Journal of King Saud University – Computer and Information Sciences 34 (2022) 6883–6894

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

# Probing AndroVul dataset for studies on Android malware classification

Namrud Zakeya [a,*], Kpodjedo Ségla [a], Talhi Chamseddine [a], Boaye Belle Alvine [b]

[a] École de Technologie Supérieure, Department of Software and IT Engineering, Montreal H3C 1K3, QC, Canada
[b] York University, Department of Electrical Engineering and Computer Science, Toronto, ON, Canada

ARTICLE INFO

*Article history:*
Received 12 May 2021
Revised 30 July 2021
Accepted 29 August 2021
Available online 22 September 2021

*Keywords:*
Mobile security
Static analysis
Reverse engineering
Mobile computing
Machine learning

ABSTRACT

Security issues in mobile apps are increasingly relevant as this software have become part of the daily life of billions of people. As the dominant OS, Android is a primary target for ill-intentioned programmers willing to exploit its vulnerabilities by spreading malwares. Significant research has been devoted to the identification of these malwares. The current paper is an extension of our previous effort to contribute to said research with a new benchmark of Android vulnerabilities. We proposed AndroVul, a repository for Android security vulnerabilities, that builds on AndroZoo – a well-known Android app dataset – and contains data on vulnerabilities for a representative sample of about 16,000 Android apps. The present paper adds confirmed malwares from the VirusShare dataset and explores more thoroughly the effectiveness of different machine learning techniques, with respect to the classification of malicious apps. We investigated different classifiers and feature selection techniques as well as different combinations for our input data. Our results suggest that the classifier MPL is the leading classifier, with competitive results that favorably compare to recent malware detection work. Additionally, we investigate how to classify (as benign or malicious) AndroZoo apps based on the number of antivirus flags they are tagged with. We found that different thresholds only marginally affect the machine learning classifier results and that the strictest choice (i.e. one flag) performs best on the confirmed malwares from VirusShare.


## 1. Introduction

With a market share of 73%.[1] Android is undoubtedly the leading mobile Operating System (OS) of our times. Besides, thanks to its openness, it is well-positioned to become the default OS for new applications centered on the Internet of Things (IoT). Security aspects are thus increasingly relevant as there are more incentives for malware developers to target Android devices. Android security has accordingly been extensively researched and that effort has been helped by the AndroZoo[2] dataset put together by Allix et al. (2016) in 2016. The AndroZoo dataset is a very useful resource for Android researchers in general but security researchers still have many hurdles to pass from the moment they discover AndroZoo to the moment they can effectively get actionable data from it.

In Namrud et al. (2019), we proposed AndroVul,[3] a repository aiming to provide researchers working on anomaly detection of Android applications with: i) a benchmark readily usable (to test hypotheses), and ii) tool that will jumpstart their data collection. Our dataset includes data on 16,180 applications from Google Play store as well as third party stores (Allix et al., 2016). Our tool extracted from these apps' binaries called APKs (Android Package Kits): 1) permissions classified as dangerous by the Android permission system; 2) data collected via AndroBugs, a popular Android vulnerability scanner[4]; and 3) security code smells, as recently defined by Gadient et al. (2017). Furthermore, we proposed preliminary experiments aiming at probing the predictive power of these various sources of vulnerability and found that it was preferable to include all of them, with dangerous permissions data being the best input.

The present paper builds on that previous work with an extensive investigation of different classifiers and feature selection techniques applied to various setups of our benchmark.

* Corresponding author.
*E-mail addresses:* zakeya.namrud.1@ens.etsmtl.ca (N. Zakeya), segla.kpodjedo@etsmtl.ca (K. Ségla), chamseddine.talhi@etsmtl.ca (T. Chamseddine), alvine.belle@lassonde.yorku.ca (B.B. Alvine).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

[1] http://gs.statcounter.com/os-market-share/mobile/worldwide.
[2] https://AndroZoo.uni.lu/.
[3] https://github.com/Zakeya/AndroVul.
[4] https://www.androbugs.com/.

https://doi.org/10.1016/j.jksuci.2021.08.033
1319-1578/

A key question that many researchers face when using AndroZoo for malware detection/ classification purposes relates to the labeling of the apps: which ones should be considered malwares? AndroZoo only gives the number of antivirus flags an app got on the VirusTotal[5] website and it is up to the researchers to know how to use that information.

Arp et al. (2014) – an influential paper published in 2016 – answered that question by classifying as a malware any app with at least 2 flags from a set of ten selected antivirus products. Since then, the set of antiviruses from VirusTotal has grown so the relevance and completeness of that initial set of ten products is questionable. More importantly, the number of antivirus flags is the information that is readily available from the main AndroZoo file; there are no information on which antiviruses flagged an app. Typically, papers who consider flags from a preselected group of antiviruses from VirusTotal are not based on AndroZoo data. Papers based on AndroZoo usually settle on a threshold for the number of antivirus flags as a way to decide whether an app should be considered a malware or not. A standard threshold is 2: any app with at least 2 antivirus flags is considered a malware (Arp et al., 2014). However, the literature also includes papers using thresholds such as 1 (Li et al., 2017), 28 (Li et al., 2017), 8 (Li et al., 2016), or half the antiviruses (Wei et al., 2017). Hence, there is no firm consensus on the threshold of antivirus flags starting from which an app in the AndroZoo dataset should be considered a malware.

In any case, this is an important question, with which we struggled as malware researchers and which the current paper investigates through various setups of our dataset, which we extended, since (Namrud et al., 2019), with confirmed Android malwares from VirusShare, a prominent repository of malware samples. In particular, the addition of malwares from VirusShare serves as an interesting addition in the way of testing the effectiveness of classifiers built using our dataset and different malware thresholds.

Moreover, we take interest in assessing possible performance differences between well-known machine learning techniques (e.g., JRIP, Naive Bayes, MLP, and J48). We also included feature selection options for a finer granularity of analysis as this – sometimes overlooked – step has proven to be a significant addition in some studies (Bhattacharya and Goswami, 2018). In short, while our first effort focused on proposing AndroVul and investigating the input data (the various vulnerability metrics), the current study expands the previous investigation on two fronts: i) the labeling (as benign or malware) of Androzoo apps; and ii) the effectiveness of the treatment (with machine learning) of the input.

The current paper aims to be self-contained and as such, it includes elements from Namrud et al. (2019). The rest of the paper is organized as follows: Sections 2 and 3 propose relevant background notions and related work on app datasets and malware classification. Section 4 and Section 5 present the tool and datasets in AndroVul. In Section 6, we lay out our study from preliminary statistical analyses to the definition of research questions and investigations aiming at answering them. Section 7 presents and discusses our results and findings. We then discuss some of the limitations and threats to validity to the present work in Section 8. Finally, Section 9 concludes the paper and outlines some future work.

## 2. Background

In this section, we briefly present the possible security vulnerabilities our dataset focuses on: dangerous permissions (as defined by the Android system), troubling code attributes (as collected by AndroBugs) and security code smells (as proposed in Gadient et al. (2017)).

### 2.1. Vulnerability

A vulnerability is a soft spot in a system, that places it in danger of an attack by hackers, viruses or any other event that will lead to breaking the security of any system (Mansourov and Campara, 2010). Mansourov and Campara (2010) define it as "a certain unit of knowledge about a fault in a system that allows exploiting this system in unauthorized and possibly even malicious ways". A fault in a system can be triggered by several factors such as human error, poor specifications of requirements, the use of processes that are poorly developed, the use of technologies that evolve rapidly, or even the poor understanding of threats. Malicious software also known as malware are a means to exploit faults in a system. Malware are usually referred to as viruses, worms, trojan horses, backdoors, keystroke loggers, rootkits, or spyware.

