Profiling and Visualizing Android Malware Datasets

Tomas Javier Concepcion Miranda

DOCTORAT BRETAGNE LOIRE / MATHSTIC

CentraleSupélec

THÈSE DE DOCTORAT DE

CentraleSupélec
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par
Tomas CONCEPCION MIRANDA

Profiling and Visualizing Android Malware Datasets

Thèse présentée et soutenue à Rennes, le 29 novembre 2022
Unité de recherche : IRISA
Thèse N° : 2022CSUP0005

Rapporteurs avant soutenance :

Jacques Klein — Professor, Université du Luxembourg
Lorenzo Cavallaro — Professor, University College London

Composition du Jury :

Présidente :	Céline Hudelot	Professeur des Universités, CentraleSupélec
Examinateurs :	Vincent Roca	Chargé de Recherches, INRIA
Dir. de thèse :	Jean-François Lalande	Professeur des Universités, CentraleSupélec
	Valérie Viet Triem Tong	Professeur, CentraleSupelec

Invité(s) :

Pierre Wilke — CentraleSupélec

Acknowledgments – Remerciements

First of all, I would like to thank Céline Hudelot, Vincent Roca, Jacques Klein and Lorenzo Cavallaro for taking an interest in my work and for accepting to be members of the jury. Especially Jacques and Lorenzo for their detailed review of this manuscript.

I'm grateful of my thesis advisors, Jean-François Lalande, Valérie Viet Triem Tong and Pierre Wilke, for all the advice they have given me throughout this time on the different obstacles and challenges of this thesis, and also for the useful advice on the subtleties and popular phrases of the French language, which I still have a lot to learn.

I would also like to thanks Michel Hurfin for being part of my *Comité de Suivi Individuel* (CSI) and my colleagues at the CIDRE team of all these years for their company and good humor.

Finalmente, en español, quiero agradecer a mi familia, amigos, y a mis padres Itza y Tomás, quienes me han dado su apoyo incondiconal para seguir adelante en mi trabajo en esos tiempos inciertos, y que gracias a ellos, pude lograr mi objetivo.

Abstract

Mobile devices are ubiquitous: nowadays most people own a mobile telephone. Because of this global presence, it is a target of interest for attackers. These attacks are delivered via malicious applications that can harm mobile device, for example by obtaining illegal control of it, gaining unauthorized access to the user's personal information or scamming a person with a ransom, among other malicious intents. Researchers in malware analysis put their effort to recognize these types of programs before they are installed on a user device. To do this, they perform experiments to automatically detect malware, for example with machine learning, where a learning phase is done over a set of already known malware and goodware. Afterwards, researchers make a performance evaluation over several new datasets, independent of the one in the learning phase. Depending on their choice of datasets, the evaluation of the experiments can yield acceptable results, or outstanding but overestimated results. Consequently, datasets with malware and benign samples are important elements to consider when designing an experiment.

Researchers in Android malware detection encounter several obstacles to create good experimental datasets. Many different markets are available to take applications from. Because of their nature, they are not suited for experiments: there is no clear and easy way to extract applications from them and to know the ground truth about being a malware or not. Additionally, malware could be underrepresented in a extract of the market. As a result, researchers use other experimental datasets, including their ground truth, that other researchers have shared to the research community. These shared datasets slowly became old and cited, and other researchers decide to craft their own experimental dataset from these shared datasets with applications from other sources. We wonder if these crafted experimental datasets have an influence over the results of conducted experiments, and if so, if some crafting methodology is more suitable than others.

This thesis presents, first, a method to evaluate the quality of datasets based on a statistical test that helps to compare a crafted dataset against a large set of applications such as markets. We show that historical datasets of the literature are of low quality, which justifies the need to create new up-to-date datasets. Second, we introduce an algorithm to update mixed datasets of malware/goodware of low quality in order to resemble a target dataset that cannot be used directly, *e.g.* a market. We evaluate the updated mixed datasets using a machine learning algorithm and we show that the detection of malware in our up-to-date dataset becomes a more difficult problem to solve. Lastly, we introduce DaViz, a dataset visualization tool for exploring and comparing Android malware datasets, which enables researchers to visualize the biases in datasets of the literature, and obtain useful information from them.

Résumé

Les dispositifs mobiles sont ubiquitaires: aujourd'hui la majorité des gens possèdent un téléphone mobile. A cause de leur présence global, ces dispositifs sont une cible d'intérêt pour les attaquants. Ces attaques sont véhiculées au travers des applications malveillantes qui peuvent nuire aux dispositifs mobiles, par exemple en obtenant leur contrôle illégale, en gagnant accès aux informations personnelles de l'utilisateur, ou en menaçant les données à l'aide d'un rançongiciel – exemples parmi d'autres types de malveillances. Les chercheurs en analyse de malware travaillent à reconnaître ces types de programmes avant qu'ils soient installés sur un dispositif utilisateur. Pour faire cela, ils réalisent des expériences pour automatiquement détecter ces malware, par exemple avec de l'apprentissage automatique, où une phase d'apprentissage est réalisée sur un ensemble de malware et des applications bénignes déjà connues. Ensuite, les chercheurs réalisent une évaluation de la performance sur plusieurs nouveaux datasets, indépendants de ceux utilisés dans la phase d'apprentissage. Selon le dataset choisi, les résultats des expériences peuvent être acceptables ou bien exceptionnellement bons parce que surestimés. Par conséquent, les datasets de malware et applications bénignes sont des éléments importants à considérer quand nous élaborons une expérience.

Les chercheurs en détection de malware Android rencontre plusieurs obstacles pour créer un bon dataset expérimental. D'abord, beaucoup de magasins d'applications sont disponibles pour récupérer des applications. Leur façon de distribuer ces applications ne convient pas pour faire des expériences: il n'y a pas de méthode aisée pour extraire des applications et connaître la vérité terrain sur le fait d'être un malware ou non. De plus, les malware peuvent être sur ou sous-représentés dans un extrait du magasin d'applications. De ce fait, les chercheurs utilisent d'autres datasets expérimentaux, incluant la vérité terrain, que d'autres chercheurs ont partagés à la communauté scientifique. Ces datasets partagés deviennent peu à peu de vieux datasets avec de nombreuses citations. D'autres chercheurs ont aussi décidé de créer leurs propres datasets expérimentaux avec des applications provenant d'autres sources. Nous pouvons alors nous demander si tous ces datasets expérimentaux ont une influence sur les résultats des expériences conduites, et si c'est le cas, si une méthodologie de création est plus propice que d'autres.

Cette thèse présente, premièrement, une méthode pour évaluer la qualité des datasets basée sur un test statistique qui aide à comparer un dataset créé avec un grand ensemble d'applications par exemple issu d'un magasin d'applications. Nous montrons alors que les datasets historiques de la littérature sont de mauvaise qualité, ce qui justifie le besoin de créer des nouveaux datasets plus à jour. Deuxièmement, nous introduisons un algorithme pour mettre à jour des datasets mixtes de malware/goodware de mauvaise qualité afin de ressembler à un dataset cible qui ne peut pas être utilisé directement, *e.g.* un magasin d'applications. Nous évaluons les datasets mixtes mis à jour en utilisant un algorithme d'apprentissage automatique et nous montrons que la détection de malware sur notre dataset mis à jour devient un problème plus difficile à résoudre. Enfin, nous introduisons DaViz, un outil de visualisation de datasets pour explorer et comparer des datasets d'applications Android. Cet outil permet aux chercheurs de visualiser les biais dans les datasets de la littérature, et d'obtenir des informations utiles à leur propos.

Résumé substantiel en français

Introduction

Les dispositifs mobiles sont ubiquitaires: aujourd'hui la majorité des gens possèdent un téléphone mobile. A cause de leur présence global, ces dispositifs sont une cible d'intérêt pour les attaquants. Ces attaques sont véhiculées au travers des applications malveillantes qui peuvent nuire aux dispositifs mobiles, par exemple en obtenant leur contrôle illégal, en gagnant accès aux informations personnelles de l'utilisateur, ou en menaçant les données à l'aide d'un rançongiciel – exemples parmi d'autres types de malveillances. Les chercheurs en analyse de malware travaillent à reconnaître ces types de programmes avant qu'ils soient installés sur un dispositif utilisateur. Pour faire cela, ils réalisent des expériences pour automatiquement détecter ces malware, par exemple avec de l'apprentissage automatique, où une phase d'apprentissage est réalisée sur un ensemble de malware et des applications bénignes déjà connues. Ensuite, les chercheurs réalisent une évaluation de la performance sur plusieurs nouveaux datasets, indépendants de ceux utilisés dans la phase d'apprentissage. Selon le dataset choisi, les résultats des expériences peuvent être acceptables ou bien exceptionnellement bons parce que surestimés. Par conséquent, les datasets de malware et applications bénignes sont des éléments importants à considérer quand nous élaborons une expérience.

Les chercheurs en détection de malware Android rencontre plusieurs obstacles pour créer un bon dataset expérimental. D'abord, beaucoup de magasins d'applications sont disponibles pour récupérer des applications. Leur façon de distribuer ces applications ne convient pas pour faire des expériences: il n'y a pas de méthode aisée pour extraire des applications et connaître la vérité terrain sur le fait d'être un malware ou non. De plus, les malware peuvent être sur ou sous-représentés dans un extrait du magasin d'applications. De ce fait, les chercheurs utilisent d'autres datasets expérimentaux, incluant la vérité terrain, que d'autres chercheurs ont partagés à la communauté scientifique. Ces datasets partagés deviennent peu à peu de vieux datasets avec de nombreuses citations. D'autres chercheurs ont aussi décidé de créer leurs propres datasets expérimentaux avec des applications provenant d'autres sources. Nous pouvons alors nous demander si tous ces datasets expérimentaux ont une influence sur les résultats des expériences conduites, et si c'est le cas, si une méthodologie de création est plus propice que d'autres.

Contributions

Cette thèse travaille sur deux axes: la qualité des datasets et la visualisation de ceux-ci. La thèse propose des contributions indépendantes sur ces deux axes bien que les contributions de visualisation des datasets aide à comprendre les enjeux de qualité.

Ainsi, la thèse débute par la présentation de l'état de l'art relatif aux datasets d'applications Android, que ce soit pour les goodware ou les malware. Nous analysons les datasets utilisés par la communauté scientifique et nous montrons que certains travaux souffrent d'irrégularités, par exemple sur l'équilibrage des dates des applications utilisés. Ces irrégularités confirment des travaux similaires conduits par Pendlebury etal et Allix etal. Cette première analyse n'étant basée que sur la lecture des méthodologies employées dans la littérature, la thèse poursuit l'investigation de ce problème en cherchant à représenter graphiquement ces irrégularités.

Une première contribution concerne donc la visualisation de datasets d'applications Android. Nous proposons une méthode intéractive pour visualiser et comparer des datasets, au travers d'un outil nommé DaViz. DaViz offre des fonctionnalités non rencontrées dans les outils existants: la possibilité de comparer plusieurs datasets, de sélectionner des sous parties interactivement, et de visualiser des caractéristiques grâce à trois représentations différentes. Le bénéfice attendu est de pouvoir réaliser des explorations de datasets inconnus ou de comparer deux à deux des datasets pour en saisir les différences notables. L'outil est exploité sur les datasets de la littérature (Drebin, AMD) mais aussi sur des extractions récentes de goodware et de malware.

Il confirme visuellement les irrégularités identifiées précédemment. Cette partie du travail de thèse nous a conduit à vouloir quantifier formellement ces différences, ce qui l'objet de la contribution suivante.

Une seconde contribution est dédiée à l'élaboration d'une méthode pour évaluer la qualité des datasets à l'aide d'un test statistique. Ce test permet de comparer un dataset créé manuellement avec un grand ensemble d'applications par exemple extrait d'un magasin d'applications. A l'aide de cette méthode statistique, nous montrons alors que les datasets historiques de la littérature sont de mauvaise qualité, ce qui justifie le besoin de créer des nouveaux datasets plus à jour. D'autres raisons, comme le manque de labels goodware/malware dans les sources possibles d'applications, par exemple le Google Play Store, nous a poussé à nous intéresser à la création de datasets dits "mixtes" i.e. contenant des goodware et des malwares. L'enjeu est alors de construire de tels datasets correctement labellisés, tout en étant proche de la réalité, c'est-à-dire de ce que l'on obtiendrait en échantillonnant un magasin d'application.

Une troisième contribution est dédiée à l'élaboration d'un algorithme qui travaille sur un dataset mixte mais biaisée, c'est-à-dire que certaines caractéristiques sont sur ou sous-représentées comparée à ce que l'on trouverait dans un magasin d'application. Notre algorithme va chercher à modifier autant que faire ce peut le dataset mixte afin de le faire ressembler au dataset dit "cible" par exemple un extrait du magasin d'application. L'intérêt de l'algorithme est de travailler sur un dataset mixte dont les labels sont connus i.e. la caractéristique "être un malware" est établie. Nous obtenons en un nombre optimal d'itérations un dataset dit "débiaisé" dont l'éloignement du dataset cible est contrôlé par un paramètre δ.

Nos expérimentations montrent qu'il est difficile de modifier les très anciens datasets comme Drebin, vu leur éloignement du dataset cible. Pour les datasets mixtes que nous avons construit avec des sources plus récentes, la convergence est plus aisée. Les datasets débiaisés produits ont été évalués en les utilisant dans des expérimentations de détection basés sur des algorithmes classiques de machine learning. Ces expériences montrent qu'il est plus difficile de détecter les malware quand on utilise des datasets débiaisés. De plus, apprendre sur des datasets débiaisés donnent de meilleurs résultats lors de la détection sur les ensembles de tests.

Conclusion

Les résultats obtenus dans ces travaux de thèses ouvrent différentes perspectives. D'autres outils statistiques pour mesurer la qualité des datasets pourraient être employés afin de mieux quantifier la "distance" existant entre deux datasets. De plus, les méthodes employées sur les datasets d'applications Android pourraient être utilisées pour des datasets qui concernent les applications x86. L'outil de visualisation pourrait être étendu pour être capable de basculer de la visualisation d'ensembles à la visualisation d'une seule application: l'enjeu est alors de produire des vues internes des applications qui soient pertinentes pour l'analyste explorant un dataset inconnu.

Contents

List of Figures

List of Tables

Prologue

1 Introduction

Mobile devices, primarily smartphones, have become ubiquitous: most people own at least one of them. They represent, according to statcounter[1] as of June 2022, 59.77% of all types of computers used in the world nowadays. Among the different mobile operating systems (OS), Android is the most popular in the world with 72.12% of market share as of June 2022[2]. These devices are used for many purposes, from everyday actions such as communication between persons and media consumption, to sensitive operations such as money transferring and business communication. Because of the number of people using smartphones, and all the possible tasks that can be accomplished with them, the Android system has become a top target for attackers.

1: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet

2: https://gs.statcounter.com/os-market-share/mobile/worldwide

Of all the types of possible attacks, there are programs that may harm a system or trick its users to obtain a gain, without the user's knowledge and consent. These types of programs are called *malware*. Although there is no practical definition of what a "malware" is, there are definitions for *types* of malicious applications based on how they work. Among these types, we can mention: spyware, which sends personal information to the attackers without the user's knowledge; ransomware, which blocks the access of users to a system and demand them a ransom to grant access to it again, and bankbots, which steal login credentials for banking applications from users.

To protect users from such threats, experts in malware analysis take measures to detect these applications before they arrive to the end user. To do this, they take applications from the places users tend to obtain them, such as Google Play, and conduct experiments to verify what these applications do. If analysts discover any malicious behavior in an application, they flag it as malware, and depending of this behavior, they can classify this malware into one of the numerous types there exists.

Analysts can operate manually, when a specific application is performing suspicious actions or upon request of users. Nevertheless, with the high number of applications on different online stores, it is impossible to rely on manual analysis for each new application or update of an application. This is why the research community has worked on automatic analysis procedures to extract relevant data from applications and eventually decide if an application is performing malicious activities.

Machine learning (ML) based algorithms have been used extensively in automating the attribution of maliciousness of applications [1–4]. By extracting properties from applications, called "features" in the ML jargon, either statically (directly from their code) or dynamically (from executing them and registering their behavior), an ML algorithm can obtain outstanding results at detecting malicious applications. Academics contributing in this research area need to compare their results with previous ones published in the literature. As a consequence, they use datasets, literally "sets of data", as inputs to compare multiple detection algorithms.

The first seminal paper that released a dataset has been published by Arp *et al.* [5]. They released the Drebin dataset. At the time, it was the largest dataset containing malware samples available to the research community. Subsequent papers reused this dataset in order to provide incremental and reproducible results. In the meantime, whereas Google was releasing new versions of the Android OS, Google Play drastically increased in the number of applications. This effort of collecting and labeling malware samples is not trivial, making Drebin and AMD [6], another bigger Android malware dataset, an easy and fast way to obtain samples. As a consequence, few new datasets were published and released in the next ten years, and the old and widely cited Drebin dataset continued to be used. At the time of writing this thesis, Drebin is cited more than 2 000 times according to Google Scholar, which is an impressive score for only a single paper.

[5]: Arp et al. (2014), 'DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket'

[6]: Wei et al. (2017), 'Deep Ground Truth Analysis of Current Android Malware'

The reader should recall that the intent behind a dataset in a malware detection experiment is to mimic, in a laboratory, the conditions encountered in the reality. This "reality" is also called the "ground truth" in the literature, and we call it the "target" set of applications in this chapter. That is, for malware detection, mimicking the difficulty for an automatic program to discover malware hidden in Google Play, where thousands of applications are uploaded each day. We believe that it would be surprising that Drebin could mimic such a problem on the current Google Play, with recent sophisticated malware.

The context of this thesis is, thus, focused on the quality of the datasets of the literature and, if the quality is confirmed to be low – which is our intuition, on the elaboration of new methods to produce datasets of high quality, *i.e.*, able to mimic the reality.

This thesis explores the creation and usage of experimental datasets for various usages, such as the typical malware detection problem. One of the issues we intend to solve is that experimental datasets are created, disregarding the target set of applications that the experiment aims. Thus, even though the experiment's results are excellent, its method would fail when operated on the target production environment.

The term "quality" for a dataset is undefined at this stage of writing, which we formalize in Chapter 4. For now, we explain what we believe are the problems of quality in datasets. We illustrate these problems with two hypotheses (that will be confirmed later in the manuscript):

- H1: Experiments with 99% accuracy are suspicious
- H2: Input datasets influence the results of the experiments

H1: Experiments with 99% accuracy are suspicious Malware detectors using ML would predict a sample as a positive (*i.e.*, is a malware), or negative (*i.e.*, is a goodware). Since we know in advance the labels (*i.e.*, which are malware and goodware), we can evaluate how well the detector predicted malware (true positives) and goodware (true negatives) correctly. Results from malware detectors using ML algorithms are usually given in the form of:

- accuracy (the concentration of true predictions among all predictions)

- precision (the concentration of true positives among all positive predictions)
- recall (the concentration of true positives among all true samples)
- F1-score (a harmonic mean of precision and recall)

For all these scores given in percentage, 100% is a perfect score. With the increasing popularity of ML techniques, analysts have focused in developing better discriminants used by these machine learning algorithms to differentiate malware from benign programs. There are several studies that show very good results with over 99% [7–9] of detection accuracy. We can think that with these types of results, Android malware should no longer be a problem if these experiments were transferred from prototypes to real antiviruses. But unfortunately, this is still not the case: we observe that malicious programs are still discovered, for example, on the Google Play. Even if their choice of discriminant features is sound, we suspect that the way the chosen experimental dataset was created may have produced overestimated results.

[7]: Idrees et al. (2017), 'PIndroid: A novel Android malware detection system using ensemble learning methods'
[8]: Karbab et al. (2018), 'MalDozer: Automatic framework for android malware detection using deep learning'
[9]: P. et al. (2019), 'A machine learning based approach to detect malicious android apps using discriminant system calls'

H2: Input datasets influence the results of the experiments During this thesis, two papers were published suggesting that input datasets influence experiments in Android malware detection. Allix *et al.* [10] discuss the influence of the applications' date in experimental datasets used in Android malware detection. The authors have found that the way researchers create their experimental datasets leads to biased results, in Android malware detection using ML. They compare a typical experiment configuration using ML to a more real-life scenario one, and the results of both experiments tended to differ. This indicates that experiments with the typical configuration are not suited to work against new malware just like in real-life, which is misleading despite having announced high detection accuracy in their results. Pendlebury *et al.* [11] also discuss the influence of date on Android malware detection experiments using ML. Results of their experiments show a decay in performance of state-of-the-art ML Android malware detection algorithms, when applications in the test dataset are newer than in the training set. The authors also indicate the proportion of malware and benign application in the dataset as a way to influence the results of experiments.

[10]: Allix et al. (2015), 'Are Your Training Datasets Yet Relevant?'

[11]: Pendlebury et al. (2019), 'TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time'

We believe that this idea of "distribution resemblance", whether it is about date or malware proportions in experimental datasets, can be generalized to a wide variety of characteristics of Android applications in order to obtain a dataset that resembles more the target set of applications the experiment aims.

As a conclusion, this thesis focus on three main research questions:

- Is it possible to quantify the quality of a dataset of applications with respect to the target set of applications? This question suggests that a metric should be elaborated to compare a dataset with the "target" set of applications.
- Can other characteristics, besides the date and the ratio of malware/benign applications, be used to detect differences between datasets? Because inconsistencies have already been identified in previous works concerning the date of applications and the ratio of malware and benign applications of the dataset, this thesis intends

to confirm and extend these works on more attributes than solely the date and malware/benign ratio.

- Given an arbitrary set of applications of low quality, can we investigate the reasons behind this low quality? This question is related to the fact that researchers may obtain datasets from different sources and eventually untrusted third parties without reliable information on the building process of this obtained dataset. In such a case, researchers would want to legitimately explore and assert the quality of the dataset.

Contributions

To answer the previous research questions, we propose three main contributions in this thesis:

1. We introduce a new tool to visually explore and compare datasets called DaViz [12]. With this tool, researchers can inspect datasets to find interesting patterns about them, and also distinguish at a glance the differences between various datasets.
2. We give guidelines on how to calculate the size of an experimental dataset, and introduce the notion of "representativity" using combinations of characteristics we call "classes" [13]. We also propose the use of a statistical test to verify that datasets are indeed different against a reference dataset chosen to study.
3. We then use these classes to create a "debiasing" method [13] for datasets to resemble another reference dataset of choice. We use this method to create several "debiased" datasets that resemble more the "target" dataset than datasets from the literature. We use one of these "debiased" datasets in a malware detection experiment based on ML algorithms, showing that the detection performance is lower for old datasets.

Associated publications

Our work on visualization and DaViz have been presented in the RESSI 2022 conference [12]. The method for debiasing datasets, alongside the code and the datasets used in the experiment, are available in the Dada dataset [14]. Lastly, the results from our debiasing experiments were published in the IEEE Transactions on Information Forensics and Security journal [13].

[12]: Concepción Miranda et al. (2022), 'DaViz: Visualization for Android Malware Datasets'

[14]: Concepcion Miranda et al. (2021), *Dada: Debiased Android DAtasets*

[13]: Concepción Miranda et al. (2022), 'Debiasing Android Malware Datasets: How Can I Trust Your Results If Your Dataset Is Biased?'

Outline

This dissertation is divided in two parts. The first part consists of this introduction and Chapter 2, that gives a background on the usage of datasets for experiments in Android malware.

The second part is dedicated to our contributions of this thesis. First, Chapter 3 explores a new visualization tool called DaViz. Next, Chapter 4 introduces and defines the notion of *representativity* of datasets. It also

shows a procedure to differentiate datasets using statistical tests. Then, Chapter 5 presents a new method for generating "debiased" datasets to resemble a studied dataset that is much more difficult to use directly.

Finally, Chapter 6 summarizes the contributions of this thesis and gives perspectives for future works.

Research Android Datasets and Where to Find Them

2

In order to start an experiment in malware research, malware samples, as well as goodware samples, must be collected to form an experimental dataset. Researchers have resorted to a variety of places to find malware samples: from VX (Virus eXchange) forums to research teams willing to share their samples to the community. The quality of the input dataset in an experiment is important, as it can have an influence in the results of the experiment. The way it is constructed, its size, and the sources, among other properties, need to be taken into account.

This chapter focuses on the study of Android datasets in malware research, the different uses of them, and the irregularities and biases encountered in their usage. In Section 2.1, we start by defining what a *dataset* is. After that, we explore the different platforms for software distribution in Android that users utilize to download and install applications, alongside the various research datasets used in security studies on Android malware. Finally, in Section 2.2, we review the biases that experiments suffer due to the construction of its experimental dataset, with a review of Android malware detection experiments to see from which irregularities they suffer, and lastly notice what missing type of bias experiments do not pay attention to.

2.1 Datasets used in the literature

In this section, we discuss the several datasets and repositories used in Android malware research.

We begin by defining the notion of *dataset*: the Cambridge dictionary defines a dataset as "a collection of separate sets of information that is treated as a single unit by a computer". For this thesis, an Android application dataset (that can be an "Android malware dataset" when it includes malware) fulfills at least one of the following criteria:

- A dataset is a collection of raw Android applications (`.apk` files)
- A dataset is a collection of results from analyses done to Android applications that includes a hash value for each application

In this matter, we can categorize the diverse datasets according to their main purpose. This section presents and discusses three types of datasets: markets, market archives, and specific datasets. Markets are repositories where users download applications. Market archives are application repositories that collect samples from markets. Specific datasets are ones created by researchers to aid in the creation of experimental datasets.

2.1.1 Markets

In this section, we start by defining what a market is, what they contain, what security mechanisms they offer to avoid the distribution of malicious

applications, and the different markets in existence and those no longer available. At the end of the section, we conclude with the value of markets for researchers.

Markets are software distribution platforms where users can find and download applications, and developers can distribute their software. These types of datasets have a *dynamic* behavior in regards to how their applications are managed: new ones can be added increasing the size of the dataset, old ones could be removed and never found again, and newer versions replace the existing ones.

Google Play

The first Android application dataset to be considered is Google Play [15] (also known as Google Play Store, and previously known as Android Market). This is the main repository where Android users can officially download applications from. It has around 2.5 million applications [16], which has peaked in 2018 [17] to a little more than 3.7 million applications. Previously, applications could be downloaded directly from the Google Play website, allowing the use of scraping methods to obtain as many applications as possible. This was done through the use of an API that Google changed at some point in time, which is not openly disclosed [18]. Nowadays, users use the official "Play Store" application to search, download and manage installed applications. Other alternative clients exits [19, 20], but they violate Google's terms of service.

[15]: Google LLC (2022), *Google Play*

[16]: AppBrain (2022), *Android and Google Play statistics*

[17]: AppBrain (2018), *Number of Android apps on Google Play*

[18]: Viennot et al. (2014), 'A Measurement Study of Google Play'

[19]: Rahul Patel (2022), *AuroraOSS*

[20]: Yeriomin (2016), *Yalp Store*

Although Google Play is the main platform for obtaining Android applications, its usage in research is limited. Firstly, because of the changing nature of the market, applications may be removed or updated, forbidding researchers to obtain samples from previous experiments and worsening reproducibility. And secondly, because of the lack of transparent API to interact with the platform, and limitations to download applications, it is difficult for researchers to get samples from the official market.

Third-party markets

Besides Google Play, other third-party markets exist that also offer software distribution to Android. They operate in a similar fashion to Google Play: they are backed by a company or organization, they have an application to contact the repository and download the applications. They also tend to have a different distribution model than the official market, or have certain regulations to what type of applications they offer.

One of those is SlideME [21]. It is one of the first third-party markets launched in 2008. Developers who want their application in this market must pass an approval process before being published. Samsung, which is a manufacturer of Android devices, has its own market called Galaxy Store [22] since 2009. It mainly comes preinstalled on Samsung Galaxy smartphones, but it can be installed in other devices. As recently as 2021, it hosted malware in the form of clones of other applications, in this case one for movie piracy [23]. Caterpillar also provides a specialized application store for their phones [24]. Aptoide [25] is another third-party

[21]: SlideME LLC (2022), *SlideME*

[22]: Samsung Electronics Co., Ltd. (2022), *Galaxy Store*

[23]: Hager (2021), *Samsung's Galaxy Store is distributing apps that could infect phones with malware*

[24]: Bullitt Mobile Ltd (2022), *Toolbox App*

[25]: Aptoide S.A. (2022), *Aptoide*

market launched in 2011 with development starting in 2009. Their main repository offer applications for users to download and install. Besides the main store, this market has allowed the "decentralization" of the "stores": it has partnered with other companies to allow the redistribution of applications throughout different stores like Softsonic, Multilaser, and Aptoko. Amazon started distributing Android applications since 2011 in its Amazon Appstore [26]. It shares few of the most popular applications also found in Google Play. F-Droid [27] is a market specialized in distributing free and open-source software (FOSS). It offers their own repository with its official application to download and manage the applications it contains. Since F-Droid offers FOSS, the source code of each application is provided in the description. This allows users and other external auditors to more easily verify the code for security flaws or for malicious behavior. The market itself builds and signs each application before distribution*. External repositories can be added to the client, which allows developers to distribute their software directly without passing through the F-Droid publishing process. AndroidOut [28] is a web-based market where applications, some that can be found in Google Play, are curated by a community of people. According to their website [29], they count with more than 2.3 million registered users and more than a million reviewed applications. It had reached over 2 million downloads in 2012 [30]. Other smaller markets we can mention are MiKandi [31] (which is defunct), 1mobile [32] and GetJar [33] (where malware have been found [34]).

[26]: Amazon.com, Inc. (2022), *Amazon Appstore*

[27]: F-Droid Limited (2022), *F-Droid*

[28]: GLOBAL PTE LTD (2022), *AndroidOut*

[29]: GLOBAL PTE LTD (2022), *Androidout corporate - Who we are - What we do Androidout.com*

[30]: Sanford (2013), *MiKandi Reaches Milestone With Mobile Devices, Registered Users*

[31]: MiKandi LLC (2022), *MiKandi*

[32]: 1Mobile Market (2022), *1Mobile Market*

[33]: Laurs (2022), *GetJar*

[34]: AndroidAdvices (2012), *Remove Android:Plankton [PUP] Virus from Android Device after Downloading Apps from GetJar*

Several database services exists that gather information about markets. These are aimed for developers to provide data about these markets for business intelligence, but may also provide useful information for researchers that study these markets. Services like AppBrain [35], MixRank [36], anroidrank.org [37], SimilarWeb [38], TapTap [39], data.ai [40] and SensorTower [41] provide information about numerous indicators, like downloads, SDK usage, release dates, etc.

[35]: Vogelzang et al. (2022), *AppBrain*

[36]: Milliken (2022), *MixRank*

[37]: Katrenic (2022), *ANDROIDRANK*

[38]: SimilarWeb LTD (2022), *Top Apps Ranking*

[39]: TapTap Pte. Ltd. (2022), *TapTap*

[40]: DATA.AI EUROPE LIMITED (2022), *data.ai*

[41]: SensorTower, Inc (2022), *SensorTower*

Regional third party markets

Besides those markets that can be accessed by everyone in the world, there are others aimed towards a specific group of people or region. These regional markets has emerged because of restrictions in their home countries, or their target audience's language. The Android market ecosystem in China is primarily ruled by regional markets, where they offer applications mainly in Chinese language. One of the reasons behind this is the restriction of Google services in China [42]. Therefore, Chinese markets target mainly only China and neighboring countries. Example of these are Baidu, 360 and Tencent, as well as markets created by phone manufacturers like Huawei, Xiaomi and Lenovo, which offer applications to the Chinese community. A study about these Chinese markets and a comparison to Google Play have been done by Wang *et al.* [43]. Another regional market is Cafe Bazaar [44], an Iranian-based marketplace founded in 2011. It is a popular market in Iran, with 97% of country's share in 2017 [45]. Uptodown [46] is a Spanish-based marketplace founded in 2002. Since 2011 [47], it provides Android applications, and nowadays it

[42]: Quinn (2012), 'Google services blocked in China'

[43]: Wang et al. (2018), 'Beyond Google Play: A Large-Scale Comparative Study of Chinese Android App Markets'

[44]: Hezardastan Information Technology Development Group (2022), *Cafe Bazaar*

[45]: Niayesh (2017), *How developers can make money in Iran's app market?*

[46]: Domínguez et al. (2022), *We are Uptodown*

[47]: Domínguez et al. (2011), *Uptodown*

* FAQ - General | F-Droid

focuses on them according to their "About" page, with a store application to access it.

Discussion

Researchers that aim to study markets face the dynamicity of these platforms. It is a problem in terms of reproducibility: since applications can be removed (or replaced by newer versions), experiments can no longer be reproduced with the same data. Besides this, none of these markets provide an API to interact with them, making it difficult to obtain applications and metadata from the sites. Malware required for experiments is very hard to find in markets, because these platforms tend to remove malicious applications once they find one. The quantity of markets available is another source of problem: because markets are different from each other, studies done on one market do not apply directly to others. And because of the difficulty to interact with markets to obtain applications, market metanalyses would take a lot of effort and time to perform. Nevertheless, since markets are the main focus of security for finding and mitigating the spread of malicious applications, they need to be considered when testing security solutions for them.

2.1.2 Market archives

In comparison to markets, market archives are software repositories that add applications from markets to their collection. They tend to use crawlers or rely on user submissions to obtain applications.

[18]: Viennot et al. (2014), 'A Measurement Study of Google Play'

One of the first archives for Google Play available was PlayDrone [18]. The issue this archive tackled is crawling the Google Play. Google limits its access by requiring clients to use registered accounts and exclusively use the Google Play application to request Google's servers. To bypass this, PlayDrone uses real accounts in combination with the API they reverse engineered from the store application. The system is designed so that it can scale with more servers. Using this crawler, the authors have collected over one million applications, which they used to make a study of the market. An archive of their applications can be found [48] in the Internet Archive [49]. APKMirror [50] is an Android applications archive, with one of their primary purposes being to "provide an archive of popular applications along with changelogs and descriptions". It contains multiple versions of the applications it hosts, but only free of cost ones can be found. Users can download and install applications from the site, or use their installer on the phone. APKPure [51], APKLinker [52] and APK4Fun [53] are also archives that provide different versions of the cost free applications they hosts. As well as in markets, archive can hosts malware [54]. Some application clients for these archives had malware found in them [55].

[48]: jakej (2022), *Android Apps*

[49]: The Internet Archive (2022), *The Internet Archive*

[50]: Illogical Robot LLC (2022), *APKMirror*

[51]: APKPure, Inc. (2022), *APKPure*

[52]: APKLinker (2022), *APKLinker*

[53]: APK4Fun (2022), *APK4Fun*

[54]: Golovin (2021), *Infected Android app store*

[55]: Golovin et al. (2021), *Malicious code in APKPure app*

Discussion

Market archives are more desirable to use for researchers because they maintain applications available for longer than markets, allowing better reproducibility of experiments. Another advantage is that we can recover missing applications from markets if the hash value is known. These market archives also provide certain metadata about the applications, such as the type of application, user reviews, and different versions of the same program. Note that, in a study aimed towards one or a set of markets, market archives cannot replace those. They can, however, be used to obtain applications previously in markets if these applications are unavailable. Despite all these advantages, researchers performing an experiment on one market archive, would obtain slightly different results from another archive or even a normal market. Particularly, there are no paid applications, and there are lot of clones of the same application, as they keep different versions of it. For these reason, using markets archives is not suitable for performing studies. This is why researchers use specific research datasets, which we explain in the next section.

2.1.3 Specific research datasets

These datasets are collections of applications made by researchers in Android malware. They tend to contain primarily malicious applications, as well as benign ones and metadata associated to the applications from results of analyses of these programs. This is the main difference between these datasets and the previous ones: these ones can contain malware deliberately, because it was used for an experiment or it was part of the design of the dataset to contain malware. These specific datasets are then shared among researchers to conduct experiments in Android malware, as malware is not easily found.

When we reviewed the literature datasets, we decided to divide these datasets in three main categories based on their usage goal:

- Malware archives: a collection of applications where we can find malware.
- Experimental datasets: collection of applications used in an experiment and made available by researchers conducting this study.
- Novelty datasets: collection fo applications made by researchers with particular properties to fulfill a specific need.

Malware archive datasets These datasets are special archives maintained by researchers to primarily gather and share malware and eventually goodware (benign applications). They are different from market archives seen in the previous section because these datasets tend to require permission to access them, where market archives are more open to end users. We can further subdivide this category into collection archives and collaborative repositories.

Collection archives are repositories whose main aim is to gather applications and sharing them to the research community. Among them, we can cite VirusTotal [56], theZoo [57], VirusShare [58] that gather all types

[56]: Hispasec Sistemas (2022), *Virus Total*

[57]: Yuval Nativ @ytisf (2015), *theZoo - A Live Malware Repository web site*

[58]: VirusShare (2011), *VirusShare.com - Because Sharing is Caring*

of malicious files. Other collection archives focus on Android malware, such as Andrubis [59] and Contagio [60].

