoul8/malwarehouse.git. With the extracted data, the authors were able to detect CPU and memory usage anomalies. After converting the exported usage information accordingly, the data were uploaded into a public database. By using machine learning algorithms on the exported metrics, the authors were able to evaluate the optional detection model with rather favorable results. The significance of this research lies in the promising results from the collection of samples analyzed and to the scalable character of the proposed solution.

Chapter 4

Feature importance in Android malware detection

4.1 Introduction

As previously mentioned in Chapter 1, anomaly-based detection employs machine learning to detect malicious behavior, i.e., deviation from a model built during the training phase. In this context, a key point, which to our knowledge is not properly addressed in the hitherto literature, is the importance of each feature category, say, permissions and intents, in mobile app classification. Simply put, which group of features in general, and which features within each group in particular do contribute the most information when it comes to classification? And, is the answer to the previous question related to the employed dataset?

To respond to the previous questions, this chapter first briefly surveys all the major datasets used in the context of app classification in the Android platform. We specifically consider datasets exploited in the respective literature from 2012 onward. Then, we concentrate on the so far most commonly used and modern datasets, namely Drebin [102], VirusShare [103], and AndroZoo [104], and try to answer the second question per dataset. Precisely, by using the average coefficients of permissions and intents for a large number of malware instances per corpus, we demonstrate the most significant feature category. Lastly, we report the top ten features per dataset and discuss similarities between these corpora.

4.2 Datasets

Heretofore, several mobile app corpora have been built and exploited by researchers to evaluate malware detection approaches on the Android platform. This section surveys in chronological order all major mobile malware datasets used in the literature. Table 4.1 compares all datasets, with regard to their age, size, access, and impact to the research community. As shown in the table, AndroZoo and VirusShare are the only datasets still being updated today. The table also includes the number of works each dataset was employed according to: (i) a corresponding list of publications as given in the dataset's website, (ii) a listing of downloads in the dataset's website, and (iii) the citations the original dataset work, if any, has received according to Google Scholar.

- Contagio mobile mini-dump [81]: It is a publicly available repository of mobile malware samples. The samples were collected in 2010 and currently the dataset contains 189 malware samples, thus being by far the smallest available corpus.

- MalGenome [105]: In 2012, the MalGenome dataset was released. This corpus contains 1,260 malware samples categorized into 49 different malware families. The malware instances are dated from Aug. 2010 to Oct. 2011. The work which introduced this dataset seems to be by far the most highly cited. Unfortunately, the MalGenome project has stopped sharing their dataset in Dec. 2015.

- VirusShare [103]: The access to the dataset's website is granted via invitation only. The dataset does not only contain mobile malware samples, but also samples from various platforms, including Windows and Linux. Furthermore, it is updated regularly and contains samples in the time span from 2012 to 2020. This dataset is also very popular in the research community, i.e., the number of works exploiting it is steadily growing every year.

- Drebin [102]: It comprises 5,560 malware across 179 different families. The samples were collected between Aug. 2010 and Oct. 2012. Drebin is one of the most popular datasets and it is referenced in more that 1.3K works in the literature. On the downside, it has not received an update since 2012.

- DroidBench [106]: Is a set of apps implementing different types of data leakage. At present, the repository comprises 120 apps. The main task of these apps is

data leak. Put simply, the samples in DroidBench are not real malware instances and are only meant to evaluate analysis tools.

- PRAGuard [107]: It currently contains 10,479 malware samples, obtained by obfuscating the MalGenome and the Contagio mobile mini-dump datasets with seven different obfuscation techniques. The samples are dated from 2010 to 2011.

- AndroZoo [104]: AndroZoo is a growing collection of Android apps collected from diverse sources, including the official Google Play store [30]. The dataset is updated regularly and it currently contains over 12M samples. The access to the dataset is granted by application only. The number of works using this dataset is also growing on a yearly basis.

- Kharon [108]: It comprises only 7 instances of malware, namely, SimpLocker, BadNews, DroidKungFu1, SaveMe, MobiDash, WipeLocker, and Cajino, which have been manually dissected and documented. The samples are dated from 2012 to 2016.

- Android Adware and General Malware Dataset (AAGM) [109]: It is generated from 1,900 apps belonging to the following three categories: 250 adware apps, 150 general malware apps, and 1,500 benign apps. Benign samples are dated from 2015 to 2016, but there is not enough information on the creation date of the malware samples.

- AMD [110]: It is a publicly shared dataset which contains 24,553 samples categorized in 135 varieties among 71 malware families. The samples are dated from 2010 to 2016. At the time of writing, the AMD website were unavailable.

4.3 Feature importance

A key factor that affects the accuracy of machine learning based malware detection methods is the importance of features contained in malware samples [111]. To obtain a clear view of this aspect, the current section presents our results on feature importance over a great mass of malware apps collected from the state-of-the-art datasets. That is, as already pointed out in section 4.2, VirusShare and AndroZoo seem to be the only

Dataset	Created	Last updated	Size	Access type	Publications/Downloads/Citations
Contagio mobile	2010	2010	189	Public	-/-/-
MalGenome	2011	2011	1,260	Unavailable	-/460/2181
VirusShare	2011	2020	Unknown*	Invitation	1307/-/-
Drebin	2012	2012	5,560	Public	-/157/1353
DroidBench	2013	2013	120	Public	-/-/-
PRAGuard	2015	2015	10,479	Application	-/133/84
AndroZoo	2016	2020	12,498,250*	Application	-/-/267
Kharon	2016	2016	7	public	-/-/20
AAGM	2017	2017	1,900*	Public	-/-/29
AMD	2017	2017	24,553	Public	-/368/171

TABLE 4.1: Outline of major datasets ordered by their creation date. Asterisk = not all samples are malicious, Dash = Not available

datasets still being updated today. Furthermore, the Drebin dataset has been used by a multitude of research works on the topic of mobile malware detection, thus making it ideal when comparing new detection methods with previous state-of-the-art.

Precisely, in the context of this section, we randomly collected 1K malware samples from each of these three datasets, as well as 1K random benign apps from Google play to create three 2K balanced datasets of both malware and benign apps. The samples are dated from 2010 to 2012, 2014 to 2017, and 2017 to 2020 for the Drebin, VirusShare and AndroZoo corpora, respectively. Static analysis was performed via the open-source tool *Androtomist* [111] to extract permissions and intents for each of the 3K malware plus 1K benign samples collected in total. Specifically, each app was decompiled to get the Manifest.xml file and log permissions and intents to create feature vectors, i.e., binary representations of each distinct feature.

The feature importance score is assigned by coefficients calculated as part of an Information Gain (IG) model. Specifically, IG is an entropy-based feature evaluation method and is defined as the amount of information provided by the feature items [112]. Put simply, low probability, i.e., rare events are more surprising and have a greater amount of information. This also means that probability distributions where the events are almost equally likely are more surprising and have larger entropy. Therefore, in our case, information entropy can be roughly thought of as how much variance the data have. For example, a dataset of only one feature would have zero entropy. On the other hand, a dataset of mixed features would have relatively high entropy. The formula to calculate

the information Entropy for a dataset with C classes is as follows:

$$E = -\sum_{i}^{C} p_i log_2 p_i$$

where p is the probability of randomly picking an element of class i, i.e., the proportion of the dataset made up of class i. It is worth noting that the entropy metric of uncertainty introduced by Shannon [113] has been exploited in several works in the information security literature [114, 115, 116].

Computing the IG for a feature involves calculating the entropy of the class label, i.e., positive (malware) or negative (benign) for the entire dataset and subtracting the conditional entropies for each possible value of that feature, in our case "exist" (1) or "not exist" (0). The entropy calculation requires a frequency count of the class label by feature value. Precisely, the instances of a dataset are selected with a feature value x. Then, the occurrences of each class are counted and the entropy for x is computed. This step is repeated for each possible value x (0,1) of the feature. The formula to calculate IG is as follows:

$$IG(D, v) = H(D)–H(D|v)$$

Where $IG(D, v)$ is the information gain for the dataset D for the variable v, $H(D)$ is the entropy for the dataset before any change, and $H(D|v)$ is the conditional entropy for the dataset given the variable v. The higher the IG score the more information is gained from this feature.

Tables 4.2, 4.3, and the left side of table 4.4 include the top 10 features observed for Drebin, VirusShare, and AndroZoo, respectively, along with their IG score. By observing the top 10 features of Drebin and VirusShare in Tables 4.2 and 4.3, it becomes obvious that there is a similarity between the top features of these corpora. More specifically, the first three features are the same for both Drebin and VirusShare's top 10. In total, 7 out of 10 features are identical in both tables divided into 6 permissions and 1 intent. On the other hand, as shown in the left side of table 4.4, AndroZoo has 1 out of 10 identical features with Drebin's top 10, and shares zero out of 10 identical features with VirusShare's top 10. Lastly, all of the Androzoo's top 10 features are intents, contrariwise to Drebin and VirusShare where only 2 and 3 out of 10 are intents, respectively. This further demonstrates the difference in feature importance among the examined datasets.

To verify our conclusions on feature importance regarding the examined datasets, we randomly selected an additional 1K malware apps from the most contemporary one, i.e., Androzoo, and also randomly added 1K new benign apps from Google Play. This doubles the number of instances contained in our AndroZoo dataset, i.e., 2K malware and 2K benign apps in total. The right side of table 4.4 contains the feature importance scores for this new double size dataset. As expected, the top 10 features in Table 4.4 are all intents too. Also, 8 out of 10 features are common to the two sides of the table, and 3 out of the 4 top features occupy the same places in both sides of the table. This further supports the observation that feature importance is tightly related to the age of the malware. Naturally, this phenomenon may negatively affect the performance of older detection methods if solely based on these two categories of features.

Figure 4.1 illustrates the average feature importance scores per dataset, for both the examined feature categories, namely permissions and intents. Note that the mean score is calculated over all the permissions and intents identified, and not solely on the top 10 values included in tables 4.2, 4.3, and 4.4. As easily observed from the figure, in the AndroZoo corpus, intents produced a much more higher - approximately triple - IG score than permissions. Emphatically, this situation applies almost equally to both the 2K and 4K datasets. Nevertheless, this picture is clearly inverted in the Drebin and VirusShare corpora, that is, in Drebin there is an $\approx +0.0045$ and in VirusShare an $\approx +0.002$ higher score than that of intents.

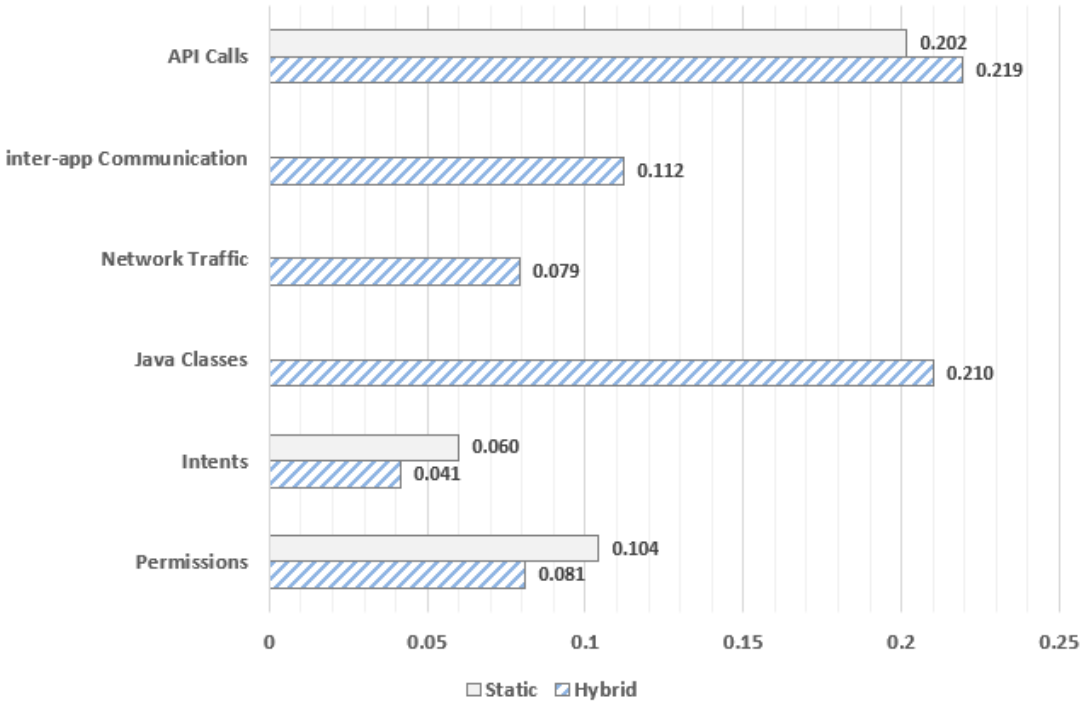

FIGURE 4.1: Average feature importance scores on all three datasets for the two feature categories (Permissions and Intents)

IG Score	Feature	Category
0.2294	android.permission.INTERNET	Permissions
0.2130	android.permission.READ_PHONE_STATE	Permissions
0.1335	android.permission.SEND_SMS	Permissions
0.0994	android.permission.WRITE_EXTERNAL_STORAGE	Permissions
0.0965	android.permission.RECEIVE_BOOT_COMPLETED	Permissions
0.0939	android.permission.RECEIVE_SMS	Permissions
0.0857	android.permission.READ_SMS	Permissions
0.0810	android.intent.action.BOOT_COMPLETED	Intents
0.0706	com.google.android.c1dm.intent.RECEIVE	Intents
0.0683	android.permission.ACCESS_COARSE_LOCATION	Permissions

TABLE 4.2: Top 10 features in the Drebin dataset.

IG Score	Feature	Category
0.2305	android.permission.INTERNET	Permissions
0.2276	android.permission.READ_PHONE_STATE	Permissions
0.1713	android.permission.SEND_SMS	Permissions
0.1477	android.permission.RECEIVE_SMS	Permissions
0.1328	android.permission.WRITE_EXTERNAL_STORAGE	Permissions
0.1067	android.permission.READ_SMS	Permissions
0.0958	android.intent.category.HOME	Intents
0.0926	android.intent.action.DATA_SMS_RECEIVED	Intents
0.0648	android.intent.action.BOOT_COMPLETED	Intents
0.0610	android.permission.WAKE_LOCK	Permissions

TABLE 4.3: Top 10 features in the VirusShare dataset.

IG Score	Feature	IG Score	Feature	Category
0,1550	android.intent.action.USER_PRESENT	0,1680	android.intent.action.USER_PRESENT	Intents
0,1401	android.intent.action.PACKAGE_REMOVED	0,1528	android.intent.action.PACKAGE_REMOVED	Intents
0,1208	android.intent.category.DEFAULT	0,1208	android.intent.category.BROWSABLE	Intents
0,0769	android.intent.action.PACKAGE_ADDED	0,1162	android.intent.action.PACKAGE_ADDED	Intents
0,0672	android.intent.category.BROWSABLE	0,0955	cn.jpush.android.intent.NOTIFICATION_RECEIVED_PROXY	Intents
0,0652	android.intent.action.VIEW	0,0812	android.intent.action.ACTION_POWER_CONNECTED	Intents
0,0582	com.google.android.c1dm.intent.RECEIVE	0,0780	org.agoo.android.intent.action.RECEIVE	Intents
0,0530	cn.jpush.android.intent.NOTIFICATION_RECEIVED_PROXY	0,0722	com.google.android.c1dm.intent.RECEIVE	Intents
0,0521	android.intent.action.ACTION_POWER_CONNECTED	0,0685	android.intent.action.MEDIA_MOUNTED	Intents
0,0518	org.agoo.android.intent.action.RECEIVE	0,0685	cn.jpush.android.intent.NOTIFICATION_OPENED	Intents

TABLE 4.4: Top 10 features in the AndroZoo dataset. Left: 2K apps dataset, right: 4K apps dataset

As a final step, the permissions and intents were used separately to classify the malware samples collected from the 4K AndroZoo corpora. Recall that vis-à-vis the rest of the datasets given in Table 4.1, AndroZoo is expected to be more challenging as it contains contemporary malware. Specifically, Table 4.5 includes the AUC and accuracy (CA) scores for permissions and intents for a number of base models, namely AdaBoost, k-NN, Logistic Regression (LR), Naive Bayes (NB), Multilayer Perceptron (MLP), Random forest (RF), Stochastic Gradient Descent (SGD) and Support Vector

Category	AdaBoost		k-NN		LR		NB		MLP		RF		SGD		SVM		Ensemble	
	AUC	CA	AUC	CA	AUC	CA	AUC	CA	AUC	CA	AUC	CA	AUC	CA	AUC	CA	AUC	CA
Permissions	0,737	0,706	**0,767**	0,616	0,758	0,705	0,762	0,692	0,739	**0,707**	0,735	0,704	0,671	0,671	0,471	0,484	0,757	0,711
Intents	0,842	0,790	0,756	0,730	0,835	0,766	0,808	0,712	0,849	0,788	**0,856**	**0,799**	0,761	0,761	0,472	0,497	0,858	0,801
Both	0,865	0,806	0,813	0,767	0,852	0,794	0,835	0,72	0,867	0,803	**0,882**	**0,814**	0,762	0,763	0,502	0,501	0,879	0.809

TABLE 4.5: AUC and accuracy comparison between permissions and intents for the 4K AndroZoo corpora. Best scores for the 8 base models are in boldface.

Machine (SVM). Note that the rightmost column of the same table shows the AUC and CA scores calculated by using an ensemble model. The ensemble model was constructed by averaging the output of all 8 base models previously mentioned. The evaluation of each model was carried out using the 10-fold cross-validation technique.

As observed from Table 4.5, when using the base models, the best AUC score if only employing the permissions as a feature category is 76.7% vis-à-vis 85.6% when only using intents. On the other hand, the best AC score is 70.7% and 79.9% when using permissions and intents, respectively. Simply put, as a sole feature category, intents scored a 8.9% and 9.2% higher AUC and AC than permissions, respectively. On the other hand, the ensemble model shows a 10.1% and 9% improvement in AUC and accuracy score, respectively. Finally, as shown in the bottom line of the Table, the best AUC and AC scores when using these two feature categories in tandem are 88.2% and 81.4%, respectively. This means that the utilization of both feature categories resulted in a betterment of 2.1% and 0.8%, on the AUC and AC respectively, when compared to the scores achieved when using only intents. Of course, this increment in performance is indeed considerable, but not exceptional.

4.4 Related work

This section presents previous work on feature importance and feature selection. As already mentioned, thus far, this topic has received little attention in the literature.

Feizollah et al. [117] categorized the available features into four groups, namely, static, dynamic, hybrid, and app's metadata. Furthermore, the authors evaluate the aforementioned features with regard to the difficulty of extraction and their popularity among the relevant literature. Finally, they offer a survey of the available datasets. On the

downside, the only available datasets at the time of this research were Contagio [81], MalGenome [105], and Drebin.

Zhao et al. [118] proposed a feature selection algorithm called *FrequenSel*. According to the authors, FrequenSel selects features which are frequently used in malware and rarely used in benign apps, thus it can more accurately distinguish between the positive and negative class. During their experiments, the authors evaluated their approach with a collection of 7,972 apps, which contained malware collected from Drebin and other public malware libraries, as well as benign apps from Google Play. Their results reported an accuracy of up to 98%. Similar to [117], the apps used in this work are nowadays considered outdated.

Kouliaridis et al. [111] introduced an online open-source tool called *Androtomist*, which performs hybrid analysis on Android apps. The authors focused on the importance of dynamic instrumentation, as well as the improvement in detection achieved when hybrid analysis is used vis-à-vis to static analysis. During their experiments, the authors compared feature importance between three datasets, namely Drebin, VirusShare, and AndroZoo. Finally, the authors elaborated on features which seem to be commonly exploited in malware and seldom in benign apps. While the datasets used in their work comprise newer apps as opposed to [117] and [118], the authors used a rather small subset of each dataset in the course of their experiments.

To the best of our knowledge, none of the above mentioned works address feature importance across multiple datasets with a large number of samples.

4.5 Conclusions

This chapter examined the literature on Android malware detection spanning the period from 2012 to 2020. The focus was on contributions exploiting machine learning and on identifying the datasets used in each relevant work. Our analysis showed that three datasets, namely Drebin, VirusShare, and AndroZoo stand out. Following, we used a significant mass of malware instances existing in each of the aforementioned datasets along with a large number of benign instances to estimate the feature importance of permissions and intents. We reported the most important features per dataset in terms of IG, as well as similarities and differences between the top features in each of them. Our

results reveal a noteworthy difference in feature importance when inspecting our partial AndroZoo datasets vis-à-vis the other two. Thus, our findings, especially those stemming from the AndroZoo dataset which consists of contemporary malware apps, demonstrate that the permissions feature alone is insufficient, and even the mixture of intents and permissions does not yield remarkable results, and therefore it should be bolstered by supplementary, more weighty features. The need for more feature categories, i.e., the ones produced via static or dynamic analysis or both, as well as their importance in the detection process is also supported and scrutinized by previous work [111]. As a way forward, we aim to improve this research by also examining the significance of features stemming from dynamic analysis, in an effort to find the best combination of feature categories which can be used against mobile malware. Moreover, based on feature importance, we aim to examine feature dimension reduction via the use of diverse techniques.

Chapter 5

Two anatomists are better than one - Dual-level Android malware detection

5.1 Introduction

As previously explained in section 2.4, mobile malware detection schemes can be categorized into two broad classes, namely signature and anomaly-based. The former collects patterns and signatures stemming from known malware, and compares them against unknown pieces of code for determining their status. The latter class employs a more lax approach; by observing the normal behavior of a piece of code for a certain amount of time and using the metrics of that normal model against any deviant behavior. App analysis on the other hand is done by means of either a static, dynamic, or hybrid method. Briefly, opposite to the dynamic approach, static analysis involves examining an app without actually running it. In the last few years, considerable work has been devoted to all the three types of app analysis for the sake of effectively detecting malware on Android [102, 119, 120, 121, 52]. However, hitherto, little attention has been paid to hybrid approaches for malware detection with emphasis on ML. Moreover, as previously explained in Section 4.5, features stemming from static analysis alone, such as permissions and intents may not be sufficient when examining concurrent mobile malware apps, thus there is a need for more weighty features. The current chapter aims

to improve the research presented in Chapter 4 by also extracting additional features stemming from dynamic analysis.

This chapter concerns the design, implementation and evaluation of Androtomist, a hybrid detection solution that combines static analysis and dynamic instrumentation to detect mobile malware on the Android platform. Instrumentation is the process of injecting trace code into a software's source code, binary code, or execution environment, and is mainly exploited for debugging, tracing, and performance and security analysis purposes. Instrumentation techniques do not necessarily modify code, but rather tamper with the app's execution or behavior based on defined constructs. The advantages of melding static analysis with dynamic instrumentation into a hybrid approach are twofold. On the one hand, the static analysis works as a fast pre-filtering mechanism capitalizing on permission analysis, API calls, and intents. On the other, by instrumenting code, one is able to scrupulously analyze and detect a plethora of behaviors during runtime, such as network traffic, information leakage threatening the privacy of the end-user [11], and so on. On top of legacy signature-based metrics, Androtomist is assessed via the use of different classification performance metrics against three well-known malware datasets.

To enable the use of Androtomist by virtually anyone, including ordinary end-users, a web app has been developed. The web-based interface caters for easily uploading and analyzing mobile apps on-the-fly. Both static and dynamic analysis in terms of feature extraction are performed automatically. Finally, unlike similar open source tools, Androtomist is easy to set up and has only a few dependencies, all of which pertain to open source software. The main contributions of this study are:

- A methodology is presented that collects groups of static and dynamic features mapping the behavior of an app, including permissions, intents, API calls, Java classes, network traffic, and inter-process communication. Especially for the dynamic analysis part, among others, we improve the hitherto related work by means of contributing features emanating from the hooking of Java classes, which is made possible due to instrumentation.
- We examine the effect of features when dynamic instrumentation is provided. It is demonstrated that the effectiveness of classification models based on either static

or dynamic analysis without instrumentation is outperformed by models using dynamic instrumentation.

- We report experimental results on three different well-known mobile malware benchmark datasets that are directly compared with state-of-the-art methods under the same settings. The performance of the approaches presented in this chapter is quite competitive to the best results reported so far for these corpora, demonstrating that the proposed methods can be an efficient and effective alternative toward more sophisticated malware detection systems.
- We propose an ensemble approach by averaging the output of all base models for each malware instance separately. The combination of all base models achieves the best average results in all three data sets examined. This demonstrates that ensembles of classifiers based on multiple, possibly heterogeneous models, can further improve the performance of individual base classifiers.
- A new, publicly available, open source software tool is provided that could be exploited for conducting further research on mobile malware detection.

5.2 Proposed Methodology

This section details on our methodology, that is, the tool implemented, the feature extraction process, and the datasets exploited.

5.2.1 Androtomist

Androtomist is offered as a web app [122], which enables the user to easily upload, analyze, and study the behavior of any Android app. Figure 5.1 depicts a high-level view of the Androtomist's architecture. Specifically, Androtomist comprises three basic components, namely virtual machines (VMs), database (DB), and a web app. Each app is installed on a VM running the Android OS. All the features collected during app analysis are stored in the DB. A user is able to interact with Androtomist via the web app.

For the needs of this work, Androtomist runs on a desktop computer equipped with an Intel Core i7-3770K CPU and 8GB of RAM. VirtualBox [123] is used to create and

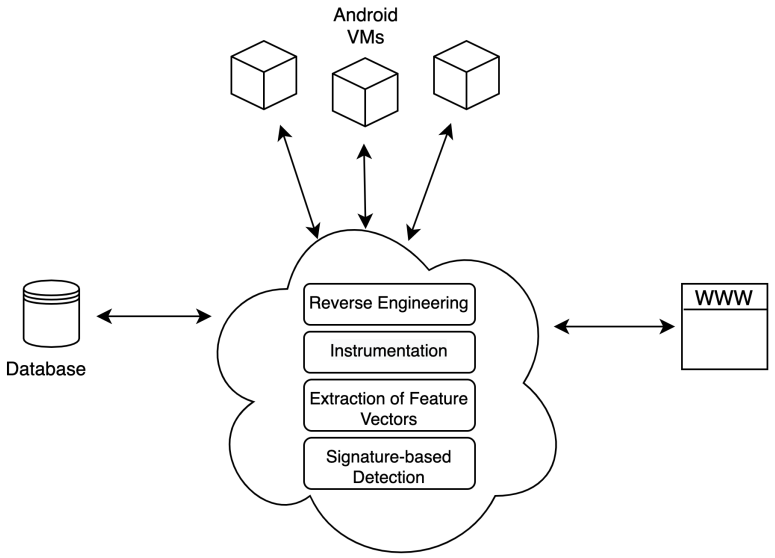

FIGURE 5.1: Androtomist's high level architecture

manage Virtual Machines (VMs). Depending on the user's preference, a VM running Android ver. 7.1, 8.0 or 9.0 is automatically deployed during an app's dynamic analysis phase in order to extract its runtime behavior. A VM communicates with Androtomist using ADB [78], which is a command-line tool that enables communication between an Android device and a computer, and also provides access to a Unix shell. Each time after the installation and testing of an app, the device (VM) is reverted to its original state. This guarantees the best possible comparison results and simultaneously ensures the forensic soundness of the data [55]. This process can be easily performed automatically when using VMs, and as result, the execution time needed for the whole process is substantially reduced. In the context of this work, VMs running the Android ver. 7.1 were used. The reason for not choosing a newer version is to achieve the maximum possible compatibility across malware samples from each dataset listed in Section 5.2.3. Finally, Androtomist is written in .NET Core and its source code is publicly available at [124] under the European Union Public Licence [125].

The process of analyzing APK files is simple and does not demand any special knowledge about malware analysis. After the user connects to the Androtomist's app, they are able to upload the desired APK file in the "files" section. Next, after clicking on the corresponding button, the system initiates the analysis process, which executes in the background. The output per APK file analyzed is available for inspection in the "results" webpage, where there is also an option to export it to a CSV-formatted file. The results webpage incorporates a summary of the app's malicious features, if any, plus an overall

percentage estimation on whether the app is malevolent or not. A detailed output report is also available for download, which contains a substantial amount of data, namely permissions, intents, network traffic, and inter-app communications. Generally, inter-app communication refers to the action of data being passed on to another app using intents during runtime. However, it is noted that inter-app communication cannot always be detected while analyzing one app at a time. Androtomist yields the following types of data:

- Data from static analysis: Permissions and intents from the manifest file, API calls retrieved from the smali files.
- Data from dynamic analysis by means of instrumentation: Network traffic, inter-app communication, Java classes.

All the above mentioned pieces of data are stored in the DB. The web app offers two types of user menu, "user" and "admin". The latter is meant only for the expert user and offers advanced functions, such as rebuilding the detection signatures from new datasets (see subsection 5.3.1), exporting feature vectors to train ML models (see subsection 5.3.2), and adding users to the app. The former is destined to the novice user, even the non-tech-savvy one, and offers basic functionality, including uploading and analyzing apps, as well as inspecting the analysis results.