### 2.2. Dangerous permissions

A key security mechanism of Android is its permission system, which controls the privileges of apps, making it so that apps must request specific permissions in order to perform specific functions. This mechanism requires that app developers declare which sensitive resources will be used by their apps. App users have to agree with the requests when installing or using the apps. Android defines several categories of permissions, among which"dangerous" ones, deemed more critical and privacy sensitive since they provide access to system features such as the camera, Internet, personal contacts, SMS, etc. Table 8 in the appendix proposes a list of the various dangerous permissions as defined by the official Android developer resource.[6]

### 2.3. AndroBugs

Androbugs is a popular security testing tool that checks Android apps for vulnerabilities and potentially critical security issues. The tool reverse engineers APKs and looks for various issues, from failures to adhere to best practices to the use of dangerous shell commands or exposure to vulnerabilities from third party libraries. AndroBugs has a proven track record of discovering security vulnerabilities in some of the most popular apps or SDKs. It is a command line tool that issues reports with four severity levels: Critical.[7], Warning.[8] Notice[9] and InfofootnoteNo security issue detected.

### 2.4. Security code smells

"Code smells" refer to code source elements that may indicate deeper problems (Shezan et al., 2017). In Gadient et al. (2017), Ghafari et al. introduced security code smells in Android apps as "symptoms in the code that signal the prospect of a security vulnerability". After reviewing the literature, they identified 28 security code smells (Gadient et al., 2017) that they regrouped into five categories, such as Insufficient Attack Protection, Security Invalidation, Broken Access Control, Sensitive Data Exposure, and Lax Input Validation.

[5] https://www.virustotal.com/gui/.

[6] https://developer.android.com.

[7] Confirmed vulnerability that should be solved (except for testing code)

[8] Possible vulnerability that should be checked by developers.

[9] Low priority issue.

## 3. Related work

### 3.1. Dataset

Dataset availability is a key issue when it comes to getting insights about a topic or evaluating approaches or hypotheses. We briefly present below some of the most notable efforts related to this issue in the context of Android apps and more specifically their possible security concerns.

A number of repositories have been proposed over the years for the study of mobile apps. F-Droid[10] is such an effort; it is a repository of free open source Android apps that have been used in an impressive number of studies. Recently, Allix et al. (2016) have proposed and continued to maintain AndroZoo, certainly the largest Android app repository, with millions of apps (and APKs) from the Google Play store and other third party markets. Even more recently, Geiger et al. (2018) made available a graph–based database with information (metadata, commit and code history) on 8,431 open-source Android apps available on GitHub and the Google Play Store. Also notable, although slightly older, is Krutz et al. (2015), with a public dataset centered on the lifecycle of 1,179 Android apps from F-Droid. Complementary to these research initiatives, there are a number of websites such as AppAnnie and Koodous that gather Android apps and perform various types of analyses, including downloads over time and advertising analytics.

When it comes to security aspects, there have been a number of papers investigating large numbers of Android apps but few propose publicly available data-sets. Among those, we can cite Munaiah et al. (2016) which propose data (e.g., app category, permissions) on reverse engineered benign applications from Google Play store and malware applications from several sources.

Our dataset on vulnerabilities of Android apps shares some similarity with the work of Gkortzis et al. (2018), which also proposes a dataset of security vulnerabilities but for open source systems (8,694). Similar to Krutz et al. (2015), we propose a subset of a well-known mobile app repository; we start with AndroZoo while Krutz et al. (2015) builds on F-Droid. Similar to Krutz et al. (2015), we also propose tool that interface with well-known reverse engineering and static analysis tools, but we do so with a focus on security vulnerabilities and use a different set of tools. Overall, our dataset and tool propose a unique offering for Android security researchers.

As touched upon in the introduction, malware research using AndroZoo involve deciding on which of the apps present in the benchmark can be considered malwares. Different approaches are present in the literature. They fall mostly into two categories based on whether they rely on a preselected subset of antivirus results from VirusTotal or a given number of antivirus flags as reported by AndroZoo data.

Zheng et al. (2012) focused on the top ten anti-virus products from VirusTotal that flagged apps as malware. Their results were subsequently used by Shen et al. (2014) to evaluate the effectiveness of antivirus tools against malware obfuscation. In Yousefi-Azar et al. (2018), Yousefi-Azar et al. considered nineteen of the most well-known antiviruses, including Kaspersky, Symantec, Avast, McAfee, AVG, Malwarebytes, etc.

Also notable is the work of Ma et al. (2019), which considered, as malwares, apps that were flagged by four well-established antiviruses (i.e. McAfee, 360 Security Guard, Kingsoft Antivirus, Norton). Some other approaches based their analysis on the number of antivirus that flagged an app as possible malware. Li et al. (2017) considered that even one flag was enough to classify an app as malware. Other studies were more lenient, with Li et al. (2016) needing 8 antivirus flags before deciding that an app is a malware, and Wei et al. (2017) needing flags from at least 50% of the anti-viruses of VirusTotal before recognizing an app as a malware. Different from these works, our paper attempts to evaluate the impact of different thresholds for malware labelling in a machine learning context.

### 3.2. Malware classification with machine learning

A primary intended use of our dataset will be as input for malware detection techniques. Malware detection approaches generally use some form of machine learning to classify candidate apps as malicious or not. The existing literature is quite extensive on that subject. In this section, we will focus on the work that is the most recent and the closest to our experiments.

Permissions, especially dangerous ones, have been used in lots of studies as inputs to various classification approaches. A particular recent work of interest is the one of Bhattacharya and Goswami (2017), which proposed a framework that gathered permissions from apps' manifest files and applied advanced feature selection techniques. The features obtained from such process were organised into four groups and used as input for fifteen different machine learning classifiers (including JRip, J48, MPL, and NB) from Weka. The authors evaluated their approach on a sample of 170 apps and reported the highest accuracy to be 77.13%. Many other research work investigated features other than permissions for malware detection purposes. For instance, Sharma and Sahay (2018) tried to leverage Dalvik[11] opcode occurrences for malware classification purposes. They selected 5531 android malwares from the DREBIN repository (Arp et al., 2014) and 2691 benign apps from the Google Play Store. They applied different machine learning techniques and reported the best detection accuracy obtained to be 79.27%. Also of interest is the work of Sharma and Sahay (2018), which focused on features extracted through Mobile Security Framework, an open source tool dedicated to mobile app security.[12] The approach proposed in Sharma and Sahay (2018) aimed at classifying apps with respect to three levels (Safe, Suspicious, Highly Suspicious). The reported experiments involved many refined machine learning classifiers applied on a corpus of 13,850 Android apps, with accuracy results up to 81.80% when considering the three proposed levels and up to 93.63% when considering a binary benign/ malicious decision.

DroidDeepLearner is a weighted malware detection technique proposed by Li et al. (2018). The method employs both dangerous API calls and risky permission combinations as features in order to build a Deep Belief Network model capable of automatically distinguishing malware from benign ones. Their method achieves over 90% accuracy with 237 features on the Drebin dataset, according to the findings.

Authors Lee et al. (2020) investigated whether the dangerous permissions are a key component of detection when determining whether an app is malicious or benign. They used a total of 10,818 malicious and benign apps. To determine the accuracy of the detection, they used four separate deep learning algorithms and measured them using the confusion matrix. The selected features resulted in about 90% accuracy. We are different then both Li et al. (2018) and Lee et al. (2020), they focus only in dangerous permissions while we investigated the use of vulnerabilities in different levels including dangerous permissions.

## 4. AndroVul-T: the tool

Our repository proposes a tool that allows, given a directory of APKs, the automatic generation of a CSV file with information on

[10] https://f-droid.org/.

[11] Dalvik is a now discontinued process virtual machine in Google's Android OS.

[12] https://github.com/MobSF/Mobile-Security-Framework-MobSF.

the apps vulnerabilities. To accommodate statistical analysis, each vulnerability corresponds to a column, to which we attach some quantitative data indicating its presence (for dangerous permissions), the certainty behind it (for AndroBugs vulnerabilities), or its weight (for security code smells).

Fig. 1 proposes an overview of the inner workings of our tool, which makes use of very well-known tools to reverse engineer any APK. The tool APKtool[13] is applied on a given APK to reverse engineer its manifest file and Smali code, which is basically a human readable description of the binary code (contained in a.dex file). Similarly, via our tool, an APK can be given to the AndroBugs tool in order to generate a report on its potential security vulnerabilities.