[59]: Lindorfer et al. (2014), 'ANDRUBIS – 1,000,000 Apps Later: A View on Current Android Malware Behaviors'

[60]: Mila (2011), *Contagio Mobile*

The biggest of all Android malware and goodware collection archives is AndroZoo [61]. This repository contains more than 19 millions Android applications from Google Play and multiple other sources. Because goodware and malware are collected from these sources, AndroZoo provides additional metadata in order to distinguish them. These metadata is calculated using VirusTotal, where the number of detections by antiviruses and the date of analysis are provided. Even if these metadata cannot be considered a totally reliable ground truth, as antivirus detection may change over time, it has become the reference for experiments in Android malware analysis because of its size, its easy to use API for requesting applications and its ever evolving dataset, continuing to add more applications since 2016. Even if VirusTotal's detection accuracy is an independent problem that can be studied, we consider in is thesis that AndroZoo's metadata is reliable. Thus, in our contributions of this thesis, we use applications and metadata from AndroZoo.

[61]: Allix et al. (2016), 'AndroZoo: Collecting Millions of Android Apps for the Research Community'

Collaborative repositories have the aim to provide not only malware samples, but also a platform for researchers to share their samples with other members of the community. Such is the case of sites like MalShare [62], Koodous [63], Malware Bazaar [64], and Malpedia [65].

[62]: Silas Cutler (2013), *MalShare*

[63]: Koodous Team (2015), *Koodous Project*

[64]: abuse.ch (2022), *Malware Bazaar*

[65]: Plohmann et al. (2018), 'Malpedia: A Collaborative Effort to Inventorize the Malware Landscape'

Compared to markets, these collections are more useful for researchers, as well as being able to reference the samples to these repositories instead of hosting their experimental datasets. There is a trade-off with the ground truth of *being malware* with these repositories. On one side, smaller repositories have less malware samples but they are more accurate in the description of their malware samples, like in the case of Malpedia [65]. On the other hand, bigger repositories have more malware but their malware samples are less accurately described, like in the case of VirusShare [58] which relies in VirusTotal detections.

[65]: Plohmann et al. (2018), 'Malpedia: A Collaborative Effort to Inventorize the Malware Landscape'

[58]: VirusShare (2011), *VirusShare.com - Because Sharing is Caring*

Experimental datasets These datasets where created for a particular experiment, and then shared to the research community. Particularly, researchers are interested in malware detection and family classification using machine learning techniques. Because there is theoretically a experimental dataset per experiment, thousands of datasets must be considered to make this section exhaustive. However, not all experiments share their experimental datasets, not even a list of hashes. We decided to take into account the datasets that are cited by others studies, because they are more discoverable than other ones, thus more prone to be used for other experiments.

The most significant of these experimental datasets is Drebin created by Arp *et al.* [5]. Their work focused on malware detection using machine learning, and the authors share the feature dataset in addition to the malware samples used in the study. Another dataset is DroidKin [66], used to evaluate a method for detecting similarities between Android applications. APIGraph [67], which is a framework to improve some of the state-of-the-art Android malware detection, also provides its experimental dataset. We made a contribution by releasing the GM19 [68] experimental dataset, used as part of an experiment on obtaining the part of the application's code that has a malicious behavior.

[5]: Arp et al. (2014), 'DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket'

[66]: Gonzalez et al. (2015), 'DroidKin: Lightweight Detection of Android Apps Similarity'

[67]: Zhang et al. (2020), 'Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware'

[68]: Viet Triem Tong et al. (2019), 'Isolating malicious code in Android malware in the wild'

The Canadian Institute for Cybersecurity (CIC) has released several datasets that accompany their experiments. These are, in chronological order: the Android Botnet 2015 dataset [69] released as part of a study on Android botnets families and behaviors, the CIC-AAGM2017 [70] "Android Adware and General Malware Dataset" used on an experiment of a new method that utilizes network traffic features to detect and classify malware, the CIC-InvesAndMal2019 [71] dataset that is a second part of the previous one from 2017 [72] with more features like permissions, intents, API calls and log files from dynamic analysis, and the CCCS-CIC-AndMal-2020 [73] [74] dataset used on a new Android malware classification method called DIDroid.

[69]: Abdul Kadir et al. (2015), 'Android Botnets: What URLs are Telling Us'

[70]: Lashkari et al. (2017), 'Towards a Network-Based Framework for Android Malware Detection and Characterization'

[71]: Taheri et al. (2019), 'Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls'

[73]: Rahali et al. (2020), 'DIDroid: Android Malware Classification and Characterization Using Deep Image Learning'

The Hacking and Countermeasure Research Lab (HCRL) in Korea has also released various datasets alongside their experiments. In chronological order: the AndroTracker [75] dataset used for malware detection using certificate features like serial numbers, Andro-AutoPsy [76] used in an Android malware detection method that uses a matching algorithm based on static characteristics, Andro-Dumpsys [77] used to evaluate a new Android malware detection system of the same name, SAPIMMDS [78] used on and Android malware detection method using suspicious API call patterns, Andro-Profiler [79] used hybrid client/server analysis method for Android malware behavior, and Andro-Simnet [80] used in an Andriod malware family classification technique.

[75]: Kang et al. (2015), 'Detecting and Classifying Android Malware Using Static Analysis along with Creator Information'

[76]: Jang et al. (2015), 'Andro-AutoPsy: Anti-malware system based on similarity matching of malware and malware creator-centric information'

[77]: Jang et al. (2016), 'Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information'

[78]: Jang et al. (2016), 'Function-Oriented Mobile Malware Analysis as First Aid'

[79]: Jang et al. (2016), 'Detecting and classifying method based on similarity matching of Android malware behavior with profile'

[80]: Kim et al. (2018), 'Andro-Simnet: Android Malware Family Classification using Social Network Analysis'

These datasets provide applications than can be used to reproduce their experiments, or as sources for researchers to take samples from and make their own. However, the use of experimental datasets come with some problems. One of the problems is that the calculation of the ground truth relies on VirusTotal. It have been shown that for Android, there is discrepancy between the expected detection rate of Android malware and the one from VirusTotal. Another problem is that static datasets tend to be useful for two to three years before they start getting old. This is the case for Drebin, which is widely used in the literature as a source of malware, despite containing samples from 2012. This influence the results of experiments using Drebin as we show in section 2.2.

Novelty datasets The main motivation for the creation of these datasets is to provide one that has desirable properties not found in other research datasets. While experimental datasets may contain the metadata used in their experiments (such as the feature vectors for machine learning), in addition to the binaries of the applications, these novelty datasets are constructed explicitly to provide one with applications and metadata that the authors deemed rare or scarce.

For example, Genome [81] is a collection of manually analyzed Android malware, with a study about what each family does. Similarly, the Kharon dataset [82] provides reversed malware samples to the research community, explaining what the sample does, which part of the code is the trigger of the malicious behavior and a visual replay of the program execution. The Android Malware Dataset (AMD) [6] dataset provides a more "up-to-date" dataset than previous efforts, notably Genome [81] which was already aging and no longer representing the current state of Android malware. CICAndMal2017 [72] is another initiative to create a reference malware dataset, arguing that a good reference dataset must

[81]: Zhou et al. (2012), 'Dissecting Android Malware: Characterization and Evolution'

[82]: Kiss et al. (2016), 'Kharon Dataset: Android Malware under a Microscope'

[6]: Wei et al. (2017), 'Deep Ground Truth Analysis of Current Android Malware'

[72]: Lashkari et al. (2018), 'Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification'

contain multiple different categories and families, and also have a large number of samples. UpDroid [83] is a dataset whose main goal is to provide up-to-date samples of malware that employ a specific type of evasion technique called "update attack". RmvDroid [84] is a dataset created due to the lack of datasets with metadata information available and recent samples. The AndroZooOpen [85] dataset is a collection of mainly open-source applications from Github, as well as F-Droid, AndroZoo and Google Play, aiming to provide a "representative picture of Github-hosted Android apps". AndroCT [86] is a recent Android dataset whose goal is to provide run-time data for studies using dynamic and hybrid analysis techniques of old and new applications. KronoDroid [87] provides a large quantity of samples with several types of metadata from static and dynamic covering a large time range. MalRadar [88] is the most recent Android malware dataset, whose goal is provide one using malware reports, such as the ones provided by TrendMicro, ESET, Kaspersky and other, as the their ground truth, instead of using VirusTotal.

[83]: Aktas et al. (2018), 'UpDroid: Updated Android Malware and Its Familial Classification'

[84]: Wang et al. (2019), 'RmvDroid: Towards A Reliable Android Malware Dataset with App Metadata'

[85]: Liu et al. (2020), 'AndroZooOpen: Collecting Large-Scale Open Source Android Apps for the Research Community'

[86]: Li et al. (2021), 'AndroCT: Ten Years of App Call Traces in Android'

[87]: Guerra-Manzanares et al. (2021), 'KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization'

[88]: Wang et al. (2022), 'MalRadar: Demystifying Android Malware in the New Era'

Discussion

Although there is a wide variety of datasets, very different from one another in terms of size and content, researchers construct their datasets with different properties (number of malware, number of goodware, date of the samples, etc) each time they perform a new experiment. On the contrary, all these malware archives, experimental and novelty datasets allow researches to either reproduce previous experiments, or take samples for their own experiment from a source that is recognized by other researchers in the field and that others can also take samples from. Using existing datasets is useful for the reproducibility of experiments, as these ones can be compared using the same data [11].

The variety of dataset results to be a problem too. Since there are too many datasets to choose from, researchers tend to select datasets like Drebin [5] and AMD [6], as they are widely used in many studies. It should be noted these datasets, Drebin and AMD, contain old samples, which may not resemble recent malware. This difference can be more properly appreciated using visual representation of the characteristics of their samples. A method to visually compare datasets is proposed in Chapter 3, where we compare these older malware dataset with newer ones.

[11]: Pendlebury et al. (2019), 'TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time'

2.2 Irregularities and biases in experimental datasets

Through the years, researchers created their experimental datasets and gathered applications by crawling the various markets, or by requesting access to other source or archive datasets. The choice of which Android applications should be included or excluded from an experimental dataset has an important impact on the measured performance of algorithms. An experiment can give bad results because of the nature of its input dataset. Conversely, it can artificially give good results with a specific

dataset, but gives bad results with any other dataset. Both scenarios are undesirable effects that researchers would want to avoid.

In this section, we will study the impact of irregularities in experimental datasets and their influence in the experiments results. First, in Section 2.2.1 we categorize the different experimental scenarios in Android malware research. Then in Section 2.2.2, we describe the possible irregularities and biases that experiments may suffer due to its experimental dataset. Finally, in Section 2.2.3, we make a review of the literature and see that these irregularities still exist in experimental datasets.

2.2.1 Experimental scenarios

We have identified three types of scenarios typically experienced by security researchers. Each of these scenarios has various issues and therefore requires the use of datasets with different expected properties.

Markets profiling Researchers may want to estimate a statistical characteristic of a market [43, 61, 89], *e.g.* "What proportion of Google Play applications have an API version lower than some threshold?". For this case, one needs to obtain a dataset that is *representative* of the market, where results of analysis on this dataset can be extrapolated to the whole market. Chapter 4 gives more details on how such a representative dataset should be extracted.

[43]: Wang et al. (2018), 'Beyond Google Play: A Large-Scale Comparative Study of Chinese Android App Markets'
[61]: Allix et al. (2016), 'AndroZoo: Collecting Millions of Android Apps for the Research Community'
[89]: Dong et al. (2018), 'Understanding Android Obfuscation Techniques: A Large-Scale Investigation in the Wild'

Algorithm evaluation Some experiments aim to compute a characteristic of interest from an application. Examples of interesting characteristics include detecting malicious behavior in an application [90], whether the sample is repackaged [91], identifying its malware family [92], detecting similar applications [93], whether it leaks personal information to the network [94]. To answer these questions, researchers implement algorithms using various analysis techniques, and run them on a dataset to estimate their precision. This evaluation dataset should be representative (formalized later in Chapter 4) and *labeled*, *i.e.* the characteristic of interest for each application should be known beforehand.

[90]: Grace et al. (2012), 'RiskRanker: Scalable and Accurate Zero-Day Android Malware Detection'
[91]: Gonzalez et al. (2015), 'Exploring Reverse Engineering Symptoms in Android Apps'
[92]: Battista et al. (2016), 'Identification of Android Malware Families with Model Checking'
[93]: Crussell et al. (2013), 'AnDarwin: Scalable Detection of Semantically Similar Android Applications'
[94]: Li et al. (2015), 'IccTA: Detecting Inter-Component Privacy Leaks in Android Apps'

Machine Learning In supervised learning methods, researchers learn a model that classifies inputs into a set of classes. Two datasets are used in this setting: a training dataset and a test dataset. The training set is used during the *learning* phase, and the test set is used to evaluate the learned model on new data. Both datasets should be labeled. The training set needs to be selected carefully to avoid biases, as discussed in more detail in Section 2.2.2. The test set should be representative, as for the *Algorithm evaluation* scenario.

For any of the three kinds of experiments above, building an appropriate dataset is a non-trivial task. Before presenting our contribution for this task in Chapter 4, we have investigated the literature to see how researchers have built their datasets. Because the number of papers for the three experimental scenarios, market profiling, algorithm evaluation and machine learning is quite important, we have restricted our review to papers dedicated to machine learning scenarios. From these reviews,

we extracted irregularities that we classified in the categories presented in the next section.

2.2.2 Categories of irregularities

In this section we present the main categories of irregularities found in datasets that cause biases in experimental results. A *bias* is defined as a systematic error that influences the results of an experiment [95]. This error may be minimal, so that the experiment yields results very close to the real value. We could identify three main irregularities in input datasets: class imbalance, presence of clones and time incoherence.

[95]: Dodge et al. (2006), *The Oxford Dictionary of Statistical Terms*

Class imbalance In supervised learning, a *class* is a property of an element that a machine learning algorithm uses to group different elements of the training set. It is also called *category* or *label*. In the case of malware detection, the possible classes of an application can be *is a malware* or *is not a malware*, and for malware family classification, each class correspond to the name of a family. Class imbalance occurs when a dataset contains significantly more units of one class than of others [96]. An overrepresented class may push a learning algorithm towards preferring this class. For malware detection, if there are more samples of goodware, this has the effect of decreasing the recall but improving the precision [11].

[96]: Weiss et al. (2001), *The effect of class distribution on classifier learning: an empirical study*

[11]: Pendlebury et al. (2019), 'TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time'

This problem has been addressed in the machine learning community [97]: the overrepresented class is undersampled, *i.e.*, some elements are removed while the underrepresented class is oversampled. The undersampling can be random or rely on a more sophisticated method, such as removing elements in dense clusters [98]. The most popular oversampling technique is SMOTE [99] that creates new instances from the interpolation of close samples. This technique would be interesting for machine learning experiments because it solves the class imbalance problem. Nevertheless, the new interpolated characteristics vector used in the experiments would not have any corresponding application. As a consequence, the application dataset cannot be provided by researchers, if is a problem for reproducibility of malware analysis experiments. Besides, class balancing is mostly restricted to binary classes [97], while we are interested in balancing more diverse classes.

[97]: Susan et al. (2020), 'The balancing trick: Optimized sampling of imbalanced datasets—A brief survey of the recent State of the Art'

[98]: Tomek (1976), 'Two modifications of CNN'

[99]: Chawla et al. (2002), 'SMOTE: synthetic minority over-sampling technique'

In the case of a dataset evaluating a detection algorithm, class imbalance does not pose a threat to the validity of the results: since a representative dataset is needed for evaluation in this scenario, it does not need to be balanced, as the proportion of malware and goodware may be disproportionate but accurate to the reality of the set of applications the experiment is interested in.

Presence of clones Another irregularity is the presence of clones, where the abundance of several very similar apps in the training set prevents the algorithm from learning unique units [100]. Because experiments do not usually share the list of used applications, it is not possible to check the presence of clones in their datasets. Pan *et al.* [101] show a method

[100]: Pfaff et al. (2019), 'Code Clones Considered Harmful? Quantifying and Exploiting the Effects of Code Clones in Static Malware Classifiers for JavaScript'

[101]: Pan et al. (2005), 'Finding Representative Set from Massive Data'

to reduce redundancy in a dataset by extracting *representatives* of cloned units using mutual information and relative entropy.

Clones in a evaluation dataset to evaluate detection algorithm are admitted, since the samples are obtained in the proper way (as shown in Chapter 4), even if clones appear, they are representative of the applications in the target set of applications of interested for the experiment.

Time incoherence This irregularity concerns the dates of the samples in the test and learning sets used in machine learning algorithms. It was shown that, when evaluating the accuracy of state-of-the-art algorithms, using a test set made of applications newer than those in the training set, the performance of these algorithms decreases [11, 102].

[11]: Pendlebury et al. (2019), 'TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time'
[102]: Fu et al. (2019), 'On the Deterioration of Learning-Based Malware Detectors for Android'

Pendlebury *et al.* [11] formalized the idea of time incoherence induced by time and features in malware classification: experimental evidence shows a decay in performance, when experiments are done with datasets that have other time distribution, rather than a 10-fold cross-validation as most of the experiments do. This induce what they call temporal bias. Besides this, in the same study, the authors introduce a "spatial bias" which, according to them [11], is a bias caused by the distribution of the class training and test sets. To remedy these biases, they propose three constraints that experiments should follow in order to avoid temporal and spatial biases:

- C1 "All the objects in the training must be strictly temporally precedent to the testing ones" [11]: the samples in the training set must be older than those in the test set.
- C2 "In every testing slot of size Δ, all test objects must be from the same time window" [11]: malware and goodware must come from the same time intervals.
- C3 "To have a realistic evaluation, the average percentage of malware in the testing must be as close as possible to the percentage of malware in the wild" [11]: while the distribution of malware and goodware in the test set must approach as the one in-the-wild, it is possible to tune this distribution in the training set (to counter class imbalance).

Even if we apply these restrictions to build an experimental dataset, it is not guarantied that applications will behave the same way in the future. In [103, 104] and [105], Cai states that the main reason that existing Android malware detection techniques do not sustain well lies in their failure to account for the evolutionary dynamics of the Android ecosystem. Android malware classifiers may not be sustainable – they would need to be retrained constantly for later use, or their performance could downgrade over time. This is further explained on a recent study by Guerra-Manzanares *et al.* [106]. A supplementary problem is one in [67], where Zhang *et al.* observes that malware samples often implement the same functionalities over time but change their implementation because of the evolution of APIs or changes of third-party libraries. These changes allow such malware to avoid detection by older classifiers.

[103]: Cai et al. (2018), 'Towards Sustainable Android Malware Detection'
[104]: Cai (2020), 'Assessing and Improving Malware Detection Sustainability through App Evolution Studies'
[105]: Cai (2020), 'Embracing Mobile App Evolution via Continuous Ecosystem Mining and Characterization'
[106]: Guerra-Manzanares et al. (2022), 'Concept drift and cross-device behavior: Challenges and implications for effective android malware detection'
[67]: Zhang et al. (2020), 'Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware'

Solutions are proposed to mitigate this phenomena. The first one consist of re-training a model, where a new test set is used to create a model using newer data. Regular re-training can solve this problem, however,

it is not always a solution, as new samples are not always available as quickly as needed or in sufficient diversity. One way to limit this problem is to choose more semantic than syntactic features, as stated by Zhang *et al.* in [67]. Finally, in [107], Xu *et al.* propose to enhance malware detection over time by making necessary updates to their detection models with evolving feature sets. For that purpose they maintain a pool of different detection models initialized with a set of labeled applications using various online learning algorithms.

[67]: Zhang et al. (2020), 'Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware'

[107]: Xu et al. (2019), 'DroidEvolver: Self-Evolving Android Malware Detection System'

Discussion

These three irregularities can be threats to the validity of experiments if present in the input dataset of an experiment. Researchers should take into account these irregularities, so that the results of their experiments give an appropriate result. In the case of an algorithm evaluation, we see that none have a significant impact on this type of experiment we are interested in. Nevertheless, a representative dataset must be used in order to ensure pertinent results when evaluating a detection algorithm. We show how to obtain a representative dataset in Chapter 4. Furthermore, we use the constraints C1, C2 and C3 proposed by Pendlebury *et al.* [11] in our experiments in Chapter 5. For machine learning, these irregularities are concerning and should be addressed.

2.2.3 Results of evaluation of the literature

We saw that researchers have plenty of sources to take applications and perform their experiments. Taking samples from heterogeneous sources may introduce irregularities into the built dataset. To confirm this hypothesis, we made a review of different experiments to see how they constructed their datasets.

We reviewed 28 articles published between 2016 and 2019 to evaluate if these irregularities may have an impact on past results. Table 2.1 presents the details of this review.

We extracted the following information from each article:

- Column "Paper" gives the name of the tool or the authors' names.
- Next, column "Goodware source" contains the datasets used in the experiment. Sometimes, several sets are used in a paper: we note these subsets $\mathbf{D}_i$ and we give their size in other columns accordingly.
- Next, columns "GS year" and "Nb Goodware" are the year range and number of goodware according to the article, if available.
- The next three colmuns give the same information for the used malware.
- Next, columns "Training set" and "Test set" correspond to the number of applications used in the respective sets.
- The notation "(M/G)" reports the Malware/Goodware for each experiment. The use of "k-fold cross validation" will be specified in these columns. Any other special cases are reported in these two columns.

Table 2.1: Details on the dataset usage in detection experiments. GP = Google Play, VT = VirusTotal, GM = GM, DB = Drebin, AZ = AndroZoo, CT = Contagio, VS = VirusShare, PD = PlayDrone, MSL = MobiSec Lab, TZ = theZoo, MS = MalShare, N/A = Not applicable

Paper	Goodware source	Goodware source year	Goodware size	Malware source	Malware source year	Malware Size	Training set (M/G)	Test set (M/G)	Class Imbalance	Time Incoherence
MADAME [108]				GM CT VS	2010-2011 2016 2012-2015	1 242 1 923	2,784/0 (Total)		X	
DroidDetector [109]	GP	2016	20 000	CT GM	2016 2010-2011	500 1 260	880/800	880/880		X
ICCDetector [110]	GP	2014	12 026	DB	2010-2012	5 264	15 561 (mixed)	1 203/526	X	X
Androdialysis [111]	GP	2016	1 846	DB	2010-2012	5 560	10-fold cross-validation	5 560/1 846	X	X
MaMaDroid [112]	PD GP	2013-2014 2016	8 447 2 843	DB VS " " "	2010-2012 2013 2014 2015 2016	5 560 6 228 15 417 5 314 2 974	10-fold cross-validation	DB/PD 2013/PD 2014/PD 2014/GP 2015/GP 2016/GP	X	
StormDroid [113]	GP	2015	4 350	MSL GM CT	2015 2010-2011 2012-2015	2 000 1 260 360	$\frac{(900+500+100)}{1500}$	$\frac{(600+300+100)}{1000}$		X
PIndroid [7]	GP AppBrain F-Droid Getjar Aptoid Mobango	up to 2016	445	CT DB GM VT TZ[57] MS[62] VS	2012-2015 2010-2012 2010-2011 ? 2016 ? 2016	60 100 1 000 70 70 25 25	80% (from 1300/445 total)	20% (from 1300/445 total)	X	X
Vinod et al. [9]	GP ($\mathbf{D}_1$, $\mathbf{D}_2$, $\mathbf{D}_3$, $\mathbf{D}_4$, $\mathbf{D}_5$) Baidu ($\mathbf{D}_1$, $\mathbf{D}_2$) Koodous($\mathbf{D}_1$, $\mathbf{D}_2$)(1,514) 1Mobile ($\mathbf{D}_1$, $\mathbf{D}_2$) 9apps ($\mathbf{D}_1$, $\mathbf{D}_2$)	2016	3 130	DB ($\mathbf{D}_1$, $\mathbf{D}_2$) Koodous ($\mathbf{D}_2$) [114] ($\mathbf{D}_4$) GM ($\mathbf{D}_5$)	2010-2012 ? 2011-2015 2010-2011	 1 514 1 000 1 206	10-fold cross-validation	$\mathbf{D}_1$ 2 520/3 130 $\mathbf{D}_2$ 1 514/3 130 $\mathbf{D}_3$ 2 474/2 474 $\mathbf{D}_4$ 1 000/14 000 $\mathbf{D}_5$ 1 206/1 206	X	X
MalDozer [8]	GP	2016	38 000	GM DB *Own* "Merged"	2010-2011 2010-2012	1 000 5 500 20 000 33 000	Each malware dataset individually/37 627		X	X
IntelliAV [115]	VT "	2011-2016 2017	10 058 2 898	VT "	2011-2016 2017	9 664 2 311	9 664/10 058 2 311/2 898		X	
Martín et al. [116]	GP	2015	~ 49 000	GP	2015	69 000	10-fold cross validation with 9 subsets of 50 000 samples, varying malware concentration from 2%, 25%, 50% of the whole subset, and the number of antivirus alerts: 1-AV, 2-AV, 4-AV (4-AV, 50% subset only contains 36 000)		X	
HinDroid [117]	Comodo Cloud Security Center	2017	920 ($\mathbf{D}_1$ Tr) 198 ($\mathbf{D}_1$ Te) 15 000 ($\mathbf{D}_2$)	Comodo Cloud Security Center	2017	914 ($\mathbf{D}_1$ Tr) 302 ($\mathbf{D}_1$ Te) 15 000 ($\mathbf{D}_2$)	914/920 ($\mathbf{D}_1$) 15 000/15 000 ($\mathbf{D}_2$)	500 ($\mathbf{D}_1$) 10-fold CV ($\mathbf{D}_2$)		
RevealDroid [118]	GP (AZ)	2016	24 600	VS DB	2016 2010-2012	22 592 5 538	10-fold CV Training-Test with date separation (Training date < Test date)		X	X
Demontis et al. [119]	DB	2010-2012	121 329	DB Mod DB and CT	2010-2012 2010-2012, 2016	5 615 10 500	10 runs, where 30 000 for training, and the rest for test		X	*Too Old*
McLaughlin et al. [120]	GM (GP) McAfee Labs "	2010-2011 2016 ? 2016 ?	863 3 627 9 268	GM McAfee Labs "	2010-2011 2016 ? 2016 ?	1 260 2 475 9 902	10-fold CV, with same GW/MW ratio In training and test			X
Alzaylaee et al. [121]	Intel Security (McAfee Labs)	2016 ?	1 222	GM	2010-2011	1 222	2/3 of GW/MW	1/3 of GW/MW		X
Melis et al. [122]	DB	2010-2012	121 329	DB	2010-2012	5 615	5 runs, where 60 000 units are randomly selected for training, And the rest for testing		X	*Too Old*
MKLDroid [123]	GP1 GP2 AndroidDrawer (AD) AnZhi (AZ) AppsApk (AA) F-Droid (FD) SlideMe (SM)	2012-2014 2012-2014 2013-2016 2013-2016 2013-2016 2013-2016 2013-2016	5 000 10 000 2 399 3 027 2 481 1 007 5 770	DR VS MYST	2010-2012 2013-2014 2015	5 560 24 317 3 000	CE1: 70% of DR + GP1 CE2: 70% of VS + GP2 CE3: DR + GP1 WEx: DR + VS + GP1 + GP2 MCLEx: ⅔ of MYST + GP1	CE1: 30% of DR + GP1 CE2: 30% of VS + GP2 CE3: VS + GP2 WEx: AD + AZ + AA + FD + SM MCLEx: ⅓ of MYST + GP1	X	X
Li et al. [124]	apkpure 360 HKUST	2019	16 753	DB AMD	2010-2011 2010-2016	5 560 16 753	5-fold CV		X	X
SMART [125]	GP	2015	5 600	DB	2010-2012	5 560	10-fold CV			X
Xiao et al. [126]	GP	2016 ?	3 536	DB	2010-2013	3 567	10-fold CV			X
DroidCat [127]	GP (AZ) " " "	2016-2017 2014-2015 2012-2013 2009-2011	5 346 6 545 5 035 439	VS, AZ VS, AZ VS, AZ, GM, DB VS, AZ, GM, DB	2016-2017 2014-2015 2012-2013 2009-2011	3 450 3 190 9 084 1 254	70% of older apps	30% of newer apps	X	
Rana et al. [128]	DB	2010-2012	5 560	DB	2010-2012	5 560	80% training and 20% testing			*Too Old*
Ma et al. [129]	AZ	?	10 010	AMD	2010-2016	10 683	Everything divided in 10 parts, Then 10-CV for each part		?	?
Huang et al. [130]	GP	2018 ?	3 312	VS	?	3 312	10-fold CV			*N/A*
Liu et al. [131]	DB	2010-2012	123 453	DB	2010-2012	5 560			X	*Too Old*
HG-Learning [132]	Tencent Security Lab (Tr) " (Te)	2018 ? 2018 ?	83 784 13 313	Tencent Security Lab (Tr) " (Te)	2018 ? 2018 ?	106 912 4 433	83 784/106 912	13 313/4 433	X	
BRIDEMAID [133]	GP	2016	9 804	DB GM CT	2010-2012 2010-2011 2016	2 794	2 974 applications tested		X	X

- Finally, columns "Class imbalance" and "Time Incoherence" indicate whether the datasets used in the experiments suffer from these irregularities, introduced in Section 2.2.

Out of the 28 articles, we found that 17 suffer from class imbalance: the goodware/malware ratio is often too high. Time incoherence happens in 15 articles: experimental datasets often contain recent applications from Google Play and older malware from Drebin or Genome. Additionally, 4 articles have a too old dataset age, compared to the publication date. Because experiments do not usually share the list of used applications, it is not possible to verify the presence of clones in their datasets. However, we can notice that 3 experiments use Drebin and Genome at the same time, and Genome is a subset of Drebin, thus having the algorithms trained and tested with several instances of the same APKs.

We notice that the test dataset is constructed from the input dataset in various ways. One common way is to perform a k-fold cross validation, where the input dataset is split into several subsets. If the input dataset is not a representative dataset of the studied set of applications, the subset of the k-fold split, and therefore the test set, will not be representative either. This invalidates results aiming to the studied set of applications. But, since we want to avoid class imbalance in the training set, the class proportion would probably be not the same as the studied set of applications, so avoiding class imbalance and being representative are incompatible properties for a training set. On the other hand, if the input dataset is a representative dataset of a market, the classes/labels *is a malware/is not a malware* are most of the time unavailable at the time of creating the representative dataset. This problem of crafting a representative and labeled is test dataset for machine learning, which also works for evaluating an algorithm, is tackled in Chapter 5.

Discussion

We saw the types of scenarios researchers in security encounter, namely market profiling, algorithm evaluation and machine learning. Specially for this last one, we saw the different irregularities that experimental datasets introduce to experiments, with proposed methods to mitigate these irregularities. Nevertheless, there is one bias that none of these experiments address: in the case of an experiment claiming that it works on a certain set of applications (*e.g.* a market), its test dataset must be either the whole set of applications or a sample of this set such that the results can be extrapolated to the whole set [134]. The previously cited works do not solve the problem of researchers having to evaluate their algorithm on an up-to-date dataset. Of course, this problem is different than adapting a classifier to an evolving market – even if this problem is of independent interest. Researcher would primarily focus on getting a dataset that mimics the whole set of applications that they targets.

Additionally, the ground truth of the dataset should be known, for example with a malware/goodware label if the study concerns a detection algorithm. These labels can be missing in the source from which applications are picked. As a consequence, we propose in the following to define the notion of representativity of a dataset against

[134]: Arp et al. (2022), 'Dos and Don'ts of Machine Learning in Computer Security'

the population it represents. A representative dataset has to be used to validate an experimental results at the time of the experiment. The representativity property is linked to a population at a period of time. Pendlebury *et al.* [11] has recommendations for this irregularity related to time, namely time incoherence. A dataset that is representative at a date will not necessarily be so in the future because the population may change significantly. Such a representative dataset will help the researcher to assess results for real-world scenarios [10].

[10]: Allix et al. (2015), 'Are Your Training Datasets Yet Relevant?'

2.3 Conclusion

In this chapter we explored the different sources of applications researchers use to create their experimental datasets. Markets are the prime source of recent, up-to-date applications. Although they are not free of malware, they provide a view of what to expect in the wild (at least for a specific market). In order to allow experiments be reproduced, a handful of researchers have provided their experimental datasets, which have also contributed to help other researchers obtain samples that would be otherwise not available. In addition to this, efforts to provide samples for researchers are still ongoing, with datasets like ViruShare, AndroZoo, Malpedia, and Malware Bazaar, aiding the reproducibility of experiments by hosting a large quantity of samples. We saw the differences between all these datasets in terms of size, dynamicity, availability, content, and experimental goal. Despite this, a more intuitive approach for comparing datasets could be performed with analysis data from their samples. This would allow to compare datasets based on their content, where we could find differences in the amount of applications with certain interesting properties. Such a comparison, and eventual exploration, of datasets can be done using visualization techniques. This can be done using our approach discussed in Chapter 3. We will see that old and new datasets are different in terms of certain properties, and that malware datasets alone are different from a population we fixed to study.

We saw the various experimental scenarios that researchers face, and the different irregularities that affect experimental results if these are present in the input dataset. While the landscape of research datasets seems to be booming, trying to provide ones with more and richer information, and with solutions to known irregularities, none of these studies seem to address the problem of representation, *i.e.* the usage of a dataset to represent another one. This representation could be useful to perform the following: instead of testing a machine learning model or an algorithm over a market, a "representative" dataset of a studied population (*e.g.* a market) could be used in place, allowing to obtain results using a sample that are very similar *as if* the test would have been performed to the population itself. A method for creating such a representative dataset is detailed in Chapter 4. Even so, samples from market do not have the properties available immediately to be used for testing. Such properties are mostly available in researches datasets that are biased, that is, they are not *representative* of the studied population. In order to use a dataset with samples from research datasets, it has to be *debiased* so that it resembles a representative dataset. We developed and test an algorithm to perform this procedure. We tested a debiased dataset produced by this technique

against other established datasets, and results show that our debiased dataset have a better performance overall than the other datasets used in the test in Chapter 5.

Contributions

Visualization of Android Malware Datasets

3

With millions of Android malware samples available from many different datasets, researchers have a large amount of applications to perform malware detection and classification, specially with the help of machine learning. Others take advantage of newer samples to perform reverse engineering and study in detail the malicious behavior of these applications. Choosing a proper dataset for an experiment is not a trivial process, and the fact that there are various, and widely different datasets to get applications from make this task harder. Researchers would appreciate to benefit from an intuitive way to differentiate or examine Android malware datasets, which would help them obtain an overview of the contents of these datasets, and help them choose one with the desired properties, *e.g.* the right amount of a certain characteristic or set of characteristics, for their experiments.

Visualization tools can help to obtain meaningful information from the large amount of data contained in a dataset. Representation of datasets allow their exploration and comparison, taking advantage of our visual system to easily spot differences between the represented values. This is a first step that helps researchers to understand the content of a dataset and infer the differences between datasets.

In this chapter we introduce a new visualization tool called DaViz. This tool is aimed at providing visual representation of datasets for exploring and comparing them. This would allow users to obtain useful information for certain tasks associated with datasets, like understanding their composition and the differences between them. In Section 3.1 we start by discussing the other visualization methods in the literature and understand the limitations of these tools for the tasks we want to perform. Then, in Section 3.2, we propose our solution to these problems, DaViz. Finally, we visualize datasets from the literature in Section 3.3, where we highlight how different datasets can be between each other and through time. This work has been published in the RESSI 2022 conference [12].

[12]: Concepción Miranda et al. (2022), 'DaViz: Visualization for Android Malware Datasets'

3.1 Motivation for Android malware datasets visualization

In order to take advantage of the abundance of applications available to study, researchers need to be able to navigate through large amount of applications in a dataset, often comparing datasets between them. In this section, we analyze the tasks involved when researchers examine datasets, then we inspect the literature in Android malware visualization for tools that aid in the achievement of these tasks.

3.1.1 Tasks and objectives for visualization

Let us imagine a situation where, in a research group or in an industry group, a dataset of unknown APKs is given to researchers working in malware analysis. A first step is to *examine the content of this dataset*, *e.g.* what types of applications can be found, what are their characteristics, whether there is any known malware/benign samples, etc. Once they recognize the content of a dataset, researchers can then *compare it against other datasets*. This comparison would allow them to discover similarities to other datasets and find correlations between them, or differences that would require a further study of the dataset in question.

From the previous remarks, we identify two main analysis tasks that researchers perform with datasets:

- an *examination* task that we call dataset exploration,
- a *comparison* task that we call dataset comparison.