5.2.2 Extraction of features and feature modeling

Under the hood, Androtomist firstly decompiles and scans the app's APK file to extract information, and secondly, installs and runs the APK on a VM, while generating pseudo-random user action sequences to extract its runtime behavior. In addition, the tool produces vectors of features that can be fed to ML classifiers to assort apps. The internal workings of both static and dynamic analysis phases is schematically shown in Figure 5.2.

Precisely, the app's APK is fed ① to the analysis controller, which first triggers static and then dynamic analysis. During static analysis ②, the APK file is scrutinised to collect static features as detailed further down. During dynamic analysis ③, a VM running the Android OS is loaded, and the APK is installed on it ④. The instrumentation code is

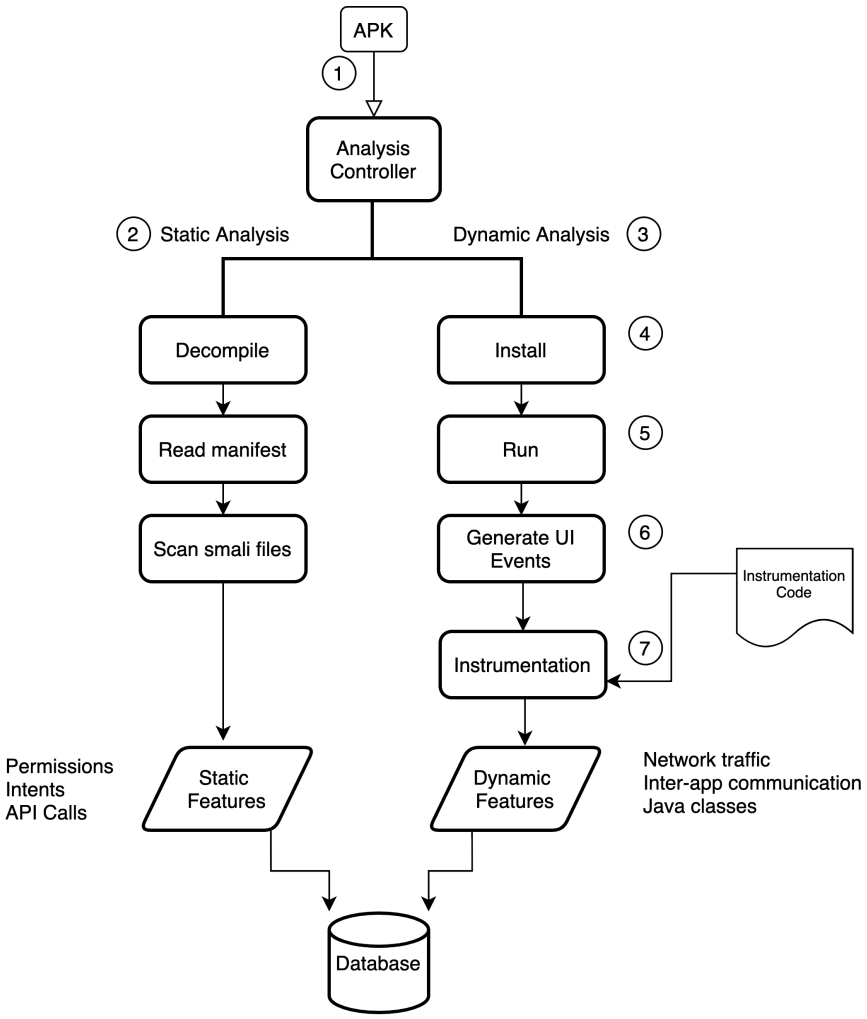

FIGURE 5.2: Androtomist's components interworking

injected during runtime ⑤. Moreover, pseudo-random user input events are generated ⑥ and the instrumentation controller logs any information emanated from the triggered events ⑦. The totality of features collected by both static and dynamic analysis are stored in the DB. The following subsections elaborate on each of the aforementioned steps of the analysis process.

Static feature extraction: The APK file is used as input to fetch static features, i.e., permissions, intents, and API calls. These features are automatically extracted with the help of reverse engineering techniques. Indicatively, for a 1Mb, 10Mb, and 20Mb app, the overall process takes correspondingly ≈10, ≈16, and ≈19 secs. In greater detail, the main steps are as follows:

- Use the Apktool [31] to decompile the APK file and get the manifest file called "AndroidManifest.xml" and the binary stream file called "classed.dex".

- Get app's permissions and intents from the manifest file.
- Decompile the binary stream to acquire the smali files and scan them to discover API calls.

Dynamic feature extraction: Androtomist collects a variety of information during app runtime by means of dynamic instrumentation. These pieces of data include the Java classes used, network communication specifics, and inter-app communication details. To do so, Androtomist capitalises on *Frida* [126], a full-fledged free dynamic instrumentation toolkit. Frida is able to hook on any function, inject code into processes, and trace the app's code without executing any compilation steps or app restarts. In particular, the instrumentation controller illustrated in Figure 5.2 feeds ⑦ the instrumentation code along with the app's package name to Frida during runtime. The instrumentation code hooks on method implementations and alters their operation. Information logged by Frida is forwarded to Androtomist in real-time using ADB. As previously mentioned, all the aforementioned functions are performed automatically. The overall process takes ≈2 min for all apps, independent of their size. Specifically, 1 min is required to load a VM and install the app, and another one is allocated to the generation of ≈1,000 pseudo-random user action sequences using the UI/app exerciser Monkey [127]. The main stages can be summarized as follows:

- Start a new VM with a clean image of the Android OS.
- Connect the device (VM) with ADB.
- Copy the selected APK file in the device (VM) and install it.
- Start the instrumentation process using the package name and the instrumentation code.
- Generate pseudo-random streams of user events into the app on the VM.
- Log the output from the instrumentation process, namely java classes, inter-app communication derived from triggered intents, and any information stemming from network communications, including communication protocols, sockets, IPs, ports, and URLs.

Androtomist allows experienced users to create and import custom Javascript instrumentation code and inject it during dynamic analysis. The code in Listing 5.1 is a simplified example of such a case and can be applied to log possible intents triggered during runtime. Specifically, the code hooks the constructors of the class *android.content.Intent*, which is used by Android apps to trigger intents. On the other hand, Listing 5.2 provides the code which can be used to hook classes and log parameters.

```
/*The primary pieces of information in intents are:
1. Action
    The general action to be performed, such as
    ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.
2. Data
    The data to operate on, such as a person record in the contacts DB,
    expressed as a URI*/

Java.perform(function() {
        var intent = Java.use("android.content.Intent");

//constructor Intent(String action)
        intent.$init.overload('java.lang.String').implementation=function(action)
        {
                console.log("action" + action);
        }

//constructor Intent(Intent o)
        intent.$init.overload('android.content.Intent').implementation=function(o)
        {
                console.log("intent" + o);
        }

//constructor Intent(String action, Uri uri)
        intent.$init.overload('java.lang.String', 'android.net.Uri')
        .implementation=function(action, uri)
        {
                console.log("action" + action + ", url" + uri);
        }
        ...
});
```

Listing 5.1: Example of instrumentation code which hooks intents

```
//Objects to hook
var objectsToLookFor = ["java.net.Socket","java.net.URLConnection","java.net.URL",
"dalvik.system.DexClassLoader"....];

for (var i in objectsToLookFor) {
        Java.perform(function () {
```

```
                Java.choose(objectsToLookFor[i], {
                        "onMatch": function (instance) {
                                console.log("\nProcess has Instantiated instance of: "
                                + objectsToLookFor[i]);
                                console.log("Details: " + instance.toString());
                        },
                        "onComplete": function () {
                        }
                });
        });
}
```

LISTING 5.2: Example of instrumentation code which hook Java classes

The data collected in the DB can be further analyzed to classify an app as malicious or benign. To this end, as it is illustrated in Figure 5.3, the gathered raw data must be first transformed into vectors of features. As already pointed out, the resulting vectors can be exported to a CSV-formatted file and used in the context of any data mining tool.

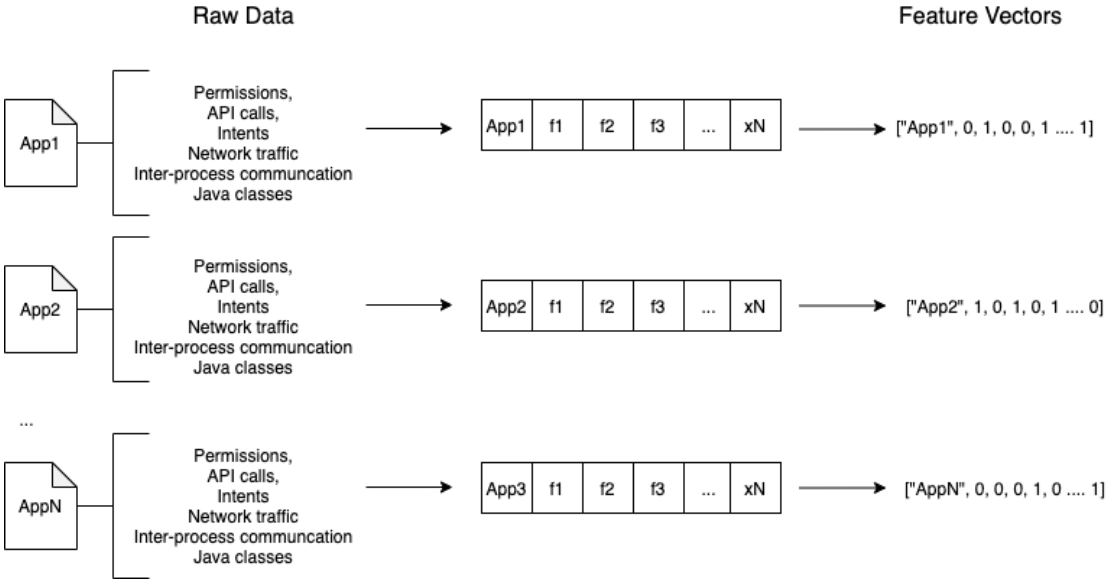

FIGURE 5.3: Feature engineering. f_i corresponds to an existing feature

5.2.3 Dataset

In the context of mobile malware detection, several datasets were built to cover manifold and exceptionally demanding malicious cases. These corpora of malware samples are considered as benchmark datasets for evaluating novel schemes that attempt to distinguish between malware and goodware samples. Initially, this chapter employed three

well-known datasets used extensively in the related literature; the full Drebin dataset [102] dated from 2010 to 2012, a collection (subset) from VirusShare [103] dated from 2014 to 2017, as well as a collection from Androzoo [104] dated from 2017 to 2020. Precisely, the number of malware samples taken from each of the aforementioned datasets was correspondingly 5,560, 10,000, and 1,520, i.e., a total of 17,080 samples. It is important to note that the Drebin dataset contains outdated samples, and therefore, these pieces of malware were only used to cross-validate our results vis-à-vis the related work. On the other hand, Androzoo samples are expected to be more challenging when it comes to detection, given that this corpus contains newer, and therefore more sophisticated malware in comparison to those in the Drebin and VirusShare datasets.

After analysing 17,080 mobile malware samples from the three datasets, a total of more than 50,000 different vectors of features were collected. Note that each vector is a binary representation of each distinct feature. Let us for example take the simplest case of just two apps. App A1 uses permission A, intent B, and initiates communication via HTTP with IP: 1.1.1.1. App A2 on the other hand, uses permission B, intent C, and triggers communication via HTTPS with IP: 1.1.1.2. As a result, as shown in Table 5.1, the vectors of features for both app A1 and app A2 have twice as many binary indicators. Of course, a real-world app would produce a much more lengthy vector.

App	permissionA	permissionB	intentA	intentB	HTTP	HTTPS	1.1.1.1	1.1.1.2
A1	1	0	1	0	1	0	1	0
A2	0	1	0	1	0	1	0	1

TABLE 5.1: Feature vectors for example apps A1 and A2.

Naturally, putting aside the capacity of the data mining tool, the overall number of 50,000 different vectors, meaning 50,000 diverse columns (features), exceeds the ability of Androtomist and virtually and proof-of-concept tool to fetch them in a reasonable time using SQL queries. So, for the needs of our experiments in subsection 5.3.2, a smaller subset of 100 or 120 malware samples per dataset was finally used. Specifically, we randomly selected 100 malware samples from each of the Drebin and VirusShare datasets and 120 from - the more challenging - AndroZoo. This sample reduction technique is also observed in previous work, including those in [128], [129] and [130]. Specifically, common malware families can produce overestimated results. Therefore, we filtered commonplace information among malware families, such as sample year and package

name from each dataset, for the sake of obtaining equally distributed samples across the different malware families. It is important to mention that all the three (reduced) datasets are balanced, containing an equal number of randomly selected malicious and benign apps. The benign apps were collected from Google Play Store [30] and are common across all the three datasets but the AndroZoo, which incorporates 20 more.

5.2.4 Classifiers and metrics

The vast majority of the state-of-the-art methods focused on the detection of malicious samples were ranked based on recall and precision of correct answers combined by the F1 measure. For each instance separately, these measures of correctness are concentrated on providing a binary answer, either a positive (malware class) or a negative (goodware class) one. This indicates that the information about the distribution of positive and negative instances is necessary in order to set this threshold. In this work, we follow exactly the same evaluation procedure to achieve compatibility of comparison with previously reported results. For each dataset, the set of the extracted scores based on the test instances are normalized in the interval of [0,1] per classification model per examined method. To this direction, the estimation of the threshold is set equal to 0.5.

Moreover, we use the AUC of the receiver-operating characteristic curve as the main evaluation measure [131]. Generally, AUC quantifies the effectiveness of each examined approach for all possible score thresholds. As a rule, the value of AUC is extracted by examining the ranking of scores rather than their exact values produced when a method is applied to a dataset. Noticeably, the estimation of the AUC measure is based on all possible thresholds. Besides, this evaluation measure does not depend on the distribution of positive and negative samples.

In this work, a sizable number of well-known and popular supervised ML algorithms are applied. That is, for each dataset, we consider seven of the most widely used binary classification models, namely Logistic Regression (LR), Naïve Bayes, Random Forest, AdaBoost, Support-vector machine (SVM), k-nearest neighbors (k-NN) and Stochastic Gradient Descent (SGD). The majority of the classification algorithms applied falls under eager learning. In this category, supervised learning algorithms attempt to build a general model of the malicious instances, based on the training data. Obviously, the performance of such classifiers strongly depends on the size, quality and representative of

the training data. On the other hand, k-NN is a weak learner (known as lazy learner) as that is not using the training data to construct a general model, but it makes a decision based on information extracted for each sample separately. For each classifier applied, the default values of the parameter settings are used. The general model of each eager classifier is built following the 10-fold cross-validation technique.

Finally, for each dataset, a simple meta-model, that is, an aggregate function, is developed combining all the base classifiers applied either in hybrid or static methods separately. This heterogeneous ensemble is based on the average score resulted by all seven binary classification models for each sample. Regarding the Android malware literature, "ensemble learning" is also exploited by the authors of References [132, 133]

5.3 Evaluation

We assess our approach using both signature and anomaly-based detection. In fact, regarding signature-based detection, Androtomist has the capacity to automatically generate detection signatures by combining static and dynamic features. We first present the signature-based results in subsection 5.3.1, and then we detail on the classification results after training a ML model with the identical set of features.

5.3.1 Signature-based detection

Generally, misuse detection, also known as signature-based detection, relies on known signatures, that is, detection rules aiming to discern between benign and malicious pieces of code. While these systems are capable of identifying previously encountered malicious software and may have a high degree of portability between platforms, they miss to recognize novel instances of malware or variations of known ones. Thus, the detection ability of a misuse detection system, as the one examined in this subsection, primarily depends on the newness of the detection rules the system has been configured with.

In the context of Androtomist, this type of detection involves comparing the features collected during static and dynamic analysis for a new ("unknown") app against those already stored in the DB for all the malicious apps encountered so far. Put differently, to

assess signature-based detection against the three datasets mentioned in section 5.2.3, a sufficient mass of malware samples had to be analyzed beforehand to create signatures.

Therefore, by using the sample reduction technique explained in Section 5.2.3, 1902 random malware samples were chosen (correspondingly 499, 1403 from Drebin, and VirusShare) and being both statically and dynamically analyzed to collect features. This number pertains to those malware samples that (a) are not included in the reduced datasets presented in Section 5.2.3, and (b) have been filtered to prune duplicate malware families by using each app's package name and hash signature.

The derived signatures comprise combinations of features collected from both static and dynamic analysis. A combination incorporates at least one feature from minimum two different categories of features, namely permissions, intents, API calls, network traffic, java classes, and triggered intents. When a new app is analyzed, Androtomist checks for a possible signature match. If the analyzed app yields results that match with two or more signatures, then the app is flagged as malicious.

On top of that, individual features which are spotted regularly among malware, but rarely among goodware are also being labeled as suspicious in the Androtomist's analysis report. Specifically, as an indication, the left column of Table 5.2 lists the top 15 most frequent permissions detected in the utilized set of 1902 malware samples. The right column, on the other hand, contains another 15 permissions which were observed in the same set of samples, but they were scarce or nonexistent in the set of 120 goodware apps selected from Google Play Store. Both these columns are ordered by percentage of occurrence in the set of 1902 pieces of malware. Finally, Table 5.3 contains the 15 most frequent permissions detected in the utilized sample of 120 apps downloaded from Google Play.

Most Common in Malware	**%**	**Least Common in Goodware**	**%**
android.permission.INTERNET	96.2	com.android.browser.permission.READ_HISTORY_BOOKMARKS	12.2
android.permission.READ_PHONE_STATE	91.8	com.android.browser.permission.WRITE_HISTORY_BOOKMARKS	11.9
android.permission.WRITE_EXTERNAL_STORAGE	85.7	android.permission.WRITE	11.4
android.permission.ACCESS_NETWORK_STATE	59.0	android.permission.ACCESS_LOCATION_EXTRA_COMMANDS	9.8
android.permission.ACCESS_WIFI_STATE	46.3	android.permission.INSTALL_PACKAGES	5.8
android.permission.ACCESS_COARSE_LOCATION	42.6	com.android.launcher.permission.READ_SETTINGS	5.5
android.permission.ACCESS_FINE_LOCATION	37.7	android.permission.MOUNT_UNMOUNT_FILESYSTEMS	5.4
android.permission.WAKE_LOCK	26.5	android.permission.RESTART_PACKAGES	4.5
android.permission.VIBRATE	25.3	android.permission.WRITE_APN_SETTINGS	4.1
android.permission.RECEIVE_BOOT_COMPLETED	24.2	android.permission.CHANGE_CONFIGURATION	2.2
com.android.launcher.permission.INSTALL_SHORTCUT	23.3	android.permission.WRITE_SECURE_SETTINGS	1.8
android.permission.CHANGE_WIFI_STATE	19.9	android.permission.ACCESS_COARSE_UPDATES	1.7
android.permission.CALL_PHONE	16.9	android.permission.DELETE_PACKAGES	1.5
android.permission.GET_TASKS	14.9	android.permission.READ_SETTINGS	0.8
android.permission.SEND_SMS	14.8	android.permission.RECEIVE_WAP_PUSH	0.5

TABLE 5.2: Top 30 suspicious permissions.

Most Common in Goodware	**%**
android.permission.INTERNET	100%
android.permission.ACCESS_NETWORK_STATE	96%
android.permission.RECEIVE_BOOT_COMPLETED	86%
android.permission.GET_ACCOUNTS	85%
android.permission.USE_CREDENTIALS	85%
android.permission.WRITE_EXTERNAL_STORAGE	85%
android.permission.WAKE_LOCK	82%
com.google.android.c2dm.permission.RECEIVE	78%
android.permission.MANAGE_ACCOUNTS	77%
android.permission.CAMERA	76%
android.permission.AUTHENTICATE_ACCOUNTS	76%
android.permission.READ_EXTERNAL_STORAGE	72%
android.permission.WRITE_SYNC_SETTINGS	70%
android.permission.READ_SYNC_SETTINGS	70%
android.permission.ACCESS_WIFI_STATE	68%

TABLE 5.3: Top 15 permissions in goodware.

Furthermore, dynamic instrumentation reveals a certain pattern in Java classes used among malware. Especially, several malware apps, i.e., correspondingly 9%, 2%, and 1% of the instances in our AndrooZoo, VirusShare, and Drebin datasets, instantiate the *DexClassLoader* class [134]. This class is used to load other classes from *jar* and *apk* type of files containing a *classes.dex* entry. This class loader can be exploited to execute

code which is not installed as part of the app per se. Another interesting observation is that the *java.net.Socket* [135] class was found quite frequently in malware instances, that is, 15% in AndrooZoo, and 10% in VirusShare and Drebin.

For the sake of comparison, Table 5.4 summarizes the results obtained when the detection engine is (a) fed only with signatures created during static analysis, and (b) signatures created during hybrid analysis. As observed, when the features of static analysis are fused with those acquired via dynamic instrumentation, the detection rate is correspondingly increased by 10%, 3%, 6% for "unknown" apps in the AndroZoo, VirusShare, and Drebin dataset. Obviously, this is because the number and variety of features have increased, thus allowing more of them to match against malicious signatures.

Dataset	**True Positive Rate (TPR)**		**Improvement Rate**
	Static	**Hybrid**	**(Hybrid)**
AndroZoo	86.6	94.1	+7.5
VirusShare	96	99	+3
Drebin	88	94	+6

TABLE 5.4: Signature-based detection scores (%).

5.3.2 Anomaly-based detection

The vectors of features derived from both static and dynamic analysis over the datasets described in subsection 5.2.3 were utilized to conduct anomaly-based detection. To do so, we exploited the open source data mining tool Orange [136]. As detailed in subsection 5.2.4, we employed seven ML algorithms used extensively in the related literature. The selected sampling technique was 10-fold cross-validation.

Table 5.5 provides a comparative overview of the classification performance scores yielded from static analysis against those obtained from hybrid analysis. The best score per metric in the table is underlined. It seems that the proposed approach based on hybrid analysis constantly outperforms the static method when the performance of AUC as well as the effectiveness of binary measures (CA, P, R, and F1) is considered. As can be observed, in almost all cases of all three datasets, hybrid analysis is more effective

than static one; only in Derbin corpora static outperforms hybrid when Naïve Bayes and LR models are applied. This shows that the proposed hybrid analysis is clearly a better option than static one.

Logistic Regression and AdaBoost hybrid models seem to be the overall best performing models for AndroZoo dataset. Moreover, a larger number of the examined classifiers, namely LR, Random Forest, AdaBoost and SVM help hybrid to achieve exceptional results in comparison to static method for the VirusShare dataset. However, the vast majority of the examined hybrid models seems to be exceptionally successful (the only exception is the Naïve Bayes model) in the Drebin dataset. In general, the hybrid approach achieves more stable performance across all three datasets in comparison to the method based on static analysis. It is also remarkable that the biggest improvement of hybrid models is achieved in AndroZoo corpus. It is noticeable that the improvement in performance in terms of AUC of the best hybrid models with respect to that of the best static ones is higher than 10% when the AndroZoo corpora is considered. All these indicate that the hybrid approach is much more reliable and effective in more challenging malware cases where there are new and more sophisticated malware conditions.

The version of our method based on static analysis is also very effective achieving the best results in VirusShare as well as Drebin corpus. As concerns the AndroZoo dataset, static analysis performs poorly. Figures 5.5, 5.6 and 5.7 and Figures 5.8, 5.9 and 5.10 depict the effectiveness of ROC curves when the LR classifier is applied on each dataset for the presented version of the hybrid as well as the version of static analysis, respectively.

		AUC		CA		F1		Precision		Recall	
Dataset	**ML Classifier**	**Static**	**Hybrid**	**Static**	**Hybrid**	**Static**	**Hybrid**	**Static**	**Hybrid**	**Static**	**Hybrid**
AndroZoo	Logistic Regression	0.870	0.978	0.888	0.888	0.894	0.888	0.906	0.890	0.888	0.888
	Naïve Bayes	0.777	0.941	0.811	0.854	0.825	0.854	0.852	0.855	0.811	0.854
	Random Forest	0.938	0.972	0.895	0.888	0.900	0.888	0.910	0.889	0.895	0.888
	k-NN	0.914	0.954	0.867	0.871	0.868	0.871	0.869	0.874	0.867	0.871
	AdaBoost	0.915	0.971	0.909	0.901	0.913	0.901	0.923	0.901	0.909	0.901
	SGD	0.808	0.918	0.881	0.918	0.882	0.918	0.883	0.918	0.881	0.918
	SVM	0.900	0.930	0.846	0.850	0.856	0.849	0.880	0.858	0.846	0.850
VirusShare	Logistic Regression	0.993	1.000	0.975	1.000	0.975	1.000	0.975	1.000	0.975	1.000
	Naïve Bayes	0.975	0.997	0.910	0.965	0.910	0.965	0.910	0.967	0.910	0.965
	Random Forest	0.994	1.000	0.975	0.995	0.975	0.995	0.975	0.995	0.975	0.995
	k-NN	0.990	0.995	0.980	0.990	0.980	0.990	0.981	0.990	0.980	0.990
	AdaBoost	0.965	1.000	0.965	1.000	0.965	1.000	0.965	1.000	0.965	1.000
	SGD	0.955	0.990	0.955	0.990	0.955	0.990	0.957	0.990	0.955	0.990
	SVM	0.983	1.000	0.950	0.995	0.950	0.995	0.955	0.995	0.950	0.995
Drebin	Logistic Regression	0.998	0.990	0.989	0.989	0.989	0.989	0.989	0.989	0.989	0.989
	Naïve Bayes	0.986	0.970	0.923	0.967	0.924	0.967	0.924	0.967	0.923	0.967
	Random Forest	0.998	1.000	0.984	0.995	0.984	0.995	0.984	0.995	0.984	0.995
	k-NN	0.982	0.995	0.962	0.995	0.962	0.995	0.965	0.955	0.962	0.995
	AdaBoost	0.973	1.000	0.973	1.000	0.973	1.000	0.973	1.000	0.973	1.000
	SGD	0.990	1.000	0.989	1.000	0.989	1.000	0.989	1.000	0.989	1.000
	SVM	0.999	1.000	0.984	1.000	0.984	1.000	0.984	1.000	0.984	1.000

TABLE 5.5: Results per dataset and classification performance metric.

5.4 Discussion

Regarding signature-based detection, the scores in Table 5.4 suggest that when static analysis is combined with dynamic instrumentation into a hybrid approach, the detection rate can be augmented by at least 3% and up to 10%. Another salient observation is that the classification performance differs significantly among the three datasets. Specifically, as observed from the same Table, the detection rate vary considerably from 87% to 96% for static analysis, but this divergence is much lesser, from 94% to 99%, for the hybrid scheme. This also support the view that hybrid analysis by means of dynamic instrumentation leads to more robust classification results. Particularly, the detection rate for - the more challenging - AndroZoo dataset, is the lowest among all scores when using only static analysis, but it augments by +10% when is backed up by the results of dynamic instrumentation. The same observation stands true for the VirusShare or Drebin datasets. This results actually do not come as a surprise, and is verified by relevant works in the literature [137].

The same general inference is deduced from Table 5.5 with respect to the results obtained from anomaly-based detection. That is, in their vast majority, the scores regarding the most two important classification performance metrics, namely AUC and F1, are superior when hybrid analysis is involved, irrespective of the dataset and the ML classifier used.

To further demonstrate the usefulness of the examined approaches, Figure 5.4 depicts the evaluation results (AUC) of hybrid and static analysis in the employed datasets when applying the seven classifiers. As can be clearly seen, the basic patterns are the same across all the datasets. The proposed hybrid method is more effective, surpassing the approach based on static analysis in almost all cases. As already pointed out in subsection 5.3.2, the only exceptions are LR and Naïve bays in the Drebin dataset. In both VirusShare and Drebin corpora, the static method is also very effective in all cases.

With respect to the results on AndroZoo corpora, the results suggest that the use of hybrid analysis helps the approach to become more stable. As can be seen, the proposed hybrid approach outperforms in all cases the static method, by a large margin (more than 10%). Recall from subsection 5.2.3 that the AndroZoo dataset embraces newer instances of malware in contrast to VirusShare and Drebin ones. It seems therefore that this challenging condition significantly affects the performance of static models. The same patterns are provided in case where F1 measure is considered as it is illustrated in Figure 8 in the Appendix.

In addition, we also consider the combination of the base classification models (seven classifiers) by averaging their answers per sample on each dataset separately. Figure 5.12 demonstrates the percentage in performance (AUC) of both these ensemble methods based on the hybrid and static analysis, respectively. Especially, vis-à-vis the results of Table 5.5, the effectiveness of both the ensemble approaches in terms of AUC, is better than that of any single base classifier in the vast majority of the cases. Although, the version of static analysis is effective enough, the version based on hybrid analysis consistently outperforms its performance. Overall, as can be observed, the proposed hybrid ensemble is the most effective one surpassing that of static analysis and improving the reported results of all base classifiers across all the three datasets considered. With respect to the results on AndroZoo corpora, the proposed hybrid ensemble clearly seems to be the top-performing model which achieves the best results not only among all

base models, but also in comparison with the static one. The improvement in average performance of the hybrid ensemble model with respect to that of the static is higher than 4%.