Once we get these three artefacts from AndroBugs and Apktool, our tool proceeds on parsing dangerous permissions from the Android Manifest, various vulnerabilities from AndroBugs and security code smells from the Smali code. Our treatment of the extracted information is illustrated in Fig. 2 and further detailed in the subsections below.

### 4.1. Dangerous permissions extraction

Extracting dangerous permissions is a relatively straightforward process, after which we fill in the csv file information about the presence (1) or absence (0) of any dangerous permission. This is summed up in Eq. 1, with $V_i$ standing for the inherent vulnerability coming with the granting of a given dangerous permission $i$.[14]

$$V_{i=1..24} = \left\{ \begin{array}{cc} 1 & \text{if the permission is requested} \\ 0 & \text{otherwise} \end{array} \right\} \quad (1)$$

### 4.2. AndroBugs extraction

As for the AndroBugs report, we parse it to extract vulnerabilities tagged Critical or Warning (see Section 2.3). To quantify the collected information for every vulnerability(V), we give a weight of 1 to Critical, 0.5 to Warning, and 0 otherwise. This is summed up in Eq. 2, with $V_i$ standing for Critical or Warning-level security vulnerabilities in our dataset.

$$V_{i=1..41} = \left\{ \begin{array}{cc} 1 & \text{if } V_i \text{ is present and Critical} \\ 0.5 & \text{if } V_i \text{ is present and a Warning} \\ 0 & \text{if } V_i \text{ is not present} \end{array} \right\} \quad (2)$$

### 4.3. Code smell extraction

We used regular expressions to parse the Smali code and extract security code smells defined in Gadient et al. (2017) and used successfully in Habchi et al. (2019) and Gadient et al. (2019).

After which, we compute (see Eq. 3) for each vulnerability posed by a security code smell, a ratio indicating the relative presence of that vulnerability; said ratio is obtained by dividing the number of identified instances of the code smell ($NS_i$) by the number of lines of code in the Smali format ($LOC_{SMALI}$).

$$V_{i=1to9} = \frac{NS_i}{LOC_{SMALI}} * 100 \quad (3)$$

## 5. AndroVul-D: the dataset

Androvul-D is our dataset of 78 vulnerability metrics collected on a sample of Android apps from AndroZoo. Fig. 3 illustrates the process through which AndroVul-D was generated and can serve as a blueprint for other researchers willing to generate vulnerability datasets for their own sample of AndroZoo apps. The figure starts with a researcher (carefully) selecting the Android apps she wants in her study or preliminary tests, and follows up with the application of the scripts of AndroVul-T to generate csv files filled with vulnerability metrics about each app of the dataset. Furthermore, since AndroZoo does not have all the information related to the apps it archived, the researcher may, as we did, have to go fetch some metadata (e.g., category) about an app from its store. Additionally, a researcher may have to add known malwares from other sources.

### 5.1. Data selection

For our data selection, we resorted to the AndroZoo data-set which contained 5,848,157 apps when we started our investigation. The AndroZoo dataset proposes data on the APKs it archived in a main CSV file containing important information for each application, including hash keys (such as sha256, sha1, md5), size information (for APKs and DEX), date of the binary, package name, version code, market place as well as information about how well the app fared on the VirusTotal website (number of antiviruses that flag the app as a malware, scan date).[15] Using a sample size calculator,[16] we computed that to get a representative sample with very high confidence level (99%) and confidence interval (1%), we ought to consider 16,586 apps. After removing duplicates and some entries that are not actual apps, we ended up with 16,180 APKs that were downloaded and used as input for the AndroVul-T scripts.

To complement the above data, we resorted to data from VirusShare[17] as a way to obtain malware data that has been validated as such. VirusShare is a website that collects virus data, whether from desktop or mobile software, from a variety of sources (Zhu et al., 2018). The data is offered in big archive files with malwares for desktop or mobile environments. We used two uploaded archive files (dating from 2019–08-08 & 2019–06-02), which comprises around 127 K desktop or mobile apps, of which we were able to recover 3,978 Android APK files. These files come with no information other than a hash for file integrity.

### 5.2. Dataset structure

The dataset we propose consists of CSV files containing information (as illustrated in Fig. 4) about the 16,180 apps from AndroZoo and the 3,978 apps from VirusShare (Forensics, 2020), one app per line. There are 78 columns in the file, each with a header clearly indicating the information it provides. There are four types of information in the CSV:

1. Information from the AndroZoo dataset, if applicable,[18] as described in Section 5.1
2. The nine (9) code smells extracted from the reverse engineered Smali code (see Table 9 in the Appendix)
3. The twenty-four (24) dangerous permissions, as parsed from the app's manifest file
4. The forty (40) metrics derived from the six types of vulnerabilities provided by AndroBugs

Overall, the file contains 78 metrics about info from AndroZoo, dangerous permissions, Smali code smells and AndroBugs-tagged vulnerabilities.

[13] https://ibotpeaches.github.io/Apktool/.
[14] $i$ is the index assigned to a given possible vulnerability in our dataset of vulnerabilities.
[15] More information herehttps://AndroZoo.uni.lu/lists.
[16] https://www.surveysystem.com/sscalc.htm.
[17] https://virusshare.com/
[18] Malicious apps from VirusShare do not have any info other than a hash.