We investigated how such tasks should be represented by following the methodology by Muzner [135]. Using her methodology, we took into consideration the volume and type of data to visualize, and the type of tasks we found. With this knowledge, we examined the types of visualization techniques (*e.g.* charts, graphs), alongside their interactions (*e.g.* selection, filter, zoom) to proposed a visualization method suited to the data, the tasks and the possible interactions using the tool. During this phase of investigation using the methodology of Muzner, we extracted the following properties about the analysis tasks:

[135]: Munzner (2015), *Visualization Analysis and Design*

Dataset exploration The first task, dataset exploration, has a "discovery" and "exploration" goals according to the different goals defined by Munzner [135]. By "discovering" datasets, users can gain new knowledge, verify hypothesis, or generate new sub datasets from the data at hand. For example, users can verify that more than half of the applications contain a specific set of characteristics. Users can also just "explore" the datasets when they do not know what they are looking for and where. This can be useful to users that look for outliers or unanticipated patterns in the data.

Dataset comparison The second task is associated to a "comparison" goal according to Munzner [135]: once users choose the datasets they are interested in, they can create charts to show differences between datasets. Multiple datasets can be compared at once, but a one by one comparison allows a more focused approach by concentrating on only two datasets at a time. Users can verify visually the difference of two datasets by comparing the proportion of applications with certain characteristics, or the presence of characteristics in either dataset (*e.g.* whether they characteristics A and B, A and not B, or B and not A).

3.1.2 Malware analysis visualization techniques

Now we examine the literature for visualization techniques for Android malware that fulfill both exploration and comparison tasks. The use of visualization for malware analysis is not new [136], whether the results are

[136]: Yoo (2004), 'Visualizing Windows Executable Viruses Using Self-Organizing Maps'

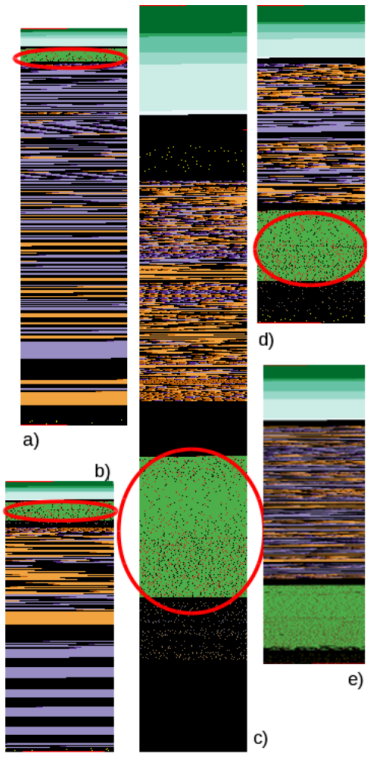

Figure 3.1: Five applications shown as images using the method by Jain *et al.* to recognize the usage of DexGuard in applications (a) and (b) (taken from [139])

used as input for an image recognition machine learning algorithm [137], or for aiding malware analysts understand what an application does and how [138]. We divide these methods into two groups: those for visualizing a single application and those for visualizing sets of applications.

[137]: Panas (2008), 'Signature Visualization of Software Binaries'

[138]: Quist et al. (2009), 'Visualizing compiled executables for malware analysis'

Visualization for single applications

In this part we present techniques that show a visual representation of a single application in general.

2D image of an APK's file structure Jain *et al.* [139] propose a method to visualize the program's `.dex` file into a color 2D image to help recognize patterns in the bytecode. Figure 3.1 shows five different applications (*a* to *d*) in form of figures using the method elaborated by Jain *et al.* [139]. We see a representation of APK files as a 2D image, where the sections of the application are shown with different colors: the header of the file is shown as a red line in the top left corner of the image, followed by the index list of strings, types, prototypes, fields, methods and classes in different shades of green, and then the data section. In this last section we can find the string list, where string constants, class names and method names can be found. In Figure 3.1 it is encircled in red. Applications produced by DexGuard tend to have their string list before classes code in the data section, as shown in Figure 3.1 (a, b), whereas it should normally be after the classes code (like in Figure 3.1 (c, d, e)). These types of patterns can be easily detected by experts when searching for applications obfuscated using DexGuard.

[139]: Jain et al. (2015), 'Enriching reverse engineering through visual exploration of Android binaries'

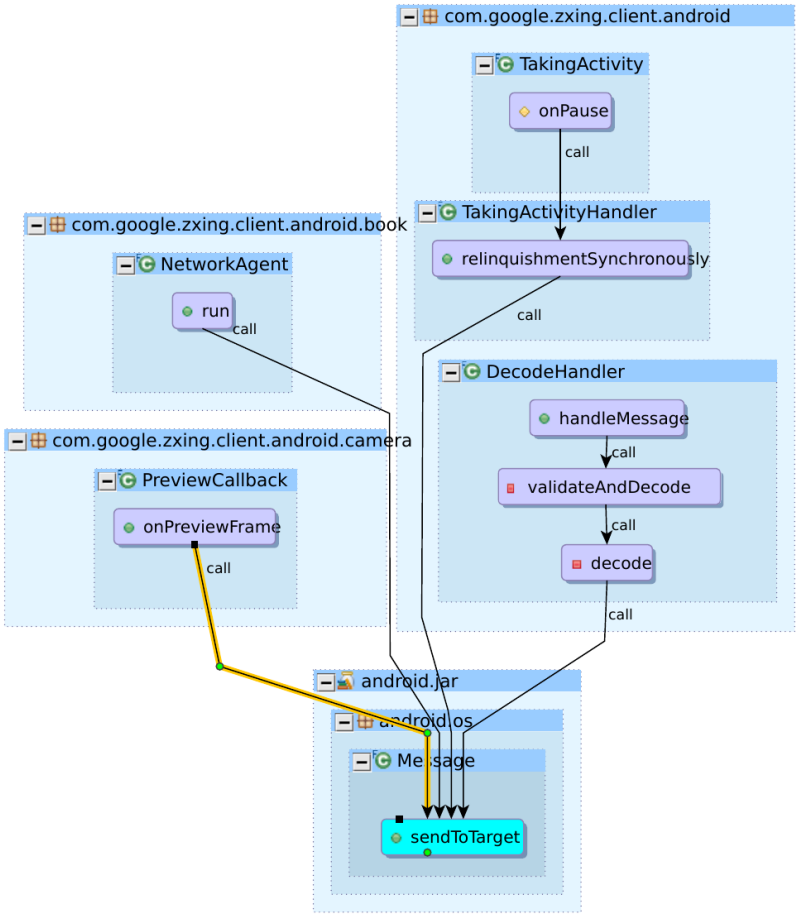

Figure 3.2: Reverse call graph used by Santhanam *et al.* (taken from [140])

[140]: Santhanam et al. (2017), 'Interactive Visualization Toolbox to Detect Sophisticated Android Malware'

Android Malicious Flow Visualization Toolbox Santhanam *et al.* [140] present a tool to visualize artifacts of an Android program (control flow graphs, system flow graphs, information flow graphs, or a combination of these) in order to help an analyst studying an unknown application decide if it violates confidentiality, integrity or availability. As an example of view proposed by the tool, we explain part of the case study of a confidentiality leak in an application. Part of the process involves the hypothesis that the application sends sensitive information in a message and sends it via the method *Message.sendToTarget*. To verify this, users can check the callers of this function to discover which other function is responsible for the possible information leak. Figure 3.2 is a "reverse call graph", where it shows all the functions that call `sendToTarget`. It the graph, we can see that this method is called by `Camera.PreviewCallback.onPreviewFrame`, an API overridden by the application. By inspecting the `PreviewCallback.onPreviewFrame`, users can see, in this example, that the preview image used by this method is sent using *Message.sendToTarget*, thus in part validating the hypothesis.

[141]: Lalande et al. (2020), 'GroDDViewer: Dynamic Dual View of Android Malware'

GroDDViewer: dynamic analysis replayer Lalande *et al.* [141] propose GroDDViewer for representing the dynamic execution of an application's system flow graph with a replay option, alongside its method control flow. Figure 3.3 shows an example of the tool's interface analyzing a ransomware, in this case a Simplelocker sample. On the left side, we see the "system flow graph", a graph showing the interactions between

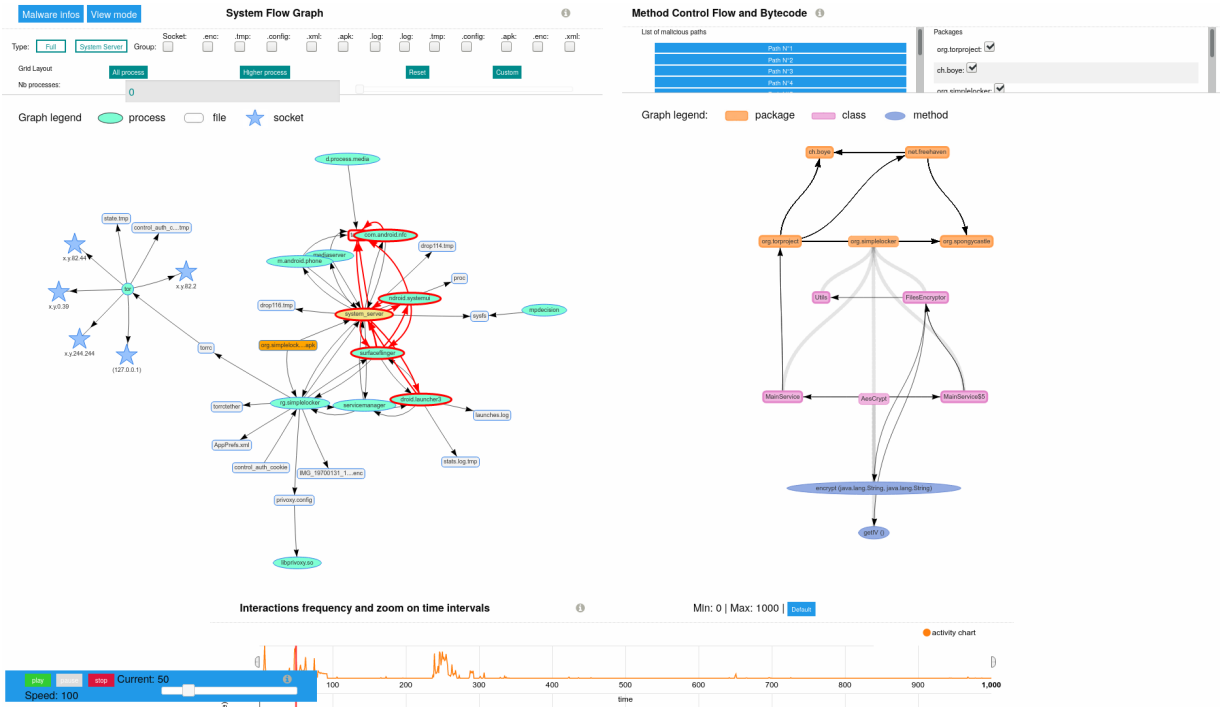

Figure 3.3: The GroDDViewer [141] interface showing the analysis of a Simplelocker sample

processes, sockets and files seen as nodes in the graph. The red outline shows what nodes are interacting at that moment. We see, in this case, calls done between various processes. At the bottom of the figure, we can see a timeline in orange, with the ordinate representing the frequency of information flow event at kernel level. On the right side, is the "method flow graph", a graph showing methods, classes and packages as nodes, and explicit calls between methods as edges. This dual view allows to see the events occurring during application's execution on the right side of the interface, and the static structure of the application on the left side for further inspection of the code on the left side.

We mention with a more brief description other tools used in Android malware analysis, since these tools do not present much interesting information about an application. Yoo *et al.* [142][143] present a tool for visualizing security risks of an application on the smartphone. StoryDroid [144] generates a *storyboard* of the application's activities, that also functions as a graph of transitions between them. Jenkins *et al.* [145] propose a tool to better understand ICC and intents in an application. These tools use diverse techniques to deliver information for very specific tasks. Different parts of an application are exploited to obtain meaningful information about the studied application.

[142]: Yoo et al. (2017), 'Personal Visual Analytics for Android Security Risk Lifelog'

[144]: Chen et al. (2019), 'StoryDroid: Automated Generation of Storyboard for Android Apps'

[145]: Jenkins et al. (2017), 'Dissecting Android Inter-Component Communications via Interactive Visual Explorations'

These techniques are useful for analysts as they provide a visual aid for the analysis of an application. Analysts can also extract significant information from a single program by using these tools. However, these methods can not be directly used to visualize a collection of applications at once.

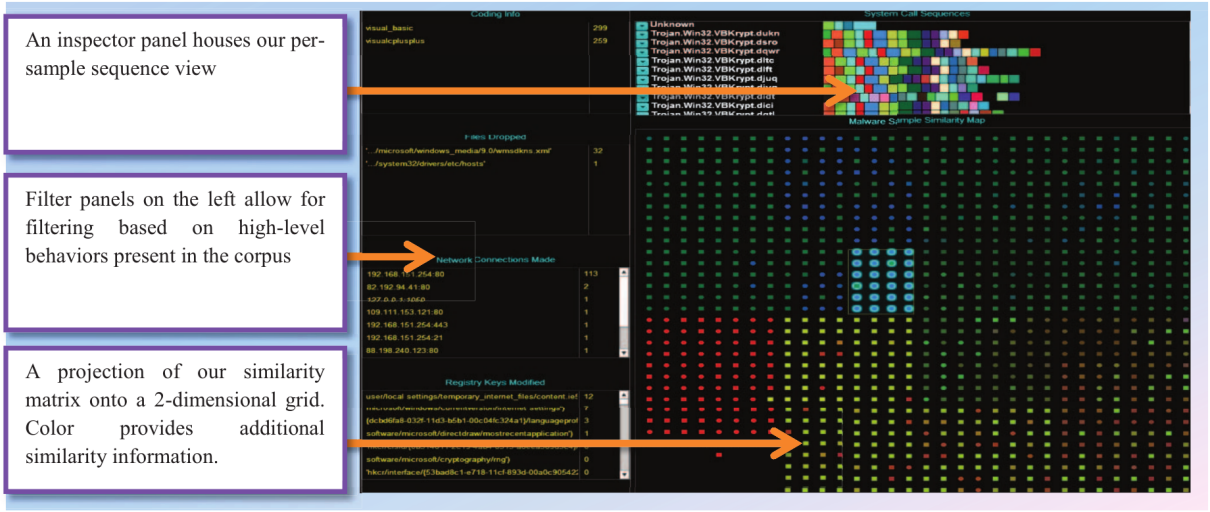

Figure 3.4: Screenshot of the tool's prototype proposed by Saxe *et al.* (taken from [146])

Visualization of datasets

Instead of visualizing a single sample, other visualization tools focus on a set of programs. As far as we know, there are no tools dedicated for Android applications for this type of task, however there exists tools tested with x86 programs [146, 147]. Thus, we decided to review these tools in order to see if they meet the same goals than our tasks, and if some visualization method could be adopted. The following papers try to represent a collection of applications, by either presenting information of every element it has, by showing similarities between subsets of elements, or both.

[146]: Saxe et al. (2012), 'Visualization of Shared System Call Sequence Relationships in Large Malware Corpora'

Similarity in system call sequences Saxe *et al.* [146] present a tool that shows similarities between applications in a malware dataset, according to their sequence of system call sequences. Semantic sequences of system calls in logs are calculated, and then compared between the different applications in the dataset. Figure 3.4, taken from the study, shows an overview of the tool's interface with their different parts. On the top left we see various colored blocks besides the name of various samples, each block representing a sequence of system calls, with unique colors for different sequences. On the left side, a filter panel of behavioral traits such as "registry key modifications, network communications, and file drops" allows to highlight the samples on the right that contain this behavior. And on the right is a grid called "Sample Similarity Map", computed using principal component analysis and project in 2D using a Hilbert curve. The grid shows the samples grouped in such a way that similar samples are put together, and take identical colors, with known samples as circles, and unknown ones as squares. It allows users to inspect a dataset for applications with similar behavior based on their system call sequences, where they can visually recognize similar samples grouped together by their colors. Although this tool allows the exploration of a single dataset, it does not allow for comparison of various datasets at the same time.

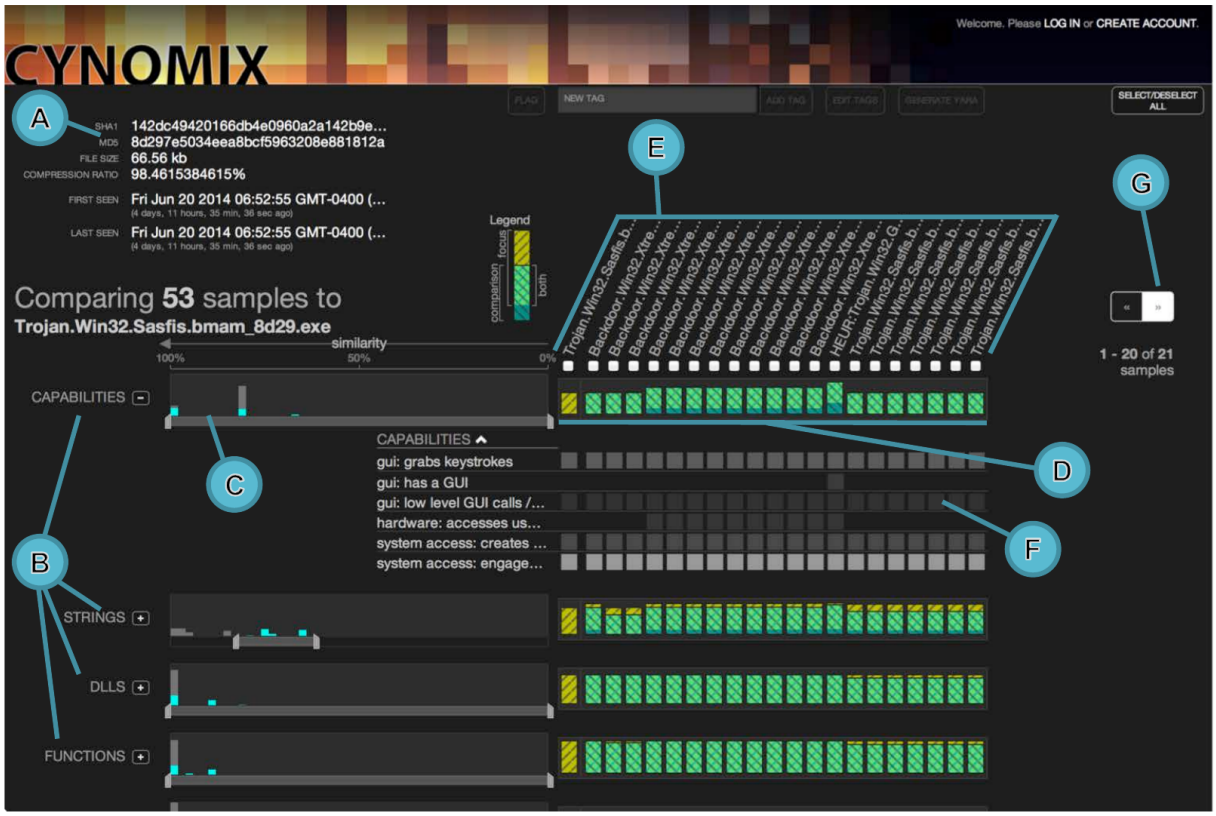

Figure 3.5: Screenshot of the SEEM interface by Gove *et al.* (taken from [147])

SEEM: comparing a sample to a dataset Gove *et al.* [147] propose a tool, called SEEM, to compare sets of attributes of a single malware sample (called focal sample) to a dataset. Figure 3.5 shows an overview of the tool's interface, with selected parts listed from A to G. On the top left (A), it shows several metadata about the applications such as different hash values, file size, and dates. From the left, we see multiple categories (B) such as capabilities, strings, DLLs, and functions. These are followed by a "similarity axis" (C), where it shows how similar is the focal sample with respect to the dataset, and allows users to filter samples to a range. Besides it on the left, a "Venn diagram list" (D) shows Venn diagrams for the top 20 most similar malware samples of the dataset to the focal sample (E). These Venn diagrams show the type of similarity for each category, whether there is an overlap, strict subset, or disjoint between the focal sample and an application of the dataset. It helps reverse engineers to speed up their work with new applications, and analysts to discern whether new samples are based on other older samples, or if they are different applications. As this tool does, we also use Venn diagrams, in our case to represent relationships between characteristics as we explain in section 3.2.3.

[147]: Gove et al. (2014), 'SEEM: A Scalable Visualization for Comparing Multiple Large Sets of Attributes for Malware Analysis'

Despite the ability to compare the samples a dataset between them, the tool by Saxe *et al.* [146] does not allow a comparison between sets of applications. And on the other hand, with the tool by Gove *et al.* [147], an exploratory task is not possible without having a comparison of one sample with a dataset.

Discussion

Visualization tools for single applications provide analysts with visual aid when performing reverse engineering and manual malware detection and classification, but they lack the ability to visualize more than one application at a time. Visualization for sets of applications so far enable researchers to: compare samples between them, or compare a single sample to a dataset.

For now, the tools for comparing features are specialize on x86 applications. For the comparison and exploration of Android malware datasets, no tool performs these tasks simultaneously.

3.2 DaViz - Dataset Visualization for Android malware datasets

So far we saw in Section 3.1 the tools used to visualize Android applications and datasets. We realized that there are no tools in the state-of-the-art for exploring and comparing Android datasets. This section presents DaViz*, shown in Figure 3.6,a new visualization tool with the goal of aiding in the exploration and comparison of Android malware datasets. We know intuitively that researchers in malware analysis can identify differences in datasets when the characteristics of these are shown visually. This new knowledge can help researchers select better datasets fitted for their experiments than randomly selecting applications from different sources, or using an already made dataset.

Section 3.2.2, we explain the characteristics used as input data for visualization. Next, in section 3.2.3, we explain the different charts used by DaViz and what type of information can be obtained from them. After that, in section 3.2.4, we explain the interactivity features exploited by DaViz. Later, in section 3.2.5, we describe the technical implementation of DaViz. Finally, in section 3.3 a use case to exemplify an usage scenario of DaViz.

3.2.1 Interface

Figure 3.6 presents an overview of DaViz's interface. On the top right of the figure, we see the name of the current's workspace, named as default "Workspace 1". Just below we see the user selection interface to create charts. On the left there is a dropdown list to select which graph to create:

- Venn diagram
- Bar chart
- Bar chart for categorical characteristics
- Heat maps

Next to it, a button "Select characteristics" for choosing the characteristics to display in the chart (see section 3.2.2), in this case for a Venn diagram. Next to this is a "Draw" button to create the chart once the user has

* Short for ***Da****taset* ***Viz****ulisation*

Figure 3.6: Overview of the DaViz's interface in Firefox 104

chose the characteristics to display. Finally in the left, there is a "Current selection", which display the characteristics chosen by the user before creating the chart. Since there is currently no characteristic chosen, "Current selection" display "(none)"

Below is the chart section of the display, where the charts are created. We can see three datasets displayed at the same time:

- AndroZoo 30k 2020, an extract of 30,000 applications from AndroZoo in 2020,
- VirusShare 2018, a collection of all Android malware collected by this platform in 2019,
- and AMD, a collection of malware from 2010 to 2016.

AndroZoo 30k 2020 is hidden, but can be unfolded by pressing the triangle button next to its name. Below is VirusShare 2018 showing three charts from left to right:

- a Venn diagram presenting the relationship of proportions of applications with the characteristics "Use reflection", "Use AccessibilityService", "DexClassLoader", and "LoadClass",
- a bar chart displaying the number of applications by MinSDK version,
- and a heatmap showing the proportion of applications that have "Use Java JNI", "Recieve SMS", "Send SMS" on abscissa, and "Use MMS", "Place calls", "Get CPU ABI", and "loadClass" on the ordinate.

And below this one is AMD, showing the same charts for this dataset.

On the top left corner there are four elements to change the current view. First, on the right, a dropdown list displaying "List View", which allows the user to select between this current view ("List View") and a "Compare View". Next, on the right, there are three icons:

- the left one allows to save a workspace with all the selections applied to the last chart,
- the middle one allows to switch between saved workspaces,
- and the right one allows to save the session, including their workspaces, for future use.

3.2.2 Input data: characteristics

The data we used to represent the datasets are *characteristics*: properties extracted from applications, *e.g.* their size, the number of classes they contain, whether they make a specific API call, etc. These characteristics are computed using Droidlysis [148], a "property extractor for Android apps" [†]: for each application it outputs different boolean characteristics from patterns in the code, alongside other ones from the application's Manifest file (like permissions, intents, activities, application size, number of classes, etc.).

Some characteristics, like the date and size, are shown in intervals parameterized by the user, like the date in the range of years, while the rest of characteristics with boolean attributes are shown in a characteristics tree, as we explain later. New characteristics can be added accordingly, by modifying Droidlysis or by using other analysis tools that output data about programs. Using this output data from Droidlysis, we can calculate the proportion of applications that have a certain characteristic in the dataset. With the computed proportions, we propose several visualization methods, using user interactivity.

Since there are 171 characteristics calculated using Droidlysis, we decided to arrange them in categories when the user is invited to select the set of characteristics to display. They are displayed as a tree, divided in categories, sub-categories, and finally the characteristics as shown in Figure 3.7. These categories are the following:

- "code": These are code-based characteristics that represent what the application can potentially do.

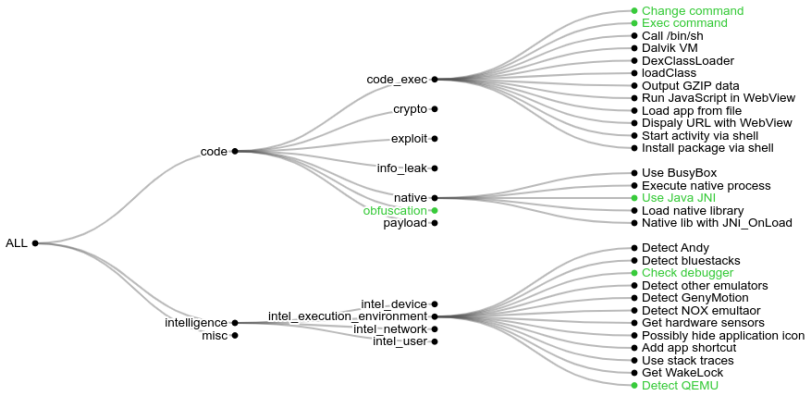

Figure 3.7: Hierarchical visualization of characteristics divided in several categories. Selected characteristics are highlighted in green

[†] Droidlysis' github page

- "code_exec": These are actions linked to external code execution, *e.g.* any "change" command (`chmod`, `chown`, `chattr`), load a DEX class, start activity via shell, etc.
- "crypto": These are properties linked to the usage of cryptography, *e.g.* compute CRC32 and traces of encryption schemes usage (cipher method and key specification).
- "exploit": These are traces of known vulnerabilities found in several CVEs specific to Android applications, *e.g.* CVE-2020-0069, CVE-2012-0025, among others.
- "info_leak": These are methods to allow attackers obtain information collected by the application, *e.g.* usage of SSH, create a network socket, usage of HTTP GET, etc.
- "native": These are properties linked to the usage of native code in the application, *e.g.* usage of Java JNI, load native libraries, use of BusyBox, etc.
- "obfuscation": These are traces that indicate the use of methods to hinder the analysis of the application, *e.g.* use of reflection, use of APKProtect, usage of base64 strings, etc.
- "payload": These are methods mostly used in the application's payload, *e.g.* send and recieve SMSs, stop/kill processes, usage of cryptominers, etc.

► "intelligence": These are information collection methods the application potentially can perform.

- "intel_device": These methods obtain information related to the current device the application runs in, *e.g.* IMEI, hardware information, bootloader's version, etc.
- "intel_execution_environment": These methods obtain information related to the application's execution environment, *e.g.* emulation detection (like Andydroid, bluestacks, QEMU and other emulators), hardware sensors, check for debugger, etc.
- "intel_network": These methods obtain information related to network connectivity of the phone: *e.g.* MAC address, WiFi SSID and RSSI, scan for WiFi hotspots, and cellular operator name.
- "intel_user": These methods obtain information related to the user: *e.g.* call logs, IP address, phone email, SIM country, access to microphone, etc.

► "misc": These are other miscellaneous characteristics that could not fit in the last two categories.

- "integrity": These are actions the application does that modifies the system in one way or another, *e.g.* install an APK, mute microphone, place calls, bypass idling mechanisms, use adb internally, etc.
- "interesting_behavior": These are actions the application performs that may be of interest in combination with other characteristics, *e.g.* IP addresses found in files, phone numbers found in files, call intent chooser, parse URI, etc.

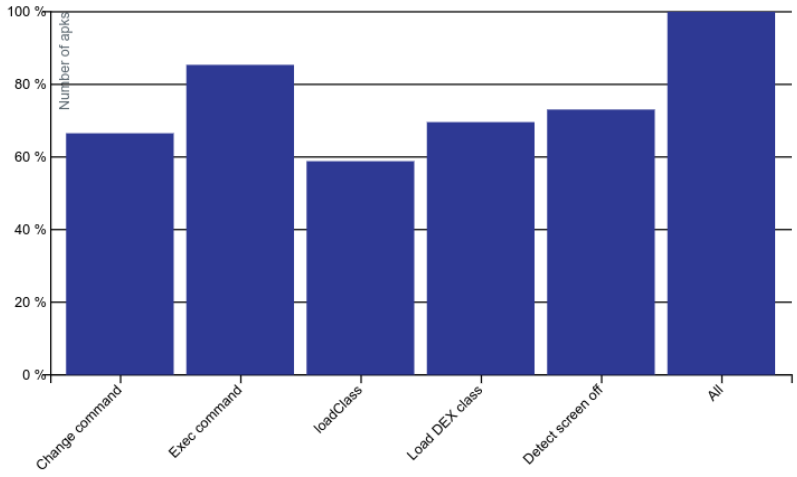

Figure 3.8: Example of a bar chart, showing for each bar the proportion, in percentage, of applications in the dataset with the given characteristic

3.2.3 Output views: charts

DaViz allows users to browse datasets using different types of charts and the multiple characteristics available. We will explain what these charts are, what type of information they convey, and in what cases they can be used.

Bar charts These charts show the size relationship between bars. An example of a bar chart is given in Figure 3.8. Each bar represent a single characteristic and its size represents a value such as:

- Number of applications
- Average average size
- Average Min SDK version
- Average Max SDK version

Users specify both abscissa and ordinate before creating the chart. In the case of characteristics such as size and the application's date year, users specify the range for each bar.

According to Munzner [135], bar charts are used to lookup the values and compare them. In the dataset exploration task, users can identify which characteristics are present the most or the least in a dataset. On the other hand, in the dataset comparison task, users can compare which characteristics are more present in one dataset than the other, or equally present in both. These observations can lead to further investigation concerning the characteristics found to be, for example, more present in a goodware dataset than in a malware dataset. However, with bar charts, it is not possible to make the same analysis for proportion of applications with more than one characteristic simultaneously.

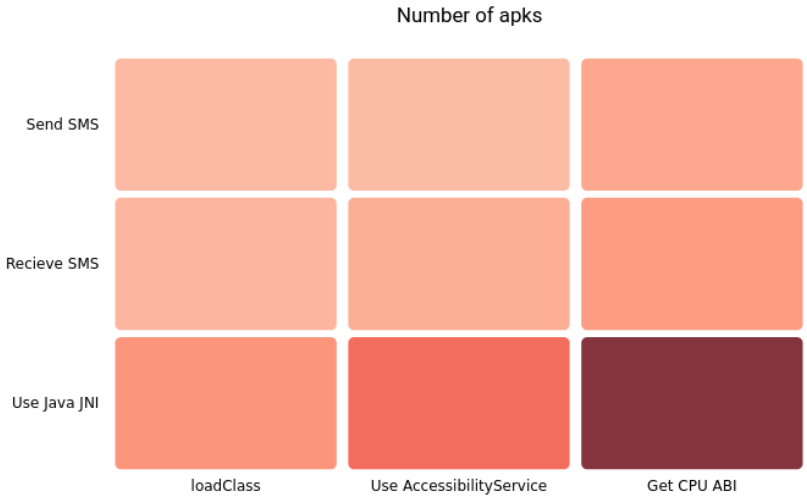

Figure 3.9: Example of a three by three heatmap, with the number of APK as color intensity

Heatmaps These charts show the magnitude between two dimensions with a color hue. An example of a heatmap chart is given in Figure 3.9. This example shows that the dataset contains mostly applications that use the Java JNI, and inform Android which application binary interface (ABI) it uses to load the correct native program or library (in x86, ARM or MIPS). Each cell represents the proportion of applications that have the two corresponding characteristics in abscissa and ordinate at the same time in the dataset. A "third" dimension is used, the color hue, to represent a value corresponding to one element of the abscissa and one of the ordinate. Users can specify the different characteristics for the abscissa and ordinate, and lastly the value for each intersection (number of applications having both characteristics, the average size, etc.).

According to Munzner [135], heatmaps are used to find clusters and outliers, but also to summarize the data. In the dataset exploration task, users can compare, within the dataset, the proportion of pairwise presence of several characteristics in applications at once. For the dataset comparison task, users can compare, between datasets, these pairwise proportions to identify a difference in presence of the chosen characteristics. For example, when looking for a pair of characteristics highly used at the same time, the heatmap will show a cell with a more intense color than the rest of the cells, which can be spotted easily. When this chart is used for dataset comparison, users take advantage of the color intensity to locate proportion differences with pairs of characteristics between datasets. Although with heatmaps we can observe the proportion of applications with two characteristics in a dataset, we show in the following how it is possible to generalize these characteristics' relationships using Venn diagrams.

Venn (Euler) charts In order to see the proportion of applications that have multiple characteristics, we have to treat each characteristic as a *set* of applications that present it. A common way to do this is through the use of Venn diagrams. They allow to see the set relationships between different characteristics as sets, where the size of each set corresponds to the number of applications with that characteristic. Figures 3.10a and 3.10b show examples of Venn diagrams. Intersections between sets correspond to applications that have all the characteristics of the intersecting sets.

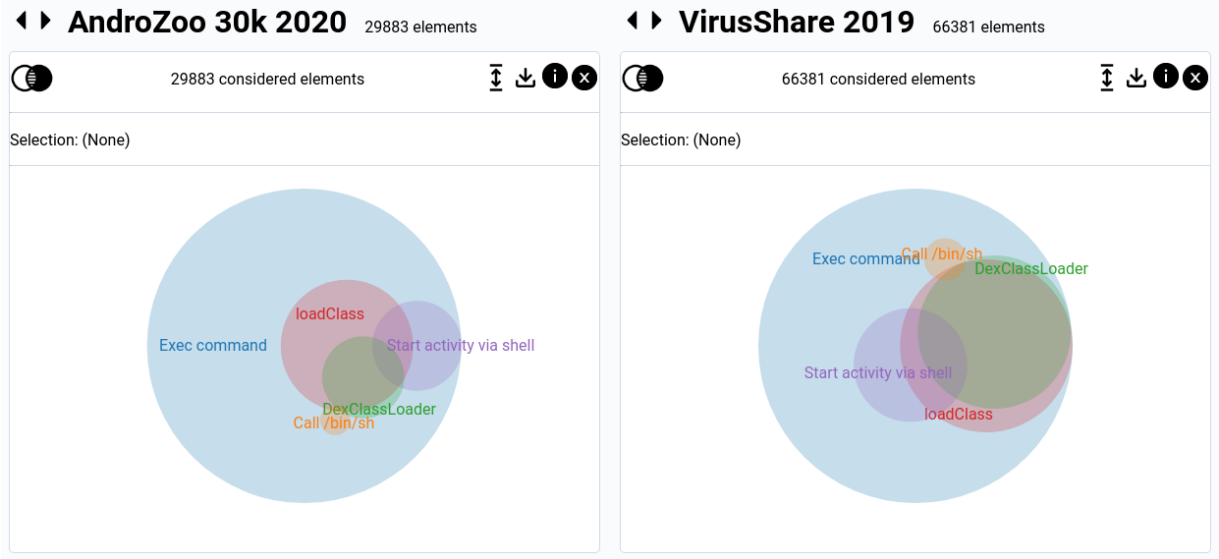

Figure 3.10a: A Venn diagram in DaViz

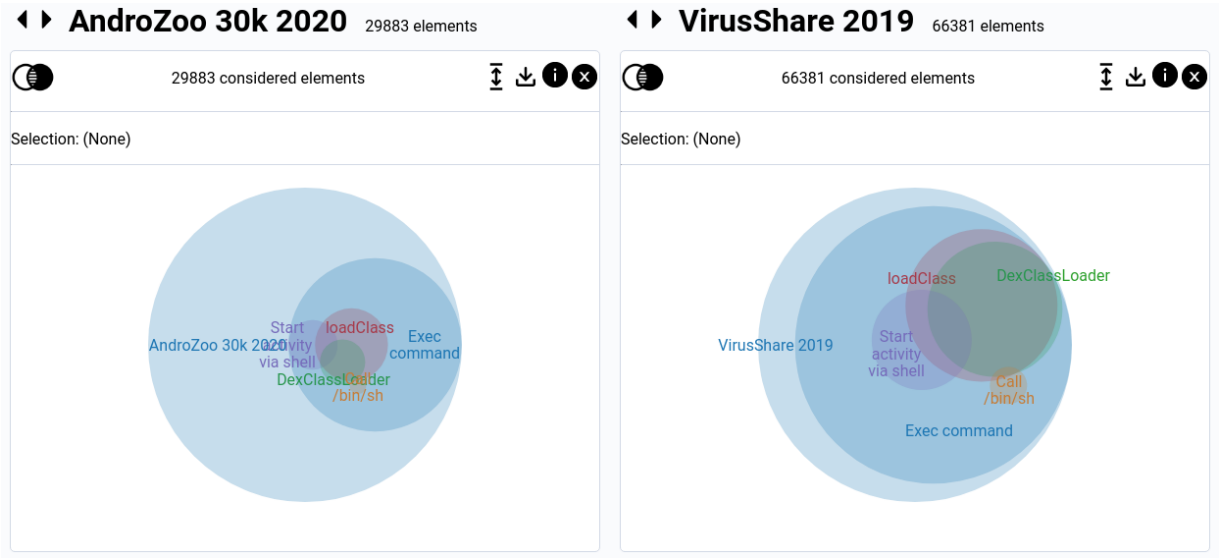

Figure 3.10b: The same Venn diagram with the whole dataset included

The methodology by Munzner [135] does not cover the use of Venn diagrams. However, the study made by Alsallakh *et al.* [149] explores the uses of Venn diagrams to show and present sets. They propose different tasks that users may want to perform with these sets. These tasks can be related to elements, to sets and set relations, or element attributes. Among the tasks established by Alsallakh *et al.* [149], DaViz allows to perform the following tasks related to sets and set relations:

[149]: Alsallakh et al. (2016), 'The State-of-the-Art of Set Visualization'

- (B2) Analyze inclusion relations, *e.g.*, identify which sets are fully included in other sets, intersections or unions.
- (B3) Analyze inclusion hierarchies, *e.g.*, identify if a set A is included in B, and B is then included in C, and so on.
- (B4) Analyze exclusion relations, *e.g.*, if a set does not intersect with another, or if it does not have any intersection at all.
- (B5) Analyze intersection relations, *e.g.*, if certain sets form intersections.
- (B7) Identify the sets that constitute a certain intersection. Users can hover the intersections and see which are the sets constitute it.
- (B8) Identify set intersections contained in a specific set.
- (B10) Analyze and compare set and intersection cardinalities. This is done by comparing the sizes shown in diagram.