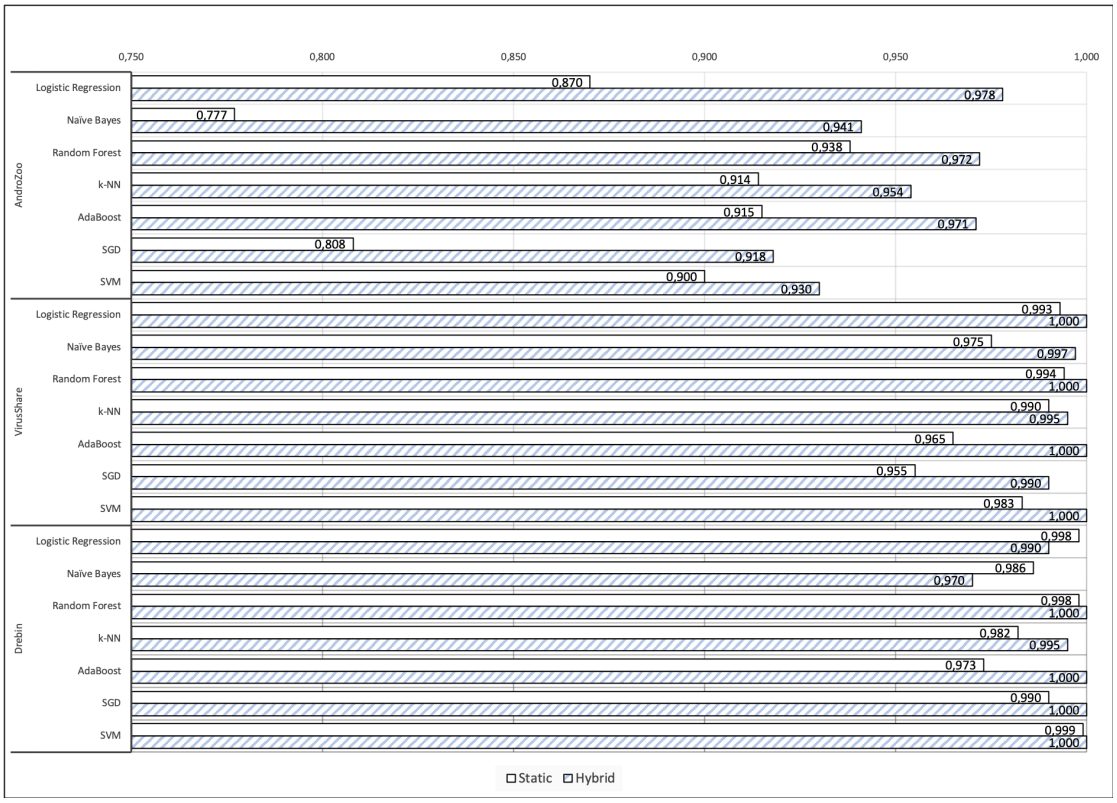

FIGURE 5.4: The performance (AUC) of the proposed static and hybrid analysis on AndroZoo, VirusShare, and Drebin corpora for different classification models

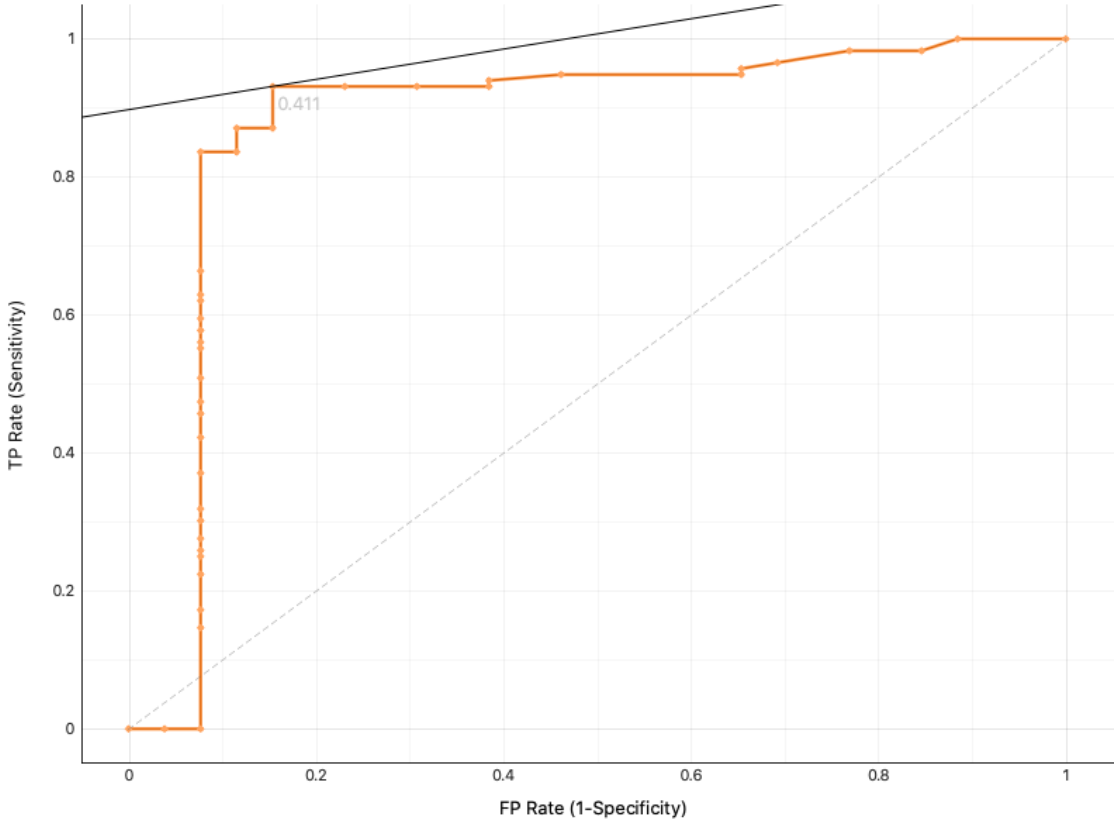

FIGURE 5.5: AndroZoo: ROC (Static Analysis)

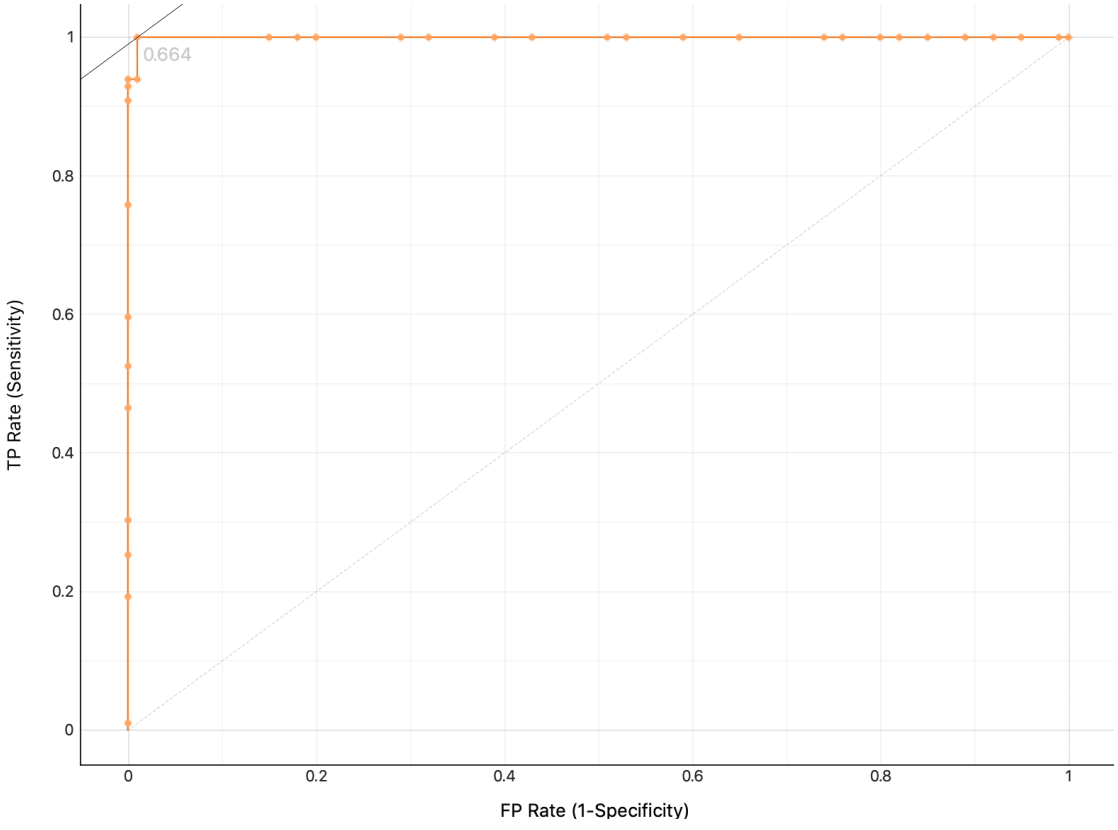

FIGURE 5.6: VirusShare: ROC (Static Analysis)

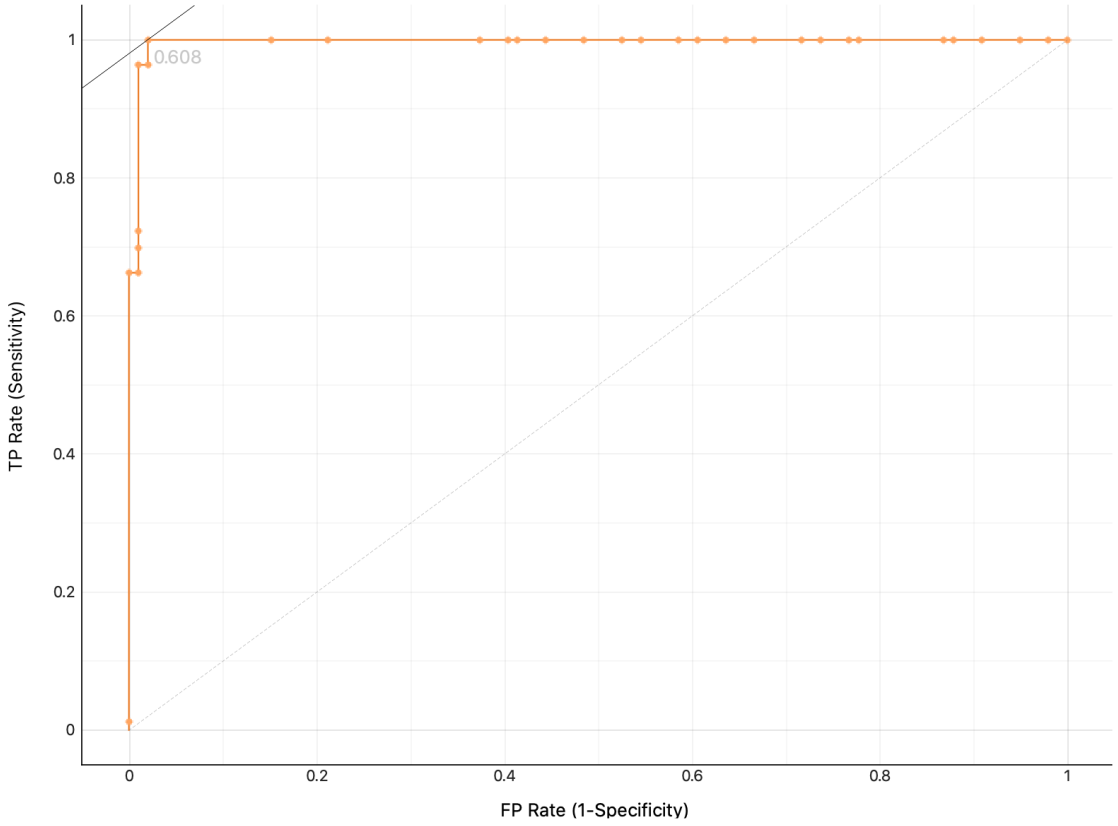

FIGURE 5.7: Drebin: ROC (Static Analysis)

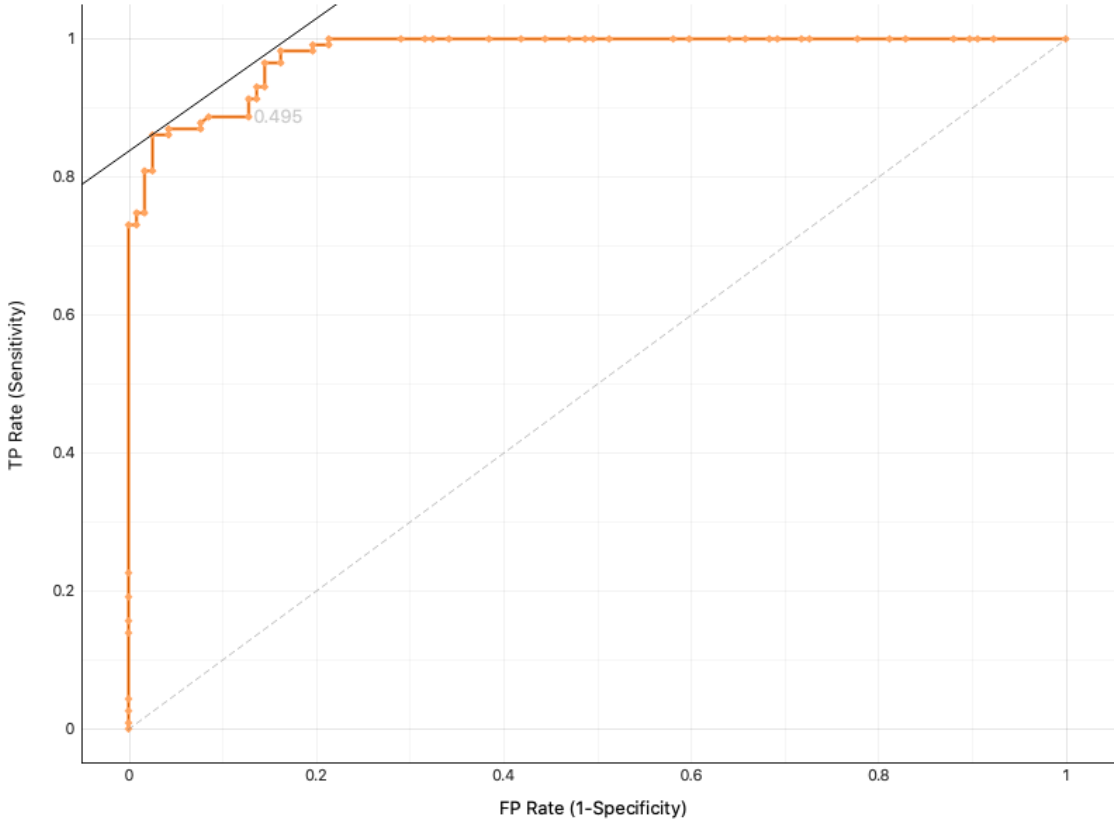

FIGURE 5.8: AndroZoo: ROC (Hybrid Analysis)

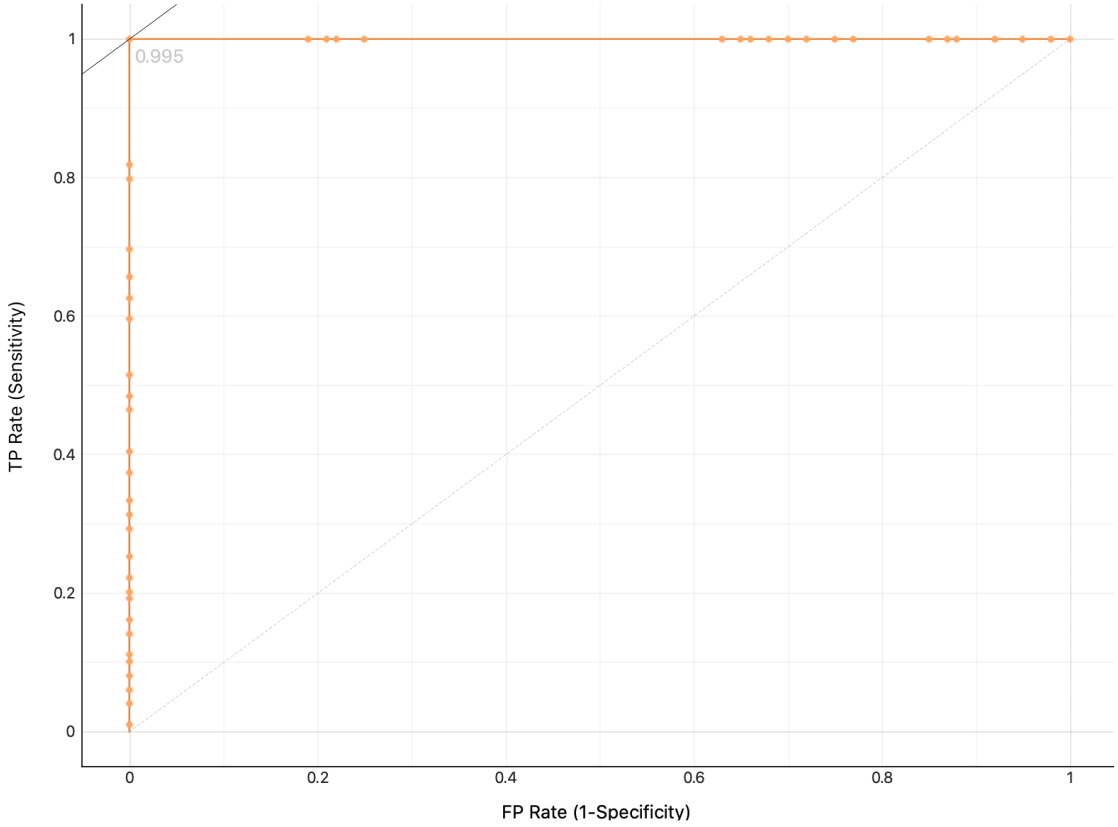

FIGURE 5.9: VirusShare: ROC (Hybrid Analysis)

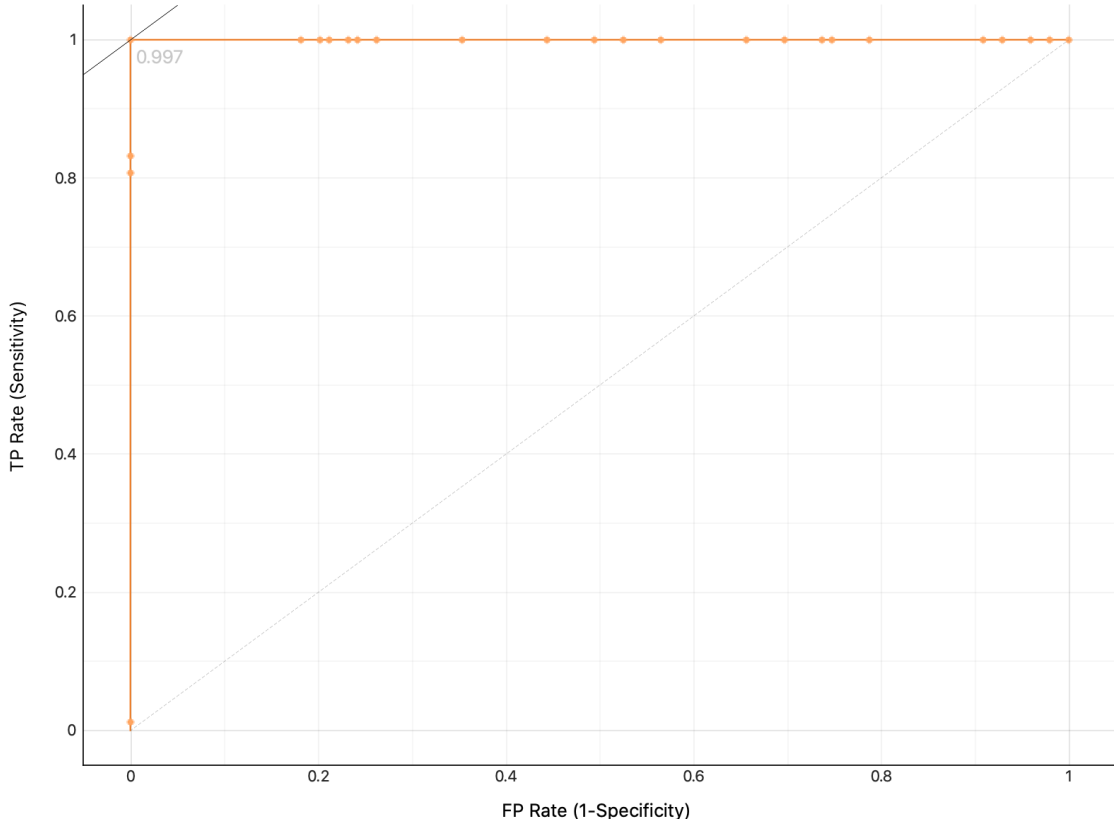

FIGURE 5.10: Drebin: ROC (Hybrid Analysis)

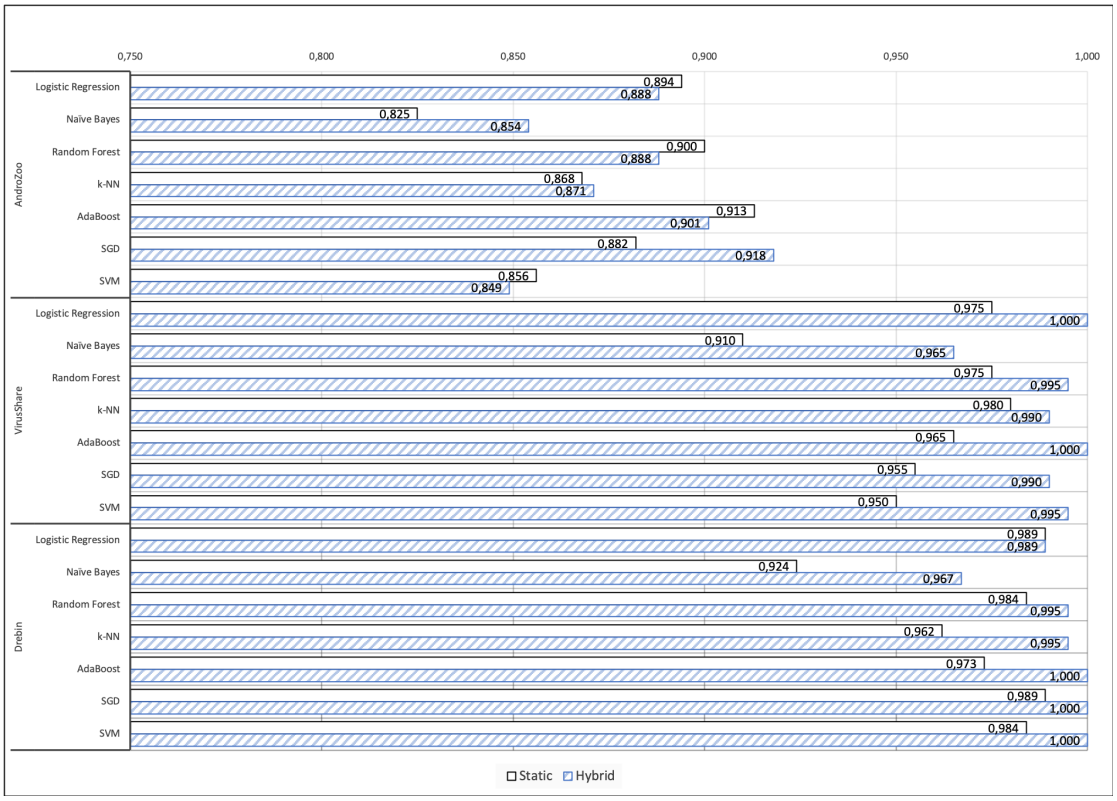

FIGURE 5.11: The performance (F1) of the proposed static and hybrid analysis on AndroZoo, VirusShare, and Drebin corpora for different classification models

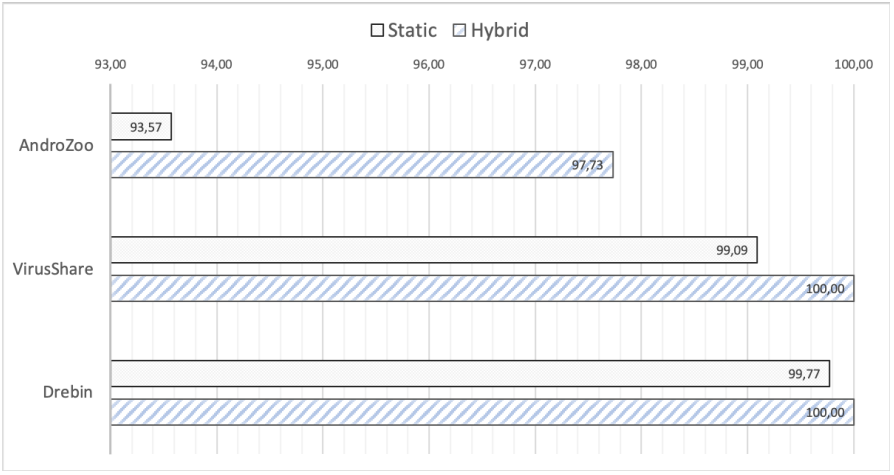

FIGURE 5.12: The percentage in performance (AUC) of hybrid and static methods by averaging the output of the base classifiers on the three datasets

Table 5.6 demonstrates the improvement in performance, i.e., difference of AUC scores, of the individual and ensemble models when the hybrid analysis is considered in comparison to the case where only the static one is used. Statistical significance of these differences is estimated using an approximate randomization test [138]. The null hypothesis is that there is no difference between the two cases and we reject this hypothesis

when p <0.05. As can be seen, in most of the cases, the models extracted from the hybrid analysis are significantly (p <0.05) better than static ones. Notable exception is Naïve Bayes on Derbin dataset where the variance with the corresponding static model is slightly decreased. In AndroZoo dataset, all hybrid base models are more improved than the corresponding static ones as well as the hybrid ensemble model gains more than the static one. This indicates that hybrid analysis is superior, able to handle challenging datasets and better suits more sophisticated malware.

On the other hand, the differences between the hybrid and static ensembles are not statistically significant on the VirusShare and Drebin data sets. As concerns the individual models, in the first corpus, the results of Naïve Bayes, AdaBoost and SGD are improved the most, while in the latter only the difference in results of k-NN and SGD models appears to be significant. The betterment of base models based on static analysis seems not to be correlated with the relative increase in the number of malware samples used to extract the static classification models. Apparently, an increasing number of malware instances on AndroZoo corpus does not help static analysis to achieve better performance.

Data Set	LR	Naïve Bayes	Random Forest	k-NN	AdaBoost	SGD	SVM	Ensemble
AndroZoo	**0.108**	**0.164**	**0.034**	**0.040**	**0.056**	**0.110**	**0.030**	**0.042**
VirusShare	0.007	**0.022**	0.006	0.005	**0.035**	**0.035**	0.017	0.009
Drebin	-0.008	-0.016	0.002	**0.013**	0.027	**0.010**	0.001	0.002

TABLE 5.6: Improvement in performance (difference in AUC) between methods using hybrid analysis (base models and ensemble) and static analysis (base models and ensemble). Statistically significant differences ($p < 0.05$) are indicated in boldface. A negative value means a decrease in performance.

A key factor that affects the performance of malware detection methods is the importance of features contained in malware samples. To study which feature matters more for detecting malware in both the static and hybrid method, we concentrate on the Logistic Regression model, which according to Table 5.5, seems to be the top performer across all datasets. To this end, Figure 5.13 illustrates the average feature importance scores on all three datasets when Logistic Regression is applied for both hybrid and static method. As feature importance scores have been directly used the average coefficients of each feature category explored following a linear regression algorithm. More specifically, the features importance scores are assigned by coefficients calculated as part of a linear regression model per set of features.

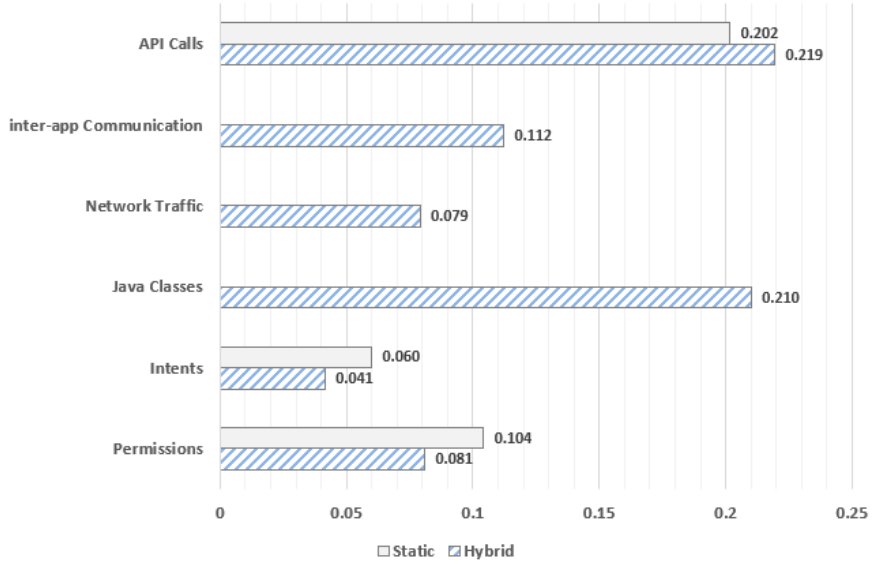

FIGURE 5.13: Average Feature Importance scores of static and hybrid analysis on all three datasets for a varying set of feature categories.