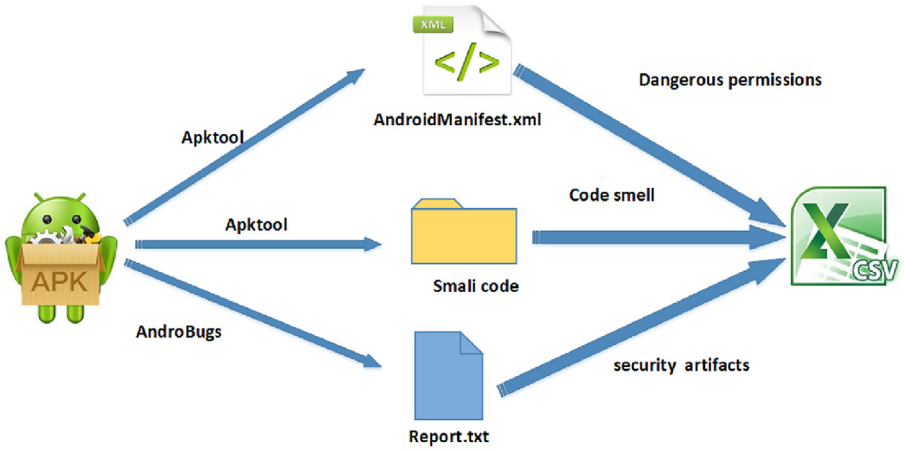


**Fig. 1.** Overview of the AndroVul tool.

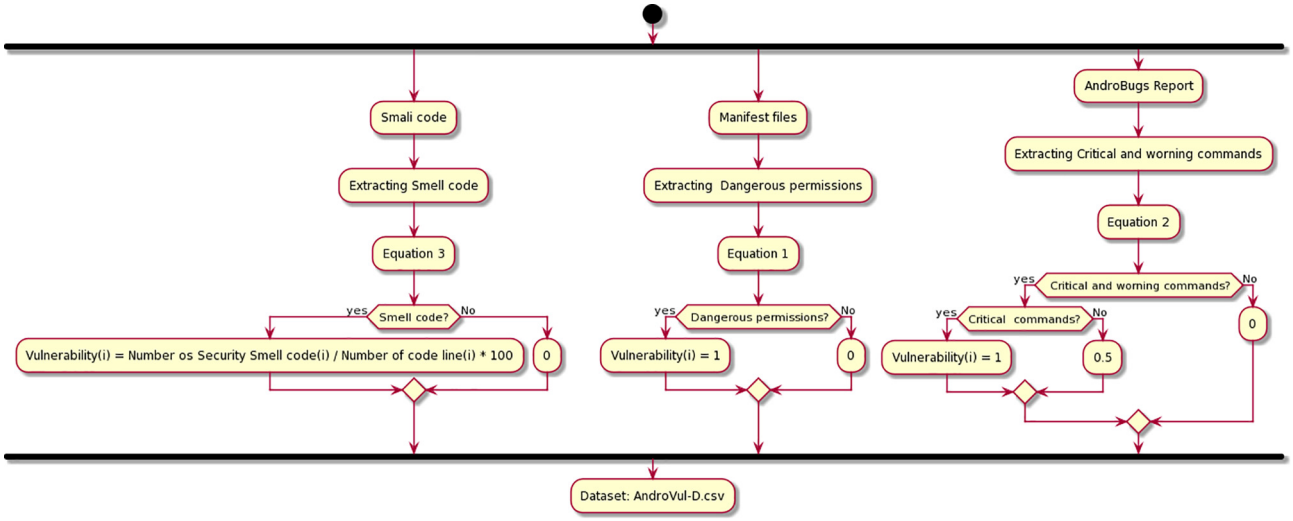


**Fig. 2.** Parsing and Quantifying vulnerability data (i refers to the specific vulnerability).

### 5.3. Dataset description

**Apps from the AndroZoo dataset:** Here, we provide some descriptive statistics on our data-sets, relatively to the date, the category, the store, APK size and number of antivirus flags. With respect to the binary dates, 3.37% of the apps display an unreliable date (1980). About 1 out of 4 apps are from 2016 to 2018, 2 out of 3 apps are from 2014 to 2018. The APK sizes range from 7 KB to 330 MB, with an average of 9 MB and a standard deviation of 12 MB. Marketplace–wise, the most dominant stores are the Google Play Store (74%), appchina (10%), mi.com (2%) and anzhi (1%). When it comes to information from the antiviruses of VirusTotal,[19] the apps in the dataset have between 0 (74% of the apps) and 40 flags, out of the 63 antiviruses; the average is 2 flags and the standard deviation is 5.11. We also collected data related to the apps' categories. A sizable part of the apps (about 43%) could not be mapped to a category, mainly because they are no longer available on the market stores. Another 14% of the apps are only available in Chinese markets. Overall, we could find the category information for only 43% of the apps. Table 1 lists all the categories that make for at least 1% of the dataset.

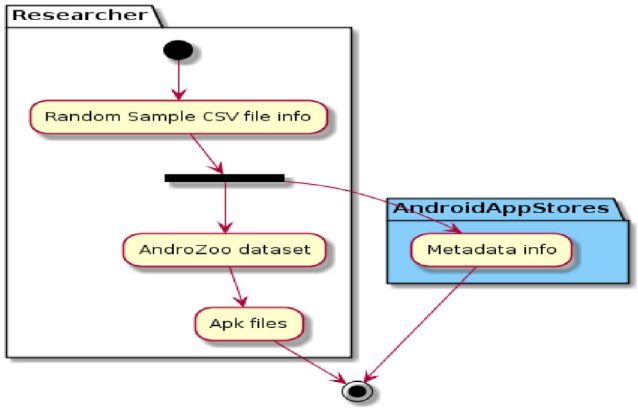


**Fig. 3.** Data Selection and Gathering.

**Apps from the VirusShare dataset:** There are no additional information coming with the APKs in this dataset. We took interest in getting an estimate about how many of these malicious apps would be classified as such, using a threshold of 3 antivirus flags from VirusTotal, threshold that we used in our previous work (Namrud et al.,

[19] https://www.virustotal.com/.

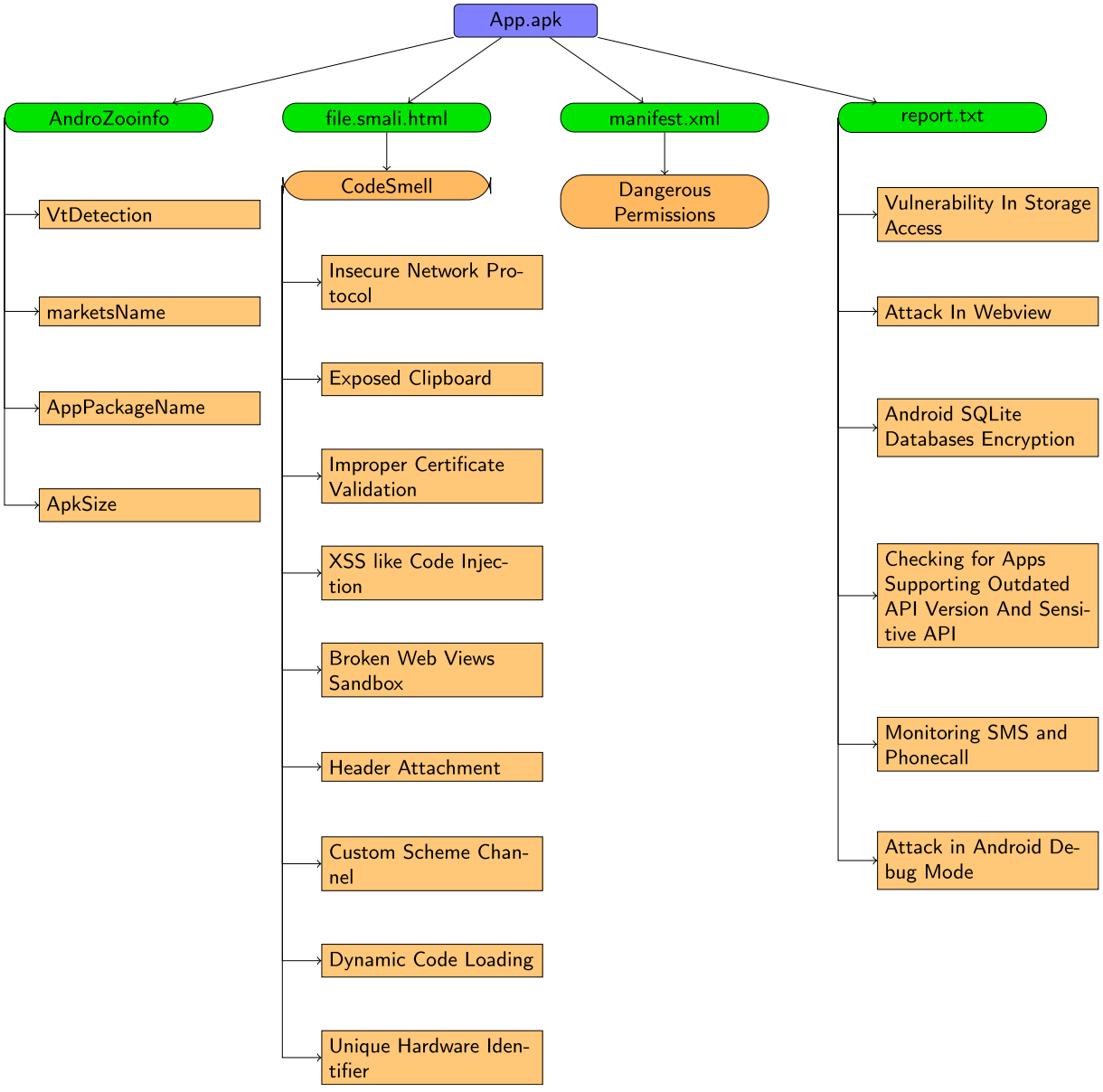


**Fig. 4.** Information and vulnerabilities extracted from an Android app.

**Table 1**
App categories in our dataset.

| CATEGORY | Proportion |
|---|---|
| GAME | 1231 (7.61%) |
| EDUCATION | 571 (3.53%) |
| LIFESTYLE | 540 (3.34%) |
| BUSINESS | 437 (2.70%) |
| ENTERTAINMENT | 423 (2.62%) |
| TOOLS | 416 (2.57%) |
| PERSONALIZATION | 397 (2.45%) |
| BOOKS AND REFERENCE | 373 (2.31%) |
| TRAVEL AND LOCAL | 307 (1.90%) |
| MUSIC AND AUDIO | 282 (1.74%) |
| NEWS AND MAGAZINES | 254 (1.57%) |
| PRODUCTIVITY | 245 (1.52%) |
| HEALTH AND FITNESS | 211 (1.31%) |
| FINANCE | 193 (1.19%) |
| COMMUNICATION | 169 (1.05%) |
| SPORTS | 165 (1.02%) |
| SOCIAL | 161 (1.00%) |

2019). There were issues in automating the scanning process from VirusTotal, so we proceeded with a small random sample of 94 apps, giving us a confidence level of 95% with an interval of 20%.[20] We found that 72% of these apps were flagged three times or more.[21]

## 6. Study design

In this section, we lay out the design of our study, from preliminary sanity checks to research questions and experimental design.