Venn diagrams allow to identify the relationship of the chosen characteristics: if many of them are mutually exclusive, they will appear separate from each other. On the contrary, if they are highly present in a group of applications, the diagram will show them overlap each other, up to the point of perfectly fit in each other if there is a one to one correlation between (*e.g.* a group of applications all have these characteristics, but there are none without one of them). In the dataset exploration task, this helps to visually identify correlations between several characteristics. In the dataset comparison task, these correlations can be compared, identifying differences in the usage of a set of characteristics between datasets.

3.2.4 Interactivity

Interactivity is one of the key features of DaViz: once a chart is created, users can create more charts that will appear from left to right (or from top to bottom) of the first chart, and an interaction performing a selection on one of the charts will affect the charts that follows (on the bottom or on the right, depending of the chart's flow).

The intent behind this behavior is to allow users select sub-parts of the datasets based on the selection of characteristics in a chart. This will affect all the following charts of the flow, as they will only consider elements with the selected characteristics in the previous charts where the selection was made. For example, Figure 3.11 shows, on the left, an extract of F-Droid in 2021 with a Venn diagram followed by a bar chart on the bottom. The chart shows the proportion for 3,587 elements. When we select the set in the middle of the Venn diagram, the intersection of "Get SD card state" and "Get phone IMEI", the bar chart is updated, as seen on the right, to consider only elements with these two characteristics. In this case, we get, as seen in the bar chart, a higher concentration of applications that use native methods, among a new total of 82 elements.

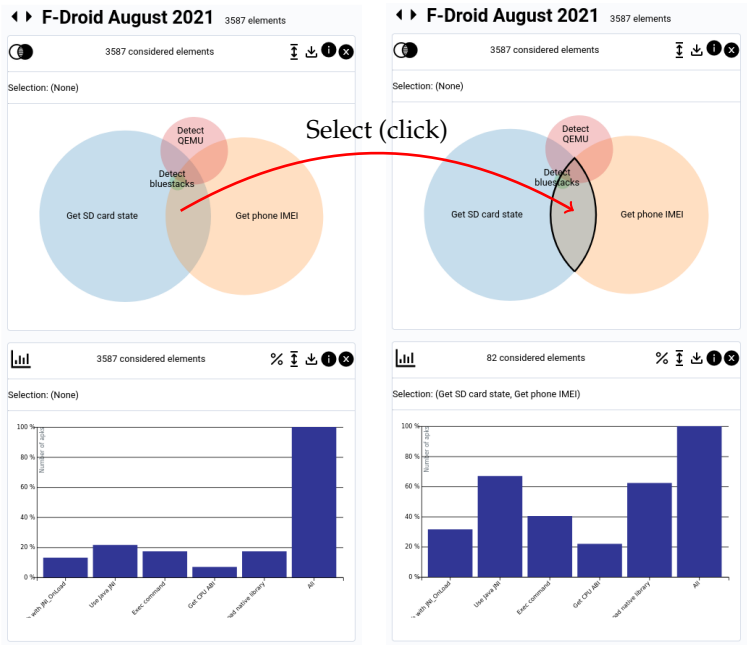

Figure 3.11: Selection of elements with "Get SD card state" and "Get phone IMEI" on the F-Droid 2021 dataset

This principle of interaction have been implemented for all three types of charts: bar charts, heat maps, and Venn diagrams.

We took inspiration from the study by Alsallakh *et al.* [149] to implement this behavior in DaViz. The study formalizes the tasks related to elements in Venn diagrams:

- (A1) Select elements that belong to a specific set. By clicking in the diagram, the user can select those elements, filtering out those that do not belong to the set.
- (A3) Select elements based on their set memberships. Users are allowed to select internal parts of intersections, without taking into account certain sets that are of the intersection.
- (A5) Filter out elements based on their set memberships. This is done by selecting sets or intersections in the diagram. The resulting selection appears in the next diagrams, as explained earlier.

Additionally, a specific interaction action have been implemented for Venn diagrams for displaying elements when they are not concerned by the characteristics, and thus not displayed. In Figure 3.10a, we can see that the "Exec command" is the biggest set (the characteristic is present the most in comparison to the others), with almost all other sets inside it, meaning that "Exec command" is always present in an application if these other are. We can see a difference in the intersection of "DexClassLoader" and "loadClass" between the two datasets: in AndroZoo 30k 2020 there is less use of both characteristics simultaneously than in VirusShare 2019. Notice that the sets' sizes represent the proportion between them in one dataset, which can be misleading for comparison: the sets' sizes cannot be visually compared because the number of considered applications is not the same. In order to fix this, users can include the whole dataset to the diagram. By selecting the scale option, the previous Venn diagram changes to Figure 3.10b. Now we can see that the "Exec command", although is the most present characteristic among the others selected, is not as present as in the malware dataset, where almost every application

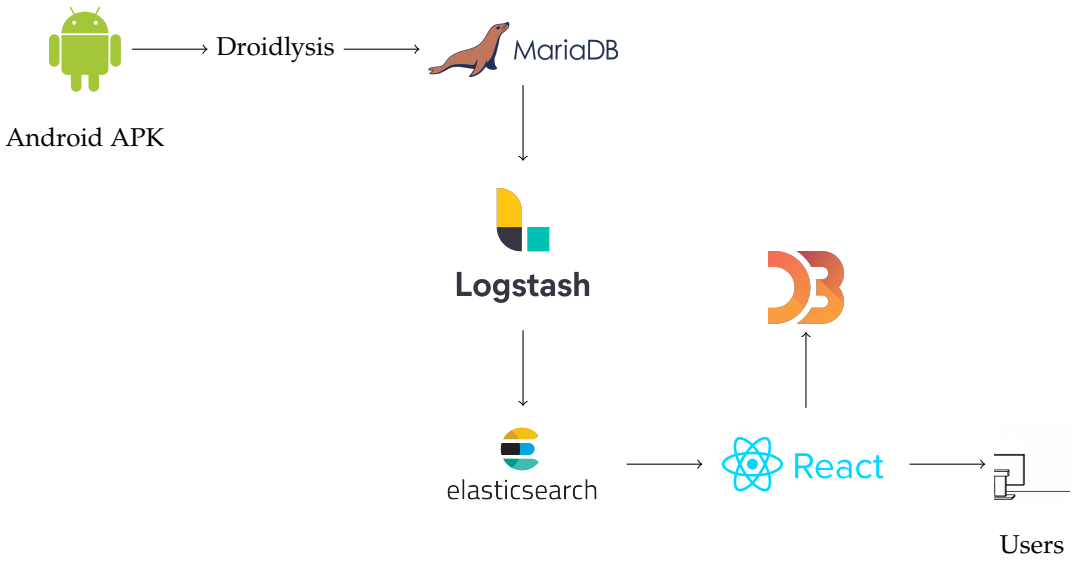

Figure 3.12: Diagram of the system implementation of DaViz

use it. The user can conclude that the VirusShare dataset contains mostly malware that try to execute code more frequently compared to the general case of applications in AndroZoo, and further inspection may be performed.

As a summary, DaViz can perform a one-to-one comparison of datasets using bar charts, heat maps and Venn diagrams. DaViz's chained charts allow users to navigate datasets with different characteristics in each chart. Moreover, the selection of characteristics in the charts allows to concentrate the view on the selected characteristics in subsequent charts. This can help filter out applications of a dataset that do not apply to the criteria selected by the user, and consolidate those of interest for further inspection.

3.2.5 Implementation

Figure 3.12 shows the diagram of the implementation for DaViz, showing the flow of data from Droidlysis analyses of APKs to the display of charts to users. We analyze each application of the datasets with Droidlysis, and store the results in a MariaDB relational database. The data is structured so that each application is represented by its SHA-256 hash value.

When users create their charts, the data has to be received as fast a possible. Delays in generating changes in the charts would impact the interactivity of the tool. To allow a more responsive response, the Elastic stack is used. It is composed of a data collection tool called Logstash and the Elasticsearch search engine. The former is used to transform the data from MariaDB to the data interpretation used by Elasticsearch. This last one handles the requests from the display client. Elasticsearch, on the other hand, is fast enough to allow rapid changes in the charts with short delays between a click on the chart and the corresponding reaction on it.[1]

A third part is the display client that the user will interact with, which is based on React to arrange the user interface and make requests to the

1: An earlier version of DaViz used MongoDB instead of Elasticsearch, and it suffered from performance issues.

Elasticsearch instance. It also uses the D3 library to create the charts and manage the interactions with them, like clicking on a bar or one of the intersections of the Venn diagram. Because analyses to malware must be performed in a secure environment, and to preserve the data from external attackers, all these instances are hosted in the High Security Laboratory (*Laboratoire de haute sécurité* or LHS) at INRIA in Rennes, France.

In the next section, we will explain our usage of interactivity to help users take advantage of the tool.

Discussion

With the variety of charts and characteristics available, users can gain knowledge from unknown datasets, by inspecting the different concentration of applications with selected characteristics in the dataset. Users can also examine the relationships between different characteristics inside a dataset. Additionally, the use of a fast implementation permit to change rapidly the views with new charts and filters, allowing users to explore continuously the data, which helps in the verification of hypotheses. This accomplishes our "Dataset exploration" goal in accordance with the methodology of Munzner [135].

By looking at the graphs, we see the differences in the proportion of individual characteristics between datasets, and differences in the proportion of multiple characteristics between datasets. This information can be useful to researchers when they create their experimental datasets. On one hand, they can inspect datasets to see the relationship between the characteristics of interest and other ones. This would allow researchers to identify unexpected patterns about the presence of these relationships. On the other hand, researchers can compare their newly created dataset to another one, to see whether their dataset does not differ with the expected proportion of characteristics of another one. This accomplishes the "Datastet comparison" goal in accordance with the methodology of Munzner [135].

With the variety of characteristics available, researchers can systematically explore a dataset. While doing this, they can also compare datasets using the same charts. The use of interactivity and fast implementations allow to rapidly make changes in the charts. A comparison between other datasets from the literature would further exemplify the utility of this new tool.

3.3 Visualization of datasets in the literature

With DaViz explained in the last section, we now show its use with a comparison of Android malware datasets. Several datasets from the literature were analyzed and imported into DaViz. In this section we are going to compare some of these datasets and see the differences between them. We start by comparing older and newer malware datasets to identify and discover key differences between them, and then comparing malware datasets from the literature with AndroZoo.

3.3.1 Differences between older and newer malware datasets

We are going to compare Drebin and AMD, particularly old but still widely used Android malware datasets, with applications dating from 2010 to 2012, to VirusShare 2018, a collection of all Android malware collected by the VirusShare platform in 2018. We will see the changes in the composition of these Android datasets malware dataset through time.

This first chart in Figure 3.13 shows the seven characteristics, automatically computed by DaViz, that have the biggest proportion difference between these datasets. At first glance, we see that around 60% of all applications VirusShare 2018 contains one of these characteristics, while Drebin never exceeds 20%. Among these, there is "Exec command", "Load DEX class", "Change command", "Load class". These are mainly code execution techniques used to hide the payload somewhere else, and then load it when the malware is ready to activate its malicious activity. On the other hand, few applications in Drebin have these characteristics. From this, we can assume that execution methods have changed over the years from the samples in Drebin to VirusShare 2018. And so, applications from Drebin are less complex compared to those in VirusShare 2018. As a consequence, we can suspect that studies using Drebin as an evaluation dataset will have significantly different results, compared to an evaluation using VirusShare 2018.

Let us examine the influence of "Change command"[2], one of the characteristics with the most concentration in VirusShare 2018, in these datasets: we click over "Change command" in the bar chart of Figure 3.13 and we obtain Figure 3.14. In this last figure, we can see that the number of applications in Drebin is reduced to 168. Indeed, it is only 3.02% of Drebin, compared to 66.64% of VirusShare 2018.. Nevertheless, with these few samples, we see the use of "Start activity via shell" and "Load native library" in almost the same proportions in Drebin as in VirusShare 2018 in relation to "Use Java JNI", but with more applications using these characteristics at the same time in VirusShare 2018 than in Drebin. We also see that "Load native library" and "Use Java JNI" are present simultaneously in both datasets. These observation indicate a shift in the behavior of applications to one using more native methods.

2: "Change command" refer to any of either `chown`, `chmod`, `chgrp`, `chcon` or `chattr` patterns found in the application's code

The same analysis is done with AMD and VirusShare 2018. Figure 3.15 shows again the seven characteristics, automatically computed by DaViz, with the biggest proportion difference between these datasets. Even if samples in AMD are older by two years, the figure presents the same characteristics as the previous comparison of Drebin and VirusShare 2018, just with a slightly different order. We can see that characteristics such as "Detect screen off", "Load DEX class", "Load class" and "URL in executable" are less used in VirusShare 2018 compared to AMD.

Again, we examine the influence of the "Change command" in these datasets as seen in Figure 3.16. In this case, "Change command" is present in only 2.21% of AMD, or 526 of samples presenting this characteristic, while this characteristic is present in 66.64% of VirusShare 2018. In both AMD and VirusShare 2018 we see that "Load native library" and "Use Java JNI" are both present, almost with the same proportions. Although

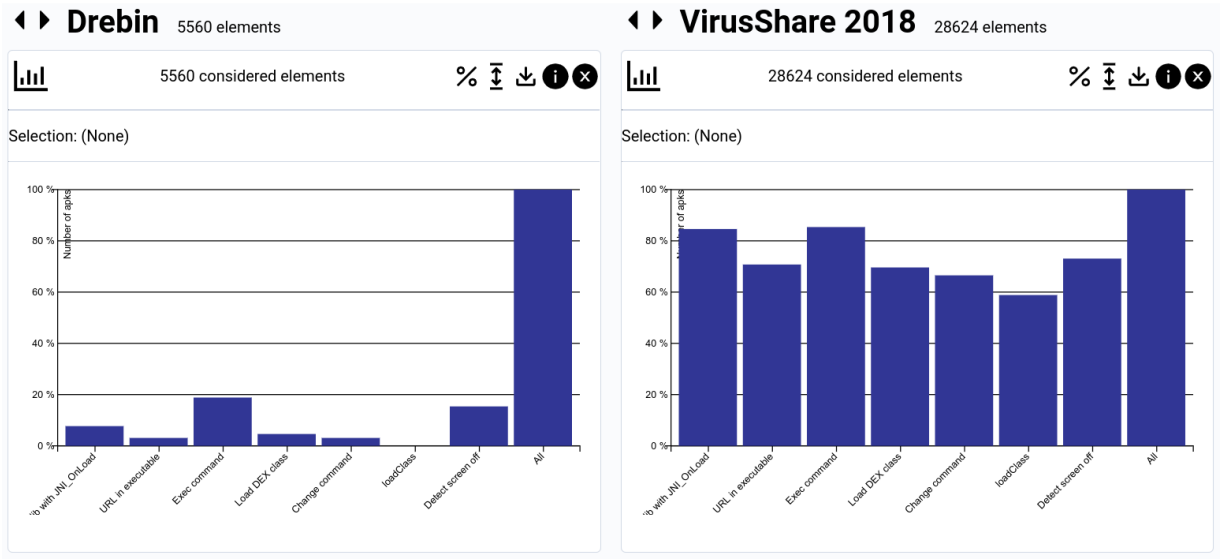

Figure 3.13: Top seven characteristics with the most proportion difference between Drebin and VirusShare 2018

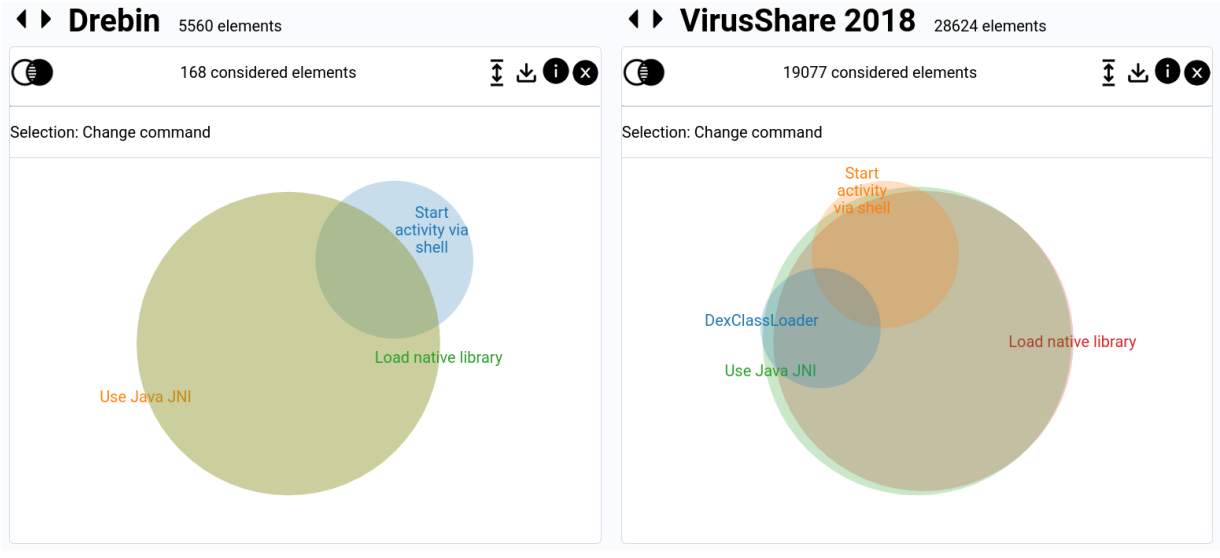

Figure 3.14: Venn diagrams showing some of the top seven characteristics with the most proportion difference between Drebin and VirusShare 2018

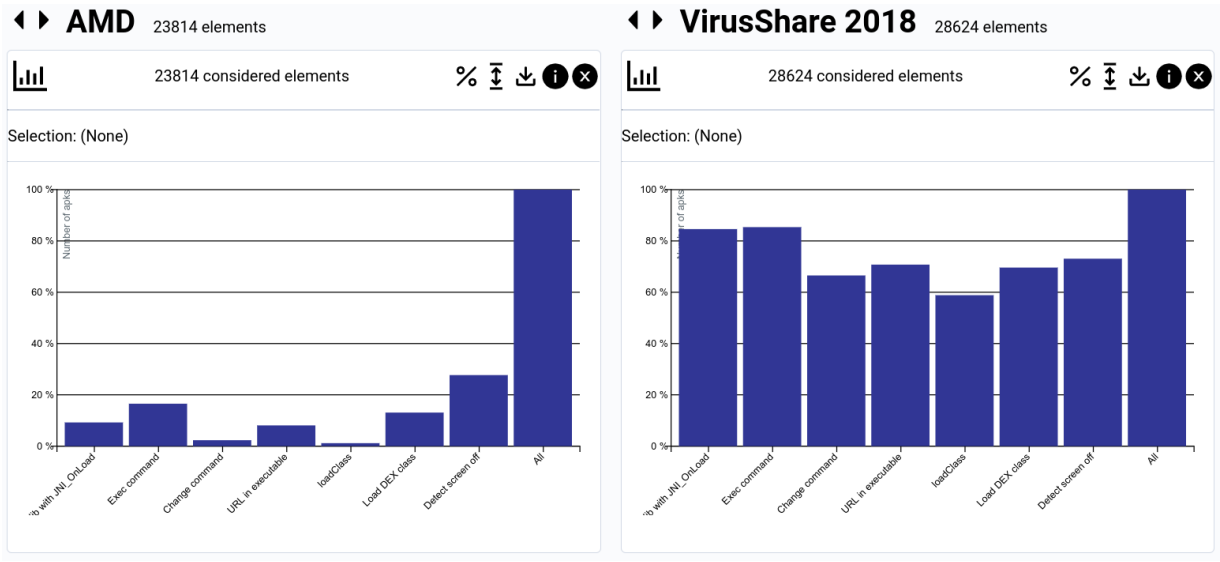

Figure 3.15: Top seven characteristics with the most proportion difference between AMD and VirusShare 2018

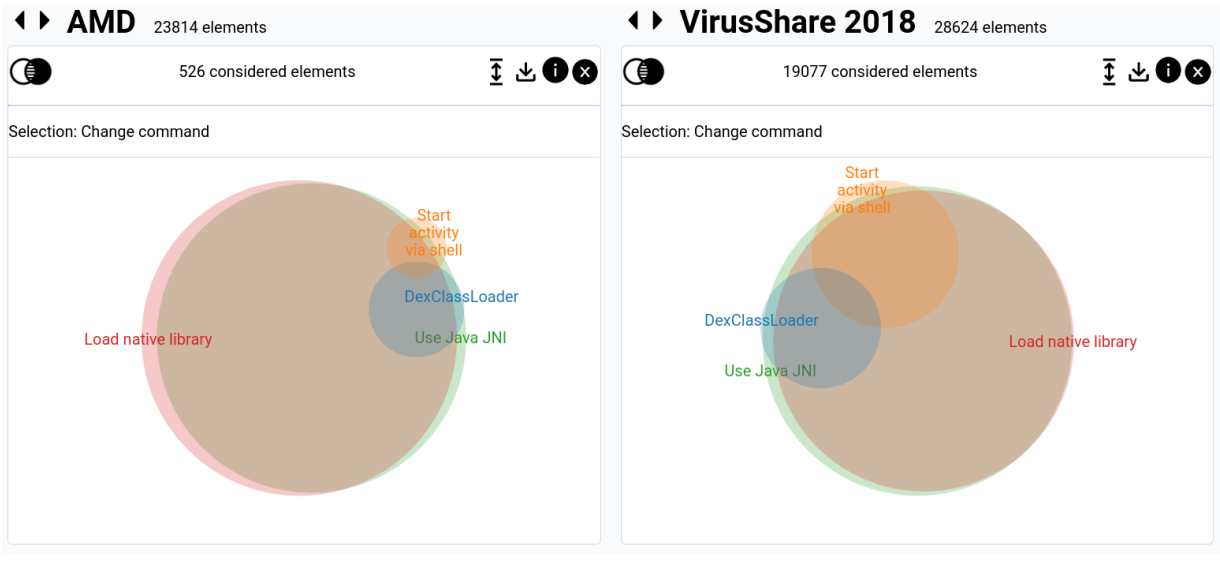

Figure 3.16: Venn diagrams showing some of the top seven characteristics with the most proportion difference between AMD and VirusShare 2018

"DexClassLoader" and "Start activity via shell" are present in AMD, we can see that there is more presence of these characteristics in VirusShare 2018. This evidences a change of behavior, where newer applications tend to use more native programs to hide their malicious behavior.

Now we compare Drebin to AMD using previous figures. Comparing Figure 3.13 and Figure 3.15, AMD has more "Detect screen off", "Load DEX class", "Load class" and "URL in executable" than Drebin. And comparing Figure 3.14 and Figure 3.16, "Change command" is present in only 2.21% of AMD, which is lower in proportion compared to Drebin but with more samples (526 vs 168). We also see applications with "DexClassLoader" and "Load native library" in AMD, whereas in Drebin there are very few with "Load native library" or none with "DexClassLoader". Again, there is a trend of more recent malware to use native programs and loading external classes than in older malware samples.

We see a significant change in malware characteristics from Drebin to ViruShare 2018: newer applications tend to use more loading techniques than older ones, most probably to hide its malicious execution, with only few old applications that use these techniques. With AMD, these differences were also the same for some characteristics, as there are no significant changes between this dataset and Drebin. And although there are less applications in AMD and Drebin that use native libraries than VirusShare 2018, in all three datasets the use of JNI and library loading are present simultaneously.

3.3.2 Differences between literature datasets and AndroZoo

In this section, we intent to compare the same specific datasets, Drebin and AMD, with the "target" dataset that represent markets. Because markets cannot be easily retrieved, we use AndroZoo for this purpose. Indeed, as we said in the previous chapter, AndroZoo is a collection of applications from many sources, particularly from Google Play and Chinese markets. This dataset broadly represent applications found in the wild, that is, in environments where users can download them. As with any other market, malware can also be present in a AndroZoo sample, as they could be undetected at the moment AndroZoo collected them, so malware may appear in the sample. Drebin, being a malware dataset with applications dating between 2011 and 2012, should present a significant difference in certain characteristics with respect to AndroZoo.

Figure 3.17 shows the seven characteristics with the biggest proportion difference between Drebin and a 30 000 applications sample of AndroZoo in 2020. The biggest difference between these datasets is the lack of "Google Play services" in Drebin versus 60% of applications in AndroZoo using it. Another interesting but expected observation is the more prominent use of "Send SMS" in Drebin than in AndroZoo. In fact, SMS is one of the attack vectors of malware, as they use it to subscribe the victim to a premium rate service, or to make the victim pay a service for the attacker through SMS. The IMEI is a number used to identify phones in the GSM network, and can be used to blacklist a stolen phone from being used in a network. It can be also used to identify a phone regardless of the phone number. Malware, such as the Gemini [150] and

[150]: F-Secure (2010), *Trojan:Android/Geinimi*

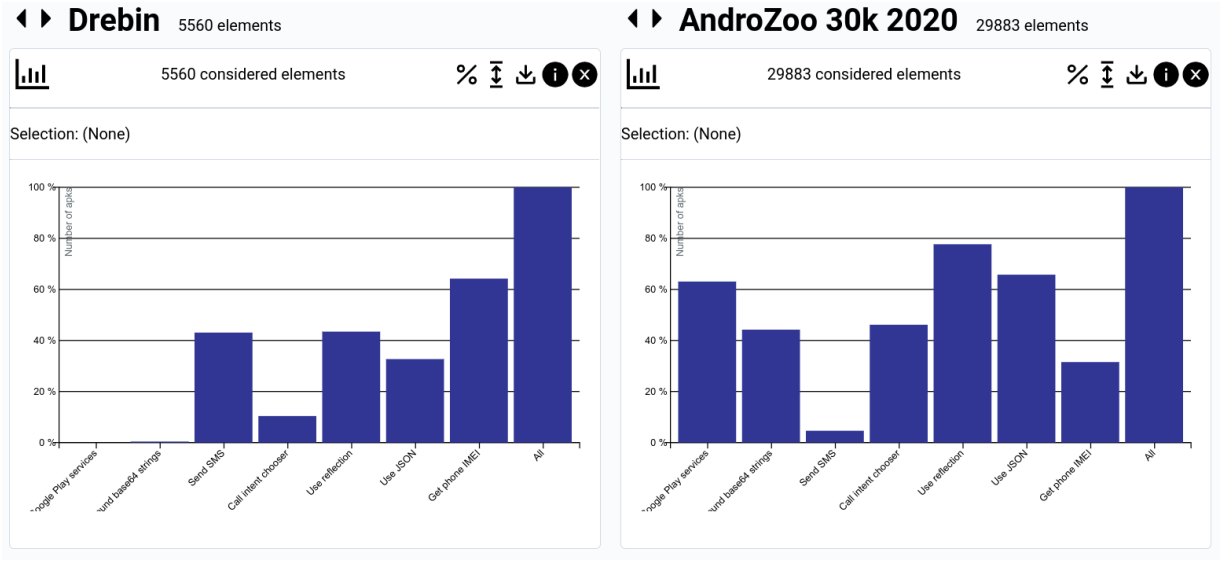

Figure 3.17: Top seven characteristics with the most proportion difference between Drebin and a 30 000 sample of AndroZoo in 2020

Figure 3.18: Top seven characteristics with the most proportion difference between AMD and a 30 000 sample of AndroZoo in 2020

GingerMaster [151] families, tend to use this to identify victims and gather as much information about them as possible. We see that more than 60% of malware in Drebin gets the phone's IMEI compared to AndroZoo (around 30%). This indicate clearly identifiable characteristics of Drebin, such a the lack of Google Play services, the use of SMS, and the extraction of identifiable information from the phone.

[151]: FortiGuard Labs (2011), *Android/GingerMaster.A!tr*

In the case of a comparison between AMD and AndroZoo, some changes appear compared to Drebin. First, as wee see in Figure 3.18, samples in AMD collect the phone's number (40% of application with the "Get phone number" characteristic) in addition to the IMEI (63%). They also appears to collect MAC addresses (46% of applications with "Get MAC address" characteristic), as well as the SIM operator numeric name (43% of applications with "Get SIM operator" characteristic). Another interesting behavior is "Get installed packages" in more than 40% of applications in AMD. These information are not only of interest to malware authors. This feature can be exploited to profile users' behavior by legitimate application [152], where it is mostly third party libraries that perform this type of data collection [153].

[152]: Scoccia et al. (2020), 'Leave My Apps Alone! A Study on How Android Developers Access Installed Apps on User's Device'

[153]: Pham et al. (2019), 'HideMyApp: Hiding the Presence of Sensitive Apps on Android'

As seen previously, both Drebin and AMD have very few applications with "Native lib with JNI_OnLoad", "Exec command", "Load DEX class", "Change command", and "loadClass" compared to VirusShare 2018. Comparing these characteristics with AndroZoo, we see that, again, both Drebin (in Figure 3.19) and AMD (in Figure 3.20) have less applications with these characteristics than AndroZoo. This is another notable difference between recent applications and older malware. If we examine a comparison between VirusShare 2018 and AndroZoo, Figure 3.21 shows us that almost all applications in VirusShare have these characteristics. More precisely, in Figure 3.22, we see that more than 80% of applications in VirusShare have "Native lib with JNI_OnLoad" and "Exec Command", and around 60% have "Load DEX class", "Change command" and "loadClass", while these characteristics are only present in around 20% of applications in AndroZoo. Indeed, there is an increase in usage of native methods in malware nowadays than ten years ago.

Discussion

Through the use of visualization we make a comparison between older and newer malware datasets, and between widely used malware datasets of the literature, Drebin and AMD, and AndroZoo. We see differences in terms of characteristics proportions such as "Native lib with JNI_OnLoad", "Exec command", "Load DEX class" and "Change command", which leads us to notice the evolution of malware towards the use of native code. We also made a comparison of a sample from AndroZoo against these three malware datasets from the literature. We see that malware in general tend not to use Google Play services; on the other hand, they tend to ask more frequently identifiable data such as the phone's IMEI, MAC address or SIM operator, besides the use of SMS.

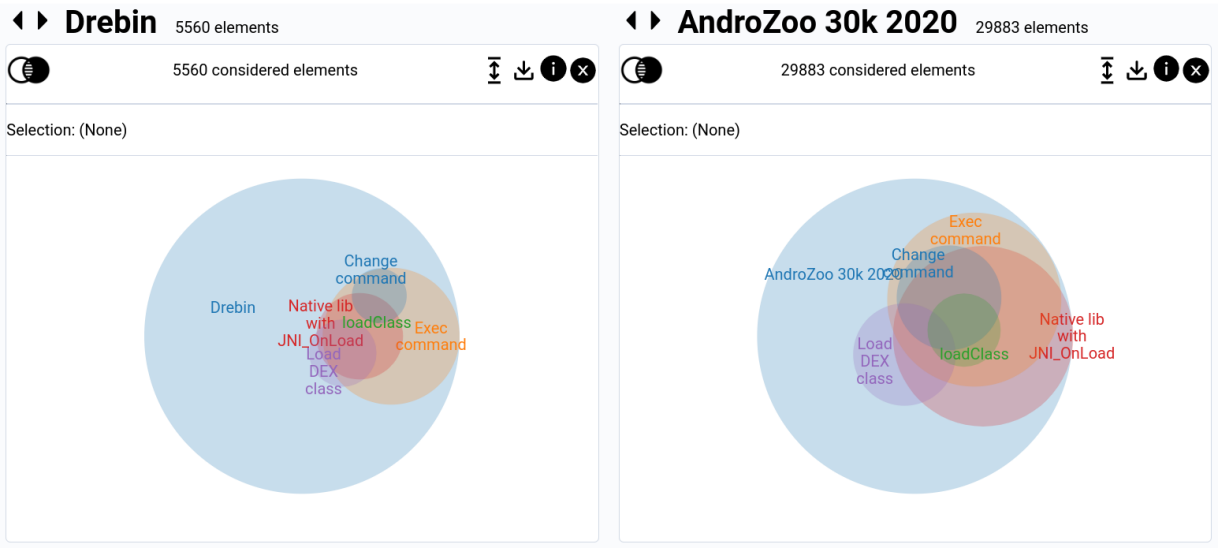

Figure 3.19: Venn diagrams showing the relationships between "Native lib with JNI_OnLoad", "Change command", "loadClass", "Exec command", "Load DEX class" in Drebin and a 30 000 sample of AndroZoo in 2020

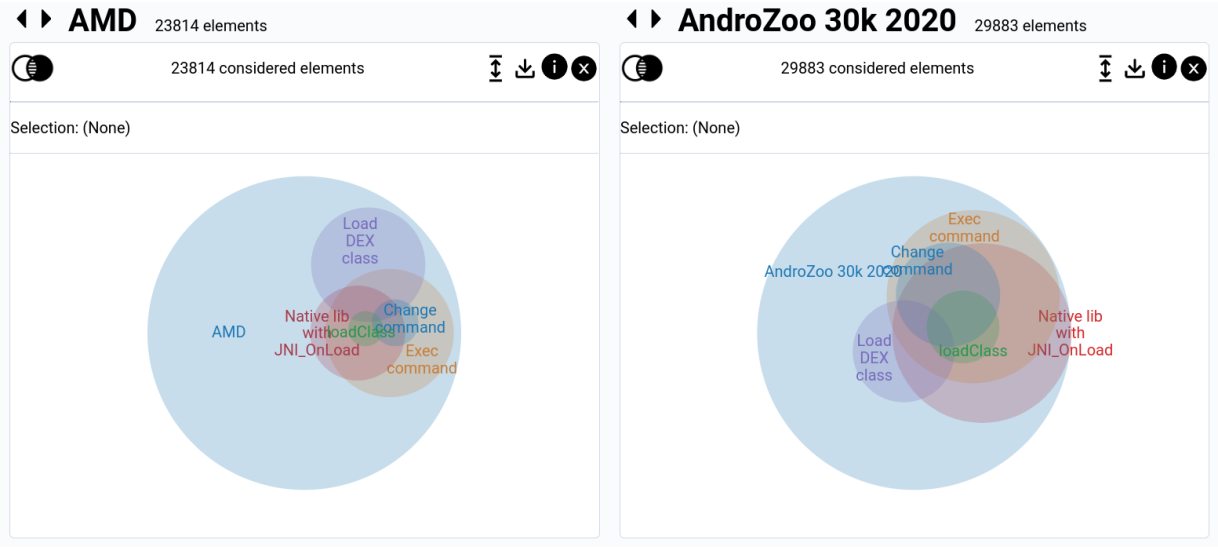

Figure 3.20: Venn diagrams showing the relationships between "Native lib with JNI_OnLoad", "Change command", "loadClass", "Exec command", "Load DEX class" in AMD and a 30 000 sample of AndroZoo in 2020

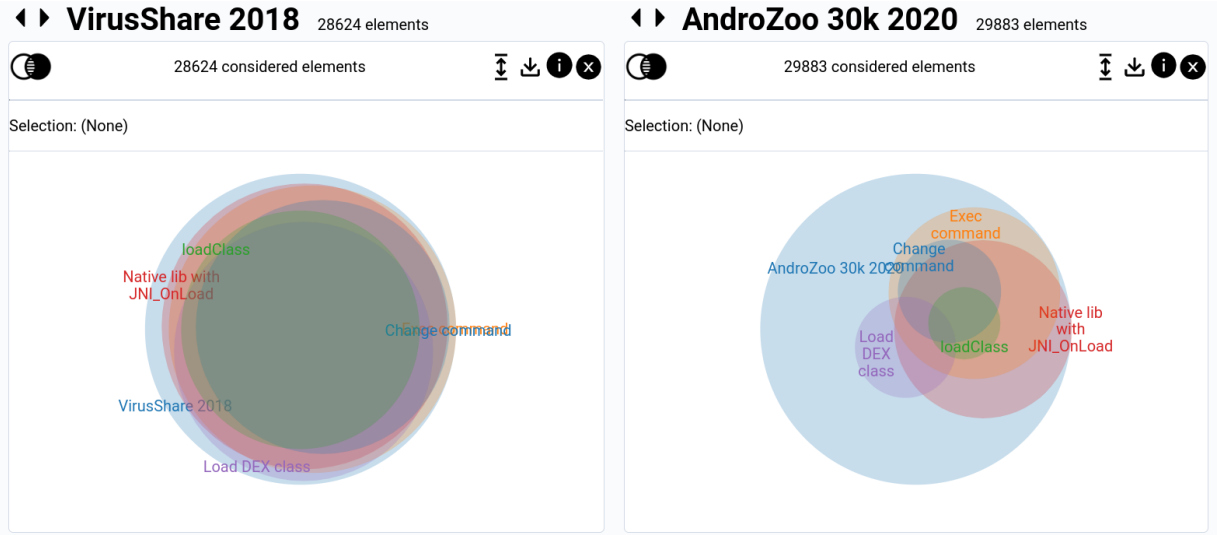

Figure 3.21: Venn diagrams showing the relationships between "Native lib with JNI_OnLoad", "Change command", "loadClass", "Exec command", "Load DEX class" in VirusShare 2018 and a 30 000 sample of AndroZoo in 2020

Figure 3.22: Top eight characteristics with the most proportion difference between VirusShare 2018 and a 30 000 sample of AndroZoo in 2020

3.4 Conclusion

In this chapter we presented DaViz, a new dataset visualization tool to explore and compare Android malware datasets. It uses different techniques such as filtering and reordering, that users can take advantage of to gain an insight of malware datasets. We presented the usage of Venn diagrams, bar charts, and heat maps to represent multiple values in different ways. We took advantage of interactivity to allow the user to perform two main analysis tasks: dataset exploration and comparison of datasets.