As already mentioned, six categories of features are explored in total; permissions, intents, API calls, network traffic, inter-app communication and Java classes, where only the first three apply to static analysis. In general, from Figure 5.13 it is observed that the API calls category is more influential and tends to improve both static and hybrid methods. It is also noteworthy that the category of Java classes achieves also significantly high average importance scores enhancing the effectiveness of hybrid method on all three datasets. On the other hand, the contribution of both intents and permissions categories is lower not only in static, but also in hybrid method. Lastly, network traffic is a category that includes features with slightly better coefficients than intents and permissions categories when hybrid analysis is considered. The case of inter-app communication category seems also to be of particular importance for hybrid method. However, its average coefficient scores similar to intents and permissions categories are significantly smaller than those of both API calls and Java classes categories on hybrid approach.

For the sake of completeness, Table 5.7 provides a comparison of the best-case accuracy performance between HADM [139], SAMADroid [67], Bridemaid [140], Surendran et al [141], and Androtomist. The comparison reveals that in all the applicable cases Androtomist is able to improve the best results per matching dataset. Precisely, Androtomist surpasses the rest of the state-of-the-art detection systems when using malware from

the Drebin or VirusShare datasets. A direct comparison with Bridemaid is not feasible because this system was evaluated using a mixture of malware samples from different datasets, i.e., *Genome*, which is not shared anymore, and *Contagio* which is considered obsolete. Naturally, as a rule of thumb, the performance of any detection model depends on the malware samples used during both its evaluation and testing phases [142].

For easy reference, the same Table lists the different kind of features collected during the analysis phase per detection system. Recall from section 5.2.1 that during static analysis, Androtomist gathers the most common features seen in the relevant work, namely permissions, intents, and API calls. The same static features are collected in [139, 67]. On the other hand, for dynamic analysis, Androtomist relies on dynamic instrumentation, enabling the extraction of a wide variety of features. Contrariwise to other schemes included in Table 5.7, system calls are consciously neglected as a feature, since its exploitation has been significantly restricted from Android v.7.0 onward [143]. Precisely, similar to SELinux [144], Android uses the *Seccomp* kernel module [18] to further limit access to the kernel by blocking certain system calls. Put simply, starting from Android v.7.0, Seccomp kernel has been applied to processes in the media frameworks. Moreover, as of v.8.0, a Seccomp filter is installed into the *zygote*, i.e., the forking handling process from which all Android apps are derived. Seccomp blocks access to certain system calls, such as swapon/swapoff, which were the root cause for certain security attacks [145]. Additionally, Seccomp blocks key control system calls, which are not useful to apps. Therefore, when analysing newer Android apps, as those in the AndroZoo dataset, the merit of system calls as a feature in malware detection schemes has been greatly diminished. Last but not least, some of these tools such as SAMADroid are intended to run on the smartphone. In this case, the app's features are locally collected when the user interacts with it. Then, the collected features are sent to a remote host in which a decision is made about the app. Androtomist on the other hand only runs on a remote host, and therefore the app must be analysed prior to its installation.

5.5 Related Work

As of today, the topic of mobile malware detection via the use of hybrid schemes has received significant attention in the Android security literature [6, 9]. This section along

Detection System	Groups of features collected	Accuracy (AC) achieved per dataset			
		Drebin	Virusshare	AndroZoo	Mixed
HADM [139]	**API calls, Permissions, Intents, System calls**	N/A	94.70%	N/A	N/A
SAMADroid [67]	**API calls, Permissions, Intents, System calls**	99.07%	N/A	N/A	N/A
Bridemaid [140]	**n-grams classification, SMS, System calls,**	N/A	N/A	N/A	99.70%
Surendran et al [141]	**Admin privileges Permissions, API calls, System calls**	99.00%	N/A	N/A	97.00%
Androtomist	**API calls, Permissions, Intents, Network traffic, Java classes, Inter-process communication**	100%	100%	91.80%	N/A

TABLE 5.7: Comparison of state-of-the-art hybrid systems in terms of collected features and classification accuracy (best case). "Mixed" means a mixed, but not strictly defined dataset, containing records from Drebin, Genome, and Contagio datasets

with Table 5.8 offer a succinct chronologically arranged review of the most notable and recent works on this topic. Specifically, we concentrate on contributions published over the last four years, that is, from 2015 to 2020.

Patel et al. [146] proposed a hybrid approach for mobile malware detection, which combines app analysis and ML techniques to classify apps as benign or malicious. The features collected vary from app's permissions, intents, and API calls during static analysis to network traffic in the course of dynamic analysis. According to the authors, it takes around 8 to 11 min to scan an app. Cam et al. [128] presented *uitHyDroid*. Their system firstly uses static analysis to collect user's interface elements, followed by dynamic analysis to capture possible inter-app communication and link partial sensitive data flows stemming from static analysis. However, their proposed system has been evaluated with a rather small dataset of 23 apps.

Tuan et al. [147] proposed eDSDroid, an analysis tool which uses both static and dynamic analysis to identify malware targeting inter-app communication. Specifically, at a first stage, static analysis detects information leakage, while dynamic analysis is employed to help eliminate the false positives of the first stage. On the downside,

their approach has only been evaluated with a tiny corpus of 3 apps. Martinelli et al. [140] introduced *BRIDEMAID*. Their system operates in three consequent steps, namely static, meta-data, and dynamic. During static analysis, BRIDEMAID decompiles the apk and analyzes the source code for finding possible similarities in the executed actions. This is done with the help of *n-grams*. Dynamic analysis exploits both ML classifiers and security policies to control suspicious activities related to text messages, system call invocations, and administrator privilege abuses. In their evaluations, the authors used the Drebin dataset, which by now is mostly considered outdated.

Xu et al. [139] proposed *HADM*. With HADM, the features extracted during static analysis are converted into vector-based representations, while those fetched during dynamic analysis, namely system call invocations, are converted into vector-based and graph-based representations. Deep learning techniques were used to train a neural network for each of the vector sets. Finally, the hierarchical multiple kernel learning technique was applied with the purpose of combining different kernel learning results from diverse features, and thus improve the classification accuracy. According to the authors, their model is weak against code obfuscation techniques because dynamic analysis is guided based on the results obtained from static analysis.

Ali-Gombe et al. [129] presented *AspectDroid* an hybrid analysis system, which analyzes Android apps to detect unwanted or suspicious activities. The proposed system employs static bytecode instrumentation to provide efficient dataflow analysis. However, static instrumentation is unable to detect apps which use anti-unpacking and anti-repackaging obfuscation mechanisms. Furthermore, the authors used the Drebin dataset to evaluate their model, which is mostly considered obsolete nowadays. Finally, their tests were conducted on an outdated Android version (ver. 6.0).

Arshad et al. [67] introduced a hybrid malware detection scheme, namely *SAMADroid*. According to the authors, SAMADroid delivers high detection accuracy by combining static and dynamic analysis, which run both in a local and remote fashion. ML is used to detect malicious behavior of unknown apps and to correctly classify them. On the negative side, their scheme was tested on an outdated Android version (ver. 5.1).

Tsutano et al. [148] contributed an Android analysis framework called *JITANA*. According to the authors, JITANA can be used to identify inter-app communications as well as potential malicious collaborations among apps. A number of tests conducted

by the authors demonstrated the effectiveness of JITANA in three different Android devices with multiple installed apps. On the negative side, the authors neither provide the detection rate and precision achieved nor any result about malicious apps which do not use intents.

Wang et al. [130] proposed a hybrid approach coined *DirectDroid*, which combines fuzzy logic with a new type of app testing process called "on-demand forced execution". According to the authors, by using this technique one can trigger more hidden malicious behaviors. On top of that, DirectDroid avoids app crashes with the use of fuzzy logic and static instrumentation. DirectDroid was evaluated against 951 malware samples from 3 datasets, namely Drebin, GitHub, and AMD, and it is reported that it can detect more malicious behaviors via a method called "augmenting fuzzing". Nevertheless, given that code obfuscation techniques render detection harder for virtually any static approach, DirectDroid cannot detect malware which employs code obfuscation or have malicious code in native payload. Put simply, any obfuscated code is invisible to DirectDroid's static analyzer.

Fang et al. [149] suggested a hybrid analysis method which performs dynamic analysis on the results of static analysis. During the static analysis phase, they decompiled the app's APK file to extract permissions from the manifest file as well as any occurrence of API features existing in *smali* files. Regarding dynamic analysis, they generated user input and logged the system calls. Finally, they used ML algorithms to classify apps. After experimentation, they reported a detection accuracy of 94.6% using a balanced dataset of 4,000 benign and 4,000 malicious apps.

Lately, Surendran et al. [141] implemented a Tree Augmented Naive Bayes (TAN) model that combines the classifier output variables pertaining to static and dynamic features, namely API calls, permissions and system calls, to detect malicious behavior. Their experiments showed an accuracy of up to 97%. The authors did not provide information about the Android version they employed or about how they generated user input events for the purposes of dynamic analysis.

As observed from Table 5.8, scarce research is hitherto conducted toward the use of dynamic instrumentation in hybrid detection approaches, as none of the above mentioned contributions exploits such a scheme. Furthermore, none of the previous work capitalises on information stemming from hooking Java classes. Moreover, none of them is offered

Work	Year	Method	Dataset	Limitations
[146]	2015	Permissions, intents, API calls, network traffic. Employs ML to classify apps.	DroidKin, Contagio	Slow (8 to 10 min to analyze an app)
[128]	2017	Analyzes APK to fetch possible data flows. Analyzes UI elements. Detects inter-app communication.	DroidBench	Small dataset (23 samples)
[147]	2017	Analyzes APK to retrieve API calls and data flows. Detects inter-app communication and data leakage.	ToyApps	Small dataset (3 samples)
[140]	2017	Logs system calls, privileges, and text messages. Uses n-gram classification.	Drebin, Genome, Contagio	Outdated dataset
[139]	2018	Combines app's features with others derived by deep learning collected during static and dynamic analysis to classify an app.	VirusShare	Unable to detect malware which uses obfuscation techniques
[129]	2018	Uses static bytecode instrumentation and dataflow analysis.	Drebin	Small dataset (100 samples) Outdated Android version (6.0)
[67]	2018	Creates feature vectors from permissions, intents, API calls, system calls, and uses ML to classify apps.	Drebin	Outdated Android version (5.1)
[148]	2019	Detects inter-app communication.	Unspecified	Metrics and results are insufficient
[130]	2019	Augments fuzzing with on-demand forced execution	Drebin, AMD, GitHub	Uses obsolete Android version (4.4) Weak against obfuscation techniques
[149]	2019	Analyzes permissions, API references, and system calls	VirusShare	Limited information is given about the examined malware samples collected
[141]	2020	Analyzes permissions, API references, and system calls and uses TAN to predict malicious behavior	Drebin, AMD AndroZoo, Github	Android version is not provided User input method is not provided

TABLE 5.8: Outline of the related work

as a web app and only the work in [148] is open source. ML techniques are exploited in half of the works included in the Table, namely [146, 140, 139, 67, 149]. Finally, it is worth mentioning that there exists a number of free online Android app analysis tools, including *AMAaaS* [150] and *VirusTotal* [151], which reportedly apply hybrid analysis. Unfortunately, AMAaaS's source code is kept private, and the tool provides insufficient

information regarding its analysis and classification engines. On the other hand, VirusTotal is not an *apk* analysis tool, but rather a vote-based system that leverages analysis from multiple engines.

5.6 Conclusions

This chapter introduced Androtomist, an automated and configurable hybrid analysis tool, which combines static analysis with dynamic instrumentation to analyse app behavior in the Android platform. The results over three different datasets show that this dual analysis of mobile apps can significantly enhance the detection capabilities of a detection model. Furthermore, a comparison between static and hybrid analysis by means of instrumentation reveals asymmetry, that is, the latter is able to yield better classification results even throughout diverse datasets. By collating the performance of our proposal vis-à-vis similar state-of-the-art detection systems we demonstrate its superiority. Ultimately, Androtomist not only offers an easy-to-use environment for virtually everyone to analyze mobile apps, but it is also configurable to the point where researchers can import custom dynamic instrumentation hooks and scripts.

In the introduced work, both the proposed approaches use a randomly chosen subset of malware samples per dataset, a typical choice in similar studies. It could be interesting to investigate how this selection process can be optimized by means of clustering toward including a set of the best possible representative malware samples showing a common behavioural pattern. Both the heterogeneous ensemble approaches rely on base models with default parameter settings. This could be used to further enrich the pool of our base verifiers considering several versions of the same approach with different fixed and tuned parameter settings. Another future work direction could focus on combining the methods based on hybrid and static analysis in a more complex approach.

Chapter 6

Improving Android malware detection through dimensionality reduction techniques

6.1 Introduction

Up to now, state-of-the-art mobile malware detection approaches in the literature have been evaluated using older datasets, including Contagio mobile [81], MalGenome [105], and Drebin [102]. As previously stated in Chapters 4 and 5, this raises an issue of whether such schemes can accurately detect current pieces of malware. In this chapter, we use a wide range of base verifiers covering the most prominent classification algorithms in the relevant literature. Moreover, we apply and compare two of the most common dimensionality reduction techniques, namely Principal Component Analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) on a collection of concurrent malware from the AndroZoo dataset [104], dated from 2017 to 2020. Then, we propose two ensemble learning approaches. First, a simple ensemble which combines the outputs of base models. These outputs are extracted dealing exclusively with either the original or a transformed feature set, respectively. Second, a more complicated ensemble approach that aggregates the answers of a larger size and possibly heterogeneous set of base models constructed from both original and transformed (Original, PCA and

t-SNE) feature sets. Experimental results on Andozoo benchmark dataset exhibit the effectiveness of both the aforementioned approaches.

The main contributions of this study are:

- We propose a simple ensemble approach by aggregating the output of each instance separately, for a number of malware detection base models. The combination of base models achieves the best results in comparison to each particular base model on the Androzoo corpus examined. This evidently demonstrates that an ensemble of classifiers based on a larger size and probably heterogeneous base models is the most appropriate and able to handle challenging malware detection scenarios, and thus can further improve the performance of each individual base classifiers.
- We examine the usefulness of two well-known dimensionality reduction technique namely, PCA and t-SNE when exclusively applied on malware detection base verifiers as well as ensembles, respectively. It is demonstrated that both transformations are able to considerably increase the performance of each base model as well as the proposed ensembles. However, the implementation of t-SNE is more effective than PCA transformation and assists base models and malware detection ensemble methods to further increase their effectiveness in all the examined cases.
- We report detailed experimental results on malware detection under the Androzoo dataset that are directly compared with state-of-the-art methods under the same settings. The performance of the approaches presented in this chapter is quite competitive to the best results reported so far for this malware detection corpus, demonstrating that the proposed methods can be an efficient and effective alternative toward more sophisticated malware detection systems.

6.2 The Proposed Method

The core idea of our proposed approach is to exploit powerful low-level features, like app's permissions and intents and apply common techniques toward reducing dimensionality and extracting more compact and less sparse representations of samples. In a more detailed description, we use a set of popular and widely used classification algorithms as malware detection base models. More, specifically, we implemented eight

classifiers handling the dataset examined either based only on the original feature set or utilizing entirely the transformed set of features accrued by applying dimensionality reduction techniques. Then, simple heterogeneous ensemble approaches are proposed by fusing the output of all base models for each instance separately. These meta-models are developed exploring two options. The first one is to use exclusively the available base models extracted from either the initial feature set of data or the modified set of features. The second option concerns to enrich the meta learner with a larger and possibly heterogeneous set of base models. Thus, we consider the answers of all the available malware detection base models extracted (based on both the Original and reduced feature set). In this way, a heterogeneous ensemble included a mixed set of base models is formed.

6.2.1 Dimensionality Reduction

Dimensionality reduction is one of the most useful processes that decisively contributes to analyze large volumes of data providing a simple way to transform them from the original high-dimensional and sparse feature space into a small set of new features. Several algebraic techniques have been applied in time series analysis for dimensionality reduction providing a less sparse representation of signals. This way, the reduced space is less redundant, and the resulting data are more compact and less noisy. In this study, we consider the two most widely used dimensionality reduction techniques, namely PCA and t-SNE.

- Principal Component Analysis (PCA) is one of the simpler and well-known linear transformation techniques. Specifically, it is one of the most multivariate and state-of-the-art statistical techniques in the field of dimensionality reduction achieving to reduce the dimensions of a d-dimensional dataset by projecting it onto a new(k)-dimensional subspace (where $k < d$) following some of the most important linear algebra concepts in order to increase the computational efficiency, while retaining most of the information. More specifically, PCA analysis aims to identify patterns in data detecting the correlation between variables and yielding the directions or eigenvectors (the principal components) that maximize the variance of the data. In other words, by applying PCA the observations are represented by their projections as well as the set of variables are represented by their correlations. It is also

worth noticing that PCA based on a deterministic technique in a consequence of utilizing a strictly mathematical approach. To make it clearer, it is important to be mentioned that in the approximation of a small dimensional space, PCA can be accomplished by a matrix algebra technique obtaining the eigenvectors and eigenvalues from the covariance matrix or correlation matrix [152]. In this way, a less sparse matrix with significantly lower dimension in comparison with the original matrix is built.

- t-distributed stochastic neighbor embedding (t-SNE) is a nonlinear dimensionality reduction technique which leads to a powerful and flexible visualization of high-dimensional data. It uses the local relationships between points to create a low-dimensional mapping. This allows it to capture non-linear structure. It creates a probability distribution using the Gaussian distribution that defines the relationships between the points in high-dimensional space. Then, it employs the Student t-distribution to recreate the probability distribution in low-dimensional space. Unlike methods like PCA, t-SNE is non-convex, meaning it has multiple local minimal and is therefore much more difficult to optimize. This technique enables the correct visualization of data which lie on curved manifolds or which incorporate clusters of complex shape. In this way, t-SNE opens the way towards a visual inspection of nonlinear phenomena in the given data. t-SNE is a more recent DR technique that belongs to the class of non-parameric techniques [153].

6.3 Experiments

6.3.1 Description of Data

In the context of malware detection between 2010 and 2019, several corpora were built covering multiple degrees of difficulty and incorporating older or newer malware/goodware instances. These corpora are usually exploited to evaluate new malware detection approaches. In this chapter, we consider the most contemporary benchmark corpora, namely AndroZoo [104]. This is a well-known and widely used real-world collection of Android apps collected from assorted sources, including the official Google Play app market [30]. Particularly, the collection of AndroZoo apps we used in the context of this chapter is dated from 2017 to 2020 and enclosed 1K malware apps, each of which

has been cross-examined by a large number of antivirus products. It is important to be mentioned that AndroZoo can be considered as a challenging corpora since it includes new and more sophisticated malware samples in comparison to older datasets, namely Drebin [102]. We also chose a set of 1K benign apps from Google Play.

Static analysis was performed on all the collected apps using the open-source tool Androtomist [10]. Specifically, each app was decompiled to get the *Manifest.xml* file and log permissions and intents to create a feature vector. Each vector is a binary representation of each distinct feature. For example, given two apps, a1 and a2, where the first uses permissions p1, p2, p3, and intent i1 and the second uses permissions p2, p3, p4 and intents i1, i2, the analysis leads to a 6-dimensional feature vector (p1, p2, p3, p4, i1, i2), and thus the feature vectors for these two apps will be (1, 1, 1, 0, 1, 0) and (0, 1, 1, 1, 1, 1), respectively. Naturally, the scrutiny of a real-world app results to a much more lengthy vector. Precisely, the analysis of the largest set of malware and benign apps used in our experiments, i.e., 1K malware instances along with all of the 1K benign apps collected from Google Play, yielded 1,002-dimensional feature vectors.

6.3.2 Experimental Setup

As already pointed out, the entire dataset of malware apps used contains a collection of 1K malware apps randomly selected by AndroZoo corpus. Moreover, 1K benign apps were downloaded by the Google Play to comprise the negative category.

To set the base malware detection models eight well-known classifiers were applied, namely AdaBoost, k-nearest neighbors (k-NN), Logistic Regression (LR), Naive Bayes (NB), Multilayer Perceptron (MLP), Stochastic Gradient Descent (SGD), Support Vector Machine (SVM) and Random forests (RF). It is important to note that seven of these classification algorithms applied fall under eager learning. In this category, supervised learning algorithms attempt to construct a general model of the malware detection samples, building upon the training data. Apparently, the effectiveness of such classifiers is completely determined by the size, quality and representative of the training set. On the other hand, k-NN is a weak learner (known as lazy learner) as that makes a decision in terms of information extracted per sample separately without needing the training part of data to construct a general model. The construction of each eager classification model is built following the 10-fold cross-validation technique. In this technique, a number of

10 different randomly segmented and equally sized sub-datasets is generated from the initial set of data. To extract each malware detection base model we consider the set of parameter settings with the default values.

As already mentioned, apart from the original samples, two very popular and widely used dimensionality reduction technique is applied, namely PCA and t-SNE. Each base model is evaluated on either PCA or t-SNE new feature set. In this way, we examine the performance of the eight base models considered by handling the Original, PCA and t-SNE set of features, separately.

A simple meta-model is developed combining the output of all base classifiers applied following two options: in the former, we combine the answers of the 8 base models (AdaBoost, k-NN, LR, NB, MLP, SGD, SVM and RF) based exclusively on the Original, PCA and t-SNE representations, separately. In the latter, a complicated and possibly more heterogeneous ensemble model is constructed to combine the outputs of all malware detection base models. More specifically, the answers of 24 base models based on Original, PCA and t-SNE representations are totally combined to build a mixed meta-model. For each one of the above ensemble malware detection models presented, the outputs of base models is merged per instance separately by following two common aggregate functions namely, average (AVG) and majority vote (MV) technique.

The vast majority of the state-of-the-art methods in the detection of malware cases are mainly evaluated by using binary measures of correctness. Noticeably, these measures always provide a binary answer, either a positive (malware class) or a negative (benign class) one for each examined instance separately. This indicates that the information about the distribution of positive and negative instances is necessary for the sake of setting a threshold value. In this chapter, we follow exactly the same evaluation procedure to achieve compatibility of comparison with previously reported results. More specifically, the classification performance measure of accuracy is considered, where TP, TN, FP, and FN represent correspondingly True Positives, True Negatives, False Positives, and False Negatives.

- Accuracy : $\frac{TP+TN}{TP+TN+FP+FN}$. The number of correctly classified patterns over the total number of patterns in the sample.

For each dataset, the set of the extracted scores based on the test instances are normalized in the interval of [0,1] per classification model per examined method. To this direction, the estimation of the threshold is set equal to 0.5. Moreover, we use the AUC of the receiver-operating characteristic curve as the main evaluation measure [154].

- Area Under Curve (AUC): The higher positive-over-negative value ranking capability of a classifier.

The AUC metric quantifies the effectiveness of each examined approach considering all possible threshold values. In general, the AUC value is extracted by examining the ranking scores rather than their exact values produced when a method is applied to a dataset. Noticeably, the estimation of the AUC measure is based on all possible thresholds.

6.3.3 Results

To evaluate the improvement in classification effectiveness when dimensionality reduction techniques are in force, we employ 8 well-known classifiers as well as ensemble approaches on benchmark Androzoo corpora following two options. The first one is to use the entire set of features on the dataset used (the case of a=1). The second option is to select a random subspace of the initial feature set. In this case, we use a fix rate equal to 0.5 (a=0.5). Table 6.1 reports the results of AUC and Accuracy (AC) measures of all malware detection base models (AdaBoost, k-NN, LR, NB, MLP, SGD, RF and SVM), utilizing either Original or transformed (based on PCA or t-SNE techniques) feature sets of data when a=1 or a=0.5, respectively.

As concerns the performance of dimensional reduction techniques, it seems that the proposed malware detection base models based on PCA or t-SNE transformation are particularly effective and outperform the corresponding original ones in the most of the cases, especially when the whole feature set is considered (a=1). It is discernible that t-SNE aids malware base models to achieve improved results in comparison to PCA transformation when a=1.

On the other hand, we also observe that the performance of both PCA and t-SNE models is negatively affected by utilizing the fixed rate of features (a=0.5). It seems that these

very challenging conditions significantly affect the performance of transformed models. However, the important improvement in the original models when a=0.5 in almost all cases (except of LR and SVM models) verifies the previous outcome that dimensionality reduction techniques are better able to handle a larger size of feature set than the original ones. The same patterns are consistent in both the examined performance measures.

Clearly, the top-performing model seems to be the k-NN, which achieves the best results in both the examined performance metrics, especially when combined with the t-SNE technique. In particular, the use of t-SNE assists the k-NN model to become more stable surpassing all other base models. The same pattern applies to both AUC and AC performance measures.

DR method	AdaBoost		k-NN		LR		NB		MLP		RF		SGD		SVM	
	AUC	CA	AUC	CA	AUC	CA	AUC	CA	AUC	CA	AUC	CA	AUC	CA	AUC	CA
Original	0.864	0.806	0.812	0.767	0.866	0.794	0.835	0.720	0.877	0.803	0.875	0.814	0.767	0.763	0.504	0.501
Original (50%)	0.876	0.807	0.840	0.772	0.862	0.782	0.842	0.731	0.881	0.809	0.880	0.801	0.823	0.799	0.499	0.480
PCA	0.861	0.805	0.814	0.774	0.866	0.729	0.835	0.690	0.878	0.771	0.880	0.800	0.769	0.704	0.505	0.484
PCA (50%)	0.873	0.810	0.870	0.800	0.816	0.790	0.755	0.708	0.827	0.788	0.880	0.812	0.714	0.692	0.591	0.589
t-SNE	0.883	0.820	**0.885**	0.844	0.820	0.799	0.792	0.770	0.849	0.803	0.882	0.837	0.799	0.744	0.619	0.604
t-SNE (50%)	0.867	0.802	0.867	0.809	0.736	0.690	0.755	0.701	0.820	0.800	0.882	0.804	0.678	0.678	0.564	0.538

TABLE 6.1: AUC scores of the proposed malware detection base models on the Androzoo corpora

In addition, we also consider the combination of the base models by fusing their answers for each sample separately. More specifically, two fusion functions are applied namely, average (AVG) and Majority Vote (MV) technique. Table 6.2 shows the effectiveness in terms of AUC and AC measures of the proposed ensemble malware detection models when either AVG or MV functions are considered on our AndroZoo dataset. Note that the ensemble approaches are not only tested on the original ($AVG_{original}$, $MV_{original}$), but also on transformed (in terms of PCA (AVG_{PCA}, MV_{PCA}) and t-NSE (AVG_{t_SNE}, MV_{t-SNE})techniques) feature sets when a=1 and a=0.5, respectively. Moreover, the AVG_{Mixed}, MV_{Mixed} ensemble models are examined. These models are extracted by combining the entire set of base models (Original, PCA and t-SNE) when either a=1 or a=0.5. In this case, the output of 24 malware detection base models is combined according to AVG and MV aggregate functions.

As can be seen in Table 6.2, the proposed AVG_{Mixed} model is the most effective one in all cases improving the best reported results for the specific dataset. Its performance is higher when a=1 in comparison to the case where a fix rate of features is considered

($a = 0.5$). This sounds reasonable since transformed base models are less enhanced in smaller feature set. Nevertheless, in case of a=0.5, AVG_{Mixed} also provides very good results. This indicates that the size of base models is an important factor that influences the performance of the presented ensemble methods. Given that our ensembles include the entire set of base models, it seems rational that their performance is improved when the number of base models augments.

It is important to be mentioned that ensemble models seem to be positively influenced when dimensionality reduction transformation are considered. Ensembles based on PCA and t-SNE base models are better than the corresponding ones based on original models for both a=1 and a=0.5. Again, t-SNE models surpass PCA models with a wide margin in the most of the cases. This shows that the proposed t-SNE ensemble models are better able to handle malware detection cases and are clearly better options than PCA ones. These patterns are consistent in both the evaluation measures.

In general, the version of ensemble models based on AVG is the most effective in all cases achieving more balanced performance on both AUC and AC measures. On the other hand, the performance of MV models seems not to be highly competitive. At the same time, MV ensemble models is competitive enough achieving its best performance when the output of mixed malware detection base models is considered. It appears again that combining the output of mixed base models achieves the best performance. This verifies that ensembles of classifiers based on multiple, possibly heterogeneous models, can further improve the performance of individual malware detection models.