### 6.1. Identifying malwares

A first point to be addressed is the identification of malwares in the AndroZoo dataset. Androzoo does not explicitly tag apps as

[20] as computed from https://www.surveysystem.com/sscalc.htm.
[21] This means we can be 95% confident that between 52% and 92% of the malicious Android apps from VirusShare (Forensics, 2020) have 3 or more antivirus flags from VirusTotal.

malwares. Rather, it provides the number of antiviruses from VirusTotal that flagged the app, with malware researchers left to decide which number of antivirus flags is enough to label an app from AndroZoo as a malware.

In addition to AndroZoo, we gathered malware apps from the VirusShare (Forensics, 2020) repository to strengthen the generalisability of our experiments.

Overall, we took into account three malware datasets:

1. AZM: AndroZoo apps flagged as possible malwares.
2. VSM: malware dataset from VirusShare.
3. MM: dataset mixing malwares from AndroZoo and VirusShare.

### 6.2. Correlation analysis on the Androzoo data

To get a quick initial sense of how much the collected metrics can contribute to malware detection, we first proceeded to some statistical correlation analysis of the metrics to: i) the number of antivirus flags; and ii) a binary value representing the benign/malicious classification: 0 for benign or 1 for malware (with the threshold of 3 flags used in our previous work (Namrud et al., 2019) to label an app as a malware). We computed Pearson correlations for all metrics and found some interesting values in all 3 categories:

- for permissions, *READ_PHONE_STATE* returns the highest correlations with respectively 0.35 for the number of antivirus flags and 0.38 for a benign/malware decision;
- for code smells, the *Dynamic Code Loading* metric yields 0.4 for the number of flags and 0.38 for the binary decision;
- and for Androbugs, the vulnerability *Using critical function* returns 0.34 (number of flags) and 0.31 (binary decision) as correlation values.

All three metrics mentioned above returned p-values significantly lower than 0.05 (the commonly accepted statistical significance threshold), as is the case for all but a few metrics in the dataset.

### 6.3. Used classifiers

We selected four classifiers representing four types of machine learning algorithms commonly used in the Android malware research community. More precisely, we used the well known machine learning software Weka and selected NaiveBayes (NB) from its *bayes* category, MLP classifier from its *function* category, JRip from its *rules* category, and J48 from its *tree* category as shown in Table 2. Using these classifiers, we proceeded to the commonly used statistical method that is the K-fold cross-validation. In short, it consists in splitting, after random shuffling, the dataset in K groups; after which, each group is used as a test group while the other K-1 groups are used for training. More specifically, we chose, in accordance to many similar studies (e.g., Bhattacharya and Goswami, 2017), K = 10 for a 10-fold cross validation study, in which 90% of the data is used for training and 10% for testing (prediction).

### 6.4. Feature selection

The features (vulnerability metrics in our case) extracted from our data constitute a relatively large set that is likely to contain some duplication. To tackle this, along with reducing risks of overfitting, feature selection is to be considered. It allows identifying the best features and excluding the least important features. To do so, it generally relies on assessing the information gained or lost by adding or removing a particular feature. Various techniques have been proposed and implemented in tools for this purpose.

**Table 2**
Selected one Classifies form each four known Machine learning categories.

| Classifier | Category |
|---|---|
| MLP | Function |
| NaiveBayes | Bayes |
| JRip | Rules |
| J48 | Tree |

In this work, we relied on the well-established open source machine learning software Weka and selected the following three attribute evaluators: ChiSquaredAttributeEval (CS), InfoGainAttributeEval (IG), and ReliefFAttributeEval (RF).

### 6.5. Performance indicators

The application of classifier results in decisions about individual apps that can be quantitatively evaluated through various measures. As it relates to the detection of malwares, we refer to True Positive (TP) as the number of malwares actually classified as such, True Negative (TN) as the number of benign apps classified as such, False Positive (FP) as the number of benign apps wrongly classified as malwares and finally False Negative (FN) the number of malwares wrongly classified as benign. From these basic measures are derived more insightful measures, commonly used in malware detection research work, such as:

- Precision: It is the ratio of actual malwares in the set of apps classified as such: TP/(TP + FP)
- Recall: It is the ratio of malwares that were detected as such: TP/(TP + FN)
- Accuracy: It is the percentage of correctly classified apps: (TP + TN)/(TP + TN + FP + FN)
- F1-Measure: It is a performance indicator that takes into account both precision and recall of the obtained classification: 2 * (Recall * Precision)/ (Recall + Precision)
- Area under ROC Curve (AUC): It is a measure of the predictive power of the classifier that basically informs on how much the model is capable of distinguishing between classes (here benign apps vs malwares).

For all these measures, the higher, the better, with 1 being the perfect value.

### 6.6. Research questions

Our research questions flow from the above considerations (as laid out in the previous subsections) and aim at answering the following research questions:

**RQ1**: Which classifiers and feature selection techniques perform the best? **RQ2**: Relatively to Androzoo, which subset of apps should be labeled as malwares? More specifically, how many antivirus flags from VirusTotal are enough to label an app from the AndroZoo dataset as a malware.

To answer these questions, we propose the following experiments based on different slicing of the AndroVul data.

### 6.7. Answering the research questions

Our research questions are designed around two key elements pertaining to malware classification: i) the input (the "best" dataset to use) and ii) the treatment (the "best" machine learning technique to choose).

**Table 3**
Information about the datasets' sizes and the selected thresholds for all experiments.

| | AZ Benign flag— Size | AZ Malware flag—Size | VS Malware Size | All Malware Size |
|---|---|---|---|---|
| Exp 0 | **0**—11,928 | **1+** — 4,201 | 3,978 | 8,179 |
| Exp 1 | **0–1** — 13,086 | **2+** — 3,042 | 3,978 | 7,020 |
| Exp 2 | **0–2** — 13,553 | **3+** — 2,621 | 3,978 | 6,599 |
| Exp 3 | **0–4** — 13,933 | **5+** — 2,195 | 3,978 | 6,173 |
| Exp 4 | **0–9** — 14,713 | **10+** — 1,397 | 3,978 | 5,375 |
| Exp 5 | **0–19** — 15,722 | **20+** — 406 | 3,978 | 4,487 |
| Exp 6 | **0–1** — 13,086 | **10+** — 1,397 | 3,978 | 5,375 |

The first point is an important one when it comes to using AndroZoo for malware classification. Given that AndroZoo only provides the number of antivirus flags (from VirusTotal) for a given app, researchers typically have to decide which threshold to use to consider a given app malicious or benign. The choice of a threshold is somewhat arbitrary and rarely motivated so, in the following, we propose experiments to probe the choice of a good threshold through the lens of its contribution to effective malware classification, especially when it comes to classifying correctly confirmed malwares (elements from VirusShare). We mainly focus on a single threshold below which apps are considered benign: thresholds of 1, 2, 3, 5, 10 and 20 but also consider one experiment with two thresholds: i) benign apps being those apps with 1 or zero antivirus flags, and ii) malware apps being those apps with 10 or more flags.

The choices outlined above delineate subsets of our benchmark that ought to be clarified. Taking as an example the threshold 1, meaning apps with one or more flags are considered as malwares, we define three (3) malware datasets:

1. $AZM_1$: the set of AndroZoo apps with 1 or more antivirus flags,
2. *VSM*: the set of malwares from VirusShare (Forensics, 2020), and
3. $MM_1$: the mixed set of apps from $AZM_1$ and *VSM*.

In single threshold experiments, the choice of a threshold also defines which apps from AndroZoo should be considered benign; in this case, these are the apps with 0 antivirus flags: $AZB_0$. This means that, with a threshold of 1 for malware decision, the complete datasets we use as input for malware classification are the following:

1. $AZB_0$ (benign apps) $\cup$ $AZM_1$ (malicious apps),
2. $AZB_0$ (benign apps) $\cup$ *VSM* (malicious apps), and
3. $AZB_0$ (benign apps) $\cup$ $MM_1$ (malicious apps).