Then, we presented a comparison of Drebin and AMD, two old and recognized malware datasets from the literature, and a more recent malware dataset VirusShare 2018. We also presented a comparison of Drebin and AMD against an extract of AndroZoo. We found that the samples in the malware datasets are very different from AndroZoo. This is an expected result, because malware have, in general, a very different behavior than an average application found in markets. We also found that the malware datasets we inspected are different between each other. This confirms the works of Allix *et al.* [10] and Pendlebury *et al.* [11] over other characteristics than just the date.

DaViz gives an intuitive and visual difference between all these datasets. This tool has been presented at the RESSI 2022 conference [12]. In the next chapter, we introduce a statistical procedure in order to qualify formally the differences we witnessed intuitevely with DaViz.

[12]: Concepción Miranda et al. (2022), 'DaViz: Visualization for Android Malware Datasets'

Representativity in Experimental Datasets

4

Before performing experiments, researchers set the purpose of the study, *i.e.* malware detection, family classification or a market study, and the set of applications that the experiments are aimed for. They then create their experimental datasets by either using another one found in other experiments, or creating their own by gathering applications from various sources. We have seen that such a process can lead to irregularities in the dataset. These irregularities can be observed graphically as presented in Chapter 3. The term irregular suggests that we express a comparison against a real production environment, *e.g.* Google Play, called the "target" dataset. The expected consequence is that, if an experiment obtains good results with a regular dataset, then it should have the same good results on the larger dataset, that we call in this thesis the target dataset. Because of this, for the rest of this thesis, we introduce the notion of representativity between two datasets. An experimental dataset is said to be *representative* if it comes from this larger target dataset, and if it has similar proportions of characteristics compared to this larger target dataset.

In this section, we will define the notion of being *representative*: how we define representativity, how to sample such a dataset, and how we can measure a difference between datasets. In section 4.1, we start by defining what is a representative dataset with the new concept of δ-representativity. Then, in section 4.2, we establish a method to create a representative dataset from a target set of applications. Later, we propose the usage of a statistical test to calculate if two datasets are statistically homogeneous to each other. And lastly, in section 4.3, we evaluate various datasets from the literature and compare them to a representative dataset of our choice.

4.1 Definition of a representative dataset

In statistics, population is a set of objects from which a study wants to be conducted [154]. In most science studies, such as medicine, psychology and biology, researchers pick a set of individuals from a defined population to study. In the case of research in Android applications, the population would be the set of programs from a defined source or sources. For example, a study of a new Android malware detection algorithm claims its solution works properly in Google Play. To verify this assumption, the researchers must test their new method with applications from Google Play.

[154]: Cochran (1977), *Sampling Techniques*

Instead of performing a census of the population, which means to study all the individuals of a population, it is convenient to obtain a sample from the population. A sample, then, would be an subset of programs from a specified population. From here on, we refer to a sample as a "dataset", since it satisfies the characteristics of one as seen in the previous chapter. Obtaining a sample from the population have two main advantages:

1. Performing a census could be infeasible, as it would require too much time or resources that the researchers do not have. By taking a sample, it significantly reduces the number of individuals to study.
2. If the sample is representative of the population, the results from the experiment using this sample would be valid for the population.

In the case of evaluating a detection algorithm against applications from Google Play, the researchers then must get a sample from this market to perform their tests.

To formalize the definition a representative dataset, we consider a population $\mathcal{P}$ that contains N applications. Then, a dataset $\mathbf{D}$ of size n is representative of $\mathcal{P}$ if it satisfies the following properties:

1. n is significantly smaller than N
2. evaluating the proportion $\hat{p}$ of a specific characteristic on $\mathbf{D}$ gives similar results as evaluation p on $\mathcal{P}$ for the same characteristic
3. evaluating an algorithm on $\mathbf{D}$ gives similar results as performing the same evaluation on $\mathcal{P}$

The individuals that compose a sample have characteristics inherent to them, and are of interest for a study. A characteristic is a property of an individual. In the case of Android applications, a characteristic can be properties such as the file size (evaluated as a integer), the number of classes in the code (an integer), or whether the applications contain a certain library (a boolean).

For a single characteristic, we could consider that some dataset $\mathbf{D}$ is representative of a population $\mathcal{P}$ for a single characteristic if the proportion of applications exhibiting is the same in $\mathbf{D}$ and $\mathcal{P}$, (*i.e.* $p = \hat{p}$). Despite of this, there are cases where this is not possible: if $p = 1/N$, meaning that only one element in $\mathcal{P}$ has this characteristic, then, in order to have exactly the same proportion in $\mathbf{D}$, the whole population have to be sampled. This would result in a contradiction of the first property about representative datasets. Therefore, a margin of error δ must be used, which is a difference between $\hat{p}$ and p one allows to have considering that $\mathbf{D}$ is representative of $\mathcal{P}$. And so, a dataset is representative of a population if it has the same proportion for a given characteristic up to a maximum difference of δ.

The way δ is considered for a single characteristic can be replicated for multiple characteristics. Let us consider a set of d characteristics $\{c_1, ..., c_d\}$ that can take the values $v_1, ..., v_d$ respectively. This generates $v_1 \times ... \times v_d$ different combinations. In the following, these combinations will be considered as *classes*, and $\mathcal{K}$ as the set of these classes. A class, then, is the set of applications that share the same values for the same set of characteristics. This term is not related to malware family names, which correspond to tags given to a set of malware that have the same behavior [155]. Then, a dataset $\mathbf{D}$ could be considered representative of $\mathcal{P}$ for a set of classes $\mathcal{K}$ if the proportion of each class $k \in \mathcal{K}$ is equal to $\mathcal{P}$ and $\mathbf{D}$.

Definition 4.1.1 *Let $0 \leq \delta \leq 1$. A dataset $\boldsymbol{D}$ is δ-representative of a population $\mathcal{P}$ for a finite set of classes $\mathcal{K}$ if $\forall k \in \mathcal{K}, |p_{\boldsymbol{D},k} - p_{\mathcal{P},k}| \leq \delta$, where $p_{\boldsymbol{D},k}$ (resp. $p_{\mathcal{P},k}$) is the proportion of class k in $\boldsymbol{D}$ (resp. $\mathcal{P}$).*

4.2 Sampling representative datasets

Once we have a definition for a representative dataset, we can talk about how to sample one from a target dataset. We take again the example of malware classification, where researchers have to take a sample from Google Play for doing the evaluation of their method. However, Google Play prevents user to crawl the platform to obtain information about a large quantity of applications, and so it is very difficult to obtain a complete snapshot of Google Play to get a representative sample from it. As a substitute, for the rest of this thesis, we use AndroZoo as a the population or our target dataset. As we explain in Chapter 2 (Section 2.1.3), this dataset is the largest dataset available with 19 553 370 (as of June 14, 2022), it is regularly updated with new applications from Google Play and other markets, and it keeps the applications they collect. Because of this, AndroZoo is the best option available to represent Google Play. This does not makes the assumptions of this work less relevant, as the methods proposed works beyond any specific dataset.

To obtain a representative sample of $\mathcal{P}$ of size n, a *simple random sampling* is performed [154]. This technique consist in drawing n units out of the N without replacement. Following definition 4.1.1, we fix a maximum difference δ that we accept. This will precondition n to a minimum sample size in order to get an error less or equal than δ. In statistics, studies quantify the probability of obtaining an error smaller than δ. This is denoted as a *confident level* C. For example, "with a confident level $C = 95\%$, the error is smaller than δ". Such estimations are performed because of the random sampling, as it may yield to differences in proportions bigger than δ just by chance. To minimize this, a minimum n is choose based on δ.

4.2.1 Sample size for one boolean characteristic

Once the maximum error δ and the confidence level C is are defined in accordance to the experiment's requirements, a lower bound for n can be calculated. The margin of error formula [156] with finite population correction [154] is used for this:

[156]: De Veaux et al. (2016), *Stats: Data and Models*

$$\delta \leq z(C)\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\sqrt{\frac{N-n}{N-1}}$$

This formula is used to calculate the margin of error δ given the sample size n, the population size N, the sample proportion $\hat{p}$ and the standard deviation $z(C)$ at confident level C for the standard normal distribution with mean 0 and standard deviation 1. The term $\hat{p}(1-\hat{p})$ is maximized when $\hat{p} = 0.5$, so we can bound this value by $0.5(1-0.5) = \frac{1}{4}$, namely:

Table 4.1: Sample size of representative datasets of Google Play and AndroZoo for one or multiple characteristics (δ = 0.01 and C = 99%).

Dataset	Size	Sample size for one characteristic (δ = 0.01 and C = 99%)	Sample size for 36 864 characteristics (δ = 0.015 and C = 99%)
Google Play	2 664 893	16 485	29 062
AndroZoo	19 553 370	16 574	29 383

$$\delta \leq z(C)\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\sqrt{\frac{N-n}{N-1}} \leq z(C)\sqrt{\frac{N-n}{4n(N-1)}}$$

Since we already know δ, we can rearrange the inequality to bring n to one side of the equation:

$$n \geq N\left(\frac{4(N-1)\delta^2}{z(C)^2}+1\right)^{-1} \tag{4.1}$$

By using this formula, there are some light assumptions about the sample and population. First, we assume that n is not to small and that it has at least 30 elements [157]. Secondly, that the population proportion p is not close to 0 or 1. In the case these assumptions fail, other confidence interval formulas can be used, like the Hoeffding inequality [158].

[157]: Hogg et al. (2015), *Probability and Statistical Inference*

[158]: Bardenet et al. (2015), 'Concentration inequalities for sampling without replacement'

Notice that the population size N influence the sample size n in the formula. Despite of this, if N is relatively large, this influence is negligible. For example, the size of Google Play is estimated to be 2 664 893 applications as of June 06, 2022. For a representative sample of Google Play with δ = 0.01 and C = 99%, the minimum required sample size is n = 16 485, or only 0.62% of the total population. Surprisingly, for a representative dataset for AndroZoo, using the same δ and C, its size n = 16 574, or 0.08%, a difference of 89 applications compared to the size of a representative sample of Google Play. Table 4.1 shows a summary of the sample size of a representative dataset for Google Play and AndroZoo respectively, for one or multiple characteristics. Next, we explain how to calculate the sample size of a representative dataset for multiple characteristics.

4.2.2 Sample size for multiple characteristics

The previous formula for calculating the minimum size for a representative dataset is only valid for a single boolean characteristic. This only applies for $|\mathcal{K}| = 2$ classes, as a single boolean characteristic only has two possible values. When more than one boolean or non-boolean characteristics are considered, the number of classes increase, and so increases the sample's size, as more combinations have to be taken into account for the sample to be representative.

Just as in the case of a single boolean characteristic, the margin of error formula can be used with a slight modification. According to the Bonferroni correction [159], representing a set of classes $\mathcal{K}$ with a confidence level C using random sampling is more likely than correctly

[159]: Dunn (1961), 'Multiple Comparisons among Means'

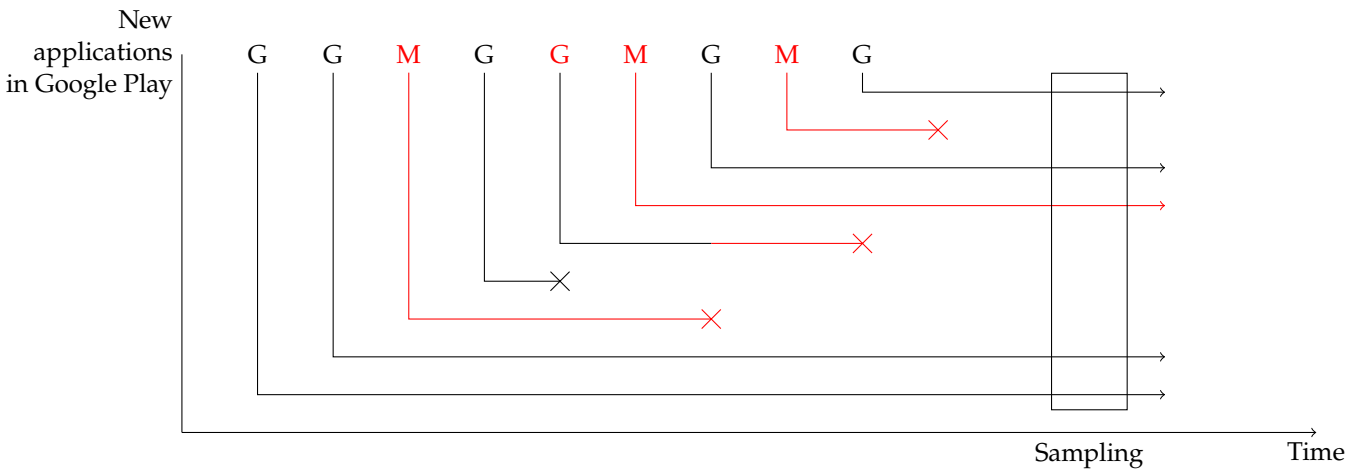

Figure 4.1: Diagram of the evolution of Google Play and the sampling of a representative dataset. From Google's point of view, there are four malware samples to detect, while the sample taken on the right would only have one malware to detect.

representing one boolean characteristic with a confidence level $1-(1-C)/(|\mathcal{K}|-1)$. This correction implies the modification of the standard deviation $z(C)$ in Equation 4.1 to $z(1-\frac{1-C}{|\mathcal{K}|-1})$. And so, the size of a representative dataset $\mathbf{D}$ of $\mathcal{P}$ with $|\mathcal{K}|$ classes with a margin of error δ and a confident level of Cneed to be at least:

$$n \geq N\left(\frac{4(N-1)\delta^2}{z\left(1-\frac{1-C}{|\mathcal{K}|-1}\right)^2}+1\right)^{-1} \tag{4.2}$$

Considering a set of classes with $|\mathcal{K}| = 36\,864$ (the set of classes of the considered characteristics as seen in Section 4.3.1), a margin of error $\delta = 0.015$, and a confident level $C = 99\%$, the size of a representative sample of Google Play would be at least $n = 29\,062$, the size of one for AndroZoo $n = 29\,339$, and for an arbitrarily large population $n = 29\,383$.

4.2.3 Application to security experiments

Sampling applications from AndoZoo with the required value of n gives a representative dataset for a chosen set of multiple characteristics. Unfortunately, two problems arise that limits this approach.

Problem 1: some characteristics may be hard-to-compute characteristics We define as hard-to-compute, a characteristic that the researcher cannot compute with reliability, for example being a malware or not, containing a packer, etc. These characteristics may be missing from the population (for example Google Play) and recomputing them would require a manual analysis of the sampled applications.

Problem 2: underrepresentation of samples with the hard-to-compute characteristic Even if we suppose that the hard-to-compute characteristic could be calculated for the samples found up to now, sampling

applications from a population would lead to new problems for the produced dataset. Indeed, the previous equation that have been presented in this chapter guaranty that the characteristics are homogeneous with the population for the given time. Let us imagine that a detection experiment is performed to Google Play at Google. The researcher's goal would be to create a dataset that resembles Google Play with the same malware/goodware proportion Google is facing, *i.e.*, considering all the applications added to Google Play throughout all 10+ years this platform have been available. We represented this scenario in Figure 4.1. We see that, during this time:

- Goodware and malware are added to the store
- Applications continue to stay over the years
- Goodware are removed
- Malware may be detected and removed

A researcher using our sampling method would obtain a representative dataset of Google Play *at that time*. We see that, for Google, the detection experiment would be to detect four malware out of nine applications, while for a researcher, the detection experiment would be to only detect one malware out of five applications. As a consequence, the drawn dataset would contain malware, but underrepresented compared to the reality of the detection problem. These two problems, a detection experiment at Google and a detection experiment for a researchers with representative dataset of Google Play, are indeed different.

Consequently, for these two reasons, **we cannot push further a method based on simple random sampling of applications**. As a fallback, we will switch to methods that consist in mixing applications that have different values for the hard-to-compute characteristics. For the malware/goodware example, it consists in mixing malware samples and goodware samples. The resulting dataset may be very heterogeneous from the population. In the next section, we propose a method, using a statistical test, to judge whether two datasets are different.

4.3 Evaluating existing datasets

In this section, our aim is to measure if a dataset **D** is statistically different from the population. This test will be based on the considered characteristics of applications. Before moving on the precise definition of the test, we need to discuss what happens if we include in our test all the possible characteristics of applications.

We recall the reader that our goal is to compare mixed datasets containing malware and goodware with the population. If the population is Google Play, there is little chance that our test answers that a mixed dataset is statistically the same as the population. Indeed, a lot of characteristics related to malware behavior would be different for the malware part of the dataset. On the contrary, the goodware part could be very similar to an extract of Google Play. These remarks hold only if Google Play does not contain too much malware, which is a reasonable hypothesis. As a result, comparing a set of malware with an extract of the Google Play is useless: there is a high chance that the characteristics are very different. As our mixed dataset of goodware/malware should have a significant

amount of malware, comparing a mixed dataset to an extract of Google Play will give the same results: they will be different.

It comes from the previous remarks that we should more precisely discuss the considered characteristics. We can split the characteristics into two groups:

- Characteristics related to security: permissions, sensitive APIs, native libraries, etc.
- Characteristics not related to security: size, date, number of classes, etc.

The group of security related characteristics should be obviously different for malware compared to the population. If they are not, it means that malware are disguised perfectly as goodware and will not be spotted by any detection algorithm; this hypothesis is very uncertain, and we expect a difference. The group of non-security related characteristics should be similar to the population. For example, the date has no reason to be different in a malware dataset, in term of distribution, than in the population.

As a consequence, we propose to build a test of homogeneity considering the characteristics **not related to security**, letting other characteristics ignored because they are not homogeneous. Such a test would help to compare a mixed dataset to the population. Our goal is to compare the datasets of the literature, such as Drebin and AMD to the population. Later, in Chapter 5, our debiasing algorithm that modify datasets will also work on the group on non-security related characteristics, for the same reasons.

4.3.1 Non-security related characteristics to consider

When we choose these non-security related characteristics, it is important to notice that the number of combinations grows exponentially with the number of characteristics, and eventually grow bigger than the size of the dataset. With three boolean characteristics we have 8 classes, with four we have 16 classes, and with five there are 32 classes. As a result, we have to limit the number of these classes, because too much of them will lead to have classes with no application belonging to them. For numerical characteristics, they can be split into intervals so to have a value associated to each interval These intervals can be designed according to the study parameters. For example, one of those characteristics would be the file size, where the values are integers and this range can be split in intervals to designate "small", "medium", or "big" applications.

For this experiment, we decided to use the following characteristics that are *not-security related*:

- APK size (4 values)
    0. 1 MB < size
    1. 1 MB ≤ size ≤ 5 MB
    2. 5 MB ≤ size ≤ 20 MB
    3. size > 20 MB
- year (9 values): {< 2011, 2011-2012, 2012-2013, 2013-2014, 2014-2015, 2015-2016, 2016-2017, 2017-2018, > 2018}

- Internet permission (bool)
- External or internal storage access (bool)
- Uses Google play services (bool)
- Generates UUIDs (bool)
- Vibrate phone permission (bool)
- NFC permission (bool)
- Bluetooth permission (bool)
- Performs HTTP request (bool)
- Uses JSON objects (bool)
- Specifies User-Agent (bool)

These characteristics were extracted using Droidlysis [148] as we also explain in Chapter 3. These characteristics are widely different and easy to evaluate statically, lowering the computational resources needed to extract and use them in the following experiments. Although dynamic characteristics could be used, this requires executing the application. Some applications may not execute in emulated environments [121], nor run on modern versions of the Android OS because of deprecated APIs or will just crash during execution [160, 161].

With these 12 characteristics, there is a total of 36 864 classes: The APK size counts for four possible values for each interval, the year counts for nine values for the reason, and there are 10 boolean characteristics that count for two possible values each, so multiplying $(4 \times 9 \times 2^{10}) = 36\,864$. These characteristics will be used to study the homogeneity of a number of datasets from the literature against an extract of the population, AndroZoo in 2020, with a statistical test that we explain in the next section.

4.3.2 Statistical test of homogeneity for datasets

[162]: Pearson (1900), 'X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling'

In statistics, the χ^2 statistical test for homogeneity [162] is a method to compare two samples and verify that these are *not* statistically homogeneous, *i.e.*, they are not from the same population. To perform this test, we need a dataset $\mathbf{D}$ and a second one $\mathbf{D}_r$, a sample drawn from a population $\mathcal{P}$. Then, we establish a *null hypothesis* [156] (denoted H_0), a conjecture proposing that there is no effect or no difference in the possibilities established in it. In our case, H_0 is the following: *$\boldsymbol{D}$ and $\boldsymbol{D}_r$ are drawn from the same population $\mathcal{P}$ (with replacement)*. Then we apply a formula (explained later in the section) using the proportion of applications of each class in $\mathcal{K}$ for $\mathbf{D}$ and $\mathbf{D}_r$, and we obtain the χ^2 test statistic for these two datasets.

The test result itself does not give a relevant indicator of how different or equal the datasets are. Instead, we look at the p-value of the result after performing the test. In plain terms, the p-value is the probability of obtaining a test result as the one obtained given that H_0 is true. Although it is a rather difficult concept to grasp even for statisticians, it is important to notice that a low p-value represents a *statistically significant* result, meaning that obtaining this result has a low chance of happening given that H_0 is true.

The χ^2 test, as well as other statistical tests such as the Student's t-test or the Z-test, can only reject the null hypothesis but not confirm it.

Rejecting the null hypothesis means that we admit the dataset $\mathbf{D}$ is not homogeneous to $\mathbf{D}_r$, and therefore dissimilar to the population $\mathcal{P}$. To do so, a threshold to which the p-value is low enough to be significant is established. This threshold is typically set to be 0.05, but it can be lower, or higher depending of the experiment.

The test is applied as follows: we consider the finite set of classes $\mathcal{K}$. We denote $m_{\mathbf{D},k}$ (respectively $m_{\mathbf{D}_r,k}$) the number of samples in $\mathbf{D}$ (respectively in $\mathbf{D}_r$) of class $k \in \mathcal{K}$ and $p_{\mathcal{P},k}$ (respectively $p_{\mathbf{D},k}$ and $p_{\mathbf{D}_r,k}$) the proportion of elements of class k in $\mathcal{P}$ (respectively in $\mathbf{D}$, in $\mathbf{D}_r$). If the null hypothesis holds true, then the three proportions $p_{\mathcal{P},k}$, $p_{\mathbf{D},k}$ and $p_{\mathbf{D}_r,k}$ of class $k \in \mathcal{K}$ should be equal or different only by a small margin. The proportion $p_{\mathcal{P},k}$ can thus be estimated by $\frac{m_{\mathbf{D}_r,k}+m_{\mathbf{D},k}}{|\mathbf{D}_r|+|\mathbf{D}|}$ and the expected number of occurrences of class k in $\mathbf{D}$ can be estimated by $E_{\mathbf{D},k,\mathbf{D}_r} = |\mathbf{D}| \times \frac{m_{\mathbf{D}_r,k}+m_{\mathbf{D},k}}{|\mathbf{D}_r|+|\mathbf{D}|}$ and in $\mathbf{D}_r$ by $E_{\mathbf{D}_r,k,\mathbf{D}} = |\mathbf{D}_r| \times \frac{m_{\mathbf{D}_r,k}+m_{\mathbf{D},k}}{|\mathbf{D}_r|+|\mathbf{D}|}$. The χ^2 value of two datasets $\mathbf{D}$ and $\mathbf{D}_r$ is defined as:

$$\chi^2 = \sum_{k \in \mathcal{K}} \frac{(m_{\mathbf{D},k} - E_{\mathbf{D},k,\mathbf{D}_r})^2}{E_{\mathbf{D},k,\mathbf{D}_r}} + \frac{(m_{\mathbf{D}_r,k} - E_{\mathbf{D}_r,k,\mathbf{D}})^2}{E_{\mathbf{D}_r,k,\mathbf{D}}}$$

This formula is related to the χ^2 distribution, that is the generalization of all possible values of this formula for any given number of classes $|\mathcal{K}|$. The p-value can then be calculated using the cumulative distribution function (CDF) of the χ^2 distribution. The CDF is parameterized by the number of *degrees of freedom*, in this case the number of classes minus one, *i.e.* $|\mathcal{K}| - 1$. The p-value can then be looked up in tables that are available in most statistics textbooks and statistics software (for example the Python library `scipy`).

In the following, we use the χ^2 test to measure the difference between various Android malware datasets of the literature and a representative dataset of AndroZoo in 2020.

4.3.3 Studied Android datasets

To perform this test, several Android datasets from the literature were taken and analyzed with Droidlysis. These datasets are the following:

- 6 **malware datasets: Drebin** [5], **AMD** [6], as well as **VS 2015**, **2016**, **2017** and **2018**, containing all the malware collected by VirusShare for each of these years;
- 2 **groups of datasets extracted from AndroZoo [61]:**
  - $\mathbf{AZ19_{100k}}$, 100 000 applications randomly drawn in 2019, and the subsets $\mathbf{AZ19_{100k}}$ in **2015**, **2016**, **2017**, **2018** restricted to applications from each year;
  - $\mathbf{AZ20_{10k}}$, $\mathbf{AZ20_{20k}}$, and $\mathbf{AZ20_{30k}}$ three datasets randomly drawn from AndroZoo in 2020, containing respectively 10 000, 20 000 and 30 000 applications.
- 1 **mix dataset:** $\mathbf{D_{mix}}$ composed of 5 560 malware from Drebin and 11 120 goodware from 2018 extracted from AndroZoo. $\mathbf{D_{mix}}$ mimics the datasets used in previous machine learning papers [110, 111].
- **DroidBench [163]:** a collection of applications with source available designed to evaluate taint-analysis tools.

[5]: Arp et al. (2014), 'DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket'

[6]: Wei et al. (2017), 'Deep Ground Truth Analysis of Current Android Malware'

[61]: Allix et al. (2016), 'AndroZoo: Collecting Millions of Android Apps for the Research Community'

[110]: Xu et al. (2016), 'ICCDetector: ICC-Based Malware Detection on Android'

[111]: Feizollah et al. (2017), 'AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection'

[163]: Arzt et al. (2014), 'FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps'

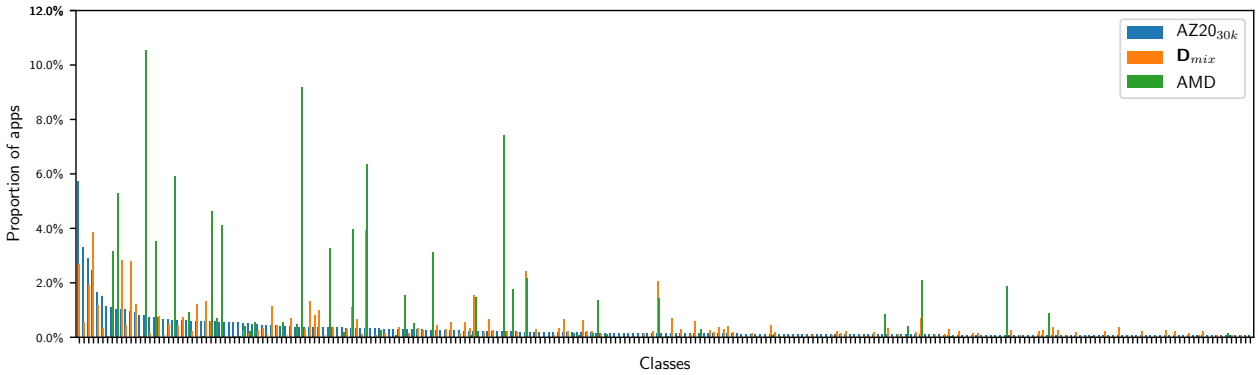

Figure 4.2: Distribution of application classes for the top most 200 over 2 874 classes of AZ20$_{30k}$ and compared with $\mathbf{D}_{mix}$ and AMD

As proposed earlier in the chapter, we use AndroZoo as the population because it is one of the easiest source of Android applications that can be used to draw a large sample. The goal is to determine which datasets commonly used in the literature are not representative of AndroZoo. This dataset contains 19 553 370 applications as of June 15, 2022. Due to its size, it would be convenient to extract a representative sample as we explained before. Hence, we represent AndroZoo through AZ20$_{30k}$, a dataset of 30 000 randomly selected applications (corresponding to $\delta = 0.015$ and $C = 99\%$ for all 36 864 theoretical classes, see Section 4.2.2).

We perform a comparison between an extract of the population, in this case AndroZoo, called AZ20$_{30k}$ with some datasets used in the literature (presented in Chapter 2). To illustrate the difference in class proportions between some of the datasets, Figure 4.2 shows the distribution of the 200 highest class proportions between AZ20$_{30k}$ (in blue), the mix dataset $\mathbf{D}_{mix}$ (in orange) and AMD (in green). The classes are sorted in descending order of size for AZ20$_{30k}$. The figure already shows the differences between these dataset. We can see that AMD has many classes with high proportions, between 4 % up to 11 %, than AZ20$_{30k}$ and $\mathbf{D}_{mix}$. On the other hand, $\mathbf{D}_{mix}$ has more classes with slightly higher proportions distributed in the center and the tail of the figure, but still different than AZ20$_{30k}$.

A further comparison is done by performing the χ^2 test for homogeneity. Table 4.2 presents the dataset size, the number of classes represented only in AZ20$_{30k}$ (and not in the studied dataset), the number of classes common to AZ20$_{30k}$ and the studied dataset, the number of classes represented only in the studied dataset (and not in AZ20$_{30k}$), the maximum and average δ of class proportions, the value of the χ^2 test, the associated p-value, and the conclusion (or lack of) of the test between the studied dataset and AZ20$_{30k}$. Right away, last column shows that all datasets, except for the last extract of AndroZoo, have a p-value lower than the threshold. For AZ20$_{20k}$, this is consistent with the definition of representativity of Section 4.1.1, as we can see a low δ and a high p-value where the test does not reject the null hypothesis of *AZ20$_{20k}$ and AZ20$_{30k}$ are drawn from the same population*. We also notice that AZ20$_{10k}$, even thought is a sample of AndroZoo, has a p-value that rejects the null hypothesis. This is mainly due to its size: AZ20$_{10k}$'s size of 9 971 applications is lower compared to AZ20$_{20k}$ which has the double. This affects the proportions for each class: since there are less applications to

Table 4.2: χ^2 test for homogeneity over $AZ20_{30k}$ for a finite set of non-security characteristics

Datasets	Dataset size	# of classes in common	# of classes only in the dataset	# of classes only in $AZ20_{30k}$	Max δ	Avg δ	χ^2value	p-value	Conclusion
$AZ20_{30k}$	29 974	2 411	–	–	0	0	0	1	
Drebin	5 304	181	141	2 230	0.1184	0.0007	27 902	0	Reject (C=99%)
AMD	23 258	128	16	2 283	0.0974	0.0007	39 771	0	Reject (C=99%)
VirusShare 2015	28 896	250	65	2 161	0.1683	0.0007	44 537	0	Reject (C=99%)
VirusShare 2016	12 651	172	28	2 239	0.1595	0.0007	30 312	0	Reject (C=99%)
VirusShare 2017	9 945	234	39	2 177	0.0898	0.0006	22 050	0	Reject (C=99%)
VirusShare 2018	28 543	728	328	1 683	0.5057	0.0006	42 617	0	Reject (C=99%)
$\mathbf{D}_{mix}$	16 424	982	447	1 429	0.0359	0.0004	17 852	0	Reject (C=99%)
$AZ19_{100k}$	99 999	2 086	2 062	325	0.0320	0.0001	12 662	0	Reject (C=99%)
DroidBench	119	6	0	2 405	0.4773	0.0008	14 121	0	Reject (C=99%)
$AZ19_{100k}$ 2015	5 794	648	291	1 763	0.0574	0.0005	15 390	0	Reject (C=99%)
$AZ19_{100k}$ 2016	27 516	1 183	921	1 228	0.0573	0.0004	30 725	0	Reject (C=99%)
$AZ19_{100k}$ 2017	6 524	688	274	1 723	0.0381	0.0004	12 378	0	Reject (C=99%)
$AZ19_{100k}$ 2018	24 549	1 216	547	1 195	0.0285	0.0003	14 754	0	Reject (C=99%)
$AZ20_{10k}$	9 971	1 370	423	1 041	0.0443	0.0003	6 296	0	Reject (C=99%)
$AZ20_{20k}$	19 927	1 361	397	1 050	0.0245	0.0001	3 560	1	*Non-conclusive*

draw, proportions are prone to be overrepresented or underrepresented to a more larger extend. We can see this in the $AZ20_{10k}$'s maximum δ for any class of 0.044 3 while for $AZ20_{10k}$ this δ is 0.024 5, almost half of $AZ20_{10k}$'s.

4.4 Conclusion

In this chapter, the notion of dataset representativity of a population was established using the definition of δ-representativity. We presented how to sample a representative dataset using this notion, for either one or more characteristics. Then, we presented a comparison between a representative dataset of AndroZoo, extracted using the techniques established in this chapter, and other Android malware datasets of the literature.

It is clear that none of these datasets are representative of AndroZoo, our studied population, and only samples of this are. Thus, only by using a representative sample of AndroZoo, a study can claim that its results works for this population. However, samples from AndroZoo, or for most other markets, tend to lack the characteristics the studies use for their experiments. These characteristics are usually hard-to-compute, like, in the case of malware detection, *being a malware*. Note that we know that AndroZoo contain information about being a malware, which comes from VirusTotal. But this information cannot really be considered as

ground truth, as VirusTotal contains false negatives and false positives. As a result, representative drawn samples cannot be used directly to perform tests.