DR method	**Ensemble (AVG)**		**Ensemble (MV)**	
	AUC	CA	AUC	CA
Original	0.878	0.821	0.815	0.755
Original (50%)	0.879	0.821	0.819	0.767
PCA	0.892	0.840	0.854	0.796
PCA (50%)	0.886	0.836	0.838	0.799
t-SNE	0.940	0.910	0.897	0.855
t-SNE (50%)	0.900	0.840	0.887	0.830
mixed	**0.951**	0.917	0.912	0.877
mixed (50%)	0.942	0.899	0.890	0.862

TABLE 6.2: Comparison of the AUC of both ensemble methods

Figure 6.1 illustrates the performance (AUC) of AVG and MV ensemble models when malware detection base models are based on Original, PCA, t-SNE representations for

a=1 and a=0.5, respectively. The AVG_{Mixed} and MV_{Mixed} ensemble models are also reported.

Apparently, the best performing model not only for the whole feature set but also when a fixed rate of features is randomly selected (a=0.5), is the AVG_{Mixed} model constructed by averaging the output of mixed malware detection base models (Original, PCA and t-SNE models). Given that these ensembles include the set of both the Original, PCA and t-SNE enhanced base models, this means that increasing the size of base models it has a positive effect on the efficiency of the proposed ensemble methods.

Again, ensembles based on original base models are outperformed by the ensembles using dimensionality reduction transformation when a=1. This clearly shows that increasing the feature set helps PCA and t-SNE techniques to improve their performance. Apparently, t-SNE ensemble models are better and more stable alternative than PCA ones with a noticeable margin in all cases indicating that t-SNE models better suits in malware detection cases.

In general, averaging the output of base malware detection models seems to be the best and more stable option achieving more balanced performance both on AUC and AC measures. This strongly indicates that the vast majority of these base models provide commonly improper votes on similar malware detection cases. Both $MV_{original}$ and MV_{PCA} perform poorly. However, it should be underlined that MV_{t-SNE} and MV_{Mixed} models are very effective surpassing the best performing base model which is based on t-SNE transformation. Again, this can be explained since the meta-learner needs as accurate base models as possible and t-SNE models are more stable and reliable than PCA ones. Moreover, it clearly demonstrates the contribution of t-SNE technique on malware detection cases.

6.3.4 Comparison with the state-of-the-art

In the course of our experiments, the performance on the Androzoo corpus is measured by the area under the receiver-operating characteristic curve as well as accuracy measures. This makes our reported results directly comparable to the ones obtained by others

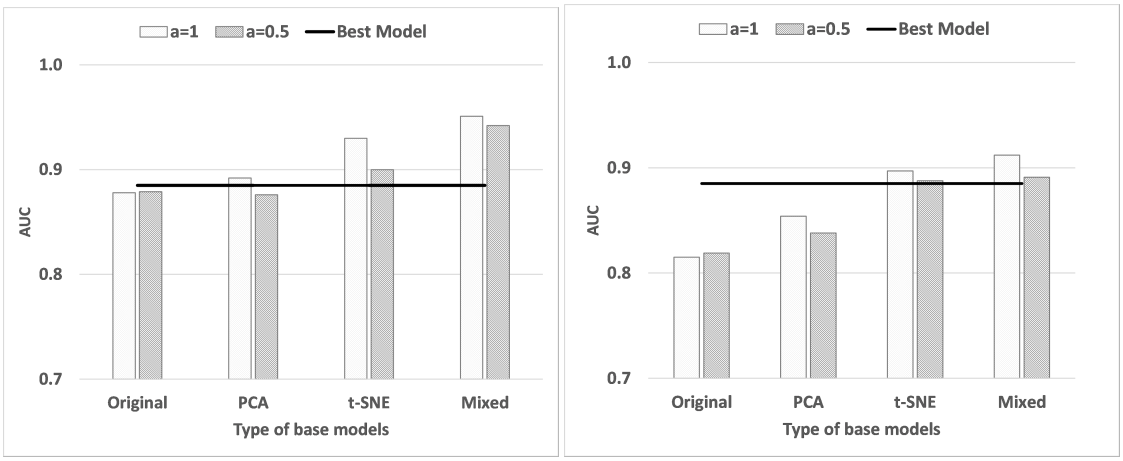

FIGURE 6.1: The performance (AUC) of the examined ensembles when a=1 and a=0.5, using either AVG (left) or MV (right) fusion techniques, on Androzoo dataset for varying types of base models, respectively. The performance of the best base model is also depicted.

published works in the framework of malware detection task. The following state-of-the-art methods (ranked in chronological order) are used to estimate the competitiveness of the proposed AVG_{Mixed} model when a=1:

- Milosevic et al. [155]: This work concentrates on the extraction of non-trivial and beneficial malicious patterns examining the usefulness of source code as well as the permissions set of features when combined with either classification or common used clustering techniques, respectively. In the experiments, the M0Droid corpus is considered and two categories of features namely, permissions and source code are utilized.

- Kouliaridis et al. [10]: In this study, a simple heterogeneous malware detection ensemble method is proposed on Adrozoo dataset. More specifically, a meta-model is constructed by averaging the output of several base models based on either static or hybrid analysis. During static analysis the authors collected features from three categories, namely permissions, intents, and API calls. The performance of this method is evaluated on several datasets, namely Drebin, VirusShare and AndroZoo.

Detection method	**Year**	**Dataset**	**Groups of Features Collected**	**AUC**	**AC**
Milosevic et al. [155]	2017	M0Droid	permissions, source code	N/A	**95.60**
Kouliaridis et al. [10]*	2020	AndroZoo	permissions, intents, API calls	93.57	90.90
AVG_{t-SNE}	2020	AndroZoo	permissions, intents	94.00	91.00
AVG_{Mixed}	2020	AndroZoo	permissions, intents	95.10	91.70

TABLE 6.3: Comparison of the proposed approach with state-of-the-art detection works in terms of collected features, accuracy and AUC score (* For this work, we only consider the results stemming from static analysis on Androzoo corpus)

Note that the published results for some of the above methods only refer to either Androzoo or M0Droid datasets. Moreover, in some works, the evaluation results are not provided on both performance measures (AUC and accuracy). Thus, we use the values of the two evaluation metrics. Table 6.3 demonstrates the effectiveness of the state-of-the-art methods per dataset and on AUC and AC evaluation measures, respectively.

Clearly, both the AVG_{Mixed} and AVG_{t-SNE} models examined in this study are the most effective, surpassing the reported results of Kouliaridis et al. [10]*, which also employs the AndroZoo benchmark dataset. In addition, the resulted evaluation values in [10] were extracted by averaging the answers of 8 base classification models. Under these settings, it is important to mention that these classifiers are similar to those combined for AVG_{t-SNE} model. In this way, it can be concluded that t-SNE transformation better suits in demanding malware detection cases and caters for an effective model handling only two feature categories (permissions and intents). Moreover, when the AVG_{Mixed} model is applied, the improved evaluation results in the challenging Androzoo dataset indicate that the examined model is not easily confused in challenging conditions and the combination of a larger set and mixed base models has a positive effect on the efficiency of the proposed model.

On the other hand, taking into consideration the work of Milosevic et al. [155] examined on a different dataset and feature set, it seems that the performance of both AVG_{Mixed} and AVG_{t-SNE} models is not so competitive. To the best of our knowledge, the M0Droid corpus is not available, therefore it is not possible for our proposed models to be tested on this corpora in order to be directly comparable. Nevertheless, the improved results of the proposed AVG_{Mixed} and AVG_{t-SNE} models in the challenging AndroZoo corpora

demonstrate that the presented models are not easily confused in demanding malware conditions and can capture useful malware information.

6.4 Previous Work

So far, several corpora have been used in the literature to evaluate mobile malware detection approaches. This chapter offers a review of works published from 2015 to 2020, categorized in three benchmark datasets, namely Drebin, VirusShare, and AndroZoo. Note that the focus is on works that present classification results based on features stemming from static analysis.

Drebin [102] is one of the oldest datasets, used in various state-of-the-art mobile malware detection approaches [156]. On the downside, Drebin is outdated and therefore newer malware samples are needed to accurately assess detection performance. Below are some notable works which employed this dataset.

- Ali-Gombe et al. [129] presented *AspectDroid* an hybrid analysis system, which analyzes Android apps to detect unwanted or suspicious activities. The proposed system employs static bytecode instrumentation to provide efficient dataflow analysis. However, static instrumentation is unable to detect apps which use anti-unpacking and anti-repackaging obfuscation mechanisms.
- Arshad et al. [67] introduced a hybrid malware detection scheme, namely *SAMADroid*. According to the authors, SAMADroid delivers high detection accuracy by combining static and dynamic analysis, which run both in a local and remote fashion. Machine Learning (ML) were used to detect malicious behavior of unknown apps and to correctly classify them.

VirusShare is another well-known dataset containing not only mobile malware samples, but also others from various platforms, including Windows and Linux. It is updated regularly and contains samples dated from 2012 onward.

- Xu et al. [139] proposed *HADM*. Their method converted classification features extracted during static analysis into vector-based representations. Other features fetched during dynamic analysis, namely system calls, were transformed into

vector-based and graph-based representations. Deep learning techniques were used to train a neural network for each of the vector sets. Finally, the hierarchical multiple kernel learning technique were applied with the purpose of combining different kernel learning results from diverse features, and thus improve the classification accuracy.

- Fang et al. [149] suggested a hybrid analysis method which performs dynamic analysis on the results of static analysis. During the static analysis phase, they decompiled the app's APK file to extract permissions from the manifest file as well as any occurrence of API features existing in *smali* files. Regarding dynamic analysis, they generated user input and logged the system calls. The authors used ML to classify apps.

Much like VirusShare, AndroZoo [104] is a growing collection of Android apps collected from diverse sources, including the official Google Play store. AndroZoo is updated regularly and it currently contains over 12M samples. To our knowledge, the only work that exploits this dataset for conducting among other static analysis is given by Kouliaridis et al. [10]. Specifically, they introduced an online open-source tool called *Androtomist*, which performs hybrid analysis on Android apps. The authors focused on the importance of dynamic instrumentation, as well as the improvement in detection achieved when hybrid scrutiny is used vis-à-vis to static analysis. In their experiments, the authors compared feature importance between three datasets, namely Drebin, VirusShare, and AndroZoo. Finally, they elaborated on features which seem to be commonly exploited in malware and seldom in benign apps.

Several works in the literature use a mixed dataset, i.e., one containing samples from two or more corpora, to evaluate their approach. Below, we refer to the most important ones.

- Martinelli et al. [140] introduced *BRIDEMAID*. Their system operates in three consequent steps, namely static, meta-data, and dynamic. During static analysis, BRIDEMAID decompiles the apk and analyzes the source code for finding possible similarities in the executed actions. This is done with the help of *n-grams*.

Dynamic analysis exploits both ML classifiers and security policies to control suspicious activities related to text messages, system call invocations, and administrator privilege abuses.

- Surendran et al. [141] implemented a Tree Augmented Naive Bayes (TAN) model that combines the classifier output variables pertaining to static and dynamic features, namely API calls, permissions and system calls, to detect malicious behavior. Their experiments showed an accuracy of up to 97%.

Overall, the effect of dimensionality reduction techniques on Android malware detection, which is the focus of the current chapter, has hitherto received scarce attention in the literature. We were able to only pick out the work of Vega et al. [157] who collected and analyzed malware samples from the Malgenome dataset with six dimensionality reduction techniques, namely Principal Component Analysis, Maximum Likelihood Hebbian Learning, Cooperative Maximum Likelihood Hebbian Learning, Curvilinear Component Analysis, and Isomap and Self Organizing Map. On the other hand, the authors did not evaluate the classification performance of these methods.

6.5 Conclusion

This chapter introduces new insights on malware detection approaches based on ensemble learning. We utilize eight popular and extensively used base classifiers namely, eager (AdaBoost, LR, NB, MLP, SGD, SVM and RF) as well as lazy (k-NN) algorithms. This collection of verifiers provides a pluralism of malware detection scores and we attempt to take advantage of their correlations by constructing two ensembles. In this way, we explore two options. The first one is consisted by a set of homogeneous malware detection base models and learns patterns of agreement or disagreement among not only original feature set but also Principal Component Analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) dimensionality reduction techniques. In other words, it learns dealing with the output of base models extracted utilizing exclusively the original feature set of malware detection cases or only a reduced set of features resulted by applying either the PCA or t-SNE transformation, respectively.

The second option is more knotty, handling the output of a larger and probably more heterogeneous set of base verifiers aggregating all the examined malware detection base

models together (Original, PCA and t-SNE base models). Both the ensemble approaches outperform a set of strong base malware detection models in the most of the cases in terms of experiments developed on benchmark Androzoo corpora. This suggests that our ensembles are able to handle demanding malware detection scenarios and are more robust than individual models. Moreover, it seems that a relatively large and mixed size of base malware detection models is required to achieve high performance.

For each of the above ensemble variation, the output of base models is combined considering two simple fusion functions namely, average (AVG) and Majority Voting (MV) technique. Except of the initial feature set, we examine the alternative of using a fixed rate equal to 0.5 of the initial feature set. This demonstrates that it is important to be defined which set of features will be retrieved and how many base classifiers will be considered. Our experiments show that both AVG and MV work better with mixed base models. This indicates that ensembles based on a large and heterogeneous set of base models are the best option and they provide a more stable performance. That is, in general, AVG are always the best performing models surpassing the corresponding MV ones. This strongly manifests that the vast majority of these base models provide commonly improper votes on similar malware detection cases.

We also focused on the use of two prominent dimensionality reduction techniques namely, PCA and t-SNE, dealing with either the whole initial feature set or a subspace randomly selected. It is demonstrated that the performance of both transformed malware detection models is notably reinforced than the original one when the entire number of features is handled. This verifies that these techniques are negatively affected when a smaller feature set is available. In general, t-SNE models are more effective and competitive than PCA ones indicating that t-SNE transformation helps malware detection approaches to become more stable.

The development of more sophisticated ensembles exploiting a larger set of dimensionality reduction techniques to achieve high diversity is an open research direction. Another possible future work direction is to try to further enrich the pool of our base verifiers considering not only a richer set of classification algorithms, but also several versions of the same approach with different fixed and tuned parameter settings.

Chapter 7

An Extrinsic Random-based Ensemble Approach for Malware Detection

7.1 Introduction

As already pointed out in Section 1, a great mass of mobile malware detection systems and methodologies leans towards static anomaly-based techniques, which employ ML to identify malicious apps [111], [7], [158], [5], [61]. Concerning performance, static analysis requires less resources, and therefore is faster than dynamic analysis. Furthermore, static analysis does not require a mobile device or a VM to run the app, thus it is straightforward to implement. However, as previously mentioned in chapters 4, 5, and 6, many state-of-the-art models have been built using outdated mobile malware datasets, which raises a question on whether features stemming from static analysis alone are sufficient to create models which can detect concurrent mobile malware.

The state-of-the-art works either compare multiple classification models to determine the best performer, or use ensemble methods, which combine multiple classifiers to obtain better predictive performance. This chapter goes one step further by proposing a dynamic ensemble selection method that concentrates on the most effective models for each malware detection case separately. That is, our method is able to take advantage of the case when multiple external malware instances are available and notably

enhances state-of-the-art performance. In particular, we present an extensive study of one dynamic method based on the best performing model and meticulously study its properties and performance. This method enriches the information that is kept in each iteration when building the random subspace ensemble. Based on an extensive experimental study using benchmark datasets that cover several malware genres, and degrees of difficulty, we show that the proposed method is more effective in most of the cases. Furthermore, we demonstrate the effect of a random subspace of features in the performance of the proposed models.

A typical malware detection problem includes a set of malware cases (or instances), all derived by the same dataset or a mixed set collected by various corpora to build the positive class. A set of benign instances are sampled to construct the negative class. A malware detection method should be able to decide whether or not the instance under examination is a malware case or benign. Apart from a binary (yes/no) answer, malware detection methods usually produce a score in [0,1] that can be viewed as a confidence estimation. Essentially, malware detection problem can be defined as a one-class classification task since only labelled samples from the positive class are available. However, there are extrinsic approaches adopted that attempt to transform it to a binary classification task by sampling the negative class, i.e., all benign instances selected by other sources. In most cases, the negative class can be huge and heterogeneous since it compromises all other possible benign apps. It is important to mention that the performance of such methods heavily depends on the quality and properties of the collected external benign instances.

In this study, we handle the malware detection problem from another point of view. In particular, we build predefined positive and negative classes composed by a set of malware and goodware apps, respectively. Moreover, we treat each test instance (either malware or benign) separately by building a random subspace ensemble where in a fix number of iterations randomly choosing a subset of features and a subset of external instances (either of the positive or negative class). In each repetition, the output of three classification binary algorithms is aggregated and a negative or positive answer is provided. Experimental results on AndroZoo [104] benchmark dataset examining different sizes and genre of external instances demonstrate the effectiveness of the proposed extrinsic approach especially in challenging cases where the set of available external instances is of limited size and in cross-genre conditions with respect to the test set.

The main contributions of this study are:

- We adopt predifined categories of external malware and benign instances and propose a more sophisticated extrinsic ensemble approach, which provides a positive or negative answer by averaging the output of the base models for each test instance separately. It is demonstrated that ensemble models can further improve the performance of each individual base classifier.
- We examine the effect of external instances when an ensemble malware detection method is provided combining different sizes and types of external instances. It is demonstrated that ensembles based on a larger and possibly homogeneous size of external instances are exceptionally effective alternative to ensembles included smaller sizes and feasibly more heterogeneous external instances.
- We examine the effect of using either the entire feature set or a random subspace of features of instances in each iteration and it is demonstrated that the latter assists an extrinsic malware detection ensemble to further increase its effectiveness.
- We report experimental results on contemporary benchmark datasets and directly compare them against state-of-the-art methods under the same settings. The performance of the method presented in this study is quite competitive to the best results reported so far for these datasets, demonstrating that an extrinsic ensemble method is much more reliable and effective for the malware detection task.

7.2 Methodology

To handle challenging malware detection cases, we propose essentially a random subspace ensemble taking into consideration a set of multiple malware and benign test instances ($Test_{instances}$). We coin this method *Extrinsic Random-based Ensemble (ERBE)*, and describe it in Algorithm 1. ERBE examines each test instance ($Test_{instance}$) separately. That is, within an iterative process, a subset of classification features stemming from static analysis, say, permissions, intents, etc., along with a subset of available malware and goodware samples (they are called as external instances, $Externals_{repetition}$) are randomly selected in each *repetition*. Note that exactly the same number of $Externals_{repetition}$ both from malware or benign ($External_{malware}$ and

$External_{benign}$) samples are considered per repetition to serve as a positive and negative class, respectively. The $classificationScore$ of a $Test_{instance}$ with both the selected $Externals_{repetition}$ (both malware and benign) samples is calculated in terms of three classification algorithms. That is, in each *repetition*, three scores ($Score_{instance}$) of a test sample are recorded. Then, a malicious detection $MalwareDetectionScore$ is calculated based on an aggregation function that combines all scores corresponding to the test sample for a number of iterations. Note that the set of classifiers applied to estimate malware detection scores per test instance and the aggregate function that combines the scores of all three classifiers for a number of repetitions can be selected among several alternatives to optimize performance in a set of preliminary experiments taken place.

As shown in Algorithm 1, the proposed method has two important parameters, $Externals_{repetition}$, and the *rate*. The former determines the size of the set of the selected external instances of both two categories. Always, an equal number of positive and negative instances are selected either from $External_{malware}$ or $External_{benign}$. The latter parameter affects the number of selected features considered for all instances (either positive or negative) examined. If it is set equal to 1, the entire set of features is utilised to represent each instance vector per iteration in order to provide the final answer (the final $MalwareDetectionScore$). On the other hand, if it is set equal to 0.5 then exactly a half amount of the initial set of features is randomly selected and used to represent a malware instance examined. In more detail, when there is exactly a fixed *rate* equal to 0.5, then a random 50% percent of the initial feature set is considered within the set of all investigated instances. As concerns the number of base learners (i) applied to estimate the $Score_{instance}(i)$ per iteration can also be used as a significant parameter of ERBE method. Note also that the proposed approach is a stochastic algorithm since it makes some random choices of features as well as both positive and negative instances for each tested sample.

7.3 Experimental Study

7.3.1 Description of Data

In this section, we considered a benchmark corpora, namely AndroZoo [104] built in the framework of malware detection task. This is a widely used and continuously spreading

Data: $Test_{instances}$, $External_{malware}$, $External_{benign}$
Parameters: $repetitions$, $|Externals_{repetition}|$, $rate$
Result: $MalwareDetectionScore$
for *each* $Test_{instance} \in Test_{instances}$ **do**
  Set $Score(Test_{instance}) = 0$;
  repeat*repetitions* times
    Select $Externals_{repetition} \subset External_{malware}$ randomly;
    Select $Externals_{repetition} \subset External_{benign}$ randomly;
    Select $rate$ % of features randomly;
    $Score_{instance}(i)$ = ClassificationLearner ($Test_{instance}$, $Externals_{repetition}$, learner(i));
    $ClassificationScore$ = aggregate ($Score_{instance}(:)$);
  end;
  $Score(Test_{instance}) = Score\ (Test_{instance}) + ClassificationScore\ /\ repetitions$;
end
$MalwareDetectionScore$ = aggregate ($Score(:)$);

Algorithm 1: The proposed *Extrinsic Random-based Ensemble* method.

real-world collection of Android apps selected from assorted sources, including the official Google Play app market [30]. Particularly, the collection of AndroZoo apps we used in the context of this chapter is dated from 2017 to 2020 and enclosed 1K malware apps, each of which has been cross-examined by a large number of antivirus products. It is important to note that AndroZoo is a challenging corpora since it includes new and more sophisticated malware samples in comparison to other datasets, including VirusShare [103] and Drebin [102]. We also chose a set of 1K benign apps from Google Play.

External cases: As already pointed out in section 7.2, given that the proposed method follows the extrinsic paradigm, it needs a set of external both malware and benign instances per each examined test instance. In this way, we follow the practice of constructing two categories to collect such a set of instances. In particular, we use a set of 800 malware cases contained in AndroZoo dataset to compose the positive category. This set of instances is randomly selected from the initial set of 1K samples in our partial AndroZoo dataset. Following the same strategy, the negative category is constructed by a randomly selected subset of 800 benign apps from the initial 1K benign cases. It should be noted that the external instances either $External_{malware}$ or $External_{benign}$ are unique and not duplicated in test set, so that they do not affect the performance scores of base models.

Feature Selection: Static analysis was performed on all the apps mentioned in subsection

7.3.1 using the open-source tool Androtomist [10]. Specifically, each app was decompiled to get the *Manifest.xml* file and log permissions and intents to create a feature vector. Each vector is a binary representation of each distinct feature. For example, think of two apps, app1 and app2. The first uses permissions p1, p2, and intent i1, while the latter uses permissions p1, p3, and intents i2, i3. This leads to the 6-dimensional feature vector (p1, p2, p3, i1, i2, i3), and thus the feature vectors for these two apps will be (1, 1, 0, 1, 0, 0) and (1, 0, 1, 0, 1, 1), respectively. Typically, the analysis of a real-world app yields a far more lengthy vector. Precisely, the analysis of the largest set of malware and benign apps used in our experiments, that is, 1K malware instances along with all of the 1K benign apps collected from Google Play, produced 1,002-dimensional feature vectors.

7.3.2 Experimental Setup

As already pointed out, the full dataset used comprises a collection of 1K malware apps randomly selected by AndroZoo corpus. Moreover, 1K benign apps were downloaded by the Google Play to comprise the negative category. The test set is constructed by randomly selecting 200 malware and 200 benign individual apps by this initial set of positive and negative instances, respectively. As already mentioned, the remaining apps per category, i.e., 800 malware and 800 benign apps, compose an initial pool of $External_{malware}$ and $External_{benign}$ samples, respectively. This way, external instances (either malware or benign) are unique and distinct with respect to the specific instances included within the test set.

As per Algorithm 1, the set of external samples ($Externals_{repetition}$) is constructed by randomly selecting a fixed number of k samples per iteration from this initial pool of $External_{malware}$ and $External_{benign}$ instances, respectively. As can be observed from Algorithm 1, the ERBE method has several parameters that need to be tuned for a particular dataset. In this section using AndroZoo benchmark data, all these parameters were tuned based on the training set.

Specifically, to simplify and make this process more efficient, we attempt to reduce independent parameters by setting fixed parameter values. In particular, we focus on fine-tuning parameter $k = |Externals_{repetition}|$ by selecting $k \in \{50, 100, ..., 300\}$ external samples per repetition optimizing the performance in the training set. Then, we set

$repetitions = 5$ based on some preliminary tests. A slight, though not consistent, difference with respect to the performance of $repetitions = 15$ and $repetitions = 5$ was noticed. As concerns the features selected, in this section, we explore two options. The first one is to use exclusively the entire set of features by setting $rate = a = 1$. This indicates that the entire vector of each instance examined is considered per iteration. The second option is to fix $rate$ of $a = 0.5$ indicating that a percent of 50% of the initial set of features are selected in each repetition. Again, doing some preliminary testing, considering $a = 0.5$ and $a = 0.75$, a significant improvement in the effectiveness of the ERBE method in comparison to the case where $a = 1$ is observed.

Moreover, three well-known and widely used supervised ML algorithms were applied, namely Logistic Regression (LR), Multiple Layer Perception (MLP), and Stochastic Gradient Descent (SGD). In a more detailed description, the entire set of the classification algorithms employed falls under eager learning. In this category, supervised learning algorithms attempt to build a general model of the malicious instances, based on the training set. Obviously, the performance of such classifiers strongly depends on the size, quality and representative of the training data. For each classifier applied, the default values of the parameter settings are used. The general model of each eager classifier is built following the 10-fold cross-validation technique, where the original dataset is randomly partitioned into 10 equal sized sub-datasets. A single sub-dataset is retained for the testing, while the remaining 9 are used for training. This process is repeated 10 times, and each time using a different sub-dataset for testing. The results are then averaged to produce a single estimation.

Our main evaluation measure is the Area Under the Receiver Operating Characteristic (ROC) curve (AUC) that quantifies the effectiveness of an examined approach for all possible malware detection score thresholds [131]. Moreover, this evaluation measure does not depend on the distribution of positive/negative instances. This is extracted by examining the ranking of malware detection scores (rather than their exact values) produced when a method is applied to a dataset. However, when one has to decide about a specific malware case, the malware detection score has to be transformed to a binary answer: either a positive (malware class) or a negative (benign class) one. To this direction, a threshold can be applied to the malware detection score - actually, the calculation of AUC is based on all possible thresholds. To set this threshold, information about the distribution of positive and negative malware detection instances

is paramount. In our dataset, positive and negative instances are equally distributed. To transform malware detection scores to binary answers, we follow exactly the same evaluation procedure to achieve compatibility of comparison with previously reported results. In a more detailed description, the set of the extracted scores based on the test instances are normalized in the interval of [0,1] per classification model per examined method. To this direction, the estimation of the threshold is set equal to 0.5. Then, all malware detection scores of the test dataset that are lower/higher than this threshold are transformed to negative/positive answers. This is in accordance to the setup of previously reported results.

Finally, we select the aggregation function used in ERBE method among average, minimum, and maximum that optimizes performance in the training set. Most of the times, average is selected. Since the proposed ERBE method makes stochastic choices in each repetition, each experiment is repeated five times and we report average performance.

7.4 Results

The following classification performance metrics are used to achieve compatibility of comparison with stare-of-the-art methods, where TP, TN, FP, and FN represent correspondingly True Positives, True Negatives, False Positives, and False Negatives.

- Accuracy $(CA) : \frac{TP+TN}{TP+TN+FP+FN}$. The number of correctly classified patterns over the total number of patterns in the sample.
- Precision $(P) : \frac{TP}{TP+FP}$. The ratio of TP values over the sum of TP and FP.
- Recall $(R) : \frac{TP}{TP+FN}$. The ratio of TP over the sum of TP and FN.
- Area Under Curve (AUC): The higher positive-over-negative value ranking capability of a classifier.
- $F1 : 2 * \frac{P*R}{P+R}$.

To demonstrate the usefulness of legacy base models in ERBE approach, figure 7.1 depicts the performance of $ERBE_{LR}$, $ERBE_{MLP}$ and $ERBE_{SGD}$ on the Androzoo

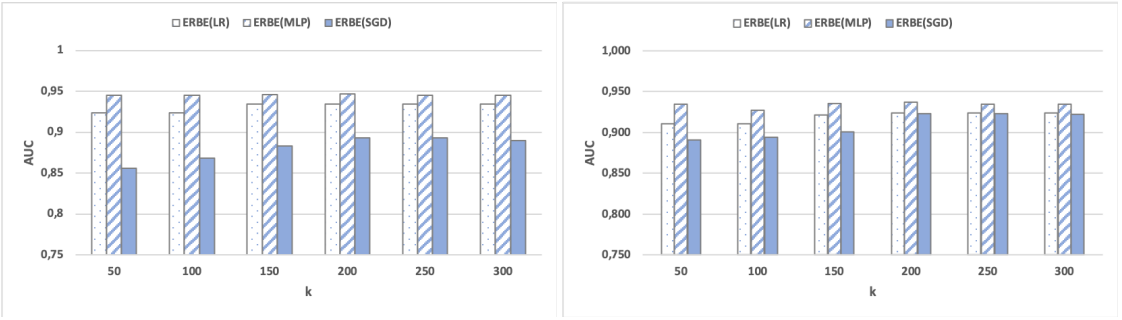

FIGURE 7.1: The performance (AUC) of the examined base models, using either $a = 0.5$ (left) or $a = 1$ (right) on AndroZoo dataset.

corpus for varying sizes of external instances (k) when $a = 0.5$ (left) and $a = 1$ (right) used for each examined instance per iteration.