Therefore, our experiments explore respectively the antivirus flag thresholds 1 ($AZB_0$ and $AZM_1$), 2 ($AZB_1$ and $AZM_2$), 3 ($AZB_2$ and $AZM_3$), 5 ($AZB_4$ and $AZM_5$), 10 ($AZB_9$ and $AZM_{10}$), and 20 ($AZB_{19}$ and $AZM_{20}$) as well as a double threshold 1 and 10 ($AZB_1$ and $AZM_{10}$).[22]

Throughout these experiments, we computed and analyzed the effectiveness of our selected classifiers (JRIP, NB, MPL, and J48) and feature selection techniques (CS, IG, and RF). To assess the results of the experiments, we relied mostly on the F1-score and the AUC measure, which are standard metrics recognised as more robust than, for instance, the accuracy measure.

The answers to our two research questions come from the analysis and comparisons of these numbers intra-experiment (RQ1: best classifiers and feature selection techniques) and inter-experiment (RQ2: best input data).

[22] Some of these configurations resulted in imbalanced data; so we used a SpreadSubsample instance filter as an imbalance reduction technique.

## 7. Experiments and results

The experiments defined in Section 6.7 are summarised in Table 3, which display, for each experiment (Exp), data about its AndroZoo benign apps (antivirus flags and data size), its AndroZoo malwares (antivirus flags and data size), its VirusShare malwares (data size), and the combined set of AndroZoo and VirusShare malwares (data size). The following sections present and discuss the performance metrics for AUC & F1 obtained from these experiments.

### 7.1. Results

Tables 4–6 present the results of all experiments with no feature selection. A first very clear output of our experiments is that the various feature selection techniques only very marginally affected the results. We thus decided, for simplicity's sake, against displaying them in the summary tables.

Table 4 presents the experiments using **AndroZoo data only**. Below are the main observations we can draw from it.

- Experiment 0 posts the worst results (AUC: 0.77–0.85, F1: 0.72–0.80).
- Experiments 1 & 2 (AUC: 0.82–0.87, F1: 0.76–0.82) propose very similar results, with values within 0.01 from one experiment to the other, suggesting unsurprisingly that there is not much difference between using two or three as the number of anti-virus flags needed to label an app as a malware.
- The same goes for Experiments 3 and 4 (AUC: 0.82–0.89, F1: 0.78–0.83), which provide results at most 0.02 from one another. Additionally, aside from the classifier NB, results from experiments 3 and 4 are relatively close to those from Experiments 1 & 2.
- Experiment 5 (AUC: 0.85–0.92, F1: 0.82–0.85) and Experiment 6 (AUC: 0.88–0.94, F1: 0.82–0.88) provide results that are distinct but suggest that higher malware thresholds translate into better results for the classifiers. It should be noted, however, for Experiment 6, which proposes the best performance measures that it leaves out a lot of apps (any app having between 3 and 15 flags).
- Overall, when it comes to classifier performance, MPL is the leading classifier, with J48 coming in second.

Table 5 presents the experiments using **AndroZoo data for benign apps and VirusShare for malwares**. The main insights are as follows:

- Results are generally better than those involving only AndroZoo apps and around 0.90 in most cases, indicating the effectiveness of the vulnerability metrics in correctly classifying VirusShare malwares.
- MPL and JRIP are the leading classifiers and exhibit remarkable consistency over the range of experiments, with their results always within 0.03 from one experiment to the other. Similar

**Table 4**
AUC and F1 results when considering only AndroZoo apps <Benign & Malicious>.

| ***AUC*** | *JRIP* | *NB* | *MPL* | *J48* |
|---|---|---|---|---|
| Exp 0 | 0.81 | 0.77 | 0.81 | **0.85** |
| Exp 1 | 0.84 | 0.82 | **0.86** | 0.83 |
| Exp 2 | 0.85 | 0.83 | **0.87** | 0.83 |
| Exp 3 | 0.84 | 0.85 | **0.89** | 0.83 |
| Exp 4 | 0.86 | 0.86 | **0.89** | 0.82 |
| Exp 5 | 0.85 | 0.91 | **0.92** | 0.85 |
| Exp 6 | 0.89 | 0.90 | **0.94** | 0.88 |
| ***F1*** | *JRIP* | *NB* | *MPL* | *J48* |
| Exp 0 | 0.78 | 0.72 | 0.77 | **0.80** |
| Exp 1 | **0.82** | 0.76 | 0.80 | **0.82** |
| Exp 2 | 0.81 | 0.76 | 0.81 | **0.82** |
| Exp 3 | **0.83** | 0.78 | **0.83** | **0.83** |
| Exp 4 | **0.83** | 0.80 | **0.83** | 0.82 |
| Exp 5 | 0.82 | 0.83 | **0.85** | **0.85** |
| Exp 6 | 0.86 | 0.82 | **0.88** | 0.87 |

**Table 5**
AUC and F1 results when considering Benign apps from <AndroZoo> & Malicious apps from <VirusShare>.

| ***AUC*** | *JRIP* | *NB* | *MPL* | *J48* |
|---|---|---|---|---|
| Exp 0 | 0.92 | 0.88 | **0.94** | 0.90 |
| Exp 1 | 0.92 | 0.88 | **0.93** | 0.90 |
| Exp 2 | **0.92** | 0.87 | **0.92** | 0.90 |
| Exp 3 | 0.91 | 0.89 | **0.93** | 0.89 |
| Exp 4 | 0.91 | 0.86 | **0.92** | 0.88 |
| Exp 5 | **0.92** | 0.85 | 0.91 | 0.88 |
| Exp 6 | 0.92 | 0.88 | **0.93** | 0.91 |
| ***F1*** | *JRIP* | *NB* | *MPL* | *J48* |
| Exp 0 | **0.90** | 0.77 | 0.88 | 0.87 |
| Exp 1 | **0.88** | 0.77 | **0.88** | 0.86 |
| Exp 2 | **0.88** | 0.74 | **0.88** | 0.86 |
| Exp 3 | **0.88** | 0.85 | 0.87 | 0.85 |
| Exp 4 | **0.88** | 0.74 | 0.86 | 0.85 |
| Exp 5 | **0.88** | 0.71 | 0.86 | 0.79 |
| Exp 6 | **0.89** | 0.77 | 0.87 | 0.86 |

**Table 6**
AUC and F1 results when considering Benign apps from <AndroZoo> & Malicious apps from <VirusShare + Androzoo>.

| ***AUC*** | *JRIP* | *NB* | *MPL* | *J48* |
|---|---|---|---|---|
| Exp 0 | 0.85 | 0.81 | **0.87** | 0.82 |
| Exp 1 | 0.86 | 0.83 | **0.89** | 0.76 |
| Exp 2 | 0.86 | 0.83 | **0.90** | 0.79 |
| Exp 3 | 0.87 | 0.84 | **0.90** | 0.86 |
| Exp 4 | 0.87 | 0.84 | **0.90** | 0.85 |
| Exp 5 | 0.89 | 0.85 | **0.91** | 0.75 |
| Exp 6 | 0.89 | 0.87 | **0.92** | 0.87 |
| ***F1*** | *JRIP* | *NB* | *MPL* | *J48* |
| Exp 0 | **0.81** | 0.69 | 0.80 | 0.75 |
| Exp 1 | 0.82 | 0.72 | **0.83** | 0.79 |
| Exp 2 | 0.82 | 0.71 | **0.83** | 0.79 |
| Exp 3 | 0.83 | 0.72 | **0.83** | 0.82 |
| Exp 4 | **0.84** | 0.72 | 0.83 | 0.82 |
| Exp 5 | **0.85** | 0.71 | **0.85** | 0.79 |
| Exp 6 | **0.86** | 0.76 | **0.86** | 0.84 |

(but somewhat lower) consistency can be observed with J48, whereas the classifier NB proposes results that fall into a broader and generally lower range, depending on the experiment, and especially for the F1 measure (0.71–0.85).