To solve this problem, we propose to design a method to push a non-representative labeled dataset to resemble a representative non-labeled dataset. This contribution is explained in the next chapter.

Debiasing Android Datasets

5

As we saw the differences between datasets, a question arises from these analyses: what are the consequences of using one dataset or another? Up to now, we explained how to sample a representative dataset from a population, and how to tell if a dataset is not representative of a population. In addition to this, in order to perform a malware detection evaluation, the dataset needs to be labeled, *i.e.* each sample have the metadata indicating if it is a malware or not. In contrast, a representative dataset is usually *unlabeled* because the characteristics have not been calculated yet, in particular the label being a malware or not is very difficult to obtain. However, the datasets in the literature are valuable for the reason of being *labeled* because they contain applications for which the characteristic of interest is already calculated. We would like to use these labeled datasets of the literature, while at the same time being representative of a population. We call such a generated dataset a *debiased* one, according to the population.

In this chapter, we propose an algorithm to obtain a representative and labeled dataset from a *representative* and unlabeled dateset, and a *labeled* dataset. In Section 5.1, we will start by proposing a new method for this goal, consisting of two algorithms. Then in Section 5.2, we present the generated debiased datasets we created from various datasets of the literature. Lastly, in Section 5.3, we use these debiased datasets in machine learning experiments, and see that they outperform old and new datasets from the literature: the malware detection becomes more difficult.

5.1 Debiasing algorithm

In this section we propose an algorithm to obtain a representative and labeled dataset from a *representative* and unlabeled dateset, and a *labeled* dataset.

Let us define the following datasets:

- a *labeled* dataset **B** that is of our interest because of the labels it contains
- a representative *unlabeled* dataset **T** of a population $\mathcal{P}$
- and a large *labeled* dataset **S**.

We propose to design an algorithm to generate a *representative* dataset **G** that resembles **T**, and that is also *labeled*. To do this, the algorithm will add or remove applications from **G**, initially a copy of **B**, until the δ between **G** and **T** is less or equal to a predefined threshold.

We will be using the δ-representativity, defined in Chapter 4, for measuring the similarity between two datasets. The margin of error δ will be assigned before executing the algorithm. We could have used the χ^2 test that, as seen in the previous chapter, is used to tell if a dataset is not homogeneous to other, but it cannot be used to tell if the datasets are

similar. In addition to this, the test results have no particular meaning on itself, and the p-value is cumbersome to interpret, while the notion of δ-representativity is easier to grasp. Therefore, this new generated dataset **G** will be δ-representative of **T** and also labeled, so that it can be used as a test set to evaluate an algorithm or a machine learning model on $\mathcal{P}$.

To implement the previous idea, we will examine two algorithms. The first one is a *constant-size* algorithm, where it keeps the size of the initial dataset **B** . The second algorithm called *optimal debiasing* will optimize the result dataset **G** by allowing the modification of the initial dataset's size.

5.1.1 Recall of characteristics and classes

As seen in Chapter 4, a *characteristic* is a property of an element in a dataset. We represent applications using a set of boolean characteristics. We call a *class* k a combination of specific values for each characteristic of this set of characteristics. For example, if we use two boolean characteristics $A = \{c_1, c_2\}$, each taking two values, we have the following four classes:

- $k_1 = \{0, 0\}$
- $k_2 = \{0, 1\}$
- $k_3 = \{1, 0\}$
- $k_4 = \{1, 1\}$

Applications sharing the same values for each characteristic in the set of characteristics are grouped in the same class. Finally, $\mathcal{K}$ is the set of all classes possible with this set of characteristics. In the previous example, $\mathcal{K} = \{k_1, k_2, k_3, k_4\}$ for the set A.

5.1.2 Constant-size algorithm

Algorithm 1 produces a dataset **G** whose size is the same as its input **B**. For this, the algorithm take as input an initial dataset **B**, a representative dataset **T** of a population $\mathcal{P}$, a large labeled dataset **S** and δ. This algorithm consists in adding elements from the source **S** to the most underrepresented classes of **B** and to remove elements from **S** for the most overrepresented class. In this case, a class k is *underrepresented* in **G** with respect of **T** if $p_{\mathbf{G},k} - p_{\mathbf{T},k} > \delta$, and is *overrepresented* in **G** with respect of **T** if $p_{\mathbf{G},k} - p_{\mathbf{T},k} < -\delta$. Each time an element is added, another one is removed in order to keep the dataset's size constant. With this last constraint, the debiasing process may not be possible for one of following reasons: either no element can be added to an underrepresented class (because the source does not contain elements of this class); or it is not possible to obtain a proportion that lies in $[p_{\mathbf{T},k} - \delta; p_{\mathbf{T},k} + \delta]$ for some class k because δ is too small.

Algorithm 1 is optimal in terms of number of modifications.* If a class k_u is underrepresented in **B**, elements are added to k_u to increase its proportion and make it correctly represented. If a_{k_u} elements are added to this class while the size of **D** remains constant (i.e. $|\mathbf{D}| = |\mathbf{B}|$), the

* Proofs are available in the appendix.

Algorithm 1: Constant size debiasing algorithm

Input: $\mathbf{B}$, $\mathbf{T}$, $\mathbf{S}$, δ
Output: Generated dataset $\mathbf{G}$

```
1  S' ← S; G ← B
2  while max_{k∈K} |p_{G,k} − p_{T,k}| > δ do
3      k_h ← arg max_{k∈K}(p_{G,k} − p_{T,k})      ▷ most overrepresented class
4      k_l ← arg min_{k∈K}(p_{G,k} − p_{T,k})      ▷ most underrepresented class
5      if |S'(k_l)| = 0 then
6          K_S = {k ∈ K | |S'(k)| > 0}
7          if p_{G,k_l} ≥ p_{T,k_l} − δ and K_S ≠ ∅ then
8              k_l ← arg min_{k∈K_S} p_{G,k} − p_{T,k}
9          else return "Impossible";
10     r ← one element of S'(k_l) chosen at random
11     Remove r from S' and add it to G
12     Remove one element of G(k_h) chosen at random
13     if p_{G,k_h} < p_{T,k_h} − δ or p_{G,k_l} > p_{T,k_l} + δ then
14         return "Impossible"
15 return G
```

proportion of class k in $\mathbf{D}$ is $p_{\mathbf{B},k} + \frac{a_k}{|\mathbf{B}|}$. A class is not underrepresented if its proportion is greater than $p_{\mathbf{T},k} - \delta$, so we are looking for the smallest a_k such that $p_{\mathbf{B},k} + \frac{a_k}{|\mathbf{B}|} \geq p_{\mathbf{T},k} - \delta$, hence $a_k \geq |\mathbf{B}|(p_{\mathbf{T},k} - \delta - p_{\mathbf{B},k})$. By definition of the ceil function $\lceil\cdot\rceil$, $\lceil|\mathbf{B}|(p_{\mathbf{T},k} - \delta - p_{\mathbf{B},k})\rceil$ is the lowest integer that verifies this inequality. So the minimal number of addition for an underrepresented class k is $\lceil|\mathbf{B}|(p_{\mathbf{T},k} - \delta - p_{\mathbf{B},k})\rceil$. If this number is negative, it means that the class is not underrepresented: in that case, no addition should be made. We denote, for any class k, the minimal number of addition as $n^-_{\mathbf{B}}(k) = \max(0, \lceil|\mathbf{B}|(p_{\mathbf{T},k} - \delta - p_{\mathbf{B},k})\rceil)$. By the same reasoning, we define the minimal number of deletions as $n^+_{\mathbf{B}}(k) = \max(0, \lceil|\mathbf{B}|(p_{\mathbf{B},k} - p_{\mathbf{T},k} - \delta)\rceil)$. In fact, we prove that Algorithm 1 makes exactly $2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$ modifications between its input $\mathbf{B}$ and its output $\mathbf{G}$.

5.1.3 Optimal debiasing algorithm

Algorithm 1 can lead to suboptimal solutions. Algorithm 2 produces a dataset $\mathbf{G}$ without size constraint which is representative of $\mathbf{T}$, with minimal modifications to $\mathbf{B}$. This algorithm calls Algorithm 1 multiple times with different dataset sizes and maintains two variables: $\mathbf{G}_{best}$ is the best dataset found so far, *i.e.* the one that minimizes the number of modifications to $\mathbf{B}$; and d_{min} is this distance between $\mathbf{G}_{best}$ and $\mathbf{B}$. The algorithm starts with $\mathbf{G}_{best} = "Impossible"$ and $d_{min} = \infty$ (line 1). First, it calls Algorithm 1 (line 4) with the initial dataset $\mathbf{B}$. If this succeeds, d_{min} is updated with the appropriate number of additions and deletions, and $\mathbf{G}_{best}$ becomes the result of Algorithm 1. Then, in lines 6 to 12, it tries to remove applications from the most overrepresented classes from $\mathbf{B}$. Finally, in lines 14 to 25, it tries to add applications to the most underrepresented classes to $\mathbf{B}$. For each modification to $\mathbf{B}$, Algorithm 1 is called to update d_{min} and $\mathbf{G}_{best}$.

Algorithm 2: Optimal debiasing Algorithm

Input: $\mathbf{B}$, $\mathbf{T}$, $\mathbf{S}$, δ
Output: Generated dataset

$d_{min} \leftarrow \infty$; $\mathbf{G}_{best} \leftarrow$ "Impossible"
if $Algo1(\boldsymbol{B}, \boldsymbol{T}, \boldsymbol{S}, \delta) \neq$ *"Impossible"* **then**
  $d_{min} \leftarrow 2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$
  $\mathbf{G}_{best} \leftarrow Algo1(\mathbf{B}, \mathbf{T}, \mathbf{S}, \delta)$
$\mathbf{G} \leftarrow \mathbf{B}$
for *i from* 1 *to* $d_{min} - 1$ **do** ▷ Search smaller datasets
  if $\boldsymbol{G} = \emptyset$ **then break**;
  $k_h \leftarrow \arg\max_{k \in \mathcal{K}}(p_{\mathbf{G},k} - p_{\mathbf{T},k})$
  Remove one random element of class k_h from $\mathbf{G}$
  $d \leftarrow |\,|\mathbf{B}| - |\mathbf{G}|\,| + 2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k))$
  if $d < d_{min}$ *and* $Algo1(\boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}, \delta) \neq$ *"Impossible"* **then**
    $d_{min} \leftarrow d$; $\mathbf{G}_{best} \leftarrow Algo1(\mathbf{G}, \mathbf{T}, \mathbf{S}, \delta)$
$\mathbf{G} \leftarrow \mathbf{B}$
for *i from* 1 *to* $d_{min} - 1$ **do** ▷ Search larger datasets
  $k_l \leftarrow \arg\min_{k \in \mathcal{K}}(p_{\mathbf{G},k} - p_{\mathbf{B},k})$
  if $|\mathbf{S}(k_l)| = 0$ **then**
    $\mathcal{K}_{\mathbf{S}} = \{k \in \mathcal{K} \mid |\mathbf{S}(k)| > 0\}$
    if $p_{\boldsymbol{G},k_l} \geq p_{\boldsymbol{T},k} - \delta$ *and* $\mathcal{K}_{\boldsymbol{S}} \neq \emptyset$ **then**
      $k_l \leftarrow \arg\min_{k \in \mathcal{K}_{\mathbf{S}}} p_{\mathbf{G},k} - p_{\mathbf{T},k}$
    else break;
  $r \leftarrow$ one element of $\mathbf{S}(k_l)$ chosen at random
  Remove r from $\mathbf{S}$ and add it to $\mathbf{G}$
  $d \leftarrow |\,|\mathbf{B}| - |\mathbf{G}|\,| + 2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k))$
  if $d < d_{min}$ *and* $Algo1(\boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}, \delta) \neq$ *"Impossible"* **then**
    $d_{min} \leftarrow d$; $\mathbf{G}_{best} \leftarrow Algo1(\mathbf{G}, \mathbf{T}, \mathbf{S}, \delta)$
return G_{best}

Theorem 5.1.1 *Let* $\boldsymbol{B}$, $\boldsymbol{S}$, $\boldsymbol{T}$ *be three datasets and* $\delta > 0$. *If there exists at least one* δ*-representative dataset of* $\boldsymbol{T}$ *that is composed of elements of* $\boldsymbol{B}$ *and* $\boldsymbol{S}$, *then Algorithm 2 produces such a dataset* $\boldsymbol{G}$ *such that the number of additions and deletions from the initial dataset* $\boldsymbol{B}$ *is minimal. If such a dataset does not exist, it returns "Impossible".*

Its worst-case temporal complexity is $O\left((\frac{1}{\delta} + |\mathbf{B}|)^2|\mathcal{K}|\right)$.

While Algorithm 2 computes the closest dataset to the original one, it may happen that the solution found contains only a few applications from the original dataset. This is an indication that the original dataset was very biased.

If $\mathbf{B}$ has been debiased towards a target dataset $\mathbf{T}$ randomly drawn from a population $\mathcal{P}$, with a tolerated difference of $\delta_{debiasing}$, we can estimate its error with the population $\mathcal{P}$ by taking into account the error $\delta_{sampling}$ and confidence interval C used to derive the size of $\mathbf{T}$ with Eq. (4.2): with confidence level C, the error between the debiased dataset and $\mathcal{P}$ is less than $\delta_{sampling} + \delta_{debiasing}$.

5.2 Generating debiased datasets

In this section, we are interested in producing datasets for experiments concerning Android malware detection. As identified in Chapter 2, three types of experimental scenarios can occur. The first scenario (market profiling) requires only unlabeled representative datasets that can be drawn randomly without requiring debiasing, as seen in Chapter 4. The two other cases, Algorithm evaluation and Machine learning, require labeled datasets with different proportions of malware/goodware samples. Therefore, we propose in Section 5.2.2 to debias datasets containing only malware or only goodware and mix them, in Section 5.2.4, to obtain mixed datasets directly usable for machine learning algorithms.

5.2.1 Characteristics used for debiasing

For our debiasing algorithm, we decided to use the non-security related characteristics as we explain in Chapter 4 Section 4.3.1. We want to homogenize a dataset to a target dataset in such a way that we leave discriminant characteristics for malware detection, *i.e.*, security related ones, intact. Otherwise, if we debias using security related characteristics, we would obtain a dataset with malware indistinguishable from goodware, which would be artificially harder for a detection algorithm to make the difference between goodware and malware. Thus, the debiased dataset should be similar to the target dataset in relation to **non-security related** characteristics.

5.2.2 Debiasing datasets

The following details the use of our algorithm to produce representative datasets and in particular the choice of the base, reference and source datasets. Through these objectives, we discuss three main questions:

1. Is it always possible to debias a strongly biased dataset such as Drebin or VirusShare?
2. How many modifications (deletions/additions) are required?
3. Is a debiased malware dataset stable over time?

Debiasing 100% **goodware datasets** We have randomly drawn a New AndrooZoo Extract (NAZE-18-G) and filtered it by relying on Virus Total to keep only goodware with a date no greater than 2018. This dataset should be close to a representative dataset of the Android app population in 2020. Indeed, by taking as reference dataset $AZ20_{30k}$ and as source dataset $AZ19_{100k}G$, we successfully build a representative goodware dataset when δ reaches 0.001, as shown in Table 5.1. Indeed, for $\delta = 0.04$ the dataset is already homogeneous: no addition/deletion is required, but the p-value is 0. Then, we decrease δ until the p-value becomes close to 1. For $\delta = 0.001$, 19.53% of the resulting dataset are new applications and more than 50% of applications have been removed from the original one.

Table 5.1: Results after debiasing datasets

Base Dataset |B|		Reference |T|	Source |S|	Diff δ	Del	Add	Debiased dataset G |G|	Add ratio	χ^2	p-value	Duration validity
Debiasing Goodware datasets											
NAZE-18-G	→	$AZ20_{30k}$	$AZ19_{100k}$G				$NAZE_{Debiased}$18-G				
11 120		29 974	72 131	0.04	0	0	11 120	0.00	12 589	0	
				0.02	438	50	10 732	0.47	12 058	0	
				0.01	1 026	223	10 317	2.16	11 291	0	
				0.005	1 885	632	9 867	6.41	9 726	0	
				0.002 5	2 685	1 280	9 715	13.18	7 998	0	
				0.001	5 808	1 289	6 601	19.53	5 104	0.999 44	
Debiasing Malware datasets											
Drebin	→	$AZ20_{30k}$	VS 15-18, AMD				$Drebin_{Debiased}$				
5 304		29 974	103 293	0.04	892	81	4 493	1.80	25 781	0	
				0.02	3 634	116	1 786	6.49	20 178	0	
				0.01	4 596	103	811	12.70	15 833	0	
				0.005							
VS 15-18	→	$AZ20_{30k}$	Drebin, AMD, AZ-17-18-M				$VS_{Debiased}$15-18				
80 035		29 974	32 562	0.04	66 475	21	13 581	0.15	16 709	0	
				0.02	75 825	31	4 241	0.73	7 406	0	
				0.01	78 074	43	2 004	2.15	4 349	1	
				0.005	79 063	29	1 001	2.90	915	1	
				0.002 5							
Debiasing the VirusShare dataset over time											
VS 15	→	$AZ19_{100k}$ 15	Drebin				$VS_{Debiased}$15				
28 896		5 792	5 304	0.04	13 877	4	15 023	0.03	29 475	0	0
				0.02	19 832	90	9 154	0.98	21 828	0	0
				0.01							
VS 16	→	$AZ19_{100k}$ 16	Drebin, VS 15				$VS_{Debiased}$16				
12 651		27 516	34 200	0.04	4 549	1	8 103	0.01	23 204	0	0
				0.02							
VS 17	→	$AZ19_{100k}$ 17	Drebin, VS 15-16, AMD				$VS_{Debiased}$17				
9 945		6 524	70 109	0.04	580	0	9 365	0.00	21 073	0	0
				0.02							
VS 18	→	$AZ19_{100k}$ 18	Drebin, VS 15-17, AMD				$VS_{Debiased}$18				
28 543		24 549	80 054	0.04	27 644	7	906	0.77	4 997	0.356 52	1
				0.02	28 189	7	361	1.94	1 809	1	1
				0.01							

Debiasing 100% **malware datasets** We were first interested in investigating whether it is possible to debias a small and old dataset (Drebin) on the one hand and a more recent and larger dataset on the other hand (malware from VirusShare between 2015 and 2018, namely VS 15, VS 16, VS 17, VS 17 and VS 15-18). We want the produced datasets to resemble AndroZoo, using an extract from 2020.

The debiasing results show us that Drebin is strongly biased as we need to add from 1.8% to 12.7% of new samples, because a lot of classes of the population are underrepresented or missing in Drebin. Even with these modifications, the resulting dataset, $Drebin_{Debiased}$, is not homogeneous to the population. On the contrary, VirusShare can be debiased with few additions (less than 2.9%) but requires removing more than 90% of the dataset. In that case, the resulting dataset $VS_{Debiased}$15-18, obtains a p-value of 1. We conclude that the VirusShare dataset already contains enough material to be representative of AndroZoo for certain classes but that a lot of these classes are overrepresented.

As seen in the results, Algorithm 2 can generate a new dataset statistically indistinguishable from the target dataset with a margin of error δ. However, the algorithm may fail (see grey cells in Table 5.1) when some classes are underrepresented in the source, as discussed in Section 5.1.2. This limitation can be overcome by providing a large and diverse enough

source dataset.

On the other hand, when the algorithm succeeds, the generated datasets' size tends to be much smaller than the base dataset. This is due to a lot of overrepresented classes, from which the algorithm removes elements at line 9. This is especially the case for historical malware datasets.

The number of classes to consider may also limit the debiasing: intuitively, more characteristics would help to resemble the target dataset better, but it spreads the elements into more classes and makes it more difficult to find elements for certain classes.

5.2.3 Visual exploration of debiased datasets

Comparing using selected characteristics In Figure 5.1, we compare VS15-18 to $AZ20_{30k}$ and see how much they differ. We see that there are very few applications with Google Play services in VS15-18, while there are more than 60% of applications with this characteristic in the AndroZoo sample. In contrast, we see, in Figure 5.2, when we compare $VS_{Debiased}$15-18 to $AZ20_{30k}$, Google Play services has increased, while the proportion of the other characteristics have not changed considerably.

Comparing using the seven characteristics with the most proportion difference This is the worst case visual comparison, *i.e.*, other characteristics are more similar than the present ones. We hide all other characteristics that are similar and we show the most similar ones. In Figure 5.3, when we compare these VS15-18 and $AZ20_{30k}$, we see right away in the first column "Google Play services" as the characteristic with the most proportion difference between them. Next, we see most of the characteristics that are present in VS15-18 (more than 40%) in comparison to $AZ20_{30k}$ (less than 30%). If we compare that with $VS_{Debiased}$15-18, as shown in Figure 5.4, we see that Google Play services appears at more than 40% of the applications in $VS_{Debiased}$15-18. On the other hand, all of the other characteristics more present in VS15-18 start to lower, up to the point that even "Exec command" is no longer present. Although we cannot visually confirm a total similarity between $VS_{Debiased}$15-18 and $AZ20_{30k}$, we can observe how much the debiased dataset is more similar than the original one. Indeed, it is an expected result because the amount of labeled source available is limited, so the algorithm cannot achieve a perfect proportion balance in the debiased dataset.

5.2.4 Building mixed datasets for machine learning

We finally produced two mixed datasets (goodware/malware) usable for training and testing machine learning algorithms. These datasets are built on top of the debiased malware datasets previously discussed. The goodware are taken from $NAZE_{Debiased}$18-G dataset. These two datasets are:

- **DR-AG**$_{Deb}$ (standing for Drebin+AndroZoo Goodware), that mixes the debiased version of Drebin with $\delta = 0.01$ and the goodware.

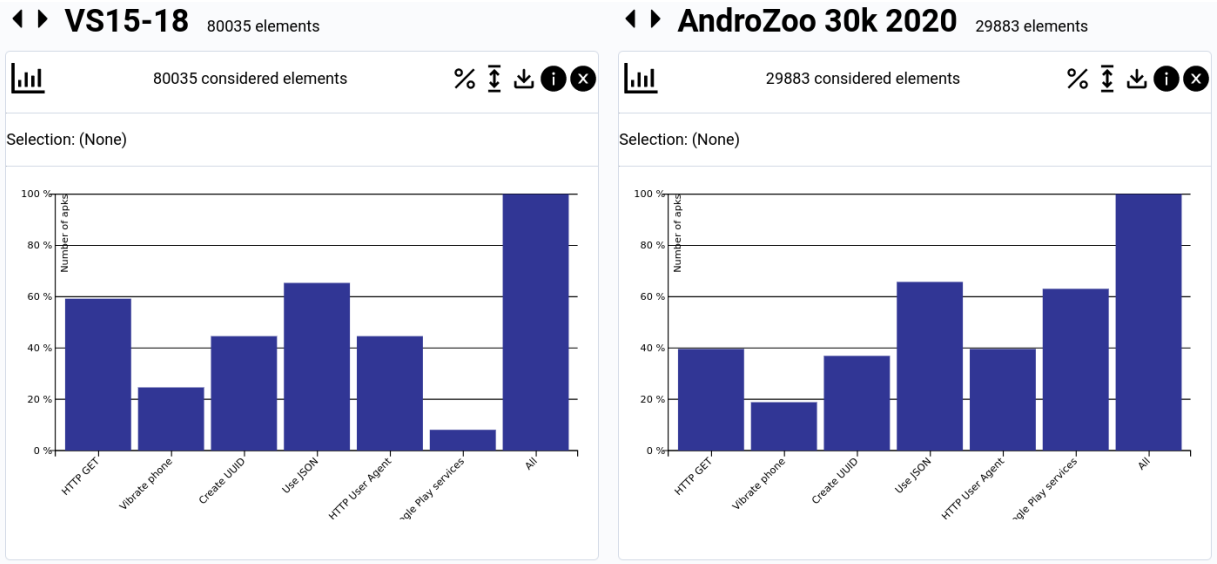

Figure 5.1: Comparison of VS15-18 to $AZ20_{30k}$ for some of the selected characteristics

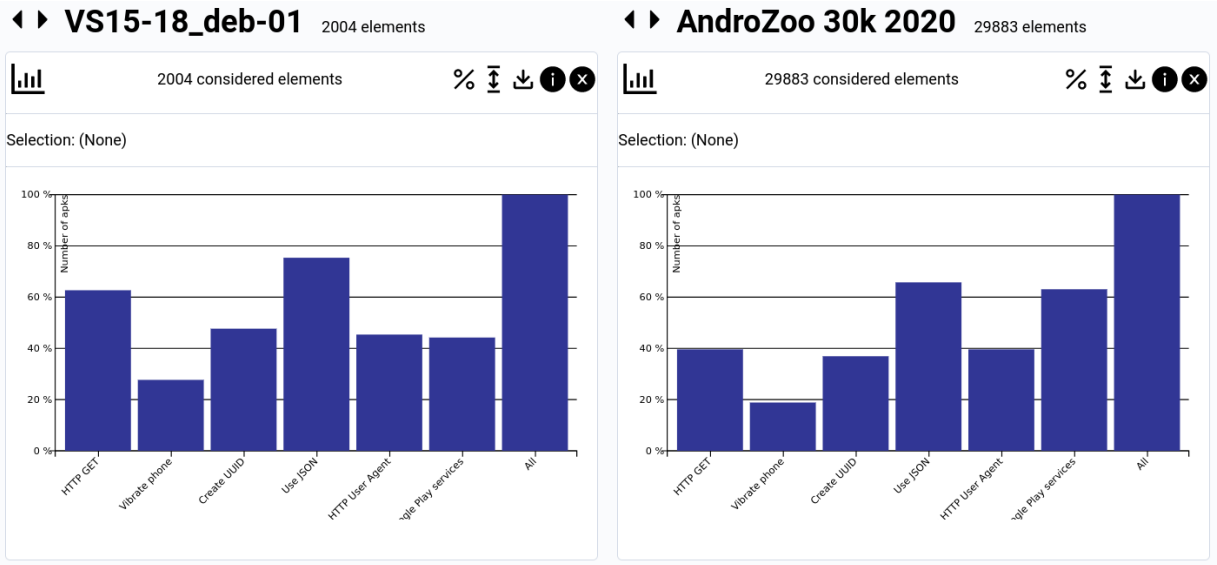

Figure 5.2: Comparison of $VS_{Debiased}15\text{-}18$ to $AZ20_{30k}$ for some of the selected characteristics

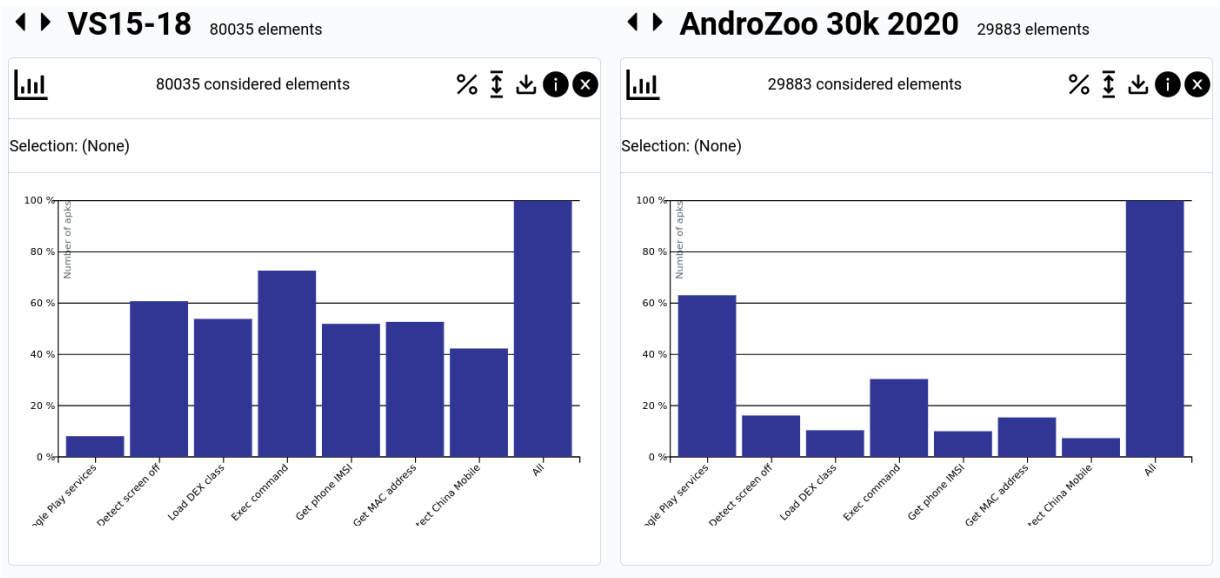

Figure 5.3: Comparison of VS15-18 to $AZ20_{30k}$ for the seven characteristics with the most proportion difference

Figure 5.4: Comparison of $VS_{Debiased}15\text{-}18$ to $AZ20_{30k}$ for the seven characteristics with the most proportion difference

- **VS-AG**$_{Deb}$ (standing for VirusShare+AndroZoo Goodware), that mixes the debiased version of VS$_{\text{Debiased}}$15-18 when $\delta = 0.1$ and the goodware.

Because the goal is to use mixed datasets for machine learning algorithms, we have to follow additional recommendations from Pendlebury *et al.* [11]. We build our mixed sets by extracting random samples from goodware and malware and by following the recommendations C1 (APK of the training set should be older than the ones of the test set) and C3 (realistic malware-to-goodware ratio, *i.e.*, 5% or 10%). In Table 5.2 we call "Pivot year" the year that delimitates the train and test sets. Note also that a consequence of C1 is that applications in the training and test sets are disjoint (which correspond to an added C4*: Train $\cap$ Test = $\emptyset$). For C2 (temporal window consistency) we do not enforce the constraint as we already balanced the classes on the year when performing the debiasing algorithms. Nevertheless, we can enforce them if needed, at the cost of reducing the output size of the training/tests sets. For example, the set VS-AG-C2$_{Deb}$ has fewer samples than VS-AG$_{Deb}$. In the rest of the paper we only use the set that does not enforce C2 to decrease the quantity of results to discuss.

We recomputed the average δ of these mixed datasets: it is still quite low (worst case of 0.0012 for VS-AG$_{Deb}$). The p-value falls to zero for some sets because applying the constraints of Pendlebury *et al.* drives the datasets away from the population. Nevertheless, we believe that respecting the time consistency for machine learning algorithms is of higher importance.

Finally, for evaluating these datasets when used for machine learning purposes, Table 5.2 includes additional datasets:

- **VS-AG**$_{Deb,\delta=0.4}$ a relaxed version of the debiasing of VS$_{\text{Debiased}}$15-18 with $\delta = 0.4$. This dataset as a p-value of 0 but contains more samples in the training set.
- **DR-AG**: a mixed version of Drebin and goodware.
- **VS-AG**: a mixed version of VS 15-18 and goodware.
- **ACT14** and **ACT17**: a mixed version of AndroCT [86] with pivot year 2014 and 2017 respectively.
- **AZL14** and **AZL17**: a mixed version of random samples extracted from AZ20$_{30k}$ for which we took the goodware/malware label of AndroZoo as an oracle.

All these datasets will be used to compare the efficiency of machine learning algorithms when using our debiased datasets. In particular, AZL14 and AZL17 are the biggest extract of the population. AndroCT [86] is a new recent datasets that cover ten years of applications: it will be used to observe its performances when learning with it and testing on AZL14/17.

Table 5.2: Mixed datasets for machine learning training and evaluation

Malware	**Goodware**	**Pivot** year	**Set**	**MalGood** propprop.	**Constraints** [11] C1	C2	C3	C4*	**Mixed dataset** Name	Size	Avg δ	p-value
$\text{Drebin}_{\text{Debiased}}$ ($\delta = 0.01$)	$\text{NAZE}_{\text{Debiased}}$18-G ($\delta = 0.001$)	2014	Training	50% 50%	✓	×	✓	✓	DR-AG_{Deb}	1 614	0.017	0
					✓	✓	✓	✓	DR-AG-C2_{Deb}	572	0.054	0
			Test	10% 90%	✓	×	✓	✓	DR-AG_{Deb}	40	0.071	1
					✓	✓	✓	✓	DR-AG-C2_{Deb}	40	0.089	1
$\text{VS}_{\text{Debiased}}$15-18 ($\delta = 0.01$)	$\text{NAZE}_{\text{Debiased}}$18-G ($\delta = 0.001$)	2017	Training	50% 50%	✓	×	✓	✓	VS-AG_{Deb}	3 492	0.012	1
					✓	✓	✓	✓	VS-AG-C2_{Deb}	2 858	0.013	1
			Test	10% 90%	✓	×	✓	✓	VS-AG_{Deb}	1 333	0.057	0
					✓	✓	✓	✓	VS-AG-C2_{Deb}	1 333	0.057	0
$\text{VS}_{\text{Debiased}}$15-18 ($\delta = 0.04$)	$\text{NAZE}_{\text{Debiased}}$18-G ($\delta = 0.01$)	2017	Training	50% 50%	✓	×	✓	✓	$\text{VS-AG}_{Deb,\delta=0.4}$	16 862	0.018	0
			Test	10% 90%	✓	×	✓	✓	$\text{VS-AG}_{Deb,\delta=0.4}$	2 095	0.057	0
Drebin	AZ19_{100k}G	2014	Training	50% 50%	✓	×	✓	✓	DR-AG	10 608	-	-
VS 15-18	AZ19_{100k}G	2017	Training	50% 50%	✓	×	✓	✓	VS-AG	133 244	-	-
			Test	10% 90%	✓	×	✓	✓	VS-AG	6 113	-	-
AndroCT [86]		2014	Training	50% 50%	✓	×	✓	✓	ACT14	19 351	0.065	0
			Test	10% 90%	✓	×	✓	✓	ACT14	12 241	0.057	0
		2017	Training	50% 50%	✓	×	✓	✓	ACT17	26 389	0.054	0
			Test	10% 90%	✓	×	✓	✓	ACT17	4 580	0.093	0
AZ20_{30k} with labels		2014	Training	50% 50%	✓	×	✓	✓	AZL14	4 890	0.015	0
			Test	10% 90%	✓	×	✓	✓	AZL14	8 785	0.057	0
		2017	Training	50% 50%	✓	×	✓	✓	AZL17	9 474	0.019	0.894
			Test	10% 90%	✓	×	✓	✓	AZL17	2 033	0.059	0

5.3 Experiments with debiased datasets

We evaluate the performance of various machine learning classifiers depending on their training datasets to assess the contribution of debiased datasets. The datasets used in this experiments are those computed in the previous section, cf. Table 5.2.

5.3.1 Classifying with machine learning

The machine learning classifiers rely on a total of 262 characteristics. The first 12 characteristics are those used to debiased the datasets that are not related to security. Additionally, we included the characteristics related to security computed from two sources:

- Droidlysis that computes 168 booleans about the use of some APIs or behaviors (loading a DEX file, using cryptography, etc.); along with 34 permissions found in the Manifest file.
- FalDroid [164] that computes 48 characteristics representing score values related to graph-based features (with parameters $\epsilon = 0.8, \theta = 0.1$ [164]). As FalDroid is intended to classify families,

Table 5.3: Mean and max AUC of machine learning classifiers depending on their training set and their test set when the pivot year is 2017. In bold face: best AUC on a test set

Train \ Test	**ACT17**		**AZL17**		**VS-AG**		**VS-AG**$_{Deb}$	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
ACT17	0.62	0.65	0.67	0.71	0.76	0.85	0.74	0.78
AZL17	**0.72**	**0.81**	**0.78**	**0.86**	0.71	0.89	0.75	0.80
VS-AG	0.62	0.68	0.66	0.73	**0.96**	**0.97**	0.74	0.84
VS-AG$_{Deb}$	0.70	0.74	0.71	0.75	0.86	0.89	**0.83**	**0.87**
DR-AG	0.54	0.57	0.53	0.60	0.57	0.63	0.54	0.58

Table 5.4: Mean and max AUC of machine learning classifiers depending on their training set and their test set when the pivot year is 2014. In bold face: best AUC on a test set

Train \ Test	**ACT14**		**AZL14**		**DR-AG$_{Deb}$**	
	Mean	Max	Mean	Max	Mean	Max
ACT14	0.69	0.73	0.71	0.74	0.69*	0.83*
AZL14	**0.75**	**0.81**	**0.76**	**0.82**	**0.72***	**0.99***
DR-AG	0.57	0.58	0.56	0.58	0.48*	0.50*
DR-AG$_{Deb}$	0.62	0.66	0.61	0.66	**0.72***	0.80*

* due to the very limited size of the test set of DR-AG$_{Deb}$, these results are unreliable

we configured the software with only two families (goodware/-malware) and we extracted features related to control flow graph that manipulates sensitve APIs.