As can be clearly seen, $ERBE_{MLP}$ model is the best option in all cases of the examined size values of the external instances on Androzoo corpora. Not only in case where a percent of 50% on the initial feature set is randomly selected, but also when the entire set of features is handled, it is more effective providing a more stable performance than the two others. Moreover, the effectiveness of $ERBE_{LR}$ model is stable and seems to be competitive enough acquiring remarkable performance when $k \geq 200$ especially in both cases. On the other hand, the effectiveness of SGD classifier is negatively affected when $a = 0.5$. Then, $ERBE_{SGD}$ model seems not to be a very stable option since its performance vary for different values of k with a large margin. It seems to get improved a lot when k increases. In particular, it achieves its best results for k=200, while in the case of shorter k values its performance seems to be especially poor. All these indicate that $ERBE_{SGD}$ models are in need of a more fixed and accurate set of features to be valid. Moreover, these results demonstrate that both $ERBE_{LR}$ and $ERBE_{MLP}$ are better and reliable models for multiple values of k while $ERBE_{MLP}$ is always superior.

Table 7.1 reports the evaluation results of ERBE method and the performance of the best base model on multiple evaluation measures over the AndroZoo dataset. Moreover, the version of ERBE when a=1 and the corresponding best performing base models, $ERBE_{MLP}$ for $k = 200$, are also reported. As can be seen, ERBE is the most effective one in all cases improving always the reported results of the best base models for the specific dataset. This verifies that ensembles of classifiers based on multiple, possibly heterogeneous models, can further improve the performance of individual malware detection base models. From the obtained results, it is also clear that the performance of

ERBE is higher when a random subspace of the initial feature set is used (in case of $a = 0.5$) in comparison to the version where the entire set of features (case of $\text{ERBE}_{a=1}$) is considered in all the examined cases. It clearly seems that ERBE method is positively affected by a random rate selection of features on examined samples per iteration. In other words, when the number of features decreases and the size of vectors in examined instances is reduced then the performance of ERBE is increased. This indicates that ERBE approach is much more reliable and effective in difficult malware cases where there are irregular and incidental subsets of features which belong to different domains in each iteration.

	AUC	**Accuracy**	**Precision**	**Recall**	**F1**
$\textbf{ERBE}_{\textbf{MLP,a=1}}$	0.936	0.917	0.903	0.930	0.916
$\textbf{ERBE}_{\textbf{MLP}}$	0.947	0.923	0.886	0.937	0.911
$\textbf{ERBE}_{a=1}$	0.978	0.950	0.940	0.959	0.949
ERBE	**0.994**	0.983	0.970	0.976	0.973

TABLE 7.1: Scores of all evaluation measures examined in ERBE malware detection method and the best performing base models with $a = 0.5$ and $a = 1$ for $k = 200$ on AndroZoo dataset.

7.4.1 Contribution of a random subspace set of features

Next, we examined the contribution of the factor $|a|$ related to the number of features considered in the proposed ensemble malware detection method ERBE. To isolate the contribution of this factor, beyond of the value of $a = 0.5$, we also used the value of the initial set of features, $a = 1$. In this way, ERBE was applied with and without considering the entire feature set for varying values of parameter k. Note that each version of the examined method was performed to the AndroZoo dataset and the aggregation function was fixed to average for this experiment.

Figure 7.2 shows the corresponding performance of ERBE method for comparative purposes. Apparently, the contribution of random selection of features is significant and assists the present ensemble method to enhance its effectiveness for all k values. The performance of ERBE without random collected features, $ERBE_{a=1}$ is also competitive enough, especially when the k value is increased ($k > 150$). In addition, ERBE is clearly better than the best base model, $ERBE_{MLP}$ for the whole range of the examined k values. It is noticeable that the contribution of random feature selection is stronger not

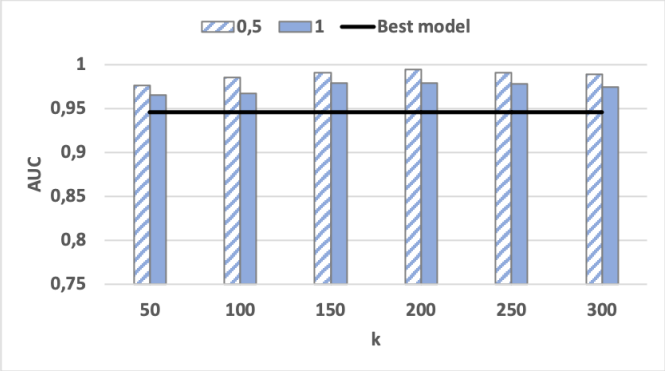

FIGURE 7.2: The performance of AUC of the proposed ERBE and ERBE$_{a=1}$. The best performing base model, $ERBE_{MLP}$ is also shown.

only for low but also for large values of k (i.e., $k > 150$), while the version of ERBE without random feature selection (when a=1) is weaker enough for relatively low values of k (up to 150).This means that k should be set to a relatively large value to reinforce the effectiveness of $ERBE_{a=1}$. In other words, this strongly suggests that when there are challenging conditions, the information of the entire feature set is less crucial for ERBE method. On the other hand, ERBE is proved to be particularly enhanced in the case where sporadic and inconstant features are selected on examined instances in each iteration.

Additionally, we examined the statistical significance of pairwise differences of both the tested versions of ERBE and base malware detection models, respectively. Table 7.2 demonstrates the improvement in performance (difference of AUC scores) of both the examined ERBE versions as well as the base classification models with $a = 0.5$ and $a = 1$ on AndroZoo dataset, respectively. The statistical significance of these differences is estimated using an approximate randomization test [138]. The null hypothesis is that there is no difference between the two cases, and we reject this hypothesis when $p < .05$.

As can be seen, in general the models extracted from the random selected feature set are more effective and clearly better options. In particular, ERBE models based on $a = 0.5$ are more improved and gain more than the corresponding ones belonging in case of $a = 1$ for all k values. As concerns the individual base models, the results of $ERBE_{LR}$ and $ERBE_{MLP}$ are improved the most, while notable exceptions are $ERBE_{SGD}$ base models, where the random selection of features per iteration is not significantly better than the case of the entire feature set (case of $a = 1$). This does not seem to correlate with the relative increase in the number of external malware instances used per repetition.

k	**ERBE**$_{LR}$	**ERBE**$_{MLP}$	**ERBE**$_{SGD}$	**ERBE**
50	**0.024**	**0.020**	**-0.035**	0.011
100	**0.024**	**0.027**	**-0.026**	**0.019**
150	0.013	**0.021**	**-0.018**	**0.012**
200	0.010	0.011	**-0.039**	**0.016**
250	**0.023**	**0.014**	**-0.037**	**0.012**
300	**0.023**	0.011	**-0.032**	**0.015**

TABLE 7.2: Improvement in performance (difference in AUC) between ensemble methods as well as base models using $a = 0.5$ and $a = 1$ on AndroZoo dataset.

7.4.2 Comparison with the state-of-the-art

In all our experiments, the performance on the evaluation data set is measured by various evaluation measures. In this way, our reported results can directly be compared with the ones of other published methods followed the static analysis that use exactly the same evaluation measures in the framework of malware detection task. The following state-of-the-art methods, ranked in chronological order, are used to estimate the competitiveness of the proposed method:

- Yerima et al. (2015): This is an ensemble malware detection method focuses on the extraction of critical Android and Java API calls from the source code, as well as the app permissions extracted from the manifest file. In all experiments, McAfee's internal dataset is considered.

- Milosevic et al. (2017): This method concentrates on the extraction of non-trivial and beneficial malicious patterns examining the usefulness of source code as well as the permissions set of features when combined with either classification or common used clustering techniques, respectively. In all experiments, the M0Droid corpus [159] is considered.

- Idrees et al. (2017): This is also a malware detection method based on ensemble learning to boost the effectiveness of base models followed a static app analysis.

This method considers a mix set of features including app's permissions and intents derived from Contagio dump, MalGenome, theZoo, Malshare, and Virushare datasets.

- Kouliaridis et al. (2020): This is a simple heterogeneous ensemble malware detection method. A meta-model is constructed by averaging the output of several base models based on either static or hybrid analysis. The feature set comprises multiple categories, such as permissions, intents and API calls. The performance of this method is evaluated on several datasets, namely Drebin, VirusShare and AndroZoo. For this work, we consider the results stemming from static analysis.

Table 7.3 demonstrates the effectiveness of the state-of-the-art methods per dataset on both AUC and Accuracy measures of evaluation. Note that the published results for some of the above methods are only provided on either AUC or Accuracy Performance measures (i.e. the work of Milosevic et al. [155]). Moreover, they are not tested on AndroZoo corpus, but only refer to a mixture of datasets. To the best of our knowledge, these mixed corpus are not accessible since they have generated by individual research groups and they are not shared with the public. As can be seen, the proposed malware detection method based on extrinsic ensemble learning, ERBE is particularly effective and outperforms the vast majority of baseline methods and simultaneously the study of Yerima et al., [160] based also on ensemble learning. In addition, ERBE seems to be highly competitive with the approach of Idrees et al., [161] examined on a similar feature set. However, the improved results of the proposed ERBE method in the challenging AndroZoo corpus indicate that the examined method is not easily confused in demanding malware conditions and the extracted extrinsic ensemble models can capture useful malware information.

Moreover, the improvement in performance of ERBE model with respect to that of the simple ensemble baseline [10] is higher than 5% on the AndroZoo dataset. All in all, given that both ERBE and simple meta-model are exclusively applied on the demanding AndroZoo dataset, it can be concluded that an extrinsic ensemble provides an effective approach in malware detection when it is fine-tuned and appropriately combined with suitable base models.

In addition, the proposed ERBE malware detection model is also very effective and clearly better option in comparison to Milosevic et al., [155] approach. The improvement

Detection method	Year	Feature set	AUC per dataset		Accuracy per dataset	
			AndroZoo	Other	AndroZoo	Other
Yerima et al.	2015	permissions, API calls	N/A	0.993	N/A	0.975
Milosevic et al.	2017	permissions, source code	N/A	N/A	N/A	0.956
Idrees et al.	2017	permissions, intents	N/A	**0.998**	N/A	**0.998**
Kouliaridis et al.	2020	permissions, intents, API calls	0.936	N/A	0.909	N/A
ERBE	2020	permissions, intents	0.994	N/A	0.983	N/A

TABLE 7.3: Comparison of state-of-the-art methods with the proposed ERBE malware detection method of this study.

in performance is higher than 2%. With respect to the categories of features applied, the method of Milosevic et al., does not seem to be positively influenced when a feature set of both app permissions and source code is available. Taking also into consideration the corresponding results of both Milosevic et al., [155] and Idrees et al., [161] methods, we notably reinforce the aforementioned outcome as a feature set of both app's permission and intents is better able to handle challenging malware conditions.

7.4.3 Genre of External cases

So far, in all the experiments the set of external instances required by the examined method stem from the AndrooZoo dataset. This means that the genre of these instances most probably match a lot to the one of the test instances in question. Taking into account that the genre of all instances is the same, the performance of the proposed method can take a significant advantage and can be probably considerably improved.

This section uses another prior released corpus that will allow us to examine this effect. As concerns the set of external malware instances required by ERBE, we explored two alternatives. First, we followed the approach used in the previous experiments selecting external instances from the rather outdated Drebin corpus. We call this alternative as ERBE_{Derbin}. Second, we used the VirusShare corpus to collect the set of external instances. VirusShare comprises newer and more challenging apps dated from 2014 to 2017. This second alternative is defined as $\text{ERBE}_{VirusShare}$. For each of the above corpus examined, 200 malware apps are randomly selected per iteration by a large pool of 1K malware samples. The benign apps included in the set of external instances are similar to those applied in ERBE and were collected from Google Play [30]. In each repetition, a set of 200 goodware apps are also randomly selected to use as external instances. Again, both the malware and benign part of the external instances was balanced with 200 positive and 200 negative cases. The parameters of the methods presented in this

	AUC	**Accuracy**	**Precision**	**Recall**	**F1**
ERBE$_{AndroZoo}$	**0.994**	0.973	0.970	0.976	0.973
ERBE$_{Derbin}$	0.971	0.938	0.936	0.940	0.938
ERBE$_{VirusShare}$	0.976	0.945	0.941	0.950	0.945
ERBE$_{mixed}$	0.987	0.941	0.925	0.960	0.942

TABLE 7.4: Scores of all evaluation measures examined of ERBE method when $k = 200$ based on different genre of external instances

study were estimated based on the training part of the corpus as described in section 7.3.2.

In addition, similar to the previous experiments, we examined a case of extracting the external malware instances by randomly selecting a similar number of apps from all three datasets. That way, a set of 900 mix external instances was obtained by extracting malware cases including all corpora examined (AndroZoo, Drebin, and VirusShare). In other words, in the case of external malware instances, the enriched collection comprises a mix of genres. This is called as ERBE$_{mixed}$ alternative. Again, in each repetition, 200 malware apps are randomly collected by a mix pool of malware external instances. We call each one of the above alternatives as "genre-agnostic" because of different genre of the set of external malware instances.

Table 7.4 shows the results on various evaluation measures of the examined ERBE malware detection method on different genres of external malware instances selected in each iteration. From the obtained results, the best results so far are obtained by the presented ERBE model. As expected, the set of AndroZoo external malware instances assists the proposed ensemble method to achieve higher scores in comparison to cases where genre-agnostic external instances are used. This sounds reasonable since AndroZoo is a demanding corpus including new and challenging apps. In this way, it is demonstrated that the meta-learner needs as accurate base models as possible and learning models exclusively on AndroZoo samples are more likely to be more accurate than models with learning on completely different genre of external malware samples.

This is verified when genre-agnostic external malware instances are concerned since then this difference is more evident. Comparing the performance of ERBE$_{Derbin}$ and ERBE$_{VirusShare}$ models, we see that the contribution of the latter is stronger, while the

version of the ERBE_{Derbin} is the most weak. This indicates that when there are cross-genre malware conditions, the information of the outdated external malware instances of Derbin corpus is less crucial on ensemble learning and inadequate to handle test instances of AndroZoo dataset. It is also remarkable that information in ensemble learning models belonging to a mix of genres can be useful to define malware cases. The ERBE_{mixed} model is better than all other variations (both ERBE_{Derbin} and $\text{ERBE}_{VirusShare}$ variations) in all the examined cases.

7.5 Related Work

As of today, the topic of mobile app classification via the use of ML has received significant attention in the Android security literature [111], [7], [158], [8], [9], [162], [11]. This section offers a chronologically arranged review of the most notable and recent works on this topic. Specifically, we concentrate on contributions published over the last six years, that is, from 2014 to 2020. We only consider highly relevant works to ours, namely those which propose or employ some type of ensemble learning.

Yerima et al. [160] contributed an approach which uses ensemble learning for Android malware detection. According to the authors, their method combines advantages from static analysis with ensemble learning to improve detection accuracy. Their results showed that the proposed method is capable of achieving 97.3 to 99% detection accuracy with low false positive rates.

Coronado-De-Alba et al. [163] presented an approach which analyzes data obtained through static analysis. According to the authors their results provided explicit evidence for classification improvement. Even more, a comparative analysis of various ensembles were presented to find the best combination of classifiers based on the evaluation of their classification results. Idrees et al. [161] presented *PIndroid*, a framework which uses permissions and intents in conjunction with ensemble learning to identify Android malware. The authors evaluated their approach by applying it to 1,745 real world apps and their results showed 99.8% accuracy.

Milosevic et al. [155] presented two ML-aided approaches for static analysis of Android apps. The first one is based on permissions and the other on source code analysis based on a "bag-of-words" representation model. The authors evaluated both these approaches

Work	ML methodology	Dataset
[160]	Ensemble Learning	McAfee's internal repository [165]
[163]	Ensemble Learning	Drebin [102]
[161]	Ensemble Learning	Contagio [81], Genome [105], theZoo [166], MalShare [167], VirusShare [103]
[155]	Ensemble learning	M0Droid [159]
[164]	Ensemble Clustering and Classification	Drebin [102]
[10]	Ensemble Learning	Drebin [102], VirusShare [103], AndroZoo [104]

TABLE 7.5: Outline of the related work

using base classification models, as well as ensemble learning along with various combinations of the selected base models. Their results showed an F-score of 95.1% and F-measure of 89% for the source code-based and permission-based classification models, respectively.

Chakraborty et al. [164] presented Ensemble Clustering and Classification (EC2), an algorithm for identifying Android malware families. Furthermore, the authors offered a performance comparison of several classification and clustering algorithms on the Drebin dataset and used the output of both supervised classifiers and unsupervised clustering to design EC2. Their experimental results on both the Drebin and other more recent malware datasets showed that EC2 is able to accurately detect malware families, outperforming several comparative baselines. According to the authors, EC2 presents an early warning system for new malware families, as well as a predictor of known families to which a malware sample belongs.

Kouliaridis et al. [10] introduced *Androtomist*, a novel tool capable of utilizing both static and dynamic analysis on Android apps. The authors concentrated on the results of static analysis when combined with dynamic instrumentation. Moreover, they proposed an ensemble approach by averaging the output of several base models for each malware instance separately. Finally, the authors evaluated their work against three well-known datasets and their results designated that *Androtomist* is superior to previous state-of-the-art mobile malware detection solutions.

7.6 Discussion

This chapter presents an extrinsic malware detection method based on ensemble learning. By utilizing a set of three well-known as well as widely used base verifiers, we attempt to take advantage of their correlations by building a more sophisticated Extrinsic Random-based Ensemble (ERBE) based on a random subspace of external instances and features for each test instance separately. The experimental results based on AndroZoo benchmark dataset demonstrate that ERBE's performance is better than any single base model and is highly competitive when compared with state-of-the-art methods. The contribution of the random subspace of features, used in ERBE, is a crucial factor to improve performance. This enables ERBE to take advantage of all the examined sizes of external instances. The extrinsic ensemble approach outperforms a set of strong baselines tested on either the benchmark AndroZoo corpus or mixed datasets. The performance of ERBE is more than 5% better than an ensemble learning baseline implemented on the challenging AndroZoo dataset too. In comparison to the best baseline (note that this method is tested on a mixed dataset), ERBE is competitive enough in terms of accuracy measure.

All extrinsic methods strongly depend on the appropriate selection of external instances. We used a set of malware instances randomly selected by AndroZoo corpus ensuring that there are similarities with the test instances under examination. Certainly, this procedure can be improved by taking into account the genre of instances. To this direction, we examined the effectiveness of ERBE method following a couple of options. In the first one, the external malware instances were exclusively collected by different malware corpora in comparison to test set. In the second, a mixed set of external malware instances derived from multiple malware corpora was considered. As it is demonstrated, ideally, the external malware instances should be sampled from the same source as the one from which the test instances are drawn. For instance, if the instance under examination belongs to AndroZoo dataset, then there is strong indication that the external instances should also be part from that dataset to ensure similarity in genre, format and edition of instances. This can be explained since the meta-model needs as accurate and reliable base models as possible. However, information about the source of test malware instances may not be available in the most of the real-world cases.

The current scheme uses three well-known classifiers as base models. An interesting future work direction could focus on a richer set of classification models, comprising eager and lazy algorithms, that can be adapted to each malware case separately. This heterogeneous ensemble approach relies on base models with default parameter settings. This could be used to further enrich the pool of our base verifiers considering several versions of the same approach with different fixed and tuned parameter settings. Another future work direction could concentrate on combining multiple malware detection methods based on hybrid and static analysis in a more complex approach. Lastly but not least, although in this work ERBE has been evaluated using Android malware datasets, it is evident that it can be easily applied for malware detection on any platform.

Chapter 8

A mapping of machine learning techniques for Android malware detection and a converging scheme

8.1 Introduction

According to a recent report from McAfee [12], 2020 was the year of mobile sneak attacks. Namely, cyber criminals and state-sponsored actors are constantly looking for ingenious ways to acquire user data. Likewise, malware writers continue to come up with new ways of hiding their attacks and frauds, making them increasingly difficult to identify and neutralize. On the positive side, new mobile malware detection techniques are also evolving to counter these threats. Indeed, Machine learning (ML) has long proved its decisive role in this ecosystem given that the vast majority of mobile malware detection solutions proposed in the literature so far are based on some kind of ML-driven scheme.

Traditionally, ML is exploited in anomaly-based detection methods. Anomaly-based detection comprises two basic phases, namely the training and the detection or testing one. That is, such solutions employ ML to detect malicious behavior, i.e., deviation from a model built during the training phase. Anomaly-based detection can be further categorized depending on the type of analysis, i.e., static, dynamic, and hybrid. Static

analysis is performed in a non-runtime environment, which analyzes an app's internal structure. Dynamic analysis on the other hand adopts the opposite approach, taking place during the app's normal operation.

As shown in Table 8.1, various app features can be extracted depending on the analysis type, either static, dynamic, or hybrid. Each of these features has its advantages and limitations. That is, features stemming from static analysis have proven efficient against older malware apps [168], but tent to be ineffective against code obfuscation and encryption techniques [169].

Analysis type	**Feature extraction method**	**Features extracted**
Static	Manifest analysis	Package name, Permissions, Intents, Activities, Services, Providers
	Code analysis	API calls, Information flow, Taint tracking, Opcodes, Native code, Cleartext analysis
Dynamic	Network traffic analysis	URLs, IPs, Network Protocols, Certificates, Non-encrypted data
	Code instrumentation	Java classes, intents, network traffic
	System calls analysis	System calls
	System resources analysis	CPU, Memory, and Battery usage, Process reports, Network usage
	User interaction analysis	Buttons, Icons, Actions/Events

TABLE 8.1: Feature extraction options per analysis method

When feeding additional features extracted through dynamic analysis to malware detection models, they can typically cope significantly better with the newest and more challenging pieces of malware [10]. However, hybrid analysis systems are inherently more complex, due to the several extra components needed by dynamic analysis, such as a virtual or real platform, and a user event and input emulator to exercise the app.

On top of that, some sophisticated malicious apps can recognize when being executed in emulated environments and avoid detection [170].

Classification is the process of categorizing data into classes. This process starts with predicting the class of given data points. The classes are often referred to as target, label, or categories. From an ML model's perspective, classification requires a training dataset with multiple instances from which the chosen ML model learns. Much like app analysis methods, each ML model also has its pros and cons based on the supplied data [171]. As detailed in section 8.2 the majority of mobile malware detection works in the literature advertise a different ML algorithm as best performer for mobile malware detection. For this reason, several performance optimization techniques have been used throughout the literature to further enhance classification performance. These techniques include:

- Feature ranking and selection by calculating feature importance scores.
- Dimensionality reduction transforms features into a lower dimension to reduce bias and noise.
- Ensemble models combine the output of multiple base models to improve the overall classification performance and can be used in conjunction with any of the previous two techniques.

Given the growing impact of ML-aided mobile malware detection schemes, deeper literature review is needed considering all state-of-the-art works available and exploring the details behind each efficient detection model. Unfortunately, while there are many contributions in the literature leveraging on ML for mobile malware detection on the Android platform, most of them rely on diverse metrics, classification models, and performance improvement techniques. The absence of a common baseline on this field can cause confusion, lead to half-true or even incorrect generalizations, and mislead future research. In an effort to mitigate these issues, the work at hand aims to:

- Provide a detailed mapping of the contemporary ML techniques regarding Android malware detection proposed in the literature during the last 7 years.
- Categorize each contribution based on four distinct criteria, namely the chosen metrics, dataset, classification models, and performance improvement techniques.

- Introduce a converging, i.e., decision-making scheme to guide future work in this ecosystem.

The remainder of this chapter is organized in the following manner. Section 8.2 details on the relevant literature and categorizes each work based on the employed ML techniques. Section 8.3 provides a discussion on the findings and introduces the proposed scheme. Section 8.4 discusses the related work. Section 8.5 draws a conclusion.

8.2 Survey of works

This section provides a detailed review on major published works devoted to the detection of Android malware in the last 7 years. Table 8.2 categorizes each work in chronological order based on the following criteria, while Table 8.3 offers condensed view of the common criteria.

- The analysis type, namely static, dynamic, or hybrid.
- The feature extraction method, namely Manifest Analysis (MA), source Code Analysis (CA), Network Traffic Analysis (NTA), Code Instrumentation (CI), System Calls Analysis (SCA), System Resources Analysis (SRA), and User Interaction Analysis (UIA).
- The features collected, as it has been listed in Table 8.1.
- The classification approach, i.e., base models and possible performance improvement techniques, including Feature importance (FI) metrics, Dimensionality Reduction (DR), and Ensemble Learning (EL).

Shabtai et al. [51] presented a system for detecting meaningful deviations in a mobile app's network behavior. That is, the system monitors the running apps to create their "normal" network behavior. Then, it is able to detect deviations from the learned patterns. The authors' main goal was "to learn user-specific network traffic patterns for each app and determine if meaningful changes occur". To evaluate their model, the authors employed the C4.5 algorithm, achieving an accuracy of up to 94%.

Canfora et al. [46] proposed a mobile malware detection approach which analyzes op-code frequency histograms. Precisely, their approach classifies malware by focusing on the number of occurrences of a specific group of op-codes. They used a detection technique, which capitalizes on a vector of features obtained from eight Dalvik op-codes. These op-codes are usually used to alter the app's control flow. Six classification models were used during evaluation, namely LadTree, NBTree, RandomForest, RandomTree and RepTree. The model were applied separately to the eight features and the three groups of features. The first group includes the move and the jump features, the second involves two well-known distance metrics, namely Manhattan and Euclidean distance, and the last embraces all the four features (move, jump, Manhattan and Euclidean features). The proposed method was evaluated on the Drebin dataset using several classifiers, namely J48, LadTree, NBTree, Random Forest, Random Tree and RepTree, and achieved an accuracy of 95%.

Jang et al. [53] developed *Andro-AutoPsy*, an anti-malware system based on Android malware similarity matching. To train the proposed model, the authors gathered both malware-centric and malware creator-centric information from anti-virus technical reports, malware repositories, community sites, and other via web crawling. They chose five footprints as features, namely the serial number of a certificate, API call sequence, permissions, intents, and system commands. According to the authors, Andro-AutoPsy can detect zero-day malware. Andro-AutoPsy was evaluated with nearly 1K malware apps obtained from the VirusShare [103] and Contagio mobile datasets [81] and more than 109K benign samples collected from Google Play [30].

Yerima et al. [160] proposed an ensemble malware detection method concentrating on the extraction of critical Android and Java API calls from the source code, as well as the app permissions extracted from the manifest file. In all the experiments, McAfee's internal (not public) dataset was considered. During the evaluation phase, several classifiers were employed, namely Naive Bayes, Simple Logistic, Decision Tree and Random Tree, scoring an AUC of up to 99.3% and accuracy of 97.5%.

Coronado-De-Alba et al. [163] presented a method which introduces a meta-ensemble algorithm. They employed static analysis on a dataset of 1,531 malware apps collected from the Drebin dataset [102] and 765 benign apps, to obtain permissions and intents.

The authors employed the RandomForest and RandomCommittee algorithms, achieving an accuracy of up to 97.5% with the use of the former.

Milosevic et al. [155] proposed a detection method which concentrates on the extraction of non-trivial and beneficial malicious patterns. This is done by examining the usefulness of source code as well as the permissions set of features when combined with either classification or common used clustering techniques, respectively. In their experiments, the M0Droid corpus [159] was considered. Several classifiers were used during the evaluation process, such as the C4.5, Random forest, Naive Bayes, Support Vector Machine (SVM), JRip, and Logistic Regression. Their results showed an accuracy of up to 95.6%.

Idrees et al. [161] proposed an Android malware detection method based on ensemble learning to boost the effectiveness of base classification models followed a static app analysis. This method considers a mixture of features, including app's permissions and intents derived from Contagio dump, MalGenome [105], theZoo [166], Malshare [167], and Virushare [103] datasets. The features with the highest feature importance score, calculated using the information gain (IG) algorithm, were selected to train the model. The authors employed the Naive Bayes, Decision Tree, Decision Table, Random Forest, and Multilayer perceptron (MLP) classifiers. Their evaluation tests demonstrated a best both AUC and accuracy score of up to 99.8%.