- There's no trend in the effectiveness of the classifiers as the malware thresholds get higher, although the results from Experiment 0 (one antivirus flag is enough to classify an AndroZoo app as a malware) are almost always the best ones.

Table 6 presents the experiments using **only AndroZoo data for benign and AndroZoo + VirusShare for malwares**. The results propose a pattern very similar to that of Table 4 (AndroZoo data only): worst results for Experiment 0, Experiments 1 & 2 as well as 3 & 4 being very close, better results as malware thresholds rise, etc.

### 7.2. Analysis and discussion of the experiments

MPL is clearly the leading classifier. It was the best (or close) classifier in all the experiments. Furthermore, looking beyond the AUC and F1 measures, it provided balanced performance for precision and recall, with their values almost always above 0.80 (and

**Table 7**
Comparison with related work.

| Papers | Tool | Accuracy | F1 | AUC |
|---|---|---|---|---|
| Bhattacharya and Goswami (2017) | Weka | 0.77 | 0.86 | 0.82 |
| Sharma and Sahay (2018) | Weka | 0.79 | / | / |
| Sachdeva et al. (2018) | Weka | 0.93 | / | / |
| AndroVul | Weka | 0.92 | 0.87 | 0.92 |

mostly above 0.85) and within 0.02 from one another meaning that the classifier provides a classification with roughly equal precision and recall. JRIP is a close second; while it visibly trails MPL on the AUC measure, it is as good as (and possibly slightly better than) MPL on the F1 measure. The worst classifier is consistently the Naive Bayes (NB) one.

Table 7 compares the results we obtained against those from recent papers. To represent our experiments, we used the average of values obtained for the experiments involving benign AndroZoo apps and malwares from VirusShare (Table 5). The table shows that the results we obtained compare mostly favorably to the ones from previous studies.

**Findings from RQ 1**

- The vulnerability metrics in our dataset classify mostly correctly benign and malicious Android apps, and their results compare favorably to recent state-of-the-art papers.
- MPL is the leading classifier; JRIP is a close second.
- Feature selection techniques are not impactful and ultimately not needed.

As for the best dataset, the emerging picture is mixed: the strictest approach (only apps with no antivirus flags are considered benign) performed slightly better on VirusShare dataset (confirmed malwares) but marginally worse on dataset configurations involving AndroZoo apps labeled as malwares. On the other hand, the laxer approaches (highest thresholds) performed equally or slightly better than mid-range thresholds. Still, the main takeaway is that the differences between the results obtained from these thresholds are small, suggesting that the choice of the threshold may not matter that much.

**Findings from RQ 2**

- Our results suggest that the choice of a given number of antivirus flags to decide whether to label. an AndroZoo app as malware is not really consequential when it comes, notably, to classify proven malwares (from VirusShare).
- The strictest threshold for benign app classification (0 antivirus flag) performed best with the VirusShare Dataset but lagged behind other dataset configurations when the malware datasets include AndroZoo apps.

## 8. Limitations and threats to validity

The present paper extends our previous paper (Namrud et al., 2019) and set out to provide a more complete benchmark for malware classification, along with empirical data about the effectiveness of different classifiers and the impact of possible data labeling decisions. As any work, it comes with limitations and some validity threats.

The main limitation of our dataset is that many of the apps do not have complete metadata (category, ratings, etc.). AndroZoo does not store that info so there is a need to retrieve it from the different stores. That retrieval may be hampered by the following scenarios: i) the apps may no longer be available in those stores; and ii) the stores themselves do not offer simple ways to automatically get the info. In these cases, HTML parsers have to be written for webpages that could be in other languages or have structures that can (and do) change. We plan future work to complement the data through the parsing of various app store clones that keep app metadata even after their deletion from the main market places.

As it relates to our study, some threats to validity are worth noting. An external threat to validity of our results is that we mostly used a sample of AndroZoo, which itself does not account for all Android Apps. However, AndroZoo is a widely used repository and we took pains to extract a random sample large enough to be reasonably representative of AndroZoo. That sample covers several different domains (see Table 1): game, education, sports, and tools, to name a few. Furthermore, different from our previous paper, we retrieved additional malware dataset, which we believe help mitigate these external validity concerns.

As for threats to internal validity, it is worth noting that our investigation of different classifiers and feature selection techniques relied mostly on default settings from the widely used statistical tool Weka. Obviously, it is possible that our results and the ranking of these classifiers could be altered with different settings or more elaborate versions of these classifiers. However, to the best of our knowledge and based on the existing literature, malware researchers generally do not engage in complex parameter tuning of the classifiers they use. Furthermore, such parameter tuning may depend on a research expertise with a given classifier and may introduce additional variability. Overall, we believe that our results provide malware researchers with an informed perspective about which classifiers to choose or avoid, especially if they do not intend to dedicate a significant amount of time to tune the parameters used by these classifiers.

Overall, we believe that the provided dataset and the accompanying experiments can be used to guide research ideas for anomaly detection, mining of safe/ dangerous patterns, etc. In addition, the accompanying scripts that we provide offer options for researchers needing their own datasets. By relying on the instructions available on the AndroVul GitHub repository,[23] they can add their APKs (in the apks folder) and run the provided scripts.

[23] https://github.com/Zakeya/AndroVul.

## 9. Conclusion

The ubiquity of smartphones, and their growing use make the security of these devices as important as that of standard computers. In this paper, we proposed a repository for Android vulnerabilities and experiments on classifier performances for different benchmarks (taken from the repository) to better support the research community engaged with anomaly detection and security issues for Android apps. Our contributions are threefold. First, we proposed a tool that harnesses well-known reverse engineering tools and greatly simplifies the generation of diverse vulnerability information (i.e. dangerous permissions, vulnerabilities from AndroBugs, and code smells in Smali code) for any app. Second, we proposed vulnerability data on a random sample of 16,180 Android apps downloaded from the well-established AndroZoo dataset, which we extended with 3,978 malwares retrieved from the VirusShare repository. Our tool and data make it so that an Android app researcher can start applying statistic analysis and machine learning experiments right away on our benchmark or right after downloading his/her own set of APKs. Third, we proposed detailed studies that provide: i) insights into the very good predictive power of the vulnerability information mined by our tool; and ii) information into which classifiers and data labelling decisions perform better. Our tool and data samples are available on GitHub and we intend to build and extend on that repository, notably by working on recovering more completely apps metadata.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Appendix A

Tables 8–10.

**Table 8**
Dangerous Permissions (Definitions from Android) (Developer.android.com, 2020).

| Permission | Description |
|---|---|
| READ_CALENDAR | Allows an application to read the user's calendar data. |
| WRITE_CALENDAR | Allows an application to write the user's calendar data. |
| CAMERA | Required to be able to access the camera device. |
| READ_CONTACTS | Allows an application to read the user's contacts data. |
| WRITE_CONTACTS | Allows an application to write the user's contacts data. |
| GET_ACCOUNTS | Allows access to the list of accounts in the Accounts Service. |
| ACCESS_FINE_LOCATION | Allows an app to access precise location. |
| ACCESS_COARSE_LOCATION | Allows an app to access approximate location. |
| RECORD_AUDIO | Allows an application to record audio. |
| READ_PHONE_STATE | Allows read only access to phone state, including the phone number of the device,current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device. |
| READ_PHONE_NUMBERS | Allows read access to the device's phone number(s). |
| CALL_PHONE | Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call. |
| ANSWER_PHONE_CALLS | Allows the app to answer an incoming phone call. |
| READ_CALL_LOG | Allows an application to read the user's call log. |
| WRITE_CALL_LOG | Allows an application to write (but not read) the user's call log data. |
| ADD_VOICEMAIL | Allows an application to add voicemails into the system. |
| USE_SIP | Allows an application to use SIP service. |
| PROCESS_OUTGOING_CALLS | Allows an application to see the number being dialed during an outgoing call with the option to redirect the call to a different number or abort the call altogether. |
| BODY_SENSORS | Allows applications to discover and pair bluetooth devices. |
| SEND_SMS | Allows an application to send SMS messages. |
| RECEIVE_SMS | Allows an application to receive SMS messages. |
| READ_SMS | Allows an application to read SMS messages. |
| RECEIVE_WAP_PUSH | Allows an application to receive WAP push messages. |
| RECEIVE_MMS | Allows an application to monitor incoming MMS messages. |
| READ_EXTERNAL_STORAGE | Allows an application to read from external storage. |
| WRITE_EXTERNAL_STORAGE | Allows an application to write to external storage. |