We base this evaluation on the area under the receiver operating characteristic curve (AUC), a classical metric in machine learning. This metric can be interpreted as the probability that a classifier considers a randomly chosen goodware to be more probably benign than a randomly chosen malware. A perfect classifier has an AUC of 1 (every goodware is ranked higher than all malware), meaning that there exists a threshold to separate exactly goodware and malware. A random classifier has an AUC of 0.5. Besides, this metric does not depend on some threshold selection.

We evaluate various machine learning techniques: k-nearest neighbors [165] (neighborhood-based technique), decision trees [166], random forest [167] (bagging ensemble technique with decision trees), Gaussian naive Bayes [168] (statistical model), and AdaBoost [169] (boosting ensemble models with decision trees). We used the scikit-learn [170] implementation with default parameters. Since the decision trees, random forest and AdaBoost learning algorithms are stochastic, we took the average AUC over 25 seeds.

5.3.2 Results

Average and max AUC are presented in Tables 5.3 (pivot year of 2017) and 5.4 (pivot year of 2014) where each line corresponds to a different training set and each column to a different test set. The mean AUC over the various classifiers helps to estimate the global quality of a training dataset for machine learning. The max AUC reflects what performances could be obtained with the best classifiers.

The datasets go by pair: there is one training set and one test set (cf. Table 5.2). In Table 5.3, we can see that generally learning on one training set yields the best results on the associated test set (i.e., the diagonal is in bold). The unique exception is that learning on AZL17 yields better results than learning on ACT17 when testing with ACT17. In fact, we can see that the AZL17 training set allows learning the best classifiers on ACT17 (max AUC: 0.77) and AZL17 (max AUC: 0.84), probably because it is a real randomly sampled dataset, and therefore it is less prone to bias. However, this training set was built using VirusTotal as an oracle, so it is, in general, not available for training.

Moreover, the most interesting results are the comparison of the performances of VS-AG and VS-AG$_{Deb}$ on ACT17 and AZL17 test sets. We see that **the debiased dataset always performs better than the original one**. Another important result is that **it is notably better to train on the debiased dataset VS**-AG$_{Deb}$ than on ACT17 (AndroCT [86]), which is the most recent dataset we have. Finally, **the debiased test dataset VS-AG$_{Deb}$ is more difficult to predict than the test set VS-AG**. It is a sign that the bias in the test set of VS-AG makes it easier for a model to correctly classify it, probably because some difficult cases are absent.

Finally, we can remark that the DR-AG training set yields poor results on all test sets, which is an expected result, as no applications in this dataset after 2015. We tried to confirm this result with an additional experiment with the pivot year of 2014, in Table 5.4. In this experiment, we evaluate if the debiased version of Drebin can obtain good results, even if it is old and quite small. The row in Table 5.4 concerning DR-AG$_{Deb}$ shows that this dataset is far from being effective compared to ACT14. This is due to its size and the failing of debiasing when δ reaches 0.05 (cf. Table 5.1).

Discussion

This section discusses the validity of the datasets through time. We also compare these results with related works [105, 171, 172] that have studied this question and the relation with Android history.

First, Cai et al. [171] focus on the evolution of usage of the system's API over time, using a dedicated analysis framework [105]. They conclude three main points: the decrease of usage of callbacks for methods of the app activities' lifecycle, a stable and small usage of inter-component communication, and a stable distribution of source/sink categories of callbacks. Second, Cai et al. [172] characterize this evolution for goodware and malware applications. Newer malware tend to access system APIs more often through third-party libraries than older malware. The use of activities and services has increased over time, while the use of ICC components has decreased. The diversity of callback categories has also increased.

Our experiments confirm this evolution at a higher level. We study the evolution of representativity of our debiased dataset over the years. More precisely, we debiased several extracts of VirusShare 2015, 2016, 2017 and 2018, and checked whether these debiased datasets stay representative of an evolving population over time. The target datasets are extracts of applications from AndroZoo 2019 ($AZ19_{100k}$) filtered on the corresponding period of time ($\mathbf{T} = AZ19_{100k}\ i$, with $i = 15, 16, 17, 18$). The source datasets are known datasets of malware such as Drebin and AMD, available at date i.

When the p-value confirms that the dataset is not statistically different, we tested the produced debiased version $VS_{Debiased}\ i$ with the next years' population, *i.e.*, $AZ19_{100k}\ i{+}1$, $i{+}2$, ... The last column indicates contains 0 if the p-value is 0: no discussion can be done for such debiased datasets. The only debiased dataset of the parts of VirusShare is VS 18. As it is the last set of applications we have, we only conclude that this set can be used during 1 year.

5.4 Conclusion

In this chapter we defined a new method for creating a dataset that is both representative of a population and labeled. This method relies on two algorithms: a first one that aims to produce a debiased dateset with the same size as the input dataset. The second one takes the solution of the previous algorithm and improves it by allowing the addition or subtraction of applications to the debiased dataset. Then, we created several debiased datasets using our new method, using AndroZoo as the population. We used datasets from the literature as sources of applications, and a representative sample of AndroZoo as the dataset we want the target dataset to resemble. When using our algorithm for a security experiment, we explained that the set of characteristics to use should be those that are not related to security. Lastly, we used our new generated datasets in a machine learning malware detection approach.

We saw in the results that, despite the size of our debiased dataset (VS-AG$_{Deb}$), it outperforms the others biased ones (like AndroCT) in almost every test. Learning with VS-AG$_{Deb}$ gives the best results. Surprisingly, testing with VS-AG$_{Deb}$ is not harder than with others, but it is very close. A possible explanation is the limited size of VS-AG$_{Deb}$ test set. The other generated debiased dataset, DR-AG$_{Deb}$, as its original dataset is too biased, the algorithm reduces considerably its size. Consequently, this reduces efficiency in accuracy for malware detection using machine learning. This indicates that, by removing the differences of unrelated characteristics between the training and test set, the model can more easily identify the differences in characteristics more linked to the detection scheme, in this case being a malware. Besides the use in malware detection, these debiased datasets can be used for other types of experiments. Researchers can use these datasets to concentrate their efforts in relevant characteristics, knowing that their dataset is similar for unrelated ones to the population they are studying.

All materials used in the experiments (SHA256 hashes of samples composing the datasets and the code) are published in the Dada dataset [14]. Our debiasing method, alongside the definition of representativity in Chapter 4, were published in the IEEE Transactions on Information Forensics and Security journal [13].

[14]: Concepcion Miranda et al. (2021), *Dada: Debiased Android DAtasets*

[13]: Concepción Miranda et al. (2022), 'Debiasing Android Malware Datasets: How Can I Trust Your Results If Your Dataset Is Biased?'

Epilogue

Conclusion 6

Contributions of this thesis

In this thesis we described three main contributions targeting the usage of experimental datasets in Android malware detection.

First, we introduced **DaViz, a visualization tool for exploring and comparing Android datasets.** This new tool allows researchers to investigate and discover new insights about a dataset, such as unusual patterns and unexpected values. It also allows researchers to more easily identify differences between datasets, for example the difference of concentration of samples with a given set of characteristics between datasets. By exploiting our visual system, researchers can easily distinguish and locate trends in chart, and so identify differences between various characteristics. Using this tool, we observed that various old and recurrent Android malware datasets from the literature, namely Drebin and AMD, are very different to our reference dataset, an extract of AndroZoo. We also compared the Drebin and AMD datasets with a more recent VirusShare 2018 dataset. We saw specific differences in certain characteristics between the old and the new datasets. This visually confirms the evolutionary changes in malware.

For our second contribution, we focused on the representativity of datasets in relation to a studied population. We introduced the notion of representativity by defining δ-representativity. We presented a method to sample a representative dataset using this notion, for either one or multiple characteristics. Hard-to-compute characteristics, such as *being a malware*, even if they could be calculated, may be over- or underrepresented in an extract of the population. Therefore, we need to use mixed datasets and evaluate their heterogeneity against the population. **We presented a comparison between datasets of the literature and a representative dataset of AndroZoo, our studied population, using a statistical test.** Using our sampling method, we saw that, for some cases, a sample of 15,000 application is statistically enough to get results with a low bias error. We showed that none of the datasets in the comparison, except an extract of AndroZoo, are representative of all AndroZoo, the studied population.

In our last contribution, we presented a method to transform a labeled non-representative dataset into a labeled *representative* dataset that resembles a *non-labeled* representative dataset extracted from the studied population. **We used this method to create several "debiased" datasets from various datasets of the literature.** We also debiased several datasets from different years and checked their validity against reference datasets of the next year. Results revealed that debiased datasets are only valid at most one year, before they are no longer representative. This confirms that static datasets, the older they get, the less they resemble other more dynamic datasets, such as markets.

We also conducted an Android malware detection experiment using machine learning techniques with one of our debiased datasets. Results

of the malware detection experiment show that our debiased dataset performs better than a bigger, more recent dataset like AndroCT, composed of 17 697 malware and 18 277 goodware samples ranging from 2010 to 2019.

Perspectives for future work

In this section we present our vision on how to improve and expand DaViz and our debiasing method.

Other types of characteristics can be included for visualization DaViz. For example, opcode frequencies, frequency of API sequences, code density, and others. These new characteristics may need modifications in Droidlysis or the use of new specialized extraction techniques to calculate them, including those that rely on dynamic analysis. This would expand the usage of our characteristics database to more machine learning experiments concerning other types of characteristics.

With these new characteristic, new challenges arise concerning how to manage these new characteristics. Because there are a lot of (potentially infinite) characteristics, users would have too much possibilities to explore datasets. This poses a problem in choosing which characteristic would be adequate to visualize. A solution would involve a recommendation system that would calculate the most pertinent charts and characteristics to explore a dataset. This way, users would choose their charts and characteristics based on the suggestions given by the system.

Additional views could be implemented in DaViz to visualize individual applications on their own, such as those showing the control-flow graph (CFG), data-flow or information-flow graphs of the application's code or execution. This would allow a more detailed exploration of the applications that compose a dataset. For example, our visualization system would switch from a datasets view to an individual applications view. Additionally, the use of graphs could be extended not only to individual applications. Indeed, in malware family classification, some techniques use CFGs to classify samples into malware families, *i.e.* malicious applications with similar behavior. [173]. An implementation idea is to calculate and show a CFG of a dataset comprised of samples from a single family. This CFG would be calculated by obtaining the most commons nodes and edges of all the CFGs in the dataset. Additionally, this technique could be used to compare datasets composed of varieties of a family, which would allow to see similarities and differences of samples inside a family. These types of analyses would help researchers in malware family classification, as this topic is an ongoing research problem [174].

[173]: Zhou et al. (2017), 'Analysis of Android Malware Family Characteristic Based on Isomorphism of Sensitive API Call Graph'

[174]: Joyce et al. (2022), 'MOTIF: A Malware Reference Dataset with Ground Truth Family Labels'

So far we use the χ^2 test to proof the statistical difference between two datasets. Unfortunately, this test cannot be used to proof the homogeneity of two datasets, as it only allows to proof that two datasets are statistically different. To evaluate the homogeneity of datasets, a distance measure could be used for this, *e.g.* using Kullback–Leibler divergence or Hellinger distance. In this case, a threshold must be established to set a boundary between similar and different when comparing two datasets. Beside this, other tests exists, just as the Kolmogorov–Smirnov test, that could be

used besides the χ^2 test. Because the data we use is primarily categorical data, as far as we know, there are not many distance measures available for this type of data. We believe that this work would require background knowledge in statistics to properly adjust existing tests and distances to categorical data.

Another scenario we consider as a possibility is the usage of our solutions, aimed toward the Android OS, for x86 programs. Of course, both DaViz and our debiasing method is platform agnostic, because they operate with characteristics rather than raw binaries. Therefore, with the extraction of characteristics from x86 and their importation to DaViz, it could be possible to explore and compare x86 malware datasets, in the same way it is done with Android datasets. Although the basic idea is easy to contemplate, x86 programs pose a problem in regard with the extraction of characteristics. In Android, the structure of an APK is known and it is possible to locate its Manifest file, native libraries and assets (*i.e.* images, documents, other code and data defined by the developer). The Manifest XML file contains important data such as permissions, intents, and others declared in it. Despite obfuscation techniques, it is possible to disassemble the code in an Android application. Thus, it is relatively easy to extract static information from an Android APK. In contrast, x86 executable formats do not follow this convention. In Portable Executable (PE) and Executable and Linkable Format (ELF) files, besides the header and the code section, no other valuable data is available for extraction. Because of this, there are less static characteristics that can be extracted from x86 programs. Despite this, it could be possible to extract more characteristics from dynamic analysis.

Once characteristics are extracted, it is possible to debias an x86 dataset to another using our method. However, even with various experimental and malware datasets available in x86, there is no particular dataset of interest. In Android, markets are specially interesting because these are the main attack vector of malware distribution. In the case of x86, there is no market or "app store" of the same scale and usage as in Android. Rather, attack vectors in x86 includes download through phishing, P2P sharing, malicious websites, and USB drives. Datasets for x86 tend to be malware datasets where the ground truth is known [174], but not a general repository of applications for users. Such a "reality" of applications for x86 and the generation of "general" datasets could be a subject of study.

Algorithm proofs A

For the sake of brevity and clarity, we introduce the notation $\Delta_{\mathbf{T}} p_{\mathbf{G},k} = p_{\mathbf{G},k} - p_{\mathbf{T},k}$. So a class k is overrepresented w.r.t. $\mathbf{T}$ if $\Delta_{\mathbf{T}} p_{\mathbf{G},k} > \delta$ and underrepresented if $\Delta_{\mathbf{T}} p_{\mathbf{G},k} < -\delta$.

Theorem A.0.1 *Algorithm 1 always halts.*

Proof. First, remark that if a class k is correctly represented at some iteration, i.e., if $|\Delta_{\mathbf{T}} p_{\mathbf{G},k}| \leq \delta$, then it will be correctly represented in the next iteration. Besides, a class that is overrepresented in $\mathbf{G}$ cannot become underrepresented in the following iteration (and vice versa) without the algorithm halting. Both remarks are ensured by the "If" block on line 13 that halts the program by verifying whether a class with an added element is overrepresented or a class with a removed element is underrepresented.

Let's assume that this case never happens. At each iteration of the algorithm, there is at least one class k such that $|\Delta_{\mathbf{T}} p_{\mathbf{G},k}| > \delta$ that is modified in $\mathbf{G}$: one element is added if k is underrepresented and one element is removed if k is overrepresented in $\mathbf{G}$. So $\sum_k |\Delta_{\mathbf{T}} p_{\mathbf{G},k}|$ is reduced by at least $\frac{1}{|\mathbf{B}|}$ in the updated value of $\mathbf{G}$. With enough iterations, $\sum_k |\Delta_{\mathbf{T}} p_{\mathbf{G},k}|$ will so low that every class will be correctly represented. At that point, the algorithm halts. □

Besides, a representative dataset always exists. This is necessary to prove the termination and the complexity of Algorithm 2.

Theorem A.0.2 *Let $\mathbf{T}$ be a dataset, $\delta > 0$ and $n > 0$ such as $n \geq \frac{1}{\delta}$. There exists a dataset $\mathbf{D}$ of size n such as $\mathbf{D}$ is δ-representative of $\mathbf{T}$.*

Proof. To prove the existence of such dataset of size n, we construct it. More precisely, we associate to each class k a number of element b_k such as this class is δ-representative of $\mathbf{T}$, i.e., $\frac{b_k}{n} \in [p_{\mathbf{T},k} - \delta; p_{\mathbf{T},k} + \delta]$. For each class k, define a_k such as $\frac{a_k}{n} < p_{\mathbf{T},k} \leq \frac{a_k+1}{n}$. Remark that $p_{\mathbf{T},k} - \delta \leq \frac{a_k}{n} < p_{\mathbf{T},k} \leq \frac{a_k+1}{n} \leq p_{\mathbf{T},k} + \delta$ since $\delta \geq \frac{1}{n}$. As $\sum_k p_{\mathbf{T},k} = 1$, $\sum_k a_k < n$ and $\sum_k (a_k + 1) \geq n$. So there exist b_k such as for all class k either $b_k = a_k$ or $b_k = a_k + 1$ and that verify $\sum_k b_k = n$.

Denote the dataset $\mathbf{D}$ such as $p_{\mathbf{D},k} = \frac{b_k}{n}$. The size of $\mathbf{D}$ is $\sum_k b_k = n$ and $\mathbf{D}$ is δ-representative of $\mathbf{T}$ since $\frac{b_k}{n} \in [p_{\mathbf{T},k} - \delta; p_{\mathbf{T},k} + \delta]$ for every class k. It proves the existence of such a dataset. □

Theorem A.0.3 *If there exists a dataset $\boldsymbol{D}$ consisting of elements of $\boldsymbol{B}$ and $\boldsymbol{S}$ (i.e., $\boldsymbol{D} \subseteq \boldsymbol{B} \cup \boldsymbol{S}$) such as $\boldsymbol{D}$ is representative of $\boldsymbol{T}$ and $|\boldsymbol{D}| = |\boldsymbol{B}|$, then Algorithm 1 returns a representative dataset. If no such dataset exists, the algorithm returns "Impossible".*

Proof. Assume such dataset **D** exists. Let us first prove that the **return** on line 9 cannot be reached. This line is reached if **S** is empty or if $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} < -\delta$.

Consider the case where $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} < -\delta$: we will prove by contradiction that this case is impossible when **D** exists. Since the algorithm never removes an element of an underrepresented class, at each iteration $p_{\mathbf{G},k_l} = \frac{\mathbf{B}(k_l)+m}{n}$ where m is the number of elements added to the class from **S**. If $\mathbf{S}(k_l) = 0$, then $p_{\mathbf{G},k_l}$ is maximized, i.e., $p_{\mathbf{G},k_l} \geq p_{\mathbf{D},k_l}$. So $\Delta_{\mathbf{T}} p_{\mathbf{D},k_l} \leq \Delta_{\mathbf{T}} p_{\mathbf{G},k_l} < -\delta$ which means that **D** is not representative of **T**. This is a contradiction: therefore this case is not possible.

Consider the case where **S** is empty and $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} \geq -\delta$. Once again, we will prove by contradiction that this case is impossible when **D** exists. The fact that $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} \geq -\delta$ implies that $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} \geq \delta$, otherwise the "While" condition would not have been true. Denote m the number of completed iterations so far. Since one element of **S** is removed at each iteration, we conclude that initially $|\mathbf{S}| = m$. Denote $\mathbf{B_D} \subseteq \mathbf{B}$ and $\mathbf{S_D} \subseteq \mathbf{S}$ such as $\mathbf{D} = \mathbf{B_D} \cup \mathbf{S_D}$. Due to the definition of k_h and the fact that $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} \geq \delta$, it means that for each previous iterations one element has been removed from an overrepresented class. So there are at least $m+1$ elements of **B** that are absent in $\mathbf{B_D}$, i.e., $|\mathbf{B_D}| \leq |\mathbf{B}| - m$. However, since $|\mathbf{B}| = |\mathbf{D}| = |\mathbf{B_D}| + |\mathbf{S_D}|$, it means that $|\mathbf{S}| \geq m+1$. This is in contradiction with the fact that $|\mathbf{S}| = m$. So **S** cannot be empty. Finally, the **return** on line 9 cannot be reached when **D** exists.

We now prove that the "If" condition on line 13 cannot be true in Algorithm 1 when **D** exists. For algorithm 1 to return an error on line 14, the condition $max_{k \in \mathcal{K}} |\Delta_{\mathbf{T}} p_{\mathbf{G},k}| > \delta$ must be true. Let us assume (without loss of generality) that there exists k' such as $\Delta_{\mathbf{T}} p_{\mathbf{G},k'} > \delta$ (the case $\Delta_{\mathbf{T}} p_{\mathbf{G},k'} < -\delta$ is similar).

Consider now the value of k_h. Since $k_h \leftarrow \arg\max_{k \in \mathcal{K}} \Delta_{\mathbf{T}} p_{\mathbf{G},k}$, we can conclude that $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} \geq \Delta_{\mathbf{T}} p_{\mathbf{G},k'} > \delta$. Denote $n = |\mathbf{B}|$ and $\mathbf{G'}$ the dataset obtained from **G** by removing one element from k_h and adding one element to k_l. The proportions of the classes of $\mathbf{G'}$ are:

$$p_{\mathbf{G}',k} = \begin{cases} p_{\mathbf{G},k} - \frac{1}{n} \text{ if } k = k_h \\ p_{\mathbf{G},k} + \frac{1}{n} \text{ if } k = k_l \\ p_{\mathbf{G},k} \text{ otherwise} \end{cases}$$

Our goal is to prove that the "If" condition will never be true, i.e., that $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_h} \geq -\delta$ and $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_l} \leq \delta$.

Let a_1 and a_2 be two naturals such as $p_{\mathbf{G},k_h} = \frac{a_1}{n}$ and $p_{\mathbf{D},k_h} = \frac{a_2}{n}$. Since $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} > \delta$ and $\Delta_{\mathbf{T}} p_{\mathbf{D},k_h} \leq \delta$ (due to **D** being representative), we can conclude that $p_{\mathbf{G},k_h} > p_{\mathbf{D},k_h}$, i.e., $a_1 > a_2$. Since $p_{\mathbf{G}',k_h} = p_{\mathbf{G},k_h} - \frac{1}{n} = \frac{a_1-1}{n}$ and $a_1 - 1 \geq a_2$, we conclude that $p_{\mathbf{G}',k_h} \geq p_{\mathbf{D},k_h}$ and $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_h} \geq \Delta_{\mathbf{T}} p_{\mathbf{D},k_h}$. Since **D** is itself representative of **T**, we know that $\Delta_{\mathbf{T}} p_{\mathbf{D},k_h} \geq -\delta$ and therefore that $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_h} \geq -\delta$.

Let us show that there exists k'_l such as $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} < \Delta_{\mathbf{T}} p_{\mathbf{D},k'_l} \leq \delta$. Since $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} > \Delta_{\mathbf{T}} p_{\mathbf{D},k_h}$ and $\sum_k \Delta_{\mathbf{T}} p_{\mathbf{G},k} = \sum_k \Delta_{\mathbf{T}} p_{\mathbf{D},k}$, we can conclude that there exists k'_l such as $\Delta_{\mathbf{T}} p_{\mathbf{G},k'_l} < \Delta_{\mathbf{T}} p_{\mathbf{D},k'_l}$. Furthermore, $k'_l \in \mathcal{K}_S$ because if **S** were empty it would not be possible for **D** to have more elements than **G** for this class.

Due to the definition of k_l, $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} \leq \Delta_{\mathbf{T}} p_{\mathbf{G},k'_l} < \Delta_{\mathbf{T}} p_{\mathbf{D},k'_l} \leq \delta$. Let b_1, b_2, b_3, b_4 four naturals such as $p_{\mathbf{G},k_l} = \frac{b_1}{n}$, $p_{\mathbf{T},k_l} = \frac{b_2}{n}$, $p_{\mathbf{D},k'_l} = \frac{b_3}{n}$ and $p_{\mathbf{T},k'_l} = \frac{b_4}{n}$. We can conclude of the previous inequalities that $b_1 - b_2 < b_3 - b_4$. Therefore $b_1 - b_2 + 1 \leq b_3 - b_4$ and $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_l} = p_{\mathbf{G},k_l} + \frac{1}{n} - p_{\mathbf{D},k_l} \leq \delta$. This proves that the "If" condition is never true if $\mathbf{D}$ exists.

Finally, if $\mathbf{D}$ exists, then Algorithm 1 returns a dataset. This dataset is representative because the "While" condition is true for the last iteration. When the algorithm outputs a dataset, it is representative and its size is the same as $\mathbf{B}$. If such dataset does not exist, the algorithm cannot output one. Since the algorithm always halts, it must return "Impossible". □

We denote $d_H(\mathbf{A}, \mathbf{B})$ the number of modifications between $\mathbf{A}$ and $\mathbf{B}$. More precisely, we extend the Hamming distance to a distance between two sets by defining $d_H(\mathbf{A}, \mathbf{B})$ as $d_H(\mathbf{A}, \mathbf{B}) = |(\mathbf{A} \setminus \mathbf{B}) \cup (\mathbf{B} \setminus \mathbf{A})|$. $d_H(\mathbf{A}, \mathbf{B})$ is the number of elements present in only one dataset, $\mathbf{A}$ or $\mathbf{B}$. In other words, it is the minimal number of addition and deletions to transform $\mathbf{A}$ into $\mathbf{B}$ (and vice-versa).

Theorem A.0.4 *Let $\boldsymbol{B}$ be a biased dataset, $\boldsymbol{D}$ a dataset representative of P such as $\boldsymbol{D} \subseteq \boldsymbol{B} \cup \boldsymbol{S}$ and $\boldsymbol{G} = Algo1(\boldsymbol{B})$. If $|\boldsymbol{B}| = |\boldsymbol{D}|$, then $d_H(\boldsymbol{B}, \boldsymbol{D}) \geq d_H(\boldsymbol{B}, \boldsymbol{G}) = 2\max(\sum_k n^+_{\boldsymbol{B}}(k), \sum_k n^-_{\boldsymbol{B}}(k))$. So Algorithm 1 outputs a representative dataset with minimal distance to $\boldsymbol{B}$.*

Proof. Let $\mathbf{A} = \mathbf{D} \setminus \mathbf{B}$ be the set of elements added to $\mathbf{B}$ and $\mathbf{R} = \mathbf{B} \setminus \mathbf{D}$ be the set of elements removed from $\mathbf{B}$. So $\mathbf{D} = (\mathbf{B} \setminus \mathbf{R}) \cup \mathbf{A}$ and $d_H(\mathbf{B}, \mathbf{D}) = |\mathbf{A}| + |\mathbf{R}|$. Due to the hypothesis $|\mathbf{B}| = |\mathbf{D}|$, we know that $|\mathbf{A}| = |\mathbf{R}|$ (there are as many additions as removals).

For any class k that is overrepresented in $\mathbf{B}$, $n^+_{\mathbf{B}}(k)$ is the minimal number of removal of elements of class k such as k is not overrepresented anymore. $n^-_{\mathbf{B}}(k) = \max(0, \lceil -|\mathbf{B}|(\Delta_{\mathbf{T}} p_{\mathbf{B},k} + \delta) \rceil)$ is similar for underrepresented classes.

To make a representative dataset from $\mathbf{B}$, one must remove at least $\sum_k n^+_{\mathbf{B}}(k)$ elements and add $\sum_k n^-_{\mathbf{B}}(k))$, i.e., $|\mathbf{A}| \geq \sum_k n^+_{\mathbf{B}}(k)$ and $|\mathbf{R}| \geq \sum_k n^-_{\mathbf{B}}(k))$. However, since $|\mathbf{A}| = |\mathbf{R}|$, we conclude that $|\mathbf{A}| = |\mathbf{R}| \geq \max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$. It means that $d_H(\mathbf{B}, \mathbf{D}) \geq 2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$.

Let now show that $d_H(\mathbf{B}, \mathbf{G}) = 2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$. At each iteration of the algorithm, if $\sum_k n^+_{\mathbf{G}}(k) > 0$, then an overrepresented class has one of its elements removed, so $\sum_k n^+_{\mathbf{G}'}(k) < \sum_k n^+_{\mathbf{G}}(k)$. For the same reason, at each iteration, $\sum_k n^-_{\mathbf{G}'}(k) < \sum_k n^-_{\mathbf{G}}(k)$. When $\sum_k n^+_{\mathbf{G}}(k) = \sum_k n^-_{\mathbf{G}}(k) = 0$, then the algorithm halts. So there are $\max(\sum_k n^+_{\mathbf{G}}(k) = \sum_k n^-_{\mathbf{G}}(k))$ iterations. Since each iteration makes two modifications of the dataset, we can conclude that $d_H(\mathbf{B}, \mathbf{G}) = 2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$.

So $d_H(\mathbf{B}, \mathbf{G}) \leq d_H(\mathbf{B}, \mathbf{D})$. □

Proof of theorem 5.1.1. Let us first prove that at each step of the algorithm, d is equal to the number of additions and deletions between $\mathbf{B}$ and $Algo1(\mathbf{G})$. This number can be decomposed as the sum of the number of additions and deletions between $\mathbf{B}$ and $\mathbf{G}$ and between $\mathbf{G}$ and $Algo1(\mathbf{G})$. The number of modifications between $\mathbf{G}$ and $\mathbf{B}$ is simply the difference

of their size, i.e., $|\,|\mathbf{B}| - |\mathbf{G}|\,|$. The number of modification between $\mathbf{G}$ and $Algo1(\mathbf{G})$ is $2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k))$ (cf. the previous proof algorithm). So the number of modifications between $\mathbf{B}$ and $Algo1(\mathbf{G})$ is $|\,|\mathbf{B}| - |\mathbf{G}|\,| + 2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k))$.

First, let us show that the algorithm always halts. This algorithm halts either if $d_{min} \neq \infty$ (so both loops have a finite number of iterations) or if the source dataset lacks some element (lines 16 to 20). If a solution $\mathbf{G}$ exists, then at some point Algorithm 2 will call Algorithm 1 with a dataset whose size is the same as $\mathbf{G}$ (either at line 2 if $|\mathbf{G}| = |\mathbf{B}|$, at line 10 if $|\mathbf{G}| < |\mathbf{B}|$ or at line 26 if $|\mathbf{G}| > |\mathbf{B}|$) and Algorithm 1 will find a solution (due to Theorem A.0.4). If at some point it is not possible to add an element to a underrepresented class because there is no such element in the source dataset, then the algorithm halts (lines 5 to 9).

Second, we show that doing $d_{min} - 1$ iterations per loop is sufficient (lines 6 and 15). Let $\mathbf{G}_{best}$ be the best dataset found so far, associated to a distance d_{min}, and a better solution $\mathbf{G'}$ associated to a distance d' such as $d' < d_{min}$. Remark that at each step, $i = |\,|\mathbf{B}| - |\mathbf{G}|\,|$ as one element is modified per iteration. Denote j the iteration at which $\mathbf{G'}$ could be found. Since $d' = |\,|\mathbf{B}| - |\mathbf{G}|\,| + 2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k)) \geq j$ and $d' < d_{min}$, we conclude that $j \leq d_{min}$ and therefore that $\mathbf{G'}$ will be found by the algorithm. □

Theorem A.0.5 *The worst-case temporal complexity of Algorithm 2 is* $O\left((\frac{1}{\delta} + |\boldsymbol{B}|)^2|\mathcal{X}|\right)$.

Proof. Remark that for all datasets $\mathbf{A}$ and $\mathbf{B}$, the number of additions and deletions required to obtain $\mathbf{B}$ from $\mathbf{A}$ is lower than or equal to $|\mathbf{A}| + |\mathbf{B}|$. Theorem A.0.2 shows that there exists a representative dataset $\mathbf{D}$ of size $\lceil\frac{1}{\delta}\rceil$. So the minimal distance d_{min} to reach a representative dataset $\mathbf{D}$ from the initial base dataset $\mathbf{B}$ is bounded by $\lceil\frac{1}{\delta}\rceil + |\mathbf{B}|$. So Algo. 2 will do at most $2(\lceil\frac{1}{\delta}\rceil + |\mathbf{B}|)$ iterations. Each iteration calls Algo. 1, whose temporal complexity is $O(|\mathbf{G}| \cdot |\mathcal{X}|)$ where the size of $\mathbf{G}$ is at most $|\mathbf{B}| + d_{min} \leq \lceil\frac{1}{\delta}\rceil + 2|\mathbf{B}|$. So the worst-case temporal complexity of Algo. 2 is $O\left((\frac{1}{\delta} + |\mathbf{B}|)^2|\mathcal{X}|\right)$. □

Bibliography

Here are the references in citation order.

[1] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 'Survey of machine learning techniques for malware analysis'. In: *Computers & Security* 81 (2019), pp. 123–147. DOI: https://doi.org/10.1016/j.cose.2018.11.001 (cited on page 3).

[2] Kaijun Liu et al. 'A Review of Android Malware Detection Approaches Based on Machine Learning'. In: *IEEE Access* 8 (2020), pp. 124579–124607. DOI: 10.1109/ACCESS.2020.3006143 (cited on page 3).

[3] Ya Pan et al. 'A Systematic Literature Review of Android Malware Detection Using Static Analysis'. In: *IEEE Access* 8 (2020), pp. 116363–116379. DOI: 10.1109/ACCESS.2020.3002842 (cited on page 3).

[4] Junyang Qiu et al. 'A Survey of Android Malware Detection with Deep Neural Models'. In: *ACM Comput. Surv.* 53.6 (2020). DOI: 10.1145/3417978 (cited on page 3).

[5] Daniel Arp et al. 'DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket'. In: *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014 (cited on pages 4, 14, 16, 63).

[6] Fengguo Wei et al. 'Deep Ground Truth Analysis of Current Android Malware'. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Polychronakis, Michalis and Meier, Michael. Cham: Springer International Publishing, 2017, pp. 252–276 (cited on pages 4, 15, 16, 63).

[7] Fauzia Idrees et al. 'PIndroid: A novel Android malware detection system using ensemble learning methods'. In: *Computers & Security* 68 (2017), pp. 36–46. DOI: https://doi.org/10.1016/j.cose.2017.03.011 (cited on pages 5, 21).

[8] ElMouatez Billah Karbab et al. 'MalDozer: Automatic framework for android malware detection using deep learning'. In: *Digital Investigation* 24 (2018), S48–S59. DOI: https://doi.org/10.1016/j.diin.2018.01.007 (cited on pages 5, 21).

[9] Vinod P., Akka Zemmari, and Mauro Conti. 'A machine learning based approach to detect malicious android apps using discriminant system calls'. In: *Future Generation Computer Systems* 94 (2019), pp. 333–350. DOI: https://doi.org/10.1016/j.future.2018.11.021 (cited on pages 5, 21).

[10] Kevin Allix et al. 'Are Your Training Datasets Yet Relevant?' In: *Engineering Secure Software and Systems*. Cham: Springer International Publishing, 2015, pp. 51–67 (cited on pages 5, 23, 53).

[11] Feargus Pendlebury et al. 'TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time'. In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 729–746 (cited on pages 5, 16, 18–20, 23, 53, 76, 77).

[12] Tomás Concepción Miranda et al. 'DaViz: Visualization for Android Malware Datasets'. In: *RESSI 2022 - Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information*. Chambon-sur-Lac, France, May 2022, pp. 1–3 (cited on pages 6, 27, 53).

[13] Tomás Concepción Miranda et al. 'Debiasing Android Malware Datasets: How Can I Trust Your Results If Your Dataset Is Biased?' In: *IEEE Transactions on Information Forensics and Security* 17 (2022), pp. 2182–2197. DOI: 10.1109/TIFS.2022.3180184 (cited on pages 6, 80).

[14] Tomas Concepcion Miranda et al. *Dada: Debiased Android DAtasets*. 2021. DOI: 10.21227/0frv-zb46 (cited on pages 6, 80).

[15] Google LLC. *Google Play*. 2022. URL: https://play.google.com/store (cited on page 10).

[16] AppBrain. *Android and Google Play statistics*. 2022. URL: https://www.appbrain.com/stats (visited on 04/23/2022) (cited on page 10).

[17] AppBrain. *Number of Android apps on Google Play*. 2018. URL: https://web.archive.org/web/20180520205012/https://www.appbrain.com/stats/number-of-android-apps (visited on 04/23/2022) (cited on page 10).

[18] Nicolas Viennot, Edward Garcia, and Jason Nieh. 'A Measurement Study of Google Play'. In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 42. 1. ACM. 2014, pp. 221–233 (cited on pages 10, 12).

[19] Rahul Patel. *AuroraOSS*. 2022. URL: https://auroraoss.com/ (cited on page 10).

[20] Sergey Yeriomin. *Yalp Store*. 2016. URL: https://github.com/yeriomin/YalpStore (visited on 04/23/2022) (cited on page 10).

[21] SlideME LLC. *SlideME*. 2022. URL: http://slideme.org/about-slideme (cited on page 10).

[22] Samsung Electronics Co., Ltd. *Galaxy Store*. 2022. URL: https://galaxystore.samsung.com/ (visited on 04/23/2022) (cited on page 10).

[23] Ryne Hager. *Samsung's Galaxy Store is distributing apps that could infect phones with malware*. Android Police. Dec. 27, 2021. URL: https://www.androidpolice.com/samsung-galaxy-store-malware-movie-piracy-showbox/ (visited on 04/23/2022) (cited on page 10).

[24] Bullitt Mobile Ltd. *Toolbox App*. https://support.catphones.com/hc/en-gb/articles/360012142819-S52-Using-the-Toolbox-App. 2022 (cited on page 10).

[25] Aptoide S.A. *Aptoide*. 2022. URL: https://www.aptoide.com/about-us (cited on page 10).