Alam et al. [57] contributed *DroidNative* for the detection of both bytecode and native code Android malware. According to the authors, *DroidNative* is the first scheme to build cross-platform (x86 and ARM) semantic-based signatures for Android and operates at the native code level. When apps are analyzed, bytecode components are passed to an Android Runtime (ART) [172] compiler to produce a native binary. The binary code is disassembled and translated into Malware Analysis Intermediate Language (MAIL) code. To evaluate their approach, the authors collected over 5,490 malware from the Drebin and Contagio mobile corpora. Their results showed a detection rate of up to 93.57% and an AUC score ranging from 97.86% to 99.56%.

Kouliaridis et al. [55] proposed *Mal-warehouse*, an open source tool performing data collection-as-a-service for Android malware behavioral patterns. An open source tool called *"MIET"* was developed, which extracts usage information over a period of time from Android devices. *Mal-warehouse* is enhanced with a detection module, which the

authors evaluated via the use of a series of base models, namely k-NN, Random Forest, SVM, Naive Bayes and AdaBoost, and achieved a top AUC score of 85.4%.

Tao et al. [49] introduced *MalPat*, an automated malware detection system which scans for malicious patterns in Android apps. MalPat detects malicious patters by analyzing API calls. The authors collected 31,195 benign apps and 15,336 malware samples. A repeated process was followed to evaluate MalPat using the Random Forest model, in which they randomly selected a percentage of both malicious and benign datasets as the training set, and the remaining part was regarded as the testing set. In their evaluations, *MalPat* achieved a 98.24% F1 score using the SVM classifier.

Shen et al. [50] suggested an approach based on information flow analysis. They introduced complex-flow as a new representation scheme for information flows. According to the authors, complex-flow is a set of simple flows that share a common portion of code. For example, if an app is able to read contacts, store them and then send them over the Internet, then these two flows would be (contact, storage) and (contact, network). The authors state that their approach can detect if an information flow is malicious or not based on the app's behavior along the flow. That is, when a new app is installed, their system compares its behavior patterns, obtained from the complex-flows representation of the app, to decide whether it is more similar to benign or malicious apps from the training set using two-class SVM classification. To test the performance of their method, the authors used four different datasets, totaling 8,598 apps. Their model achieved a best accuracy of 94.5%.

Wang et al. [56] proposed a method which employs network traffic analysis and uses a c4.5 ML algorithm. According to the authors, c4.5 is capable of identifying Android malware with very high accuracy. The authors tested their model on the Drebin dataset [102]. The obtained results showed that the proposed model performs well when compared with state-of-the-art approaches and achieves a detection rate of up to 97.89% with the aforementioned algorithm.

Kouliaridis et al. [10] proposed a simple heterogeneous ensemble malware detection method. The ensemble model is created by averaging the output of several base models based on either static or hybrid analysis. The features extracted pertain to permissions, intents and API calls, Java classes, network traffic, and inter-process communications. The performance of this method is evaluated against several datasets, namely Drebin

[102], VirusShare [103], and AndroZoo [104]. The authors evaluated their model using several classifiers, namely Logistic Regression, Naive Bayes, Random Forest, k-NN, AdaBoost, Stochastic Gradient Descent (SGD), and SVM. Additionally, the authors used the most challenging dataset, i.e., AndroZoo, and achieved an accuracy and AUC score of 97.8% and 97.7%, respectively. Finally, feature importance is calculated for each dataset and feature.

Potha et al. [173] examined the effect of an ensemble model when external instances of different sizes and types are used. The ensemble model works by combining the output of several base models, namely Logistic Regression, MLP, and SGD. Their results demonstrated that ensemble models based on a larger and possibly homogeneous size of external instances are exceptionally effective alternative to ensemble models which comprise smaller sizes, and feasibly more heterogeneous external instances. Additionally, they examined the effect of using either the entire feature set or a random sub-space of features of instances, and showed that the latter aids an extrinsic ensemble model to further augment its performance. The authors reported 99.4%, 99.3%, and 99.7% AUC and 98.3%, 98.7%, and 99.1% accuracy on the AndroZoo, VirusShare, and Drebin datasets, respectively.

Alzaylaee et al. [174] proposed *DL-Droid*, a deep learning system which detects malicious Android apps with dynamic analysis using stateful input generation. The authors collected more than 31K apps of which more than 11K being malware. DL-Droid runs using an automated platform, which is able to perform both static and dynamic analysis. The evaluation was carried out using a real Android device and the reported required time to analyze each app was approximately 190 sec. DL-Droid achieved a detection rate of up to 97.8% when using only features stemming from dynamic analysis and 99.6% when adding features stemming from static analysis, using the Random Forest classifier.

Taheri et al. [175] developed four malware detection methods based on Hamming distance. Their models aim to detect similarities between samples which are first nearest neighbors (FNN), all nearest neighbors (ANN), weighted all nearest neighbors (WANN), and k-medoid based nearest neighbors (KMNN). The authors extracted permissions, intents and API Calls from three datasets, namely Drebin [102], Contagio mobile [81], and MalGenome [105]. Using a Random Forest Regressor feature selection algorithm, the

authors selected 300 important features. Evaluation was carried out using several classifiers, namely SVM, Decision Tree, Random Forest, and MLP, and achieved an accuracy between 90% and 99%.

Millar et al. [176] presented *DANdroid*, a mobile malware detection model which uses deep learning to classify apps. DANdroid capitalizes on a triad of features, namely Opcodes, permissions, and API calls. Their model was evaluated with apps from the Drebin dataset, obfuscated with five techniques, which produced a total of nearly 70K apps. Their results demonstrated a F-score of up to 97.3% using the CNN algorithm.

Cai et al. [177] proposed *JOWMDroid*, an Android malware detection scheme based on feature weighting, with the joint optimization of weight-mapping and classifier parameters. Eight feature categories were extracted from Android apps, and then the most important features were selected using the IG algorithm. The proposed model calculates weights per feature with three base models, and then five weight-mapping models are designed to map the initial weights to the final ones. Finally, the parameters of the weight-mapping model and the base model are jointly optimized by the differential evolution algorithm. The authors collected malware from two datasets, namely Drebin and AMD. They used several classifiers to evaluate their approach, namely SVM, Random Forest, and Logistic Regression, scoring a best accuracy of 98.1%.

Kouliaridis et al. [168] examined the effect of two well-known dimensionality reduction techniques, namely PCA and t-SNE, when applied on base models as well as ensembles. It was demonstrated that both these transformations are able to considerably increase the performance of each base model as well as the constructed ensembles. Static analysis was employed to extract permissions and intents from of 1K apps of the AndroZoo dataset. The authors evaluated their model using several classifiers, namely AdaBoost, k-NN, Logistic Regression, Naive Bayes, MLP, SGD, Random Forest and SVM, and achieved a 95.1% and 91.7% AUC and accuracy scores, respectively.

Work	Year	Analysis	Method(s)	Feature(s)	Dataset(s)	ML technique(s)
[51]	2014	Dynamic	NTA	Network traffic	N/A	Base models
[46]	2015	Static	CA	Opcodes	Drebin	Base models, DR
[53]	2015	Static	MA, CA	Package name, Permissions, API calls, Intents, Opcodes	Contagio Mobile, VirusShare	Base models
[160]	2015	Static	CA	Permissions, API Calls	McAfee	EL
[163]	2016	Static	CA	Permissions, Intents	Drebin	EL
[155]	2017	Static	CA	Permissions, Source code	M0Droid	EL
[161]	2017	Static	CA	Permissions, Intents	Contagio, MalGenome, theZoo, Malshare, VirusShare	FI, EL
[57]	2017	Static	CA	Native code	Contagio Mobile, Drebin	Base models
[55]	2018	Dynamic	SRA	CPU, Memory, and Battery usage, Process reports, Network usage	N/A	Base models
[49]	2018	Static	CA	API calls	N/A	Base models
[50]	2018	Static	CA	Information flow	N/A	Base models
[56]	2019	Dynamic	NTA	Network traffic	Drebin	Base models
[10]	2020	Hybrid	MA, CA, CI	Permissions, Intents, API calls, Java classes, inter-process communication, network traffic	Drebin, VirusShare, AndroZoo	Base models, FI, EL
[173]	2020	Static	MA	Permissions, Intents	Drebin, VirusShare, AndroZoo	Base models, EL
[174]	2020	Hybrid	MA, CA, UIA	Permissions, Intents, API Calls, Actions/Events	McAfee	Base models, FI
[175]	2020	Static	MA, CA	Permissions, Intents, API Calls	Drebin, Contagio mobile, MalGenome	Base models, FI
[176]	2020	Static	MA, CA	Permissions, Opcodes, API Calls	Drebin	Base models
[177]	2021	Static	MA, CA	Permissions, Intents, Features, Components, API Calls, Intents, Shell commands	Drebin, AMD	Base models plus weighted-mapping, FI
[168]	2021	Static	MA	Permissions, Intents	AndroZoo	Base models, EL, DR

TABLE 8.2: Outline of the surveyed works

Category	**Option**	**No. of works**
Analysis type	Static	14
	Dynamic	3
	Hybrid	2
Feature extraction method	Source Code analysis	14
	Manifest analysis	8
	Network traffic analysis	2
	Code instrumentation	1
	System resources analysis	1
	User interaction analysis	1
Dataset age	2010 to 2014	11
	2015 to 2016	5
	2017 to 2020	3
ML techniques	Base models	15
	Ensemble learning	7
	Feature importance	5
	Dimensionality reduction	2
Metrics	Accuracy as a metric	13
	AUC as a metric	7
	Other metric	4

TABLE 8.3: Summary of key characteristics observed across the surveyed works

8.3 Discussion

This section wraps up a number of key findings based on the surveyed works in section 8.2. Precisely, as shown in Tables 8.2 and 8.3, most contributions, i.e., 14 out of 19, rely on static analysis alone, while only 3 and 2 apply dynamic and hybrid analysis, respectively. Additionally, the following important observations can be made about performance optimization techniques:

- Ensemble models are considered by 7 works.
- Feature importance scores are calculated in 5 works.
- Dimensionality reduction techniques are used in 2 works.

Ensemble models have started to appear in the relevant literature after 2015. Specifically, such models are used in 6 out of 11 works employing static analysis from 2015 to 2020, and in all of the works in the same year span, which employ a regularly updated malware dataset, namely VirusShare or AndroZoo.

As shown in Table 8.3, source code analysis is the most common analysis technique used in the surveyed literature. Moreover, the most widespread classification features among the surveyed works are Permissions, Intents, and API Calls, used in 12, 9, and 7 works, respectively.

Also, as shown in Table 8.3, when focusing on the datasets employed, it is deduced that numerous works rely on outdated (and not updated) datasets. Specifically, Drebin, dated back to 2012, is the most used dataset, utilized in almost half of the surveyed works. On the other hand, Contagio Mobile and MalGenome are used in 4 and 2 works, respectively. However, the former is dated back to 2010, while the latter to 2012. Recall that previous work has shown that feature importance changes across datasets of different age [171, 156].

Additionally, previous work has demonstrated that when extracting multiple feature categories from a large collection of apps, the number of features substantially increases [10], as does the computational cost and risk of overfitting due to resulting model complexity. Therefore, when evaluating a mobile malware detection approach, the choice of dataset should play a key role in choosing the classification models and performance enhancing techniques, such as ensemble learning. On the positive side, works dated after 2015 seem to also employ newer datasets, such as VirusShare and AndroZoo, which are regularly updated.

Another important factor when assessing an ML-based approach is the primary metric used to evaluate its classification performance. The top used metrics shown in table 8.3 reveal that the Accuracy and AUC are the most commonly used. However, by inspecting the works included in section 8.2, one can conclude that a wide variety of metrics has been utilized to measure classification performance, namely detection rate (DR), true positive rate (TPR), Precision, Recall, F1, Accuracy, and Area Under the Curve (AUC). This assortment can cause a series of issues, including (a) inability to compare with state-of-the-art when the same metric is not available, (b) incorrect metrics can produce

inaccurate or over-estimated results, as in the case when using the accuracy metric with imbalanced datasets [10].

Finally, Figure 8.1 illustrates the most popular base classification models among the surveyed works. The Random forest seems to be the most popular classifier used in 11 works, followed by SVM and Naive Bayes used in 8 and 6 works, respectively.

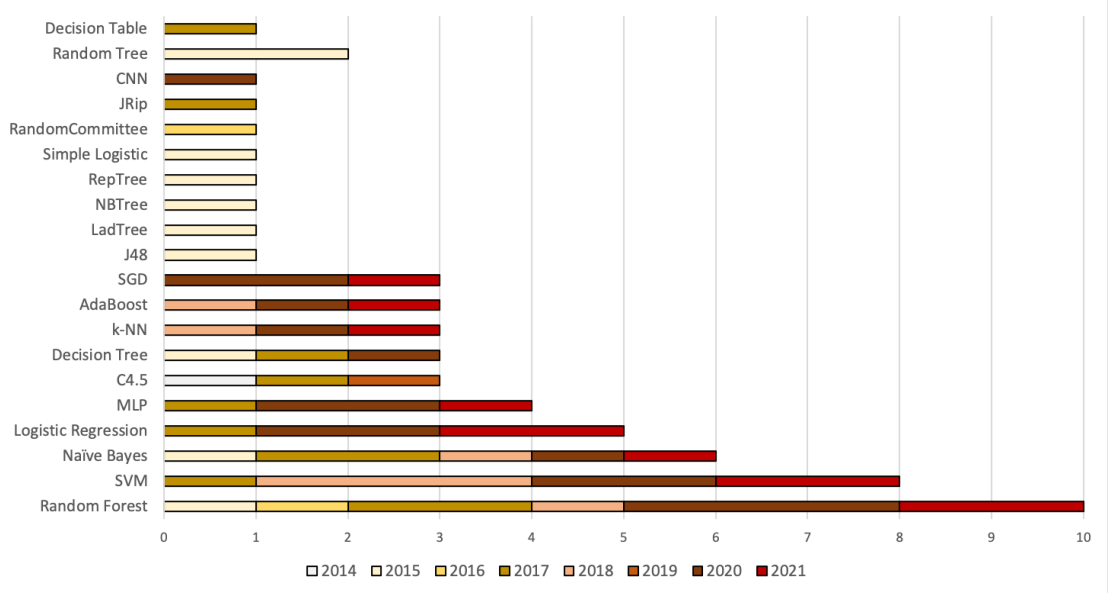

FIGURE 8.1: Number of works utilizing each base classification model per year

In an effort to address the aforementioned issues, we introduce a converging parameter selection scheme shown in Figure 8.2. Precisely, the proposed scheme aims to aid future mobile malware detection methodologies, by suggesting a unified baseline for designing more comparable and well-engineered ML-based malware detection solutions. This is achieved by considering all four key parameters into a unified typology. Simply put, the "parameters" term here refer to feature importance across datasets of different age, the increase in performance when using ensemble models instead of base models, the merit of dimensionality reduction techniques in mobile malware detection, and the advantages of each of the classification metrics. Under this mindset, Figure 8.2 comprises four steps, namely dataset age selection, analysis method selection, ML techniques selection, and performance metrics selection.

Specifically, the proposed scheme guides one in selecting optimal ML techniques based on the dataset age and analysis method chosen in the first and second step, respectively. Namely, the first two steps associate the age of the dataset used for evaluation with one

of the three analysis methods. The third step indicates the ML classification techniques to be used based on the selection made during the preceding steps. The final step depends on whether the dataset used is balanced in terms of malware and benign apps. This will determine if accuracy is indeed a trustworthy metric. In all cases however, the AUC metric is preferable, as it constitutes a more conclusive and realistic evaluation of models, even when substantially imbalanced datasets are utilized [131]. Generally, AUC quantifies the effectiveness of each examined approach for all possible score thresholds. As a rule, the value of AUC is extracted by examining the ranking of scores rather than their exact values produced when a method is applied to a dataset. And on top of everything else, AUC does not depend on the equality of distribution between positive and negative classes.

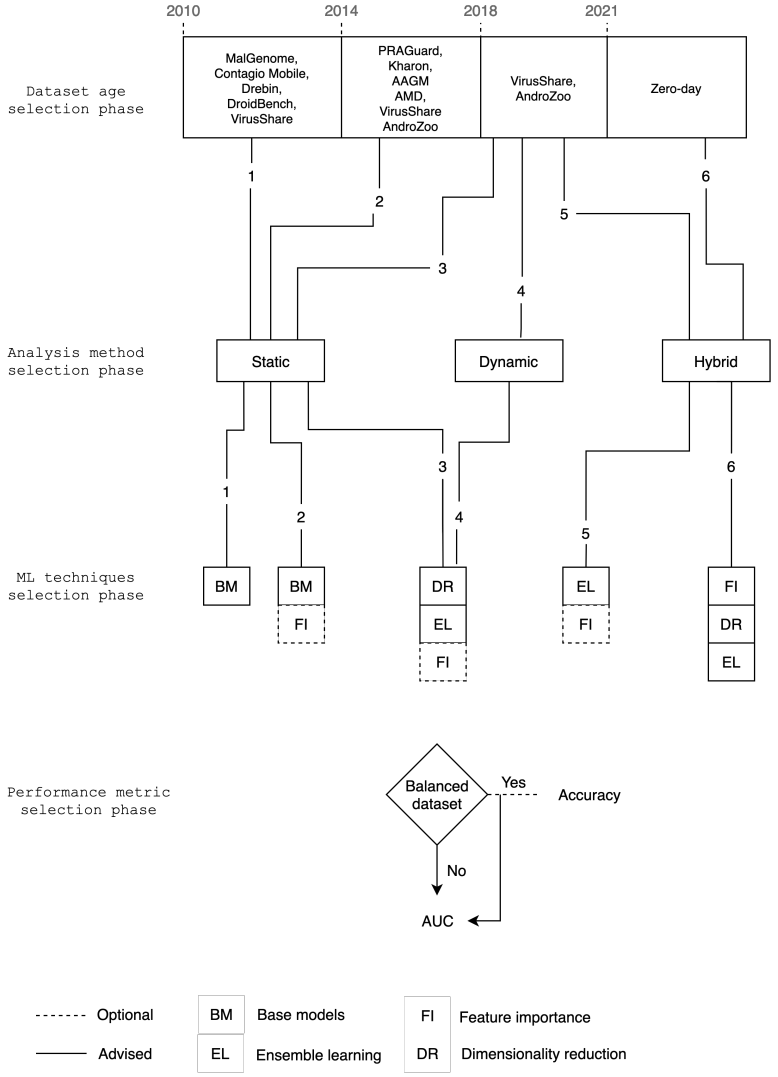

FIGURE 8.2: Baseline scheme for mobile malware detection models

8.4 Related work

As of today, the topic of Android malware detection has received plenty of attention in the literature. However, few works focus on the ML methodologies employed and, to the best of our knowledge, none of them provides a clear classification of mobile malware detection systems based on the metrics and ML techniques used. Focusing on a period spanning from 2017 to 2021, this section chronologically identifies such literature contributions and places them vis-à-vis the current work.

Yan et al. [178] offered a thorough survey on dynamic mobile malware detection approaches, summarizing a number of criteria and performance evaluation metrics for mobile malware detection. Additionally, the authors analyzed and compared the theretofore existing mobile malware detection systems based on the analysis methods and evaluation results. Finally, the authors pointed out open issues in the field and future research directions.

Odusami et al. [179] surveyed mobile malware detection techniques in an effort to identify gaps and provide insight for effective measures against unknown Android malware. Their work showed that approaches which rely on ML to detect malicious apps were more promising and produced higher detection accuracy as opposed to signature-based techniques.

Kouliaridis et al. [180] provided a holistic review of works on the topic of mobile malware detection and categorized each of them under a unique classification scheme. Precisely, the latter groups each work based on it target platform, feature selection method, and detection techniques, namely signature-based or anomaly-based detection.

Liu et al. [181] presented a comprehensive survey of Android malware detection approaches which utilize ML techniques. The authors analyzed and summarized several key topics, including sample acquisition, data preprocessing, feature selection, ML models, algorithms, and the detection performance. Finally, they elaborated on limitations of ML approaches and offered insights for potential future directions.

Gibert et al. [182] surveyed popular ML techniques for malware detection and in particular, deep learning techniques. The authors explained research challenges and limitations of legacy ML techniques and analyzed recent trends and developments in the field with

a focus on deep learning schemes. They categorized the surveyed works in three groups, namely static, dynamic, and hybrid.

As shown in Table 8.4, none of the above works offers a complete classification of each approach based on the features listed in section 8.1, namely metrics, classification models, and performance improvement techniques. Furthermore, none of them concentrate on performance improvement techniques for ML-based detection systems. The current work aspires to fill this gap and additionally introduce a decision-making scheme which will potentially guide future ML-based methodologies on deciding which analysis method, ML performance optimization technique, and metric is most fitting based on the employed dataset.

Work	**Year**	**Performance**	**PEM**	**DT**	**ML**	**AM**	**FE**	**DL**	**Datasets**	**ML PI**
[178]	2017	+	+	-	-	-	-	-	-	-
[179]	2018	-	+	+	+	-	-	-	-	-
[180]	2020	+	-	+	-	+	+	-	-	-
[181]	2020	+	-	-	+	-	+	-	+	-
[182]	2020	-	-	-	+	-	-	+	-	-
Current	2021	+	+	+	+	+	+	-	+	+

TABLE 8.4: Important topics addressed by the related works. PEM: Performance evaluation metrics, DT: Detection techniques, ML: Machine learning, AM: Analysis methods, FE: Features and feature extraction, DL: Deep learning, ML PI: ML performance improvement

8.5 Conclusions

This chapter provides a state-of-the-art survey on ML-powered Android malware detection techniques. To do so, we categorize and succinctly analyze state-of-the-art works in the literature during the last 7 years, i.e., from 2014 to 2021, based on the analysis type, feature extraction method, dataset, ML classification techniques, and metrics used in their performance evaluation. Additionally, we elaborate on our findings and research trends, as well as possible issues and future directions. From the results, it becomes

obvious that the majority of the approaches embrace a different set of basic parameters, including the dataset, the analysis (feature collection), and detection evaluation metrics. To moderate this issue, we proposed a converging scheme to serve as a baseline for future mobile ML-based Android malware detection approaches.

Chapter 9

Conclusions and Future Directions

9.1 Conclusions

Mobile devices are currently dominating the marketing world. Since the day the first Android phone appeared in 2008 [183], smartphones have mushroomed into an everyday necessity. Indeed, handheld devices are almost an indispensable part of who we are and how we communicate with the world today. But along with the increased use comes an explosion of mobile malware designed to target smartphones and tablets. Hence, it is an urgent need to investigate for malware detection methods which not only achieve high detection efficacy, but they are also able to support the decision making in critical situations. Reaching to the end of this doctoral thesis, it becomes clear that mobile malware detection is a highly active research area as the popularity of mobile devices constantly increases, spurred also by the proliferation of the new generation cellular networks, i.e., 5G and beyond. Focusing on the popular Android platform, throughout our research, we investigated the current state-of-the-art in mobile malware detection approaches, while we also introduced novel detection methodologies for providing optimal security countermeasures.

More specifically, as described in chapter 2, mobile malware detection is a research field which encompasses various techniques. Through an extensive analysis, we revealed a gamut of open research challenges, and we provide future directions and best practices

for building reliable mobile malware classification methodologies. That said, as explained in chapters 6 and 7, the present doctoral thesis introduced two novel methodologies for improving the state-of-the-art in mobile malware detection.

Also, Chapter 3, demonstrated how mobile malware behavioral patterns can be collected in an open cloud database. Specifically, using reverse engineering techniques we gathered CPU, memory and network usage, as well as process and network statistics, and used machine learning to identify behavioral patterns which can be used to detect mobile malware. Chapter 4 attempted to answer a rudimentary but decisive question; do new mobile malware families require additional feature categories to be extracted during analysis? If yes, which feature category can produce optimal results and how feature importance is related to the age of the mobile app. To answer this matter, we meticulously analysed several well-known mobile apps datasets, dated from 2010 to 2020 to gain insight on feature importance when static analysis is involved.

In Chapter 5, we focused on extracting new feature categories such as java classes and inter-process communication, along with well-known feature categories, such as network traffic by means of dynamic instrumentation. These additional feature categories not only augmented the overall classification performance, but were also used to further examine feature importance among these categories. Chapter 6 assessed the merit of two low-level static features, namely permissions and intents, as well as their contribution on the use of two well-known dimensionality reduction techniques, i.e., Principal Component Analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE).

Finally, Chapter 7 introduced an extrinsic malware detection method based on ensemble learning. We employed static analysis to get two well-known feature categories, namely permissions and intents. Next we used three classifiers as base models, and measured the correlation among mobile malware by building a more sophisticated Extrinsic random-based ensemble.

9.2 Thesis Contributions

In accordance to the objectives presented in chapter 1, this doctoral thesis aimed to shed light to the state-of-the-art methodologies, which aim to provide optimal and robust counteraction to the mobile malware threat. Additionally, with a focus on the Android

platform, the thesis concentrated on the deployment of advanced ML techniques to tackle known limitations of the mobile malware detection literature. A side-by-side comparison of the thesis objectives and our contributions in terms of publications in peer-reviewed venues is given in Table 9.1.

Objective	**Chapter**	**Contribution**	**Publication**
Obj. 1	2	A Survey on Mobile Malware Detection Techniques	[55]
Obj. 2	3	Mal-warehouse: A data collection-as-a-service of mobile malware behavioral patterns	[6]
Obj. 2	4	Feature importance in Android malware detection	[156]
Obj. 2	5	Two anatomists are better than one - Dual-level Android malware detection	[10]
Obj. 2	5	Androtomist tool	[122]
Obj. 3	6	Improving Android malware detection through dimensionality reduction techniques	[168]
Obj. 3	7	An Extrinsic Random-based Ensemble Approach for Malware Detection	[173]
Obj. 1	8	A comprehensive survey on machine learning techniques for Android malware detection	Submitted

TABLE 9.1: Overall PhD Thesis Contribution.

Precisely, in accordance to the first objective of this thesis, chapter 2 offered detailed information on current mobile malware, as well as an overview of the different machine learning algorithms, evaluation metrics, and performance enhancing techniques used to counter mobile malware. Furthermore, it provided an extensive analysis of mobile malware detection techniques resulting the following research challenges, which remain widely open among the reviewed literature.

- Lack of guiding data on classification feature importance.
- Lack of new feature categories, which may improve the classification efficacy.

- Limited evaluation data regarding newest datasets, which include more challenging (advanced) malware instances.
- Limited research towards robust hybrid malware detection approaches.

Given the aforementioned challenges, our work advocates that the topic of mobile malware detection has still many steps to take for reaching the point of providing complete counteraction and security to smartphone users. Based on the analysis provided in chapter 2 and [6], it becomes clear that there is a need for re-evaluating the merit of current classification feature categories, as well as the precise contribution of static analysis techniques. In this direction, this doctoral thesis analyzed the top three used datasets in the literature to identify the feature importance of the most widely used static feature categories, namely permissions and intents. Furthermore, we were able to find major differences in the most recent and challenging dataset, i.e., AndroZoo. This PhD thesis also developed and meticulously assessed a methodology for mobile malware detection, by combining static analysis with dynamic instrumentation into a hybrid open source solution. This methodology has been succesfully tested against the most challenging datasets, consisting of the latest mobile malware. Our work also evaluated the feature importance of each classification feature category, for both static and dynamic analysis, and for each of the three utilized mobile malware corpora. The feature importance ranking was achieved via a linear regression algorithm. According to the evaluation results presented in chapter 5 and [10], our approach was able to surpass state-of-the-art mobile detection solutions in terms performance metrics over three distinct benchmark datasets, namely Drebin, VirusShare, and AndroZoo.

In this direction and in accordance to the third objective of this doctoral thesis, we were able to solve the issues presented in 4, by introducing two novel methodologies for reliable mobile malware detection using classification features stemming from static analysis. First, based on the results provided in chapter 6, we examined the effect of dimensionality reduction techniques in mobile malware detection. Precisely, we analyzed the effect of two well-known methods namely, PCA and t-SNE when exclusively applied on malware detection base verifiers as well as ensembles, respectively. By doing so, we were able to demonstrate that both transformations are able to considerably increase the performance of each base model as well as the proposed ensembles. Secondly, we proposed a more sophisticated extrinsic ensemble approach, which provides accurate mobile

malware classification, by averaging the output of the base models for each test instance separately. Our method was tested against benchmark datasets, i.e., VirusShare and AndroZoo and is directly comparable with other state-of-the-art detection approaches.

9.3 Future Research Directions

This Phd thesis has mainly contributed to the field of mobile malware detection by introducing versatile methodologies with an eye towards addressing key limitations in this field of research. Additionally, a significant step was taken in the direction of mobile malware detection, by reporting and decomposing methodologies which have been proposed in the literature so far. Undoubtedly, the quest for novel malware analysis and detection schemes is one of the pillars of the future of mobile security. Overall, it became clear that several steps need to be taken for introducing more robust and reliable malware analysis and detection techniques that can bring the advantage to the defensive side. To this end, possible research directions to this line of research are as follows.