**Table 9**
Regular expressions used in our tool containing Smali type for code smell.

| Smell | Symptom in Smali Code and Corresponding Escaped Regexes |
|---|---|
| Custom Scheme Channel | Scheme registration code exists Lorg/apache/http/conn/scheme/SchemeRegistry;− > register*Lorg/apache/http/conn/scheme/Scheme*; Lorg/apache/http/conn/scheme/Scheme |
| Header Attachment | Header attachment code exists Lorg/apache/http/client/methods/HttpGet;− > addHeader*Ljava/lang/String*;*Ljava/lang/String*; |
| Unique Hardware Identifier | Hardware identifier access code for MACs and IMEI exists Landroid/telephony/TelephonyManager;-\\ getDeviceId\(\) Ljava/lang/String Landroid/bluetooth/BluetoothAdapter; − >, getAddress\(\)\, Ljava/lang/StringLandroid/net/wifi/WifiInfo; − >, getMacAddress \(\), Ljava/lang/String |
| Exposed Clipboard | Clipboard manipulation code existsLandroid/content/ClipboardManager; − >, getPrimaryClip\(\), Landroid/content/ClipDataLandroid/content/ClipboardManager; − >, setPrimaryClip Landroid/content/ClipData; |
| Exposed Clipboard | Clipboard manipulation code exists Landroid/content/ClipboardManager; − >, getPrimaryClip\(\) Landroid/content/ClipData Landroid/content/ClipboardManager; − >, setPrimaryClip*Landroid/content/ClipData*; |
| Insecure Network Protocol | Http connection establishment code exists Ljava/net/HttpURLConnection;− > *Ljava/net/URL*; |
| Improper Certificate Validation | Customised certificate validation code existsïmplements Ljavax/net/ssl/X509TrustManager; |
| Dynamic Code Loading | Dynamic code loading mechanism exists Landroid/content/Context;− > createPackageContext*Ljava/lang/String*;*I* Landroid/content/Context |
| XSS-like Code Injection | WebView JavaScript setting code exists Landroid/webkit/WebSettings;− > setJavaScriptEnabled*Z* |
| Broken WebView's Sandbox | WebView Java interface code exists Landroid/webkit/WebView;− > addJavascriptInterface\(Ljava/lang/Object; Ljava/lang/String;\) |

**Table 10**
Severity Level Artifacts.

| Severity Level | Description |
|---|---|
| Critical | Confirmed security vulnerability that should be solved (except for testing code) |
| Warning | AndroBugs Framework is not sure if this is a security vulnerability. Developers need to manually confirm. |
| Notice | Low priority issue or AndroBugs Framework tries to let you know some additional information. |
| Info | No security issue detected. |

## References


Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y., 2016. Androzoo: collecting millions of android apps for the research community. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), IEEE, pp. 468–471.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C., 2014. Drebin: Effective and explainable detection of android malware in your pocket., in: Ndss, pp. 23–26..

Bhattacharya, A., Goswami, R.T., 2017. Dmdam: data mining based detection of android malware, in: Proceedings of the first international conference on intelligent computing and communication, Springer. pp. 187–194..

Bhattacharya, A., Goswami, R.T., 2018. Community based feature selection method for detection of android malware. J. Global Inf. Manage. 26, 54–77.

developer.android.com, 2020.https://developer.android.com/reference/android/Manifest.permission. [Online; accessed June 29, 2020]..

Forensics, C., 2020.https://virusshare.com/. [Online; accessed June 29, 2020]..

Gadient, P., Nierstrasz, O., Ghafari, M., 2017. Security in android applications. PhD diss., Master s thesis. University of Bern..

Gadient, P., Ghafari, M., Frischknecht, P., Nierstrasz, O., 2019. Security code smells in android icc. Empirical Software Eng. 24, 3046–3076.

Geiger, F.X., Malavolta, I., Pascarella, L., Palomba, F., Di Nucci, D., Bacchelli, A., 2018. A graph-based dataset of commit history of real-world android apps. In: Proceedings of the 15th International Conference on Mining Software Repositories, pp. 30–33.

Gkortzis, A., Mitropoulos, D., Spinellis, D., 2018. Vulinoss: a dataset of security vulnerabilities in open-source systems. In: Proceedings of the 15th International Conference on Mining Software Repositories, pp. 18–21.

Habchi, S., Moha, N., Rouvoy, R., 2019. The rise of android code smells: who is to blame? In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), IEEE, pp. 445–456.

Krutz, D.E., Mirakhorli, M., Malachowsky, S.A., Ruiz, A., Peterson, J., Filipski, A., Smith, J., 2015. A dataset of open-source android applications. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, IEEE, pp. 522–525.

Lee, C., Ko, E., Lee, K., 2020. Methods to select features for android malware detection based on the protection level analysis. International Conference on Information Security Applications, Springer., 375–386

Li, L., Bissyandé, T.F., Le Traon, Y., Klein, J., 2016. Accessing inaccessible android apis: an empirical study. In: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp. 411–422.

Li, L., Li, D., Bissyandé, T.F., Klein, J., Le Traon, Y., Lo, D., Cavallaro, L., 2017. Understanding android app piggybacking: a systematic study of malicious code grafting. IEEE Trans. Inf. Forensics Secur. 12, 1269–1284.

Li, W., Wang, Z., Cai, J., Cheng, S., 2018. An android malware detection approach using weight-adjusted deep learning. In: 2018 International Conference on Computing. Networking and Communications (ICNC), IEEE, pp. 437–441.

Ma, Z., Ge, H., Liu, Y., Zhao, M., Ma, J., 2019. A combination method for android malware detection based on control flow graphs and machine learning algorithms. IEEE Access 7, 21235–21245.

Mansourov, N., Campara, D., 2010. System assurance: beyond detecting vulnerabilities. Elsevier.

Munaiah, N., Klimkowsky, C., McRae, S., Blaine, A., Malachowsky, S.A., Perez, C., Krutz, D.E., 2016. Darwin: a static analysis dataset of malicious and benign android apps. In: Proceedings of the International Workshop on App Market Analytics. pp. 26–29..

Namrud, Z., Kpodjedo, S., Talhi, C., 2019. Androvul: a repository for android security vulnerabilities. In: Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, pp. 64–71.

Sachdeva, S., Jolivot, R., Choensawat, W., 2018. Android malware classification based on mobile security framework. IAENG Int. J. Comput. Sci. 45, 514–522.

Sharma, A., Sahay, S.K., 2018. An investigation of the classifiers to detect android malicious apps. Information and Communication Technology. Springer, 207–217.

Shen, T., Zhongyang, Y., Xin, Z., Mao, B., Huang, H., 2014. Detect android malware variants using component based topology graph. In: 2014 IEEE 13th international conference on trust, security and privacy in computing and communications, IEEE, pp. 406–413.

Shezan, F.H., Afroze, S.F., Iqbal, A., 2017. Vulnerability detection in recent android apps: an empirical study. In: 2017 International Conference on Networking, Systems and Security (NSysS). IEEE, pp. 55–63.

Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W., 2017. Deep ground truth analysis of current android malware. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 252–276.

Yousefi-Azar, M., Hamey, L.G., Varadharajan, V., Chen, S., 2018. Malytics: a malware detection scheme. IEEE Access 6, 49418–49431.

Zheng, M., Lee, P.P., Lui, J.C., 2012. Adam: an automatic and extensible platform to stress test android anti-virus systems. International conference on detection of intrusions and malware, and vulnerability assessment, Springer, 82–101.

Zhu, H.J., You, Z.H., Zhu, Z.X., Shi, W.L., Chen, X., Cheng, L., 2018. Droiddet: effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing 272, 638–646.