[26] Amazon.com, Inc. *Amazon Appstore*. 2022. URL: https://www.amazon.com/gp/mas/get/android/ (visited on 04/23/2022) (cited on page 11).

[27] F-Droid Limited. *F-Droid*. 2022. URL: https://f-droid.org (cited on page 11).

[28] GLOBAL PTE LTD. *AndroidOut*. https://www.androidout.com. 2022 (cited on page 11).

[29] GLOBAL PTE LTD. *Androidout corporate - Who we are - What we do Androidout.com*. https://www.androidlista.org/#overview. 2022 (cited on page 11).

[30] John Sanford. *MiKandi Reaches Milestone With Mobile Devices, Registered Users*. https://web.archive.org/web/20180627173546/https://www.androidcentral.com/nsfw-android-adult-app-store-mikandi-bares-even-more-and-does-it-better-version-4. Accessed: 2022-04-17. Feb. 5, 2013 (cited on page 11).

[31] MiKandi LLC. *MiKandi*. https://mikandi.com/. 2022 (cited on page 11).

[32] 1Mobile Market. *1Mobile Market*. https://1mobile.market/. 2022 (cited on page 11).

[33] Ilja Laurs. *GetJar*. 2022. URL: https://www.getjar.com/about (visited on 04/23/2022) (cited on page 11).

[34] Team AndroidAdvices. *Remove Android:Plankton [PUP] Virus from Android Device after Downloading Apps from GetJar*. Android Advices. Apr. 11, 2012. URL: https://androidadvices.com/remove-androidplanktona-pup-virus-android-device-downloading-apps-getjar/ (visited on 04/23/2022) (cited on page 11).

[35] Mathijs Vogelzang and Uwe Maurer. *AppBrain*. 2022. URL: https://www.appbrain.com/info/about (visited on 04/23/2022) (cited on page 11).

[36] Scott Milliken. *MixRank*. 2022. URL: https://mixrank.com/about (cited on page 11).

[37] Jan Katrenic. *ANDROIDRANK*. 2022. URL: https://androidrank.org/ (visited on 04/24/2022) (cited on page 11).

[38] SimilarWeb LTD. *Top Apps Ranking*. 2022. URL: https://www.similarweb.com/apps/top/google/app-index/us/all/top-free/ (visited on 04/24/2022) (cited on page 11).

[39] TapTap Pte. Ltd. *TapTap*. 2022. URL: https://www.taptap.io/about-us (visited on 04/24/2022) (cited on page 11).

[40] DATA.AI EUROPE LIMITED. *data.ai*. 2022. URL: https://www.data.ai/en/about/ (visited on 04/24/2022) (cited on page 11).

[41] SensorTower, Inc. *SensorTower*. 2022. URL: https://app.sensortower.com/ (visited on 04/24/2022) (cited on page 11).

[42] Ben Quinn. 'Google services blocked in China'. In: *The Gardian* (Nov. 9, 2012). (Visited on 04/18/2022) (cited on page 11).

[43] Haoyu Wang et al. 'Beyond Google Play: A Large-Scale Comparative Study of Chinese Android App Markets'. In: *Proceedings of the Internet Measurement Conference 2018*. IMC '18. Boston, MA, USA: Association for Computing Machinery, 2018, pp. 293–307. DOI: 10.1145/3278532.3278558 (cited on pages 11, 17).

[44] Hezardastan Information Technology Development Group. *Cafe Bazaar*. 2022. URL: https://cafebazaar.ir/about?l=en (cited on page 11).

[45] Umid Niayesh. *How developers can make money in Iran's app market?* AzerNews. Sept. 4, 2017. URL: https://www.azernews.az/region/118470.html (visited on 04/17/2022) (cited on page 11).

[46] José Domínguez and Luis Hernández. *We are Uptodown*. 2022. URL: https://en.uptodown.com/aboutus/uptodown (visited on 04/23/2022) (cited on page 11).

[47] José Domínguez and Luis Hernández. *Uptodown*. 2011. URL: https://web.archive.org/web/20111112154211/http://www.uptodown.com:80/ (visited on 04/23/2022) (cited on page 11).

[48] jakej. *Android Apps*. https://archive.org/details/android_apps/. 2022 (cited on page 12).

[49] The Internet Archive. *The Internet Archive*. 2022. URL: https://archive.org/about/ (cited on page 12).

[50] Illogical Robot LLC. *APKMirror*. 2022. URL: https://www.apkmirror.com (visited on 04/24/2022) (cited on page 12).

[51] APKPure, Inc. *APKPure*. 2022. URL: https://apkpure.com (visited on 04/24/2022) (cited on page 12).

[52] APKLinker. *APKLinker*. 2022. URL: https://www.apklinker.com/ (visited on 04/24/2022) (cited on page 12).

[53] APK4Fun. *APK4Fun*. 2022. URL: https://www.apk4fun.com (visited on 04/24/2022) (cited on page 12).

[54] Igor Golovin. *Infected Android app store*. Kaspersky Lab. Apr. 9, 2021. URL: https://www.kaspersky.com/blog/infected-apkpure/39273/ (visited on 04/24/2022) (cited on page 12).

[55] Igor Golovin and Anton Kivva. *Malicious code in APKPure app*. Kaspersky Lab. Apr. 9, 2021. URL: https://securelist.com/apkpure-android-app-store-infected/101845/?/kaspersky/april8/index.html (visited on 04/24/2022) (cited on page 12).

[56] Hispasec Sistemas. *Virus Total*. 2022. URL: https://www.virustotal.com (visited on 05/03/2022) (cited on page 13).

[57] Yuval Nativ @ytisf. *theZoo - A Live Malware Repository web site*. 2015. URL: http://ytisf.github.io/theZoo/ (cited on pages 13, 21).

[58] VirusShare. *VirusShare.com - Because Sharing is Caring*. https://virusshare.com/. 2011 (cited on pages 13, 14).

[59] M. Lindorfer et al. 'ANDRUBIS – 1,000,000 Apps Later: A View on Current Android Malware Behaviors'. In: *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. Sept. 2014, pp. 3–17. DOI: 10.1109/BADGERS.2014.7 (cited on page 14).

[60] Mila. *Contagio Mobile*. 2011. URL: https://contagiominidump.blogspot.com (cited on page 14).

[61] Kevin Allix et al. 'AndroZoo: Collecting Millions of Android Apps for the Research Community'. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. MSR '16. Austin, Texas: ACM, 2016, pp. 468–471. DOI: 10.1145/2901739.2903508 (cited on pages 14, 17, 63).

[62] Silas Cutler. *MalShare*. 2013. URL: https://malshare.com/about.php (cited on pages 14, 21).

[63] Koodous Team. *Koodous Project*. 2015. URL: https://koodous.com/about (visited on 05/04/2022) (cited on page 14).

[64] abuse.ch. *Malware Bazaar*. Bern University of Applied Sciences (BFH). June 1, 2022. URL: https://bazaar.abuse.ch (visited on 06/01/2022) (cited on page 14).

[65] Daniel Plohmann et al. 'Malpedia: A Collaborative Effort to Inventorize the Malware Landscape'. In: *The Journal on Cybercrime & Digital Investigations* 3.1 (2018) (cited on page 14).

[66] Hugo Gonzalez, Natalia Stakhanova, and Ali A. Ghorbani. 'DroidKin: Lightweight Detection of Android Apps Similarity'. In: *International Conference on Security and Privacy in Communication Networks*. Ed. by Jing Tian, Jiwu Jing, and Mudhakar Srivatsa. Cham: Springer International Publishing, 2015, pp. 436–453 (cited on page 14).

[67] Xiaohan Zhang et al. 'Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware'. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 757–770 (cited on pages 14, 19, 20).

[68] Valérie Viet Triem Tong et al. 'Isolating malicious code in Android malware in the wild'. In: *MALCON 2019 - 14th International Conference on Malicious and Unwanted Software*. MALCON 2019. Nantucket, United States: IEEE Computer society, Oct. 2019 (cited on page 14).

[69] Andi Fitriah Abdul Kadir, Natalia Stakhanova, and Ali Akbar Ghorbani. 'Android Botnets: What URLs are Telling Us'. In: *Network and System Security*. Ed. by Meikang Qiu et al. Cham: Springer International Publishing, 2015, pp. 78–91 (cited on page 15).

[70] Arash Habibi Lashkari et al. 'Towards a Network-Based Framework for Android Malware Detection and Characterization'. In: *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. 2017, pp. 233–23309. DOI: 10.1109/PST.2017.00035 (cited on page 15).

[71] Laya Taheri, Andi Fitriah Abdul Kadir, and Arash Habibi Lashkari. 'Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls'. In: *2019 International Carnahan Conference on Security Technology (ICCST)*. 2019, pp. 1–8. DOI: 10.1109/CCST.2019.8888430 (cited on page 15).

[72] Arash Habibi Lashkari et al. 'Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification'. In: *2018 International Carnahan Conference on Security Technology (ICCST)*. 2018, pp. 1–7. DOI: 10.1109/CCST.2018.8585560 (cited on page 15).

[73] Abir Rahali et al. 'DIDroid: Android Malware Classification and Characterization Using Deep Image Learning'. In: *2020 the 10th International Conference on Communication and Network Security*. ICCNS 2020. Tokyo, Japan: Association for Computing Machinery, 2020, pp. 70–82. DOI: 10.1145/3442520.3442522 (cited on page 15).

[74] David Sean Keyes et al. 'EntropLyzer: Android Malware Classification and Characterization Using Entropy Analysis of Dynamic Characteristics'. In: *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*. 2021, pp. 1–12. DOI: 10.1109/RDAAPS48126.2021.9452002 (cited on page 15).

[75] Hyunjae Kang et al. 'Detecting and Classifying Android Malware Using Static Analysis along with Creator Information'. In: *International Journal of Distributed Sensor Networks* 11.6 (2015), p. 479174. DOI: 10.1155/2015/479174 (cited on page 15).

[76] Jae-wook Jang et al. 'Andro-AutoPsy: Anti-malware system based on similarity matching of malware and malware creator-centric information'. In: *Digital Investigation* 14 (2015), pp. 17–35. DOI: http://dx.doi.org/10.1016/j.diin.2015.06.002 (cited on page 15).

[77] Jae-wook Jang et al. 'Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information'. In: *Computers & Security* 58 (2016), pp. 125–138. DOI: https://doi.org/10.1016/j.cose.2015.12.005 (cited on page 15).

[78] Jae-wook Jang and Huy Kang Kim. 'Function-Oriented Mobile Malware Analysis as First Aid'. In: *Mobile Information Systems* 2016 (2016). DOI: 10.1155/2016/6707524 (cited on page 15).

[79] Jae-wook Jang et al. 'Detecting and classifying method based on similarity matching of Android malware behavior with profile'. In: *SpringerPlus* 5.1 (Mar. 2016), p. 273. DOI: 10.1186/s40064-016-1861-x (cited on page 15).

[80] H. M. Kim et al. 'Andro-Simnet: Android Malware Family Classification using Social Network Analysis'. In: *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. Aug. 2018, pp. 1–8. DOI: 10.1109/PST.2018.8514216 (cited on page 15).

[81] Y. Zhou and X. Jiang. 'Dissecting Android Malware: Characterization and Evolution'. In: *2012 IEEE Symposium on Security and Privacy*. May 2012, pp. 95–109. DOI: 10.1109/SP.2012.16 (cited on page 15).

[82] Nicolas Kiss et al. 'Kharon Dataset: Android Malware under a Microscope'. In: *The LASER Workshop: Learning from Authoritative Security Experiment Results (LASER 2016)*. San Jose, CA: USENIX Association, May 2016, pp. 1–12 (cited on page 15).

[83] Kursat Aktas and Sevil Sen. 'UpDroid: Updated Android Malware and Its Familial Classification'. In: *Secure IT Systems - 23rd Nordic Conference, NordSec 2018, Oslo, Norway, November 28-30, 2018, Proceedings*. Ed. by Nils Gruschka. Vol. 11252. Lecture Notes in Computer Science. Springer, 2018, pp. 352–368. DOI: 10.1007/978-3-030-03638-6_22 (cited on page 16).

[84] Haoyu Wang et al. 'RmvDroid: Towards A Reliable Android Malware Dataset with App Metadata'. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 2019, pp. 404–408. DOI: 10.1109/MSR.2019.00067 (cited on page 16).

[85] Pei Liu et al. 'AndroZooOpen: Collecting Large-Scale Open Source Android Apps for the Research Community'. In: *Proceedings of the 17th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 548–552 (cited on page 16).

[86] Wen Li, Xiaoqin Fu, and Haipeng Cai. 'AndroCT: Ten Years of App Call Traces in Android'. In: *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 2021, pp. 570–574. DOI: 10.1109/MSR52588.2021.00076 (cited on pages 16, 76, 77, 79).

[87] Alejandro Guerra-Manzanares, Hayretdin Bahsi, and Sven Nõmm. 'KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization'. In: *Computers & Security* 110 (2021), p. 102399. DOI: https://doi.org/10.1016/j.cose.2021.102399 (cited on page 16).

[88] Liu Wang et al. 'MalRadar: Demystifying Android Malware in the New Era'. In: *Proc. ACM Meas. Anal. Comput. Syst.* 6.2 (2022). DOI: 10.1145/3530906 (cited on page 16).

[89] Shuaike Dong et al. 'Understanding Android Obfuscation Techniques: A Large-Scale Investigation in the Wild'. In: *Security and Privacy in Communication Networks*. Ed. by Raheem Beyah et al. Cham: Springer International Publishing, 2018, pp. 172–192. DOI: https://doi.org/10.1007/978-3-030-01701-9_10 (cited on page 17).

[90] Michael Grace et al. 'RiskRanker: Scalable and Accurate Zero-Day Android Malware Detection'. In: *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. MobiSys '12. Low Wood Bay, Lake District, UK: ACM, 2012, pp. 281–294. DOI: 10.1145/2307636.2307663 (cited on page 17).

[91] Hugo Gonzalez et al. 'Exploring Reverse Engineering Symptoms in Android Apps'. In: *Proceedings of the Eighth European Workshop on System Security*. EuroSec '15. Bordeaux, France: Association for Computing Machinery, 2015. DOI: 10.1145/2751323.2751330 (cited on page 17).

[92] Pasquale Battista et al. 'Identification of Android Malware Families with Model Checking'. In: *Proceedings of the 2nd International Conference on Information Systems Security and Privacy, ICISSP 2016, Rome, Italy, February 19-21, 2016*. Setúbal, Portugal: SciTePress, 2016, pp. 542–547. DOI: 10.5220/0005809205420547 (cited on page 17).

[93] Jonathan Crussell, Clint Gibler, and Hao Chen. 'AnDarwin: Scalable Detection of Semantically Similar Android Applications'. In: *Computer Security – ESORICS 2013*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 182–199 (cited on page 17).

[94] L. Li et al. 'IccTA: Detecting Inter-Component Privacy Leaks in Android Apps'. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. 2015, pp. 280–291. DOI: 10.1109/ICSE.2015.48 (cited on page 17).

[95] Y. Dodge et al. *The Oxford Dictionary of Statistical Terms*. Oxford University Press, 2006 (cited on page 18).

[96] Gary M Weiss and Foster Provost. *The effect of class distribution on classifier learning: an empirical study*. Tech. rep. ML-TR-44. Department of Computer Science, Rutgers University, 2001. DOI: \url{https://doi.org/doi:10.7282/t3-vpfw-sf95} (cited on page 18).

[97] Seba Susan and Amitesh Kumar. 'The balancing trick: Optimized sampling of imbalanced datasets—A brief survey of the recent State of the Art'. In: *Engineering Reports* (2020), e12298 (cited on page 18).

[98] Ivan Tomek. 'Two modifications of CNN'. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6.11 (1976), pp. 769–772. DOI: 10.1109/TSMC.1976.4309452 (cited on page 18).

[99] Nitesh V Chawla et al. 'SMOTE: synthetic minority over-sampling technique'. In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357 (cited on page 18).

[100] David Pfaff, Marie-Therese Walter, and Michael Backes. 'Code Clones Considered Harmful? Quantifying and Exploiting the Effects of Code Clones in Static Malware Classifiers for JavaScript'. In: *Proceedings of the 2019 14th International Conference on Malicious and Unwanted Software*. 2019, pp. 111–118 (cited on page 18).

[101] Feng Pan et al. 'Finding Representative Set from Massive Data'. In: *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*. IEEE Computer Society, 2005, pp. 338–345. DOI: 10.1109/ICDM.2005.69 (cited on page 18).

[102] Xiaoqin Fu and Haipeng Cai. 'On the Deterioration of Learning-Based Malware Detectors for Android'. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 2019, pp. 272–273. DOI: 10.1109/ICSE-Companion.2019.00110 (cited on page 19).

[103] Haipeng Cai and John Jenkins. 'Towards Sustainable Android Malware Detection'. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ICSE '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 350–351. DOI: 10.1145/3183440.3195004 (cited on page 19).

[104] Haipeng Cai. 'Assessing and Improving Malware Detection Sustainability through App Evolution Studies'. In: *ACM Trans. Softw. Eng. Methodol.* 29.2 (Mar. 2020). DOI: 10.1145/3371924 (cited on page 19).

[105] Haipeng Cai. 'Embracing Mobile App Evolution via Continuous Ecosystem Mining and Characterization'. In: *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*. MOBILESoft '20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 31–35. DOI: 10.1145/3387905.3388612 (cited on pages 19, 79).

[106] Alejandro Guerra-Manzanares, Marcin Luckner, and Hayretdin Bahsi. 'Concept drift and cross-device behavior: Challenges and implications for effective android malware detection'. In: *Computers & Security* 120 (2022), p. 102757. DOI: https://doi.org/10.1016/j.cose.2022.102757 (cited on page 19).

[107] Ke Xu et al. 'DroidEvolver: Self-Evolving Android Malware Detection System'. In: *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. 2019, pp. 47–62. DOI: 10.1109/EuroSP.2019.00014 (cited on page 20).

[108] A. Saracino et al. 'MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention'. In: *IEEE Transactions on Dependable and Secure Computing* 15.1 (Jan. 2018), pp. 83–97. DOI: 10.1109/TDSC.2016.2536605 (cited on page 21).

[109] Z. Yuan, Y. Lu, and Y. Xue. 'Droiddetector: android malware characterization and detection using deep learning'. In: *Tsinghua Science and Technology* 21.1 (Feb. 2016), pp. 114–123. DOI: 10.1109/TST.2016.7399288 (cited on page 21).

[110] K. Xu, Y. Li, and R. H. Deng. 'ICCDetector: ICC-Based Malware Detection on Android'. In: *IEEE Transactions on Information Forensics and Security* 11.6 (June 2016), pp. 1252–1264. DOI: 10.1109/TIFS.2016.2523912 (cited on pages 21, 63).

[111] Ali Feizollah et al. 'AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection'. In: *Computers & Security* 65 (2017), pp. 121–134. DOI: https://doi.org/10.1016/j.cose.2016.11.007 (cited on pages 21, 63).

[112] Lucky Onwuzurike et al. 'MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models'. In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017 (cited on page 21).

[113] Sen Chen et al. 'Stormdroid: A streaminglized machine learning-based system for detecting android malware'. In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM. 2016, pp. 377–388 (cited on page 21).

[114] Shweta Bhandari et al. 'SWORD: Semantic aWare andrOid malwaRe Detector'. In: *J. Inf. Secur. Appl.* 42 (2018), pp. 46–56. DOI: 10.1016/j.jisa.2018.07.003 (cited on page 21).

[115] Mansour Ahmadi, Angelo Sotgiu, and Giorgio Giacinto. 'IntelliAV: Toward the Feasibility of Building Intelligent Anti-malware on Android Devices'. In: *Machine Learning and Knowledge Extraction*. Ed. by Andreas Holzinger et al. Cham: Springer International Publishing, 2017, pp. 137–154 (cited on page 21).

[116] Ignacio Martín et al. 'Android Malware Characterization Using Metadata and Machine Learning Techniques'. In: *Security and Communication Networks* 2018 (2018), 5749481:1–5749481:11. DOI: 10.1155/2018/5749481 (cited on page 21).

[117] Shifu Hou et al. 'HinDroid: An Intelligent Android Malware Detection System Based on Structured Heterogeneous Information Network'. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 1507–1515. DOI: 10.1145/3097983.3098026 (cited on page 21).

[118] Joshua Garcia, Mahmoud Hammad, and Sam Malek. 'Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware'. In: *ACM Trans. Softw. Eng. Methodol.* 26.3 (Jan. 2018). DOI: 10.1145/3162625 (cited on page 21).

[119] A. Demontis et al. 'Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection'. In: *IEEE Transactions on Dependable and Secure Computing* 16.4 (July 2019), pp. 711–724. DOI: 10.1109/TDSC.2017.2700270 (cited on page 21).

[120] Niall McLaughlin et al. 'Deep Android Malware Detection'. In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. CODASPY '17. Scottsdale, Arizona, USA: Association for Computing Machinery, 2017, pp. 301–308. DOI: 10.1145/3029806.3029823 (cited on page 21).

[121] Mohammed K. Alzaylaee, Suleiman Y. Yerima, and Sakir Sezer. 'EMULATOR vs REAL PHONE: Android Malware Detection Using Machine Learning'. In: *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics*. IWSPA '17. Scottsdale, Arizona, USA: Association for Computing Machinery, 2017, pp. 65–72. DOI: 10.1145/3041008.3041010 (cited on pages 21, 62).

[122] M. Melis et al. 'Explaining Black-box Android Malware Detection'. In: *2018 26th European Signal Processing Conference (EUSIPCO)*. Sept. 2018, pp. 524–528. DOI: 10.23919/EUSIPCO.2018.8553598 (cited on page 21).

[123] Annamalai Narayanan et al. 'A multi-view context-aware approach to Android malware detection and malicious code localization'. In: *Empirical Software Engineering* 23.3 (2018), pp. 1222–1274. DOI: 10.1007/s10664-017-9539-8 (cited on page 21).

[124] C. Li et al. 'Android Malware Detection Based on Factorization Machine'. In: *IEEE Access* 7 (2019), pp. 184008–184019. DOI: 10.1109/ACCESS.2019.2958927 (cited on page 21).

[125] Guozhu Meng et al. 'Semantic Modelling of Android Malware for Effective Malware Comprehension, Detection, and Classification'. In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ISSTA 2016. Saarbrücken, Germany: Association for Computing Machinery, 2016, pp. 306–317. DOI: 10.1145/2931037.2931043 (cited on page 21).

[126] Xi Xiao et al. 'Android malware detection based on system call sequences and LSTM'. In: *Multimedia Tools and Applications* 78.4 (2019), pp. 3979–3999 (cited on page 21).

[127] H. Cai et al. 'DroidCat: Effective Android Malware Detection and Categorization via App-Level Profiling'. In: *IEEE Transactions on Information Forensics and Security* 14.6 (2019), pp. 1455–1470 (cited on page 21).

[128] Md Shohel Rana, Charan Gudla, and Andrew H Sung. 'Android malware detection using stacked generalization'. In: *27th International Conference on Software Engineering and Data Engineering*. 2018 (cited on page 21).

[129] Z. Ma et al. 'A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms'. In: *IEEE Access* 7 (2019), pp. 21235–21245. DOI: 10.1109/ACCESS.2019.2896003 (cited on page 21).

[130] Na Huang et al. 'Deep Android Malware Classification with API-Based Feature Graph'. In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE. 2019, pp. 296–303 (cited on page 21).

[131] Xiaolei Liu et al. 'Adversarial Samples on Android Malware Detection Systems for IoT Systems'. In: *Sensors* 19.4 (2019). DOI: 10.3390/s19040974 (cited on page 21).

[132] Yanfang Ye et al. 'Out-of-sample Node Representation Learning for Heterogeneous Graph in Real-time Android Malware Detection'. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 4150–4156. DOI: 10.24963/ijcai.2019/576 (cited on page 21).

[133] Fabio Martinelli, Francesco Mercaldo, and Andrea Saracino. 'BRIDEMAID: An Hybrid Tool for Accurate Detection of Android Malware'. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '17. Abu Dhabi, United Arab Emirates: Association for Computing Machinery, 2017, pp. 899–901. DOI: 10.1145/3052973.3055156 (cited on page 21).

[134] Daniel Arp et al. 'Dos and Don'ts of Machine Learning in Computer Security'. In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3971–3988 (cited on page 22).

[135] Tamara Munzner. *Visualization Analysis and Design*. AK Peters Visualization Series. CRC Press, 2015 (cited on pages 28, 38, 39, 41, 44).

[136] InSeon Yoo. 'Visualizing Windows Executable Viruses Using Self-Organizing Maps'. In: *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*. VizSEC/DMSEC '04. Washington DC, USA: Association for Computing Machinery, 2004, pp. 82–89. DOI: 10.1145/1029208.1029222 (cited on page 28).

[137] Thomas Panas. 'Signature Visualization of Software Binaries'. In: *Proceedings of the 4th ACM Symposium on Software Visualization*. SoftVis '08. Ammersee, Germany: Association for Computing Machinery, 2008, pp. 185–188. DOI: 10.1145/1409720.1409749 (cited on page 29).

[138] Daniel A. Quist and Lorie M. Liebrock. 'Visualizing compiled executables for malware analysis'. In: *2009 6th International Workshop on Visualization for Cyber Security*. 2009, pp. 27–32. DOI: 10.1109/VIZSEC.2009.5375539 (cited on page 29).

[139] Ashutosh Jain, Hugo Gonzalez, and Natalia Stakhanova. 'Enriching reverse engineering through visual exploration of Android binaries'. In: *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*. PPREW-5. Los Angeles, CA, USA: Association for Computing Machinery, 2015. DOI: 10.1145/2843859.2843866 (cited on page 29).

[140] Ganesh Ram Santhanam et al. 'Interactive Visualization Toolbox to Detect Sophisticated Android Malware'. In: *2017 IEEE Symposium on Visualization for Cyber Security (VizSec)*. 2017, T-4 (1–8) (cited on page 30).

[141] Jean-François Lalande, Mathieu Simon, and Valérie Viet Triem Tong. 'GroDDViewer: Dynamic Dual View of Android Malware'. In: *The Seventh International Workshop on Graphical Models for Security*. Virtual Conference, France: Springer, June 2020 (cited on pages 30, 31).

[142] Sangbong Yoo et al. 'Personal Visual Analytics for Android Security Risk Lifelog'. In: *Proceedings of the 10th International Symposium on Visual Information Communication and Interaction*. VINCI '17. Bangkok, Thailand: Association for Computing Machinery, 2017, pp. 29–36. DOI: 10.1145/3105971.3105975 (cited on page 31).

[143] Sangbong Yoo et al. 'Visual analytics and visualization for android security risk'. In: *Journal of Computer Languages* 53 (2019), pp. 9–21. DOI: https://doi.org/10.1016/j.cola.2019.03.004 (cited on page 31).

[144] Sen Chen et al. 'StoryDroid: Automated Generation of Storyboard for Android Apps'. In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 2019, pp. 596–607 (cited on page 31).

[145] John Jenkins and Haipeng Cai. 'Dissecting Android Inter-Component Communications via Interactive Visual Explorations'. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2017, pp. 519–523 (cited on page 31).

[146] Josh Saxe, David Mentis, and Chris Greamo. 'Visualization of Shared System Call Sequence Relationships in Large Malware Corpora'. In: *Proceedings of the Ninth International Symposium on Visualization for Cyber Security*. VizSec '12. Seattle, Washington, USA: Association for Computing Machinery, 2012, pp. 33–40. DOI: 10.1145/2379690.2379695 (cited on pages 32, 33).

[147] Robert Gove et al. 'SEEM: A Scalable Visualization for Comparing Multiple Large Sets of Attributes for Malware Analysis'. In: *Proceedings of the Eleventh Workshop on Visualization for Cyber Security*. VizSec '14. Paris, France: Association for Computing Machinery, 2014, pp. 72–79. DOI: 10.1145/2671491.2671496 (cited on pages 32, 33).

[148] Ludovic Apvrille and Axelle Apvrille. 'Pre-filtering mobile malware with heuristic techniques'. In: *The 2nd International Symposium on Research in Grey-Hat Hacking (GreHack)*. 2013 (cited on pages 36, 62).

[149] Bilal Alsallakh et al. 'The State-of-the-Art of Set Visualization'. In: *Computer Graphics Forum* 35.1 (2016), pp. 234–260. DOI: https://doi.org/10.1111/cgf.12722 (cited on pages 41, 42).

[150] F-Secure. *Trojan:Android/Geinimi*. 2010. URL: https://www.f-secure.com/v-descs/trojan_android_geinimi.shtml (visited on 06/23/2022) (cited on page 48).

[151] FortiGuard Labs. *Android/GingerMaster.A!tr*. Sept. 24, 2011. URL: https://www.fortiguard.com/encyclopedia/virus/2992120 (visited on 06/23/2022) (cited on page 50).

[152] Gian Luca Scoccia et al. 'Leave My Apps Alone! A Study on How Android Developers Access Installed Apps on User's Device'. In: *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*. MOBILESoft '20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 38–49. DOI: 10.1145/3387905.3388594 (cited on page 50).

[153] Anh Pham et al. 'HideMyApp: Hiding the Presence of Sensitive Apps on Android'. In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 711–728 (cited on page 50).

[154] William G Cochran. *Sampling Techniques*. third. John Wiley & Sons, 1977 (cited on pages 55, 57).

[155] F. Tchakounté and F. Hayata. 'Chapter 6 - Supervised Learning Based Detection of Malware on Android'. In: *Mobile Security and Privacy*. Ed. by Man Ho Au and Kim-Kwang Raymond Choo. Boston: Syngress, 2017, pp. 101–154. DOI: https://doi.org/10.1016/B978-0-12-804629-6.00006-7 (cited on page 56).

[156] Richard D. De Veaux, Paul F. Velleman, and David E. Bock. *Stats: Data and Models*. Pearson, 2016 (cited on pages 57, 62).

[157] Robert V Hogg, Elliot A Tanis, and Dale L Zimmerman. *Probability and Statistical Inference*. Pearson, 2015 (cited on page 58).

[158] Rémi Bardenet, Odalric-Ambrym Maillard, et al. 'Concentration inequalities for sampling without replacement'. In: *Bernoulli* 21.3 (2015), pp. 1361–1385 (cited on page 58).

[159] Olive Jean Dunn. 'Multiple Comparisons among Means'. In: *Journal of the American Statistical Association* 56.293 (1961), pp. 52–64 (cited on page 58).

[160] Wei Yang et al. 'Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps'. In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACSAC 2017. Orlando, FL, USA: Association for Computing Machinery, 2017, pp. 288–302. DOI: 10.1145/3134600.3134642 (cited on page 62).

[161] Jean-François Lalande et al. 'Challenges for Reliable and Large Scale Evaluation of Android Malware Analysis'. In: *2018 International Conference on High Performance Computing & Simulation (HPCS)*. 2018, pp. 1068–1070. DOI: 10.1109/HPCS.2018.00173 (cited on page 62).

[162] Karl Pearson. 'X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling'. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50.302 (1900), pp. 157–175 (cited on page 62).

[163] Steven Arzt et al. 'FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps'. In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '14. Edinburgh, United Kingdom: Association for Computing Machinery, 2014, pp. 259–269. DOI: 10.1145/2594291.2594299 (cited on page 63).

[164] Ming Fan et al. 'Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis'. In: *IEEE Transactions on Information Forensics and Security* 13.8 (2018), pp. 1890–1905. DOI: 10.1109/TIFS.2018.2806891 (cited on page 77).

[165] Evelyn Fix and Joseph Lawson Hodges. 'Discriminatory analysis. Nonparametric discrimination: Consistency properties'. In: *International Statistical Review/Revue Internationale de Statistique* 57.3 (1989), pp. 238–247 (cited on page 78).

[166] Wei-Yin Loh. 'Classification and regression trees'. In: *Wiley interdisciplinary reviews: data mining and knowledge discovery* 1.1 (2011), pp. 14–23 (cited on page 78).

[167] Tin Kam Ho. 'Random decision forests'. In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282 (cited on page 78).

[168] David J Hand and Keming Yu. 'Idiot's Bayes—not so stupid after all?' In: *International statistical review* 69.3 (2001), pp. 385–398 (cited on page 78).

[169] Yoav Freund and Robert E Schapire. 'A decision-theoretic generalization of on-line learning and an application to boosting'. In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139 (cited on page 78).

[170] F. Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cited on page 78).

[171] Haipeng Cai and Barbara Ryder. 'A Longitudinal Study of Application Structure and Behaviors in Android'. In: *IEEE Transactions on Software Engineering* 47.12 (2021), pp. 2934–2955. DOI: 10.1109/TSE.2020.2975176 (cited on page 79).

[172] Haipeng Cai, Xiaoqin Fu, and Abdelwahab Hamou-Lhadj. 'A study of run-time behavioral evolution of benign versus malicious apps in android'. In: *Information and Software Technology* 122 (2020), p. 106291. DOI: https://doi.org/10.1016/j.infsof.2020.106291 (cited on page 79).

[173] Hao Zhou et al. 'Analysis of Android Malware Family Characteristic Based on Isomorphism of Sensitive API Call Graph'. In: *2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)*. 2017, pp. 319–327. DOI: 10.1109/DSC.2017.77 (cited on page 84).

[174] Robert J. Joyce et al. 'MOTIF: A Malware Reference Dataset with Ground Truth Family Labels'. In: *Computers & Security* (2022), p. 102921. DOI: https://doi.org/10.1016/j.cose.2022.102921 (cited on pages 84, 85).

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

CentraleSupélec

Titre : Profilage et Visualisation de Datasets d'Applications Android Malveillantes

Mot clés : Malware, Datasets, Biais, Visualization

Résumé : Les dispositifs mobiles sont ubiquitaires: aujourd'hui la majorité des gens possèdent un téléphone mobile. A cause de ce fait, ces dispositifs sont une cible d'intérêt pour les attaquants. Ces attaques sont véhiculées au travers des applications malveillantes qui peuvent nuire aux dispositifs mobiles. Les chercheurs en analyse de malware travaillent à reconnaître ces types de programmes avant qu'ils soient installés sur un dispositif utilisateur. Pour faire cela, ils réalisent des expériences pour automatiquement détecter ces malware, où ils utilisent des ensembles de malware et des applications bénignes déjà connues. Selon le dataset choisi, les résultats des expériences peuvent être acceptables ou bien exceptionnellement bons parce que surestimés. Par conséquent, les datasets de malware et applications bénignes sont des éléments importants à considérer quand nous élaborons une expérience.

Cette thèse présente, premièrement, une méthode pour évaluer la qualité des datasets basée sur un test statistique qui aide à comparer un dataset créé avec un grand ensemble d'applications par exemple issu d'un magasin d'applications. Nous montrons alors que les datasets historiques de la littérature sont de mauvaise qualité, ce qui justifie le besoin de créer des nouveaux datasets plus à jour. Deuxièmement, nous introduisons un algorithme pour mettre à jour des datasets mixtes de malware/goodware de mauvaise qualité afin de ressembler à un dataset cible qui ne peut pas être utilisé directement, *e.g.* un magasin d'applications. Nous évaluons les datasets mixtes mis à jour en utilisant un algorithme d'apprentissage automatique et nous montrons que la détection de malware sur notre dataset mis à jour devient un problème plus difficile à résoudre. Enfin, nous introduisons DaViz, un outil de visualisation de datasets pour explorer et comparer des datasets d'applications Android. Cet outil permet aux chercheurs de visualiser les biais dans les datasets de la littérature, et d'obtenir des informations utiles à leur propos.

Title: Profiling and Visualizing Android Malware Datasets

Keywords: Malware, Datasets, Bias, Visualization

Abstract: Mobile devices are ubiquitous: nowadays most people own a mobile telephone. Because of this, it is a target of interest for attackers. Researchers in malware analysis put their effort to recognize these types of programs before they are installed on a user device. To do this, they perform experiments to automatically detect malware, for example with machine learning, where they use sets of already known malware and goodware. Depending on their choice of datasets, the evaluation of the experiments can yield acceptable results, or outstanding but overestimated results. Consequently, datasets with malware and benign samples are important elements to consider when designing an experiment.

This thesis presents, first, a method to evaluate the quality of datasets based on a statistical test that helps to compare a crafted dataset against a large set of applications such as markets. We show that historical datasets of the literature are of low quality, which justifies the need to create new up-to-date datasets. Second, we introduce an algorithm to update mixed datasets of malware/goodware of low quality in order to resemble a target dataset that cannot be used directly, *e.g.* a market. We evaluate the updated mixed datasets using a machine learning algorithm and we show that the detection of malware in our up-to-date dataset becomes a more difficult problem to solve. Lastly, we introduce DaViz, a dataset visualization tool for exploring and comparing Android malware datasets, which enables researchers to visualize the biases in datasets of the literature, and obtain useful information from them.