- Feature importance - As mobile apps continue to evolve, research towards understanding a ML models logic is required for reliable and more robust classification by focusing only on the variables (features) that matter the most.
- Additional feature categories - By taking into consideration the research challenges in the field, as well as the continued security enhancements in the Android OS, it is worth of investigating additional feature categories, in which future detection systems can rely on to reliably detect future, more insidiously-engineered instances of mobile malware.
- Dimensionality reduction approaches - A ML model trained on a large number of features gets increasingly dependent on the volume of data it was trained on, resulting in an over-fitted model, which in turn lead to poor performance on real-case scenarios. In Chapter 6, we demonstrated how dimensionality reduction techniques successfully improve the performance of ML models.
- Ensemble learning approaches - As shown in [10] ensemble learning techniques in ML classification combine the result of multiple models, and thus are able to produce superior performance in terms of prediction accuracy as compared to using

a single base model. The investigation of ensemble learning approaches can still significantly benefit the effectiveness of mobile malware detection approaches.

- New challenging datasets - It has been widely reported in the literature that the community lacks of contemporary datasets that reflect the modern network conditions and malware attack characteristics. Hence, an aspirant future direction is to design multidisciplinary mobile malware datasets that combine malware families from various realms like IoT devices and smart devices, and investigate for unified and interoperable detection solutions.

Bibliography

[1] Y. D. Lin, C. Y. Huang, M. Wright, and G. Kambourakis. Mobile application security. *Mobile Application Security*, 47(6):21–23, 2014.

[2] mobile os market share. https://gs.statcounter.com/os-market-share/mobile/worldwide. Accessed: 2020-09-10.

[3] Mobile malware evolution 2020. https://securelist.com/mobile-malware-evolution-2020/101029/. Accessed: 2020-09-01.

[4] Mobile threat report. https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf, . Accessed: 2020-09-01.

[5] F.A. Narudin, A. Feizollah, N.B. Anuar, and A. Gani. Evaluation of machine learning classifiers for mobile malware detection. *Soft Comput*, 20:343–357, 2016.

[6] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, and S. Chen. A survey on mobile malware detection techniques. *IEICE Transactions on Information and Systems*, E103.D(2):204–211, 2020.

[7] P. Yan and Z. Yan. A survey on dynamic mobile malware detection. *Software Quality Journal*, 26:891–919, 2018.

[8] A. Souri and R. Hosseini. A state-of-the-art survey of malware detection approaches using data mining technique. *Human-centric Computing and Information Sciences*, 8:3, 2018.

[9] M. Odusami, O. Abayomi-Alli, S. Misra, O. Shobayo, R. Damasevicius, and R. Maskeliunas. Android malware detection: A survey. *Applied Informatics*, pages 255–266, 2018.

[10] Vasileios Kouliaridis, Georgios Kambourakis, Dimitris Geneiatakis, and Nektaria Potha. Two anatomists are better than one-dual-level android malware detection. *Symmetry*, 12(7):1128, 2020.

[11] D. Papamartzivanos, D. Damopoulos, and G. Kambourakis. A cloud-based architecture to crowdsource mobile app privacy leaks. *PCI '14*, 2014.

[12] Mobile threat report. https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf, . Accessed: 2020-09-01.

[13] Sophos 2020 threat report. https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophoslabs-uncut-2020-threat-report.pdf. Accessed: 2021-03-10.

[14] Permissions on android. https://developer.android.com/guide/topics/permissions/overview, . Accessed: 2020-09-01.

[15] Interprocess communication. https://developer.android.com/guide/components/processes-and-threads, . Accessed: 2020-09-01.

[16] Intent. https://developer.android.com/reference/android/content/Intent, . Accessed: 2020-09-01.

[17] Security-enhanced linux in android. https://source.android.com/security/selinux, . Accessed: 2020-09-01.

[18] Seccomp bpf. https://www.kernel.org/doc/html/v4.16/userspace-api/seccomp_filter.html, . Accessed: 2020-09-01.

[19] Google android enterprise security whitepaper 2018. https://source.android.com/security/reports/Google_Android_Enterprise_Security_Whitepaper_2018.pdf, . Accessed: 2020-09-01.

[20] Trusty. https://source.android.com/security/trusty, . Accessed: 2020-09-01.

[21] Android verified boot. https://source.android.com/security/verifiedboot, . Accessed: 2020-09-01.

[22] S. Peng, G. Wang, and S. Yu. Modeling the dynamics of worm propagation using two-dimensional cellular automata in smartphones. *Journal of Computer and System Sciences*, 79(5):586 – 595, 2013. ISSN 0022-0000.

[23] M. Anagnostopoulos, G. Kambourakis, and S. Gritzalis. New facets of mobile botnet: Architecture and evaluation. *International Journal of Information Security*, 12 2015.

[24] Bankbot returns on play store. https://thehackernews.com/2017/11/bankbot-android-malware.html. Accessed: 2020-09-01.

[25] J. Bickford, R. O'Hare, A. Baliga, V. Ganapathy, and L. Iftode. Rootkits on smart phones: Attacks, implications and opportunities. pages 49–54, 02 2010.

[26] Mobile threat report. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf. Accessed: 2020-09-01.

[27] Hidden miners on google play. https://www.kaspersky.com/blog/google-play-hidden-miners/21882/. Accessed: 2020-09-01.

[28] A.Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. *Proceedings of the ACM Conference on Computer and Communications Security*, 10 2011.

[29] Nickyspy. https://fortiguard.com/encyclopedia/virus/2971289/android-nickispy-a-tr-spy. Accessed: 2020-09-01.

[30] Google play. https://play.google.com/. Accessed: 2020-09-01.

[31] Apktool. https://ibotpeaches.github.io/Apktool/. Accessed: 2020-09-01.

[32] Sdk build tools. https://developer.android.com/studio/releases/build-tools, . Accessed: 2020-09-01.

[33] j.R. Vacca. Computer and information security handbook. *Morgan Kaufmann*, 2017.

[34] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 303–313, 2015.

[35] M. La Polla, F. Martinelli, and D. Sgandurra. A survey on security for mobile devices. *IEEE Communications Surveys & Tutorials*, 15(1):446–471, 2013.

[36] E. Gandotra, D. Bansal, and S. Sofat. Malware analysis and classification: A survey. *Journal of Information Security*, pages 56–64, 2014.

[37] P. Yan and Z. Yan. A survey on dynamic mobile malware detection. *Software Quality Journal*, 26:891–919, 2018.

[38] N. Idika and M. Aditya. A survey of malware detection techniques. *Purdue University*, 03 2007.

[39] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. *Proceedings of 16th ACM Conference on Computer and Communications Security*, pages 235–245, 2009.

[40] C. Chen, G. Lai, and J. Lin. Identifying threat patterns of android applications. *AsiaJCIS*, pages 69–74, 2017.

[41] 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. *AsiaJCIS*, pages 62–69, 2012.

[42] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. Garcia Bringas, and G. Alvarez. Puma: Permission usage to detect malware in android. *Advances in Intelligent Systems and Computing*, 189:289–298, 2013.

[43] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. Garcia Bringas, and G. Alvarez. Machine learning for android malware detection using permission and api calls. *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 300–305, 2013.

[44] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. Garcia Bringas, and G. Alvarez. Mast: Triage for market-scale mobile malware analysis. *Proceedings of the 6th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2013.

[45] S. Liang and X. Du. Permission-combination-based scheme for android mobile malware detection. *IEEE International Conference on Communications*, pages 2301–2306, 2014.

[46] G. Canfora, F. Mercaldo, and C. A. Visaggio. Permission-combination-based scheme for android mobile malware detection. *12th International Joint Conference on e-Business and Telecommunications*, pages 27–38, 2015.

[47] M. Yusof, M. M. Saudi, and F. Ridzuan. A new mobile botnet classification based on permission and api calls. *Seventh International Conference on Emerging Security Technologies*, pages 122–127, 2017.

[48] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225, 2018.

[49] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu. Malpat: Mining patterns of malicious and benign android apps via permission-related apis. *IEEE Transactions on Reliability*, 67(1):355–369, 2018.

[50] F. Shen, J. Del Vecchio, A. Mohaisen, S. Ko, and L. Ziarek. Android malware detection using complex-flows. *IEEE Transactions on Mobile Computing*, 2018.

[51] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici. Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security*, 43, 2014.

[52] D. Damopoulos, G. Kambourakis, and G. Portokalidis. The best of both worlds. a framework for the synergistic operation of host and cloud anomaly-based ids for smartphones. *EuroSec*, 2014.

[53] J.-W. Jang, H. Kang, J. Woo, A. Mohaisen, , and H. K. Kim. Andro-autopsy: Anti-malware system based on similarity matching of malware and malware creator-centric information. *Digital Investigation*, 14:17–35, 2015.

[54] Z. Chen, Q. Yan, H. Han, S. Wang, L. Peng, L. Wang, , and B. Yang. Machine learning based mobile malware detection using highly imbalanced network traffic. *Information Sciences*, 433-434:346–364, 2018.

[55] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, and G. Wang. Mal-warehouse: A data collection-as-a-service of mobile malware behavioral patterns. *IEEE SmartWorld*, pages 1503–1508, 2018.

[56] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia. A mobile malware detection method using behavior features in network traffic. *Journal of Network and Computer Applications*, 133:15–25, 2019.

[57] S. Alam, Z. Qu, R. Riley, Y. Chen, , and V. Rastogi. Droidnative: Automating and optimizing detection of android native code malware variants. *Computers and Security*, pages 230–246, 2017.

[58] Android runtime. https://source.android.com/devices/tech/dalvik, . Accessed: 2020-09-01.

[59] T. Fei and Y. Zheng. A hybrid approach of mobile malware detection in android. *Journal of Parallel and Distributed Computing*, 103:22–31, 2017.

[60] Z. Hanlin, P. Khanh, C. Yevgeniy, G. Linqiang, W. Sixiao, Y. Wei, L. Chao, C. Genshe, S. Dan, and Blasch. Scanme mobile: a cloud-based android malware analysis service. *ACM SIGAPP Applied Computing Review*, 16:36–49, 2016.

[61] D. Damopoulos, G. Kambourakis, S. Gritzalis, and S. O.Park. Exposing mobile malware from the inside (or what is your mobile app really doing?). *Peer-to-Peer Networking and Applications*, 7:687–697, 2014.

[62] C. Miller, D. Blazakis, D. Daizovi, S. Esser, V. Lozzo, , and R.P. Weinmann. A hybrid approach of mobile malware detection in android. *Indianapolis: John Wiley & Sons*, 2012.

[63] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis. Rage against the virtual machine: Hindering dynamic analysis of android malware. *EuroSec*, pages 1–12, 2014.

[64] A. Damodaran, F. Di Troia, C. Aaron Visaggio, T. H. Austin, and M. Stamp. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, pages 1–12, 2017.

[65] Q. Zhou, F. Feng, Z. Shen, R. Zhou, M. Y. Hsieh, and K. C. Li. A novel approach for mobile malware classification and detection in android systems. *Multimedia Tools and Applications*, 2019.

[66] S. Sharmeen, S. Huda, J. H. Abawajy, W. N. Ismail, and M. M. Hassan. Malware threats and detection for industrial mobile-iot networks. *IEEE Access*, pages 15941–15957, 2018.

[67] I. Ali-Gombe, B. Saltaformaggio, J. R. Ramanujam, D. Xu, and G. G. Richard. Samadroid: A novel 3-level hybrid malware detection model for android operating system. *IEEE Access*, 6:4321–4339, 2018.

[68] S. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica (Slovenia)*, 31:249–268, 01 2007.

[69] I. Rish. An empirical study of the naive bayes classifier. Technical report, 2001.

[70] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[71] D.W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley, 2004. ISBN 9780471654025.

[72] J. R. Quinlan. Induction of decision trees. *MACH. LEARN*, 1:81–106, 1986.

[73] P. Burman. Estimation of optimal transformations using v-fold cross validation and repeated learning-testing methods. *The Indian Journal of Statistics, Series A*, 52(3):314–345, 1990.

[74] T.G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[75] L. Van Der Maaten E. Postma and J. Van den Herik. Dimensionality reduction: a comparative review. *J Mach Learn Res*, 10:66–71, 2009.

[76] Z. Cheng and Z. Lu. A novel efficient feature dimensionality reduction method and its application in engineering. *Complexity*, 2018:1–14, 10 2018.

[77] Cyber security report 2017. https://fsecurepressglobal.files.wordpress.com/2017/02/cyber-security-report-2017.pdf. Accessed: 2020-09-01.

[78] Android debug bridge (adb). https://developer.android.com/studio/command-line/adb. Accessed: 2020-09-01.

[79] Sdk platform tools release notes. https://developer.android.com/studio/releases/platform-tools, . Accessed: 2020-09-01.

[80] Tutorial: Root shield tablet k1 (nougat). http://nvidiashieldzone.com/shield-tablet/shield-tablet-k1/android-7-nougat/root-shield-tablet-k1-nougat/. Accessed: 2020-09-01.

[81] Contagio. http://contagiominidump.blogspot.com/. Accessed: 2020-09-10.

[82] android-malware - collection of android malware samples. https://www.welivesecurity.com/2017/02/14/new-android-trojan-mimics-user-clicks-download-dangerous-malware/. Accessed: 2020-09-01.

[83] Android armour. https://nakedsecurity.sophos.com/2013/01/10/a-chink-in-android-armour/. Accessed: 2020-09-01.

[84] Badnews. https://www.infosecurity-magazine.com/news/badnews-android-malware-pushes-fraud-schemes/. Accessed: 2020-09-01.

[85] Akamai. https://blogs.akamai.com/2014/12/ios-and-android-os-targeted-by-man-in-the-middle-attacks.html. Accessed: 2020-09-01.

[86] Akamai. https://www.f-secure.com/v-descs/backdoor_iphoneos_xsser.shtml. Accessed: 2020-09-01.

[87] Coinkrypt. https://blog.lookout.com/coinkrypt. Accessed: 2020-09-01.

[88] Notcompatible. https://www.f-secure.com/v-descs/trojan-proxy_android_notcompatible.shtml. Accessed: 2020-09-01.

[89] Nickyspy. https://fortiguard.com/encyclopedia/virus/2971289/android-nickispy-a-tr-spy. Accessed: 2020-09-01.

[90] Spamsoldier. https://www.adaptivemobile.com/blog/mobile-malware-spam-bot-spamsoldier-returns. Accessed: 2020-09-01.

[91] Angry birds - lena. https://thehackernews.com/2012/04/legacy-native-malware-in-angry-birds.html. Accessed: 2020-09-01.

[92] Doctor web. https://www.drweb.com/. Accessed: 2020-09-01.

[93] Angry birds transformers. http://news.softpedia.com/news/New-Android-Malware-Poses-As-Angry-Bird-Transformers-Wipes-Device-Clean-460997.shtml. Accessed: 2020-09-01.

[94] Feabme. http://www.virusradar.com/en/Android_Spy.Feabme.C/description. Accessed: 2020-09-01.

[95] Rumms. https://www.fireeye.com/blog/threat-research/2016/04/rumms-android-malware.htm. Accessed: 2020-09-01.

[96] Trojandownloader.agent.ji. https://www.welivesecurity.com/2017/02/14/new-android-trojan-mimics-user-clicks-download-dangerous-malware/. Accessed: 2020-09-01.

[97] Orange data mining tool. https://orange.biolab.si/, . Accessed: 2020-09-01.

[98] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. *12th USENIX Security Symposium, USENIX Security Symposium*, pages 169–186, 2003.

[99] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, page 2721–2744, 2006.

[100] T. Blasing, L. Batyuk, A. D. Schmidt, S. A. Camtepe, and S. Albayrak. An android application sandbox system for suspicious software detection. *5th International Conference on Malicious and Unwanted Software*, 5:55–62, 2010.

[101] D. Damopoulos, S. A. Menesidou, G. Kambourakis, M. Papadaki, N. Clarke, and S. Gritzalis. Evaluation of anomaly-based ids for mobile devices using machine learning classifiers. *Security and Communication Networks*, 5:3–14, 2012.

[102] D. Arp, M. Spreitzenbarth, M. Huebner, H. Gascon, and K. Rieck. Drebin: Efficient and explainable detection of android malware in your pocket. *21th Annual Network and Distributed System Security Symposium (NDSS)*, 12(7):1128, 2014.

[103] Virus share. https://virusshare.com, . Accessed: 2020-09-01.

[104] K. Allix, T.Bissyandé F., J. Klein, and Y. Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471. ACM, 2016.

[105] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 12(7), 2012.

[106] Droidbench. https://github.com/secure-software-engineering/DroidBench. Accessed: 2020-09-10.

[107] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto. Stealth attacks: an extended insight into the obfuscation effects on android malware. *Computers and Security*, 51:16–31, 2015.

[108] N. Kiss, J. Lalande, M. Leslous, and V. Viet Triem Tong. Kharon dataset: Android malware under a microscope. In *Learning from Authoritative Security Experiment Results*, San Jose, United States, May 2016. The USENIX Association. URL https://hal-univ-orleans.archives-ouvertes.fr/hal-01300752.

[109] A.H. Lashkari, A.F. A.Kadir, H. Gonzalez, K.F. Mbah, and A. A. Ghorbani. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 233–23309, 2017.

[110] Amd malware dataset. http://amd.arguslab.org/. Accessed: 2020-09-10.

[111] V. Kouliaridis, G. Kambourakis, D. Geneiatakis, and N. Potha. Two anatomists are better than one-dual-level android malware detection. *Symmetry*, 12(7):1128, 2020.

[112] S. Lei. A feature selection method based on information gain and genetic algorithm. In *2012 International Conference on Computer Science and Electronics Engineering*, volume 2, pages 355–358, 2012.

[113] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[114] W.K. Ehrlich, k. Futamura, and D. Liu. *An Entropy Based Method to Detect Spoofed Denial of Service (Dos) Attacks*, pages 101–122. Springer US, 2008.

[115] Zisis Tsiatsikas, Dimitris Geneiatakis, Georgios Kambourakis, and Angelos D. Keromytis. An efficient and easily deployable method for dealing with dos in sip services. *Computer Communications*, 57:50 – 63, 2015.

[116] Z. Tsiatsikas, A. Fakis, D. Papamartzivanos, D. Geneiatakis, G. Kambourakis, and C. Kolias. Battling against ddos in sip: Is machine learning-based detection an effective weapon? In *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, volume 04, pages 301–308, 2015.

[117] A. Feizollah, N.B. Anuar, R. Salleh, and A.W.A. Wahab. A review on feature selection in mobile malware detection. *Digital Investigation*, 13:22 – 37, 2015. ISSN 1742-2876.

[118] K. Zhao, D. Zhang, X. Su, and W. Li. Fest: A feature extraction and selection tool for android malware detection. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 714–720, 2015.

[119] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, 9(11):1869–1882, 2014.

[120] Y. Aafer, W. Du, and H. Yin. Droidapiminer: Mining api-level features for robust malware detection in android. *International Conference on Security and Privacy in Communication Systems*, 2013.

[121] D. Damopoulos, G. Kambourakis, and G. Portokalidis. The best of both worlds: A framework for the synergistic operation of host and cloud anomaly-based ids for smartphones. *In proceedings of the Seventh European Workshop on System Security (EuroSec)*, (6), 2014.

[122] Androtomist. https://androtomist.com/, . Accessed: 2020-09-01.

[123] Virtualbox. https://virtualbox.com/. Accessed: 2020-09-01.

[124] Androtomist's source code. https://github.com/billkoul/Androtomist, . Accessed: 2020-09-01.

[125] European union public licence. https://ec.europa.eu/info/european-union-public-licence_en. Accessed: 2020-09-01.

[126] Frida. https://frida.re/. Accessed: 2020-09-01.

[127] Monkey. https://developer.android.com/studio/test/monkey. Accessed: 2020-09-01.

[128] N. T. Cam, V.-H. Pham, , and T. Nguyen. Detecting sensitive data leakage via inter-applications on android using a hybrid analysis technique. *Cluster Computing*, 22:1055–1064, 2017.

[129] I. Ali-Gombe, B. Saltaformaggio, J. R. Ramanujam, D. Xu, and G. G. Richard. Toward a more dependable hybrid analysis of android malware using aspect-oriented programming. *Computers & Security*, 73:235–248, 2018.

[130] X. Wang, Y. Yang, and S. Zhu. Automated hybrid analysis of android malware through augmenting fuzzing with forced execution. *IEEE Transactions on Mobile Computing*, 12:2768 – 2782, 2019.

[131] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27:861–874, 2006.

[132] F. Idrees, M. Rajarajan, M. Conti, R. Rahulamathavan, and T. Chen. Pindroid: A novel android malware detection system using ensemble learning. *Computers and Security*, 68:36–46, 2017.

[133] N. Milosevic, A. Dehghantanha, and K-K. R. Choo. Machine learning aided android malware classification. *Comput. Electr. Eng. Int. J.*, 61:266–274, 2017.

[134] Dexclassloader. https://developer.android.com/reference/dalvik/system/DexClassLoader. Accessed: 2020-09-01.

[135] java.net.socket. https://developer.android.com/reference/java/net/Socket. Accessed: 2020-09-01.

[136] Orange. https://orange.biolab.si/, . Accessed: 2020-09-01.

[137] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb. Maldozer: Automatic framework for android malware detection using deep learning. *Digital Investigation*, 24:S48–S59, 2018.

[138] E. Noreen. Computer-intensive methods for testing hypotheses: An introduction. *New York: Wiley*, 1989.

[139] L. Xu, D. Zhang, N. Jayasen, and J. Cavazos. Hadm: Hybrid analysis for detection of malware. *Proceedings of SAI Intelligent Systems Conference 2016 Lecture Notes in Networks and Systems*, pages 702–724, 2018.

[140] F. Martinelli, F. Mercaldo, and A. Saracino. Bridemaid: An hybrid tool for accurate detection of android malware. *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security - ASIA CCS 17*, 2017.

[141] X. Wang, Y. Yang, and S. Zhu. A tan based hybrid model for android malware detection. *Journal of Information Security and Applications*, 54, 2020.

[142] K. Allix, T.F. Bissyande, and Q. Jerome et al. Empirical assessment of machine learning-based malware detectors for android. *Digital Investigation*, 21(1):183–211, 2016.

[143] Android enterprise security whitepaper 2018. https://source.android.com/security/reports/Google_Android_Enterprise_Security_Whitepaper_2018.pdf. Accessed: 2020-09-01.

[144] Selinux. https://source.android.com/security/selinux. Accessed: 2020-09-01.

[145] Seccomp filter. https://android-developers.googleblog.com/2017/07/seccomp-filter-in-android-o.html, . Accessed: 2020-09-01.

[146] K. Patel and B. Buddadev. Detection and mitigation of android malware through hybrid approach. *Applied Informatics*, pages 455–463, 2015.

[147] L. H. Tuan, N. T. Cam, and V.-H. Pham. Enhancing the accuracy of static analysis for detecting sensitive data leakage in android by using dynamic analysis. *Cluster Computing*, 22:1079–1085, 2017.

[148] Y. Tsutano, S. Bachala, W. Srisa-An, G. Rothermel, and J. Dinh. Jitana: A modern hybrid program analysis framework for android platforms. *Journal of Computer Languages*, 52:55–71, 2019.

[149] X. Wang, Y. Yang, and S. Zhu. A hybrid detection method for android malware," 2019 ieee 3rd information technology. *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference*, 2019.

[150] Amaaas. https://amaaas.com/. Accessed: 2020-09-01.

[151] Virustotal. https://virustotal.com/, . Accessed: 2020-09-01.

[152] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[153] Kerstin Bunte, Michael Biehl, and Barbara Hammer. A general framework for dimensionality-reducing data visualization mapping. *Neural Computation*, 24(3): 771–804, 2012.

[154] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8): 861–874, 2006.

[155] N. Milosevic, A. Dehghantanha, and K-K. R. Choo. Machine learning aided android malware classification. *Computers & Electrical Engineering*, 61:266–274, 2017.

[156] V. Kouliaridis, G. Kambourakis, and T. Peng. Feature importance in android malware detection. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1449–1454, 2020.

[157] Rafael Vega Vega, Héctor Quintián, José Luís Calvo-Rolle, Álvaro Herrero, and Emilio Corchado. Gaining deep knowledge of Android malware families through dimensionality reduction techniques. *Logic Journal of the IGPL*, 27(2):160–176, 2018.

[158] M. La Polla, F. Martinelli, and D. Sgandurra. A survey on security for mobile devices. *IEEE Communications Surveys Tutorials*, 15(1):446–471, 2013.

[159] Mohsen Damshenas, Ali Dehghantanha, Kim-Kwang Raymond Choo, and Ramlan Mahmud. M0droid: An android behavioral-based malware detection model. *Journal of Information Privacy and Security*, 11(3):141–157, 2015.

[160] S. Y. Yerima, S. Sezer, and I. Muttik. High accuracy android malware detection using ensemble learning. *IET Information Security*, 9(6):313–320, 2015.

[161] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan. Pindroid: A novel android malware detection system using ensemble learning methods. *Computers & Security*, 68:36–46, 2017.

[162] D. Geneiatakis, G. Baldini, I.N. Fovino, and I. Vakalis. Towards a mobile malware detection framework with the support of machine learning. *Security in Computer and Information Sciences*, pages 119–129, 2018.

[163] L. D. Coronado-De-Alba, A. Rodríguez-Mota, and P. J. Escamilla-Ambrosio. Feature selection and ensemble of classifiers for android malware detection. pages 1–6, 2016.

[164] T. Chakraborty, F. Pierazzi, and V. S. Subrahmanian. Ec2: Ensemble clustering and classification for predicting android malware families. *IEEE Transactions on Dependable and Secure Computing*, 17(2):262–277, 2020.

[165] Mcafee. https://www.mcafee.com/en-us/index.html. Accessed: 2021-03-10.

[166] thezoo aka malware db. https://thezoo.morirt.com/. Accessed: 2021-03-10.

[167] Malshare project. https://malshare.com/about.php. Accessed: 2021-03-10.

[168] Vasileios Kouliaridis, Nektaria Potha, and Georgios Kambourakis. Improving android malware detection through dimensionality reduction techniques. *Machine Learning for Networking*, page 57–72, 2021.

[169] A. Bacci, A. Bartoli, F. Martinelli, E. Medvet, F. Mercaldo, and C.A. Visaggio. Impact of code obfuscation on android malware detection based on static and dynamic analysis. pages 379–385, 01 2018. doi: 10.5220/0006642503790385.

[170] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis. Rage against the virtual machine. In *Proceedings of the Seventh European Workshop on System Security - EuroSec '14*. ACM Press, 2014. doi: 10.1145/2592791.2592796.

[171] S. Roy, J. DeLoach, Y. Li, N. Herndon D., Caragea, X. Ou, V.P. Ranganath, H. Li, and N. Guevara. Experimental study with real-world data for android app

security analysis using machine learning. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, page 81–90. Association for Computing Machinery, 2015. doi: 10.1145/2818000.2818038.

[172] Android runtime (art) and dalvik. https://source.android.com/devices/tech/dalvik, . Accessed: 2020-09-10.

[173] N. Potha, V. Kouliaridis, and G. Kambourakis. An extrinsic random-based ensemble approach for android malware detection. *Connection Science*, page 1–17, Dec 2020. doi: 10.1080/09540091.2020.1853056.

[174] M K. Alzaylaee, S. Y. Yerima, and S. Sezer. Dl-droid: Deep learning based android malware detection using real devices. *Computers & Security*, 89:101663, 2020. doi: https://doi.org/10.1016/j.cose.2019.101663.

[175] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti. Similarity-based android malware detection using hamming distance of static binary features. *Future Generation Computer Systems*, 105:230–247, 2020. doi: https://doi.org/10.1016/j.future.2019.11.034.

[176] S. Millar, N. McLaughlin, J. Martinez del Rincon, P. Miller, and Z. Zhao. Dandroid: A multi-view discriminative adversarial network for obfuscated android malware detection. Association for Computing Machinery, 2020. doi: 10.1145/3374664.3375746.

[177] L. Cai, Y. Li, and Z. Xiong. Jowmdroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. *Computers & Security*, 100:102086, 2021. doi: https://doi.org/10.1016/j.cose.2020.102086.

[178] P. Yan and Z. Yan. A survey on dynamic mobile malware detection. *Software Quality Journal*, 26(3):891–919, May 2017. doi: 10.1007/s11219-017-9368-4.

[179] M. Odusami, O. Abayomi-Alli, S. Misra, O. Shobayo, R. Damasevicius, and R. Maskeliunas. *Android Malware Detection: A Survey*, page 255–266. Springer International Publishing, 2018. doi: 10.1007/978-3-030-01535-0_19.

[180] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, and S. Chen. A survey on mobile malware detection techniques. *IEICE Transactions on Information and Systems*, E103.D(2):204–211, Feb 2020. doi: 10.1587/transinf.2019ini0003.

[181] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu. A review of android malware detection approaches based on machine learning. *IEEE Access*, 8:124579–124607, 2020. doi: 10.1109/ACCESS.2020.3006143.

[182] D. Gibert, C. Mateu, and J. Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, Mar 2020. doi: 10.1016/j.jnca.2019.102526.

[183] J. Gozalvez. First google's android phone launched [mobile radio]. *IEEE Vehicular Technology Magazine*, 3(4):3–69, 2